

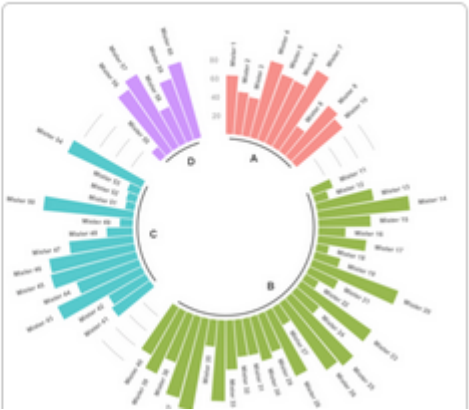
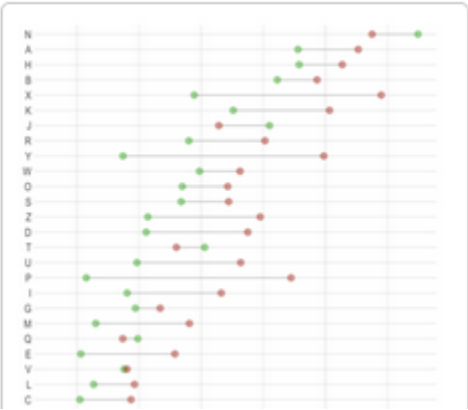
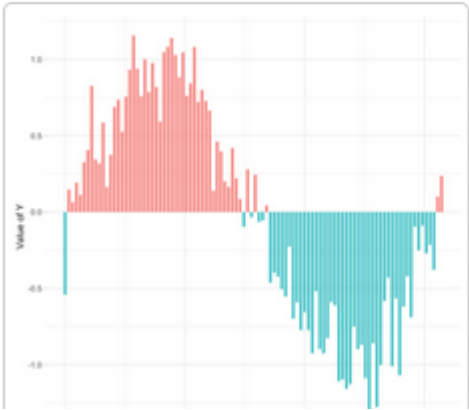
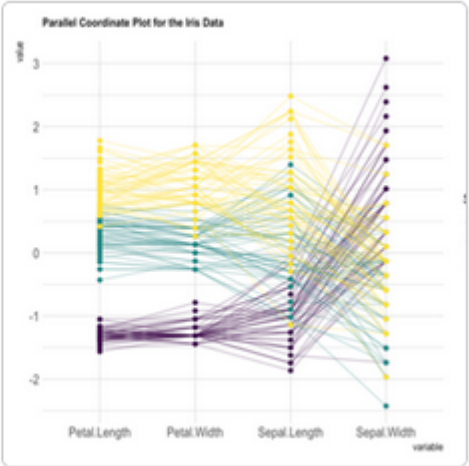
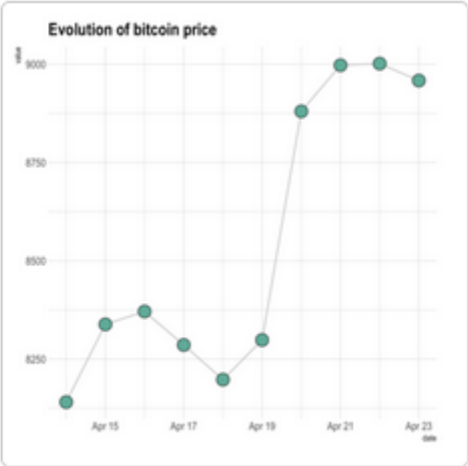
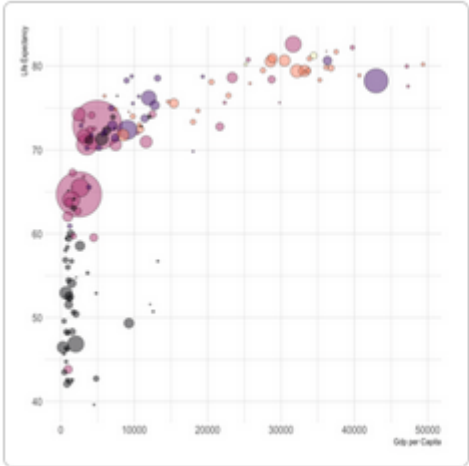
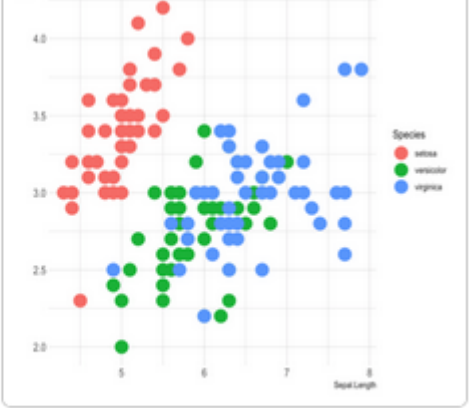
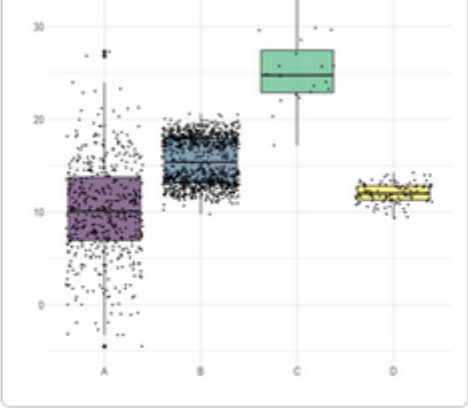
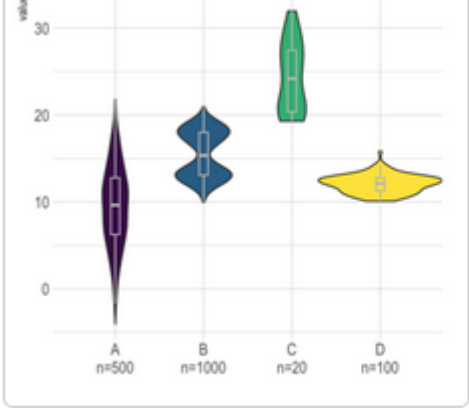
Plotting data

with **ggplot2**



A. Ginolhac | rworkshop | 2021-09-09

ggplot2



About this lecture

Learning objectives

- Learn the basic grammar of graphics
- Understand how it is implemented in `ggplot2`
 - Input data structures as `data.frame/tibble`
 - Mapping columns to display features (*aesthetics*)
 - Types of graphics (*geometries*)
 - Multiple and repeating graphics (*facets*)
 - Transforming plots (*scales*)
 - Using different coordinate systems
 - Customizing graphs with *themes*
- Make quick exploratory plots of your multidimensional data.



Introduction

ggplot2

- stands for **g**rammar of **g**raphics plot v2
- Inspired by Leland Wilkinson work on the [grammar of graphics](#) in 2005.

Idea: split a graph into layers

- such as axis, curve(s), labels.
- **3** elements are required: **data**, **aesthetics**, **geometry** ≥ 1

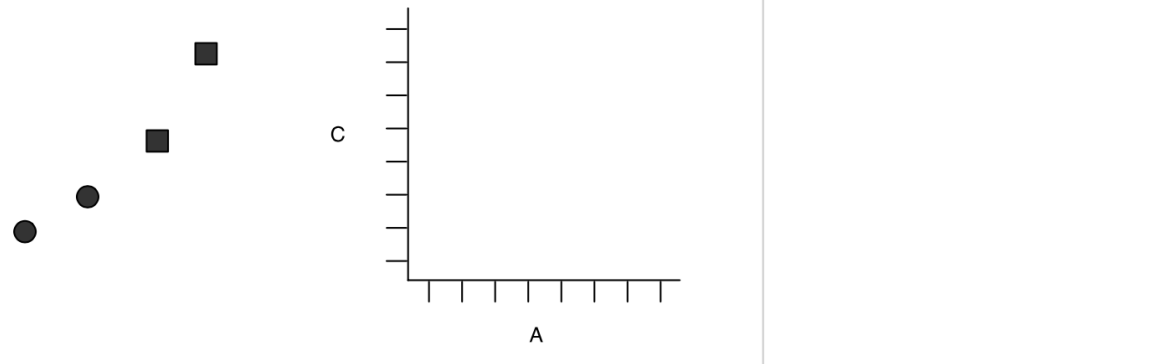


Figure 1. Graphics objects produced by (from left to right): geometric objects, scales and coordinate system, plot annotations.

[Wickham H. 2007. J. Comp. Graph. Stat. 19:3–28](#)

ggplot2 layers

Layers

Theme
Coordinates
Statistics
Facets
Geometries
Aesthetics
Data



Adapted from this [article](#) by [Colin Fay](#) ([thinkR](#))

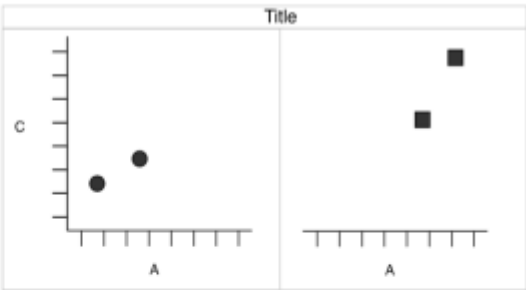
ggplot2 layers

Layers

Theme
Coordinates
Statistics
Facets
Geometries
Aesthetics
Data



Example



classic
cartesian
identity
shape
geom_point
x, y, shape

x	y	shape
25	11	circle
0	0	circle
75	53	square
200	300	square

Adapted from this [article](#) by [Colin Fay](#) ([thinkR](#))

Basic example

data

x	y	shape
25	11	circle
0	0	circle
75	53	square
200	300	square

aesthetics

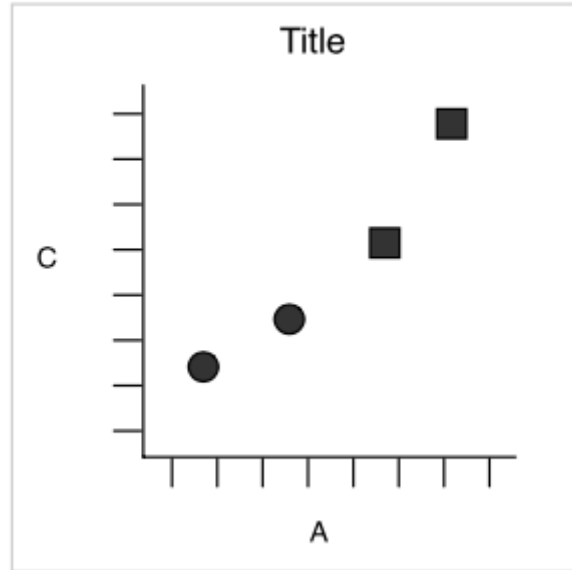
- $x = x$
- $y = y$
- $shape = shape$

geometry

- **dot / point**

Wickham: [A Layered Grammar of Graphics](#) (2007)

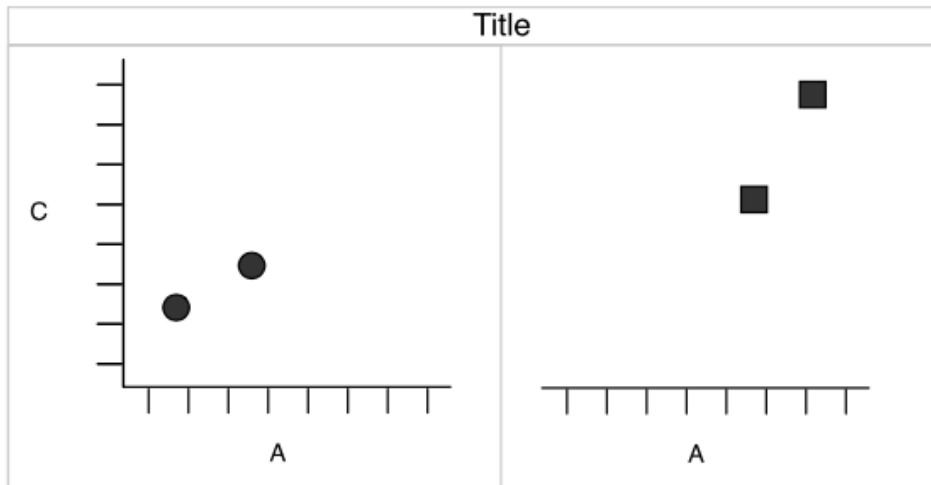
Result



What if we want to split into panels *circles* and *squares*?

Faceting

Split by shape, *aka* trellis or lattice plots



Redundancy

- **shape** and **facets** provide the same information.
- The **shape** aesthetic is free for another variable.

Wickham: [A Layered Grammar of Graphics](#) (2007)

Faceting can also carry a message



[geofacet, US states](#) by Ryan Hafen

Layers

Theme
Coordinates
Statistics
Facets
Geometries
Aesthetics
Data



data

x	y	shape
25	11	circle
0	0	circle
75	53	square
200	300	square

Motivation for this layered system

Data visualisation is not meant just to be seen but to be **read**, like written text

— Alberto Cairo

Combine layers but not with pipes!

Warning

`ggplot2` layers are combined with `+`, not `%>%`, `|>`.

This introduces **a break** in the **workflow**. (`ggplot1` would have been fine)



`ggplot` lines combined with `+`

```
library(ggplot2)
swiss %>%
  ggplot(aes(x = Education,
             y = Examination)) +
  geom_point() +
  scale_colour_brewer()
```

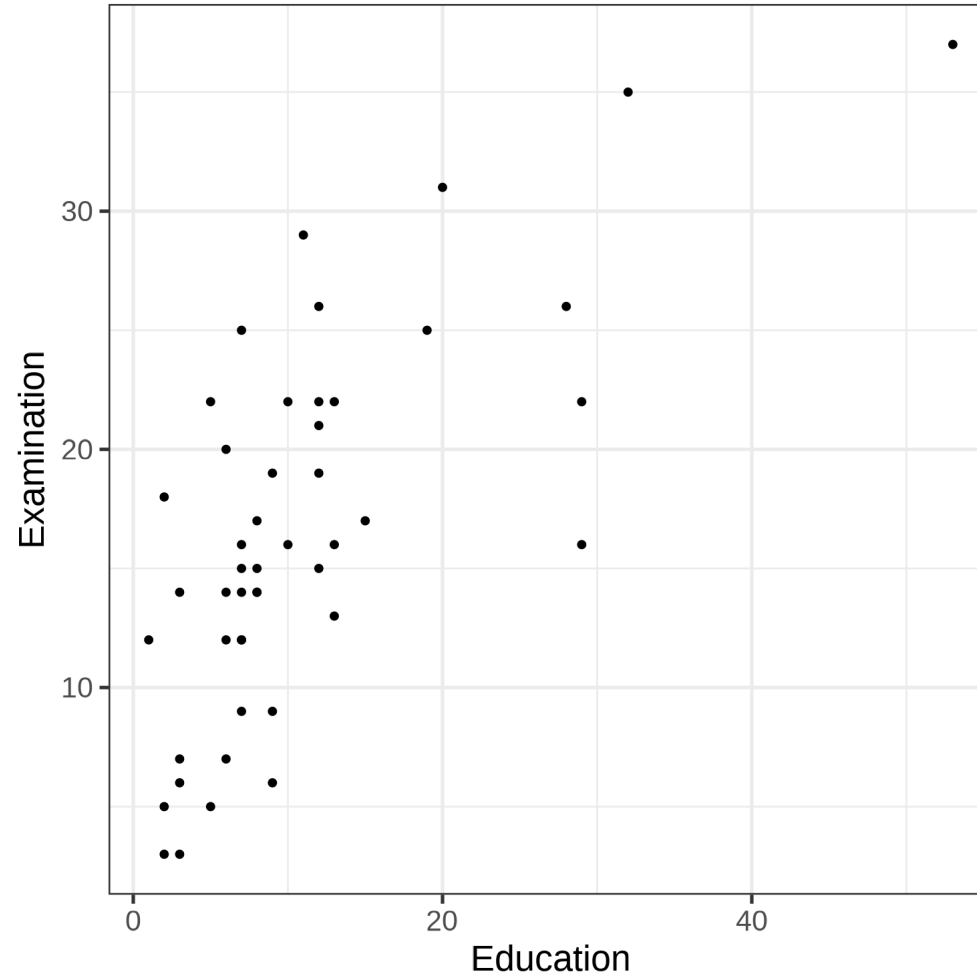
if not, the error is explicit

```
swiss %>%
  ggplot(aes(x = Education,
             y = Examination)) %>%
  geom_point() +
  scale_colour_brewer()
```

```
Error: `mapping` must be created by `aes()`
Did you use %>% instead of +?
```


Build a plot by layers

```
swiss %>%  
  ggplot() +  
  aes(x = Education) +  
  aes(y = Examination) +  
  geom_point() +  
  theme_bw(18)
```



Palmer penguins

CHINSTRAP!

GENTOO!

ADÉLIE!



install with `install.packages("palmerpenguins")`



Horst AM, Hill AP, Gorman KB (2020). [palmerpenguins: Palmer Archipelago \(Antarctica\) penguin data](#). R package v0.1.0

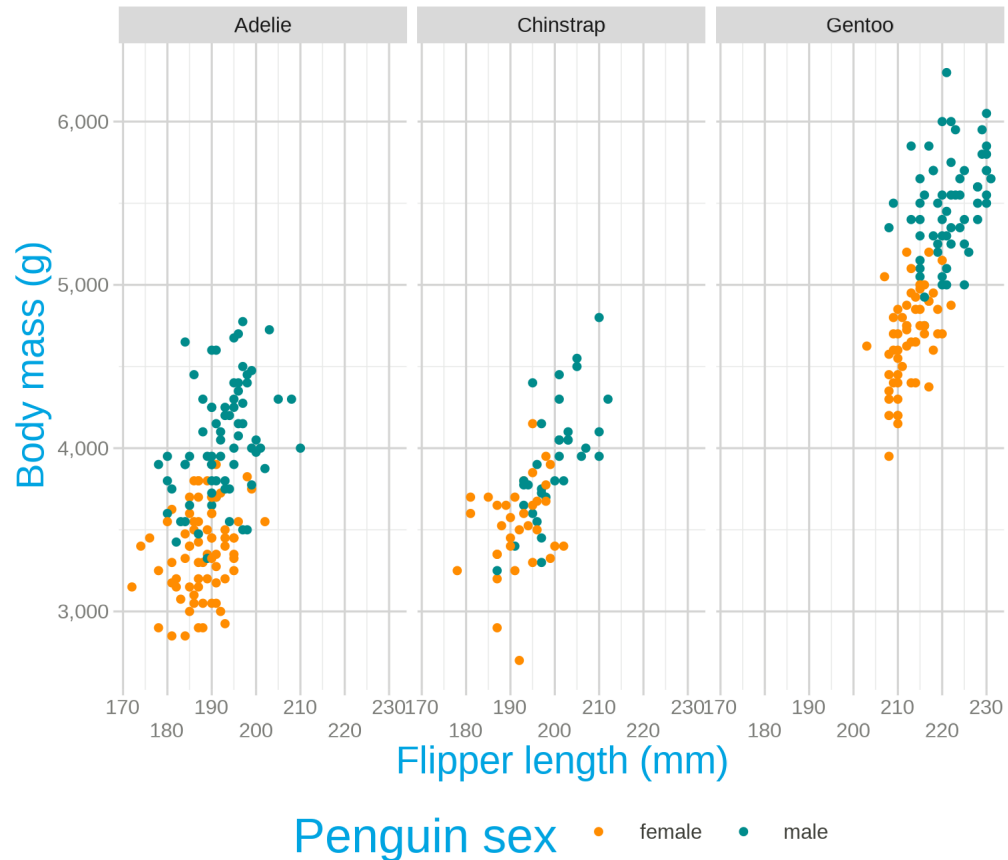
@allison_horst

A more interesting example

```
library(palmerpenguins)
penguins %>%
  ggplot() +
    aes(x = flipper_length_mm,
        y = body_mass_g) +
    aes(color = sex) +
    geom_point() +
    theme_xaringan(text_font = "Roboto Condensed", text_font_size = 14,
                    scale_color_manual(values = c("darkorange", "cyan4"), na.rm = TRUE)) +
    labs(title = "Penguin flipper and body mass",
         caption = "Horst AM, Hill AP, Gorman KB (2020)",
         subtitle = "Dimensions for male/female Adelie, Chinstrap and Gentoo Penguins at Palmer Station",
         theme(plot.subtitle = element_text(size = 13)) +
    labs(x = "Flipper length (mm)",
         y = "Body mass (g)",
         color = "Penguin sex") +
    theme(legend.position = "bottom",
          legend.background = element_rect(fill = "white", color = "black",
                                            corner.radius = 10)) +
    theme(plot.caption = element_text(hjust = 0, face = "italic", size = 10)) +
    theme(plot.caption.position = "plot") +
    facet_wrap(~ species) +
    scale_x_continuous(guide = guide_axis(n.dodge = 2)) +
    scale_y_continuous(labels = scales::comma)
```

Penguin flipper and body mass

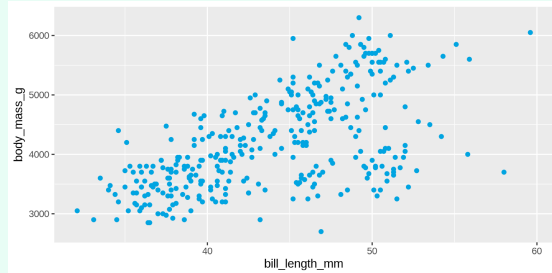
Dimensions for male/female Adelie, Chinstrap and Gentoo Penguins at Palmer Station



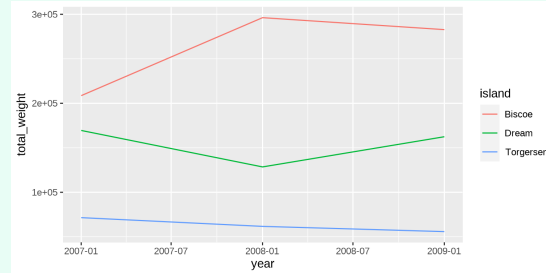
Horst AM, Hill AP, Gorman KB (2020)

Geometric objects define the plot type to be drawn

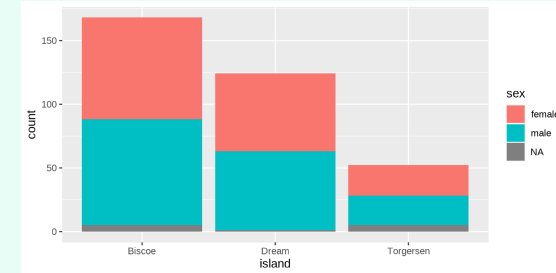
`geom_point()`



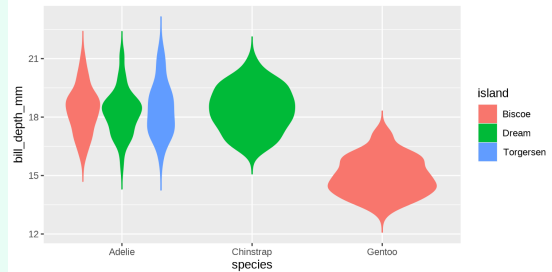
`geom_line()`



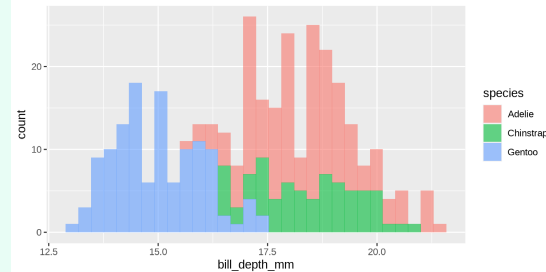
`geom_bar()`



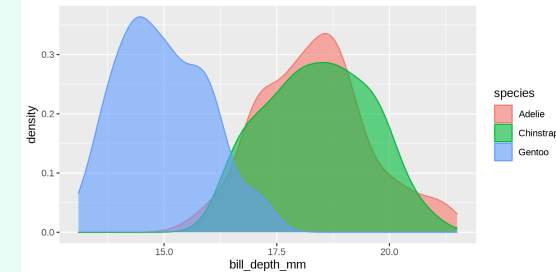
`geom_violin()`



`geom_histogram()`



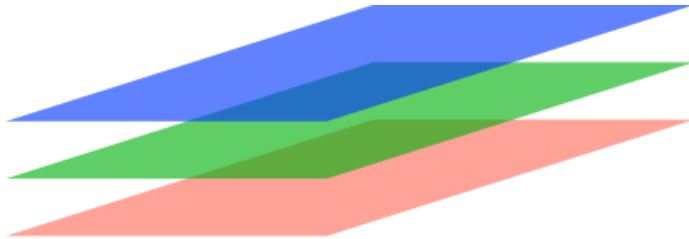
`geom_density()`



Have a look at the [cheatsheet](#) or the [documentation](#) for more possibilities.

Core layers

Geometries
Aesthetics
Data



Other layers

they are present, it works because they have *sensible* default:

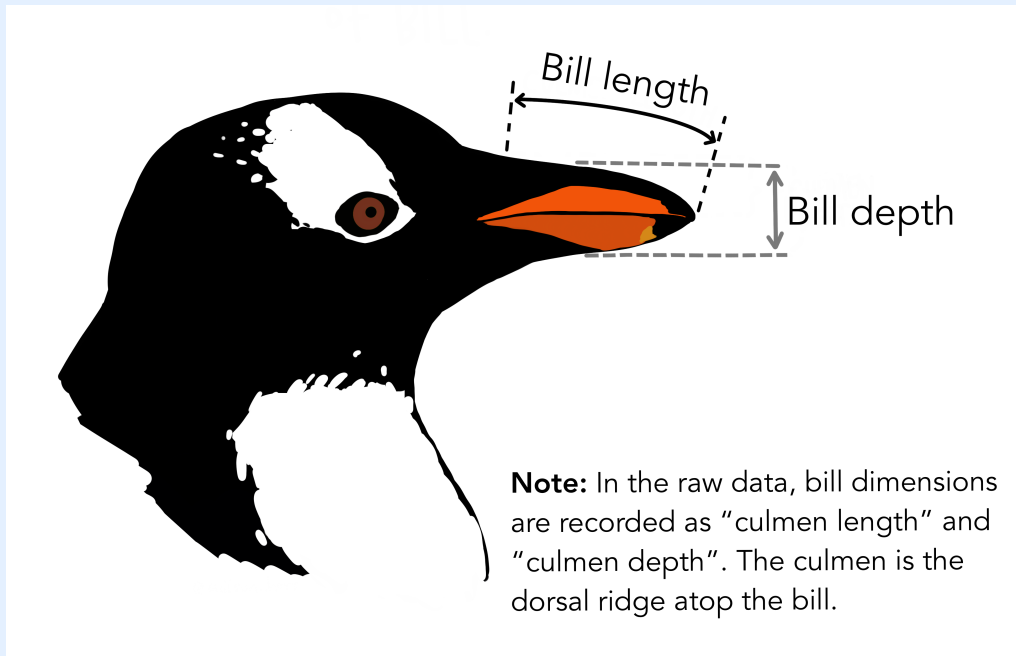
- **theme** is `theme_grey`
- **coordinate** is `cartesian`
- **statistic** is `identity`
- **facets** are `disabled`

3 layers are enough

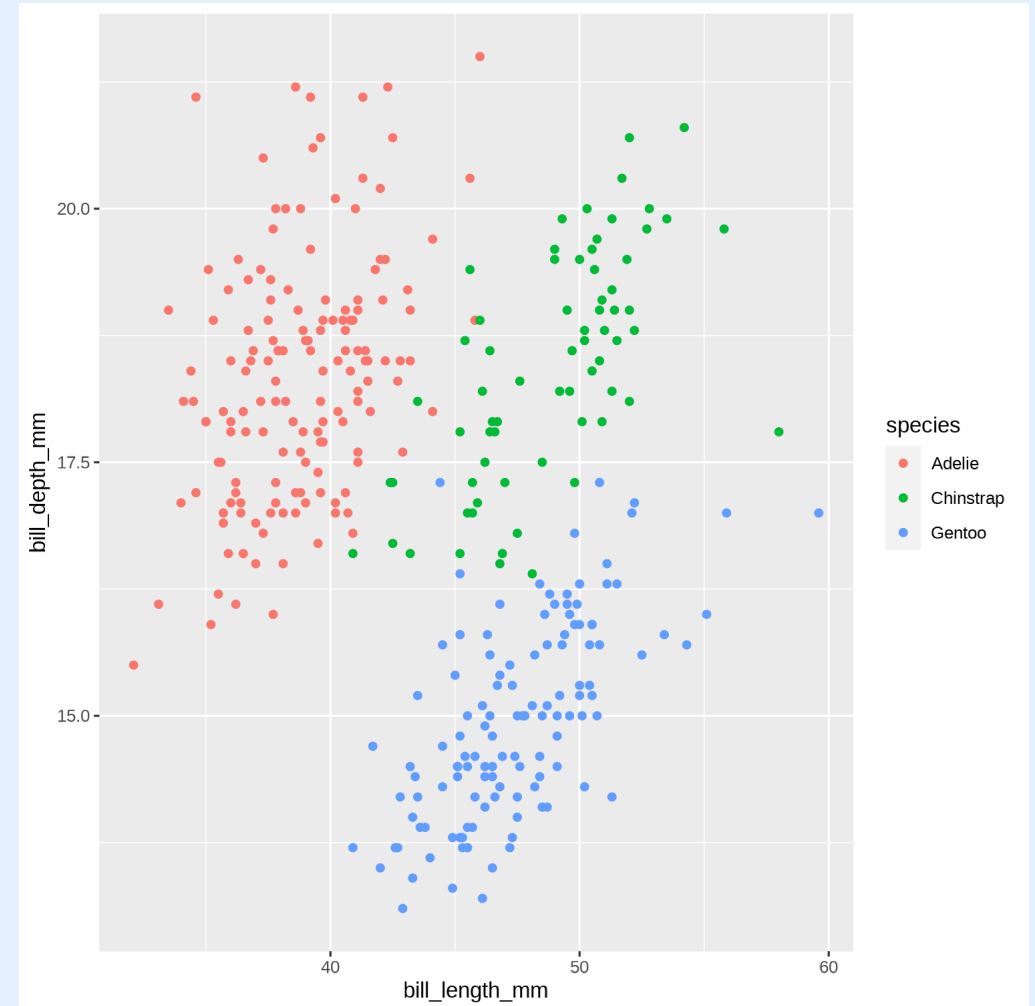
- **data**
- **aesthetics** mapping data to plot component
- **geometry** at least one

Your first plot

```
library(palmerpenguins)
library(ggplot2)
ggplot(data = penguins) +
  geom_point(mapping = aes(x = bill_length_mm,
                           y = bill_depth_mm,
                           colour = species))
```



Artwork by [@allison_horst](https://twitter.com/allison_horst)



Mapping aesthetics

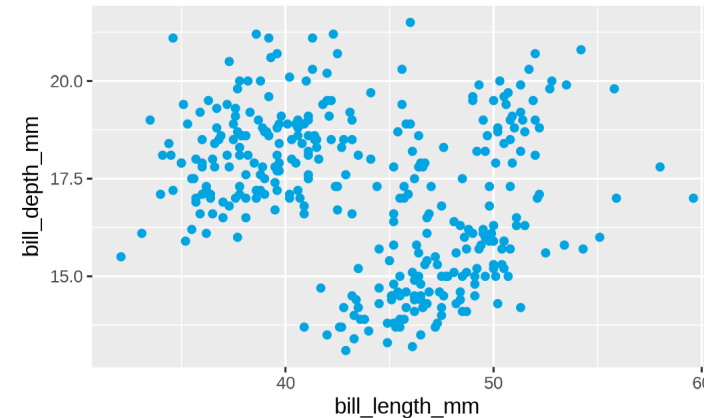
Requirements

- `aes()` map columns/variables **data** to **aesthetics**
- specific **geometries** (`geom`) have different expectations:
 - univariate, one **x** or **y** for flipped axes
 - bivariate, **x** and **y** like scatterplot
- **continuous** or **discrete** variables
 - **continuous** for color → **gradient**
 - **discrete** for color → **qualitative**

`geom_point()` requires **both** **x** and **y** coordinates

```
ggplot(penguins) +  
  geom_point(aes(x = bill_length_mm,  
                 y = bill_depth_mm))
```

- same as previous slide
- **without** **colour** mapping
- on your laptop, default to **black**

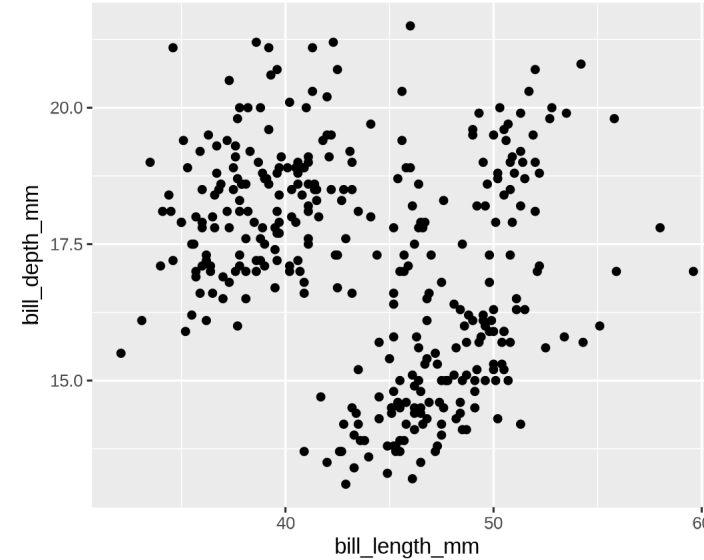


NB: `mapping` = and `data` = are often skipped.

Unmapped parameters

- `geom_point()` accepts additional arguments such as the `colour`
- Define them to a **fixed** value **without** mapping

```
ggplot(penguins) +  
  geom_point(aes(x = bill_length_mm,  
                 y = bill_depth_mm),  
             colour = "black")
```



Important

Parameters defined **outside** the aesthetics `aes()` are applied to **all** data.

Mapped parameters

mapped parameters require two conditions:

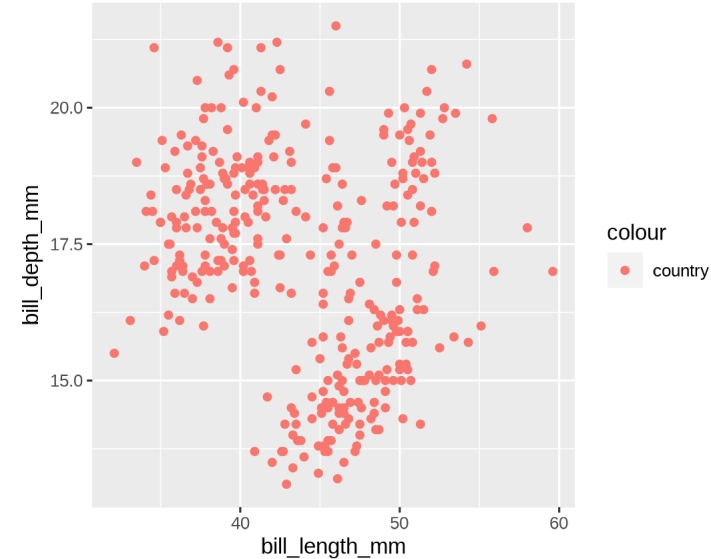
- being defined **inside** the aesthetics `aes()`
- refer to one of the column data, here: **mistake**

```
ggplot(penguins) +  
  geom_point(aes(x = bill_length_mm,  
                 y = bill_depth_mm,  
                 colour = country))
```

```
Error in FUN(X[[i]], ...): object 'country' not found
```

- passing the unknown column as **string** as a different effect:

```
ggplot(penguins) +  
  geom_point(aes(x = bill_length_mm,  
                 y = bill_depth_mm,  
                 colour = "country"))
```

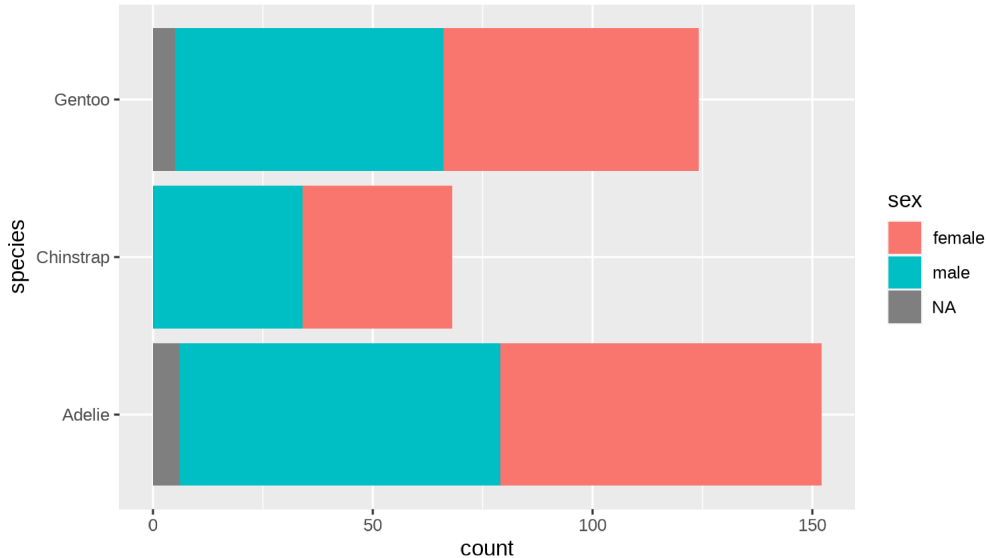


- this is hardly useful, but we shall see an application later.
- stick to the 2 mapping rules:
 - in `aes()`
 - refer to a valid column.

Mapping aesthetics correctly

In `aes()` *and* refer to a data column

```
ggplot(penguins) +  
  geom_bar(aes(y = species,  
              fill = sex))
```



`species` and `sex` are 2 **valid columns** in `penguins`

Advantages:

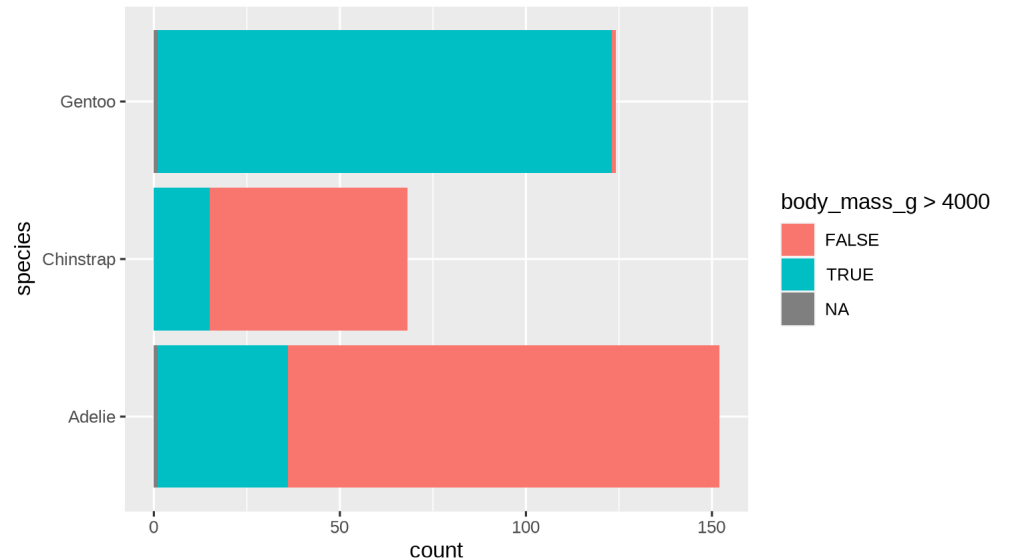
- The **legend** is/are for free
- Missing data are highlighted in grey
- Using **y** axis for categories eases reading

How not using a string for mapping is useful?

Fair question

- could we pass an **expression**?
- which penguins are above 4 kg?
- use `body_mass_g > 4000` that return a boolean to find out

```
ggplot(penguins) +  
  geom_bar(aes(y = species,  
               fill = body_mass_g > 4000))
```

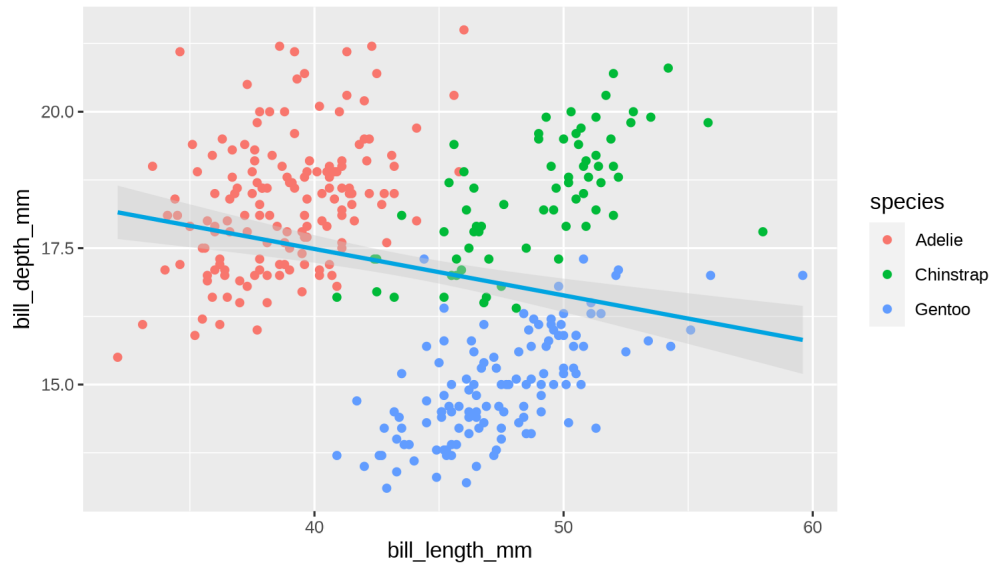


The **expression** was evaluated in `penguins` context
Obvious that **Gentoo** are bigger than the 2 other species

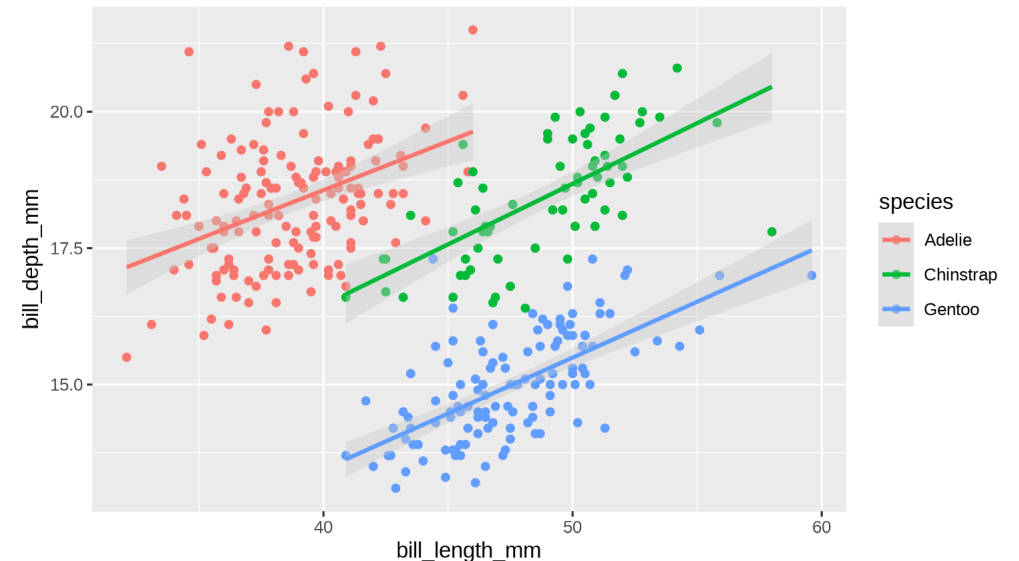
Inheritance of arguments across layers

Compare the two following (great example of a Simpson's paradox):

```
ggplot(penguins,  
  aes(x = bill_length_mm,  
      y = bill_depth_mm)) +  
  geom_point(aes(colour = species)) +  
  geom_smooth(method = "lm", formula = 'y ~ x')
```



```
ggplot(penguins,  
  aes(x = bill_length_mm,  
      y = bill_depth_mm,  
      colour = species)) +  
  geom_point() +  
  geom_smooth(method = "lm", formula = 'y ~ x')
```

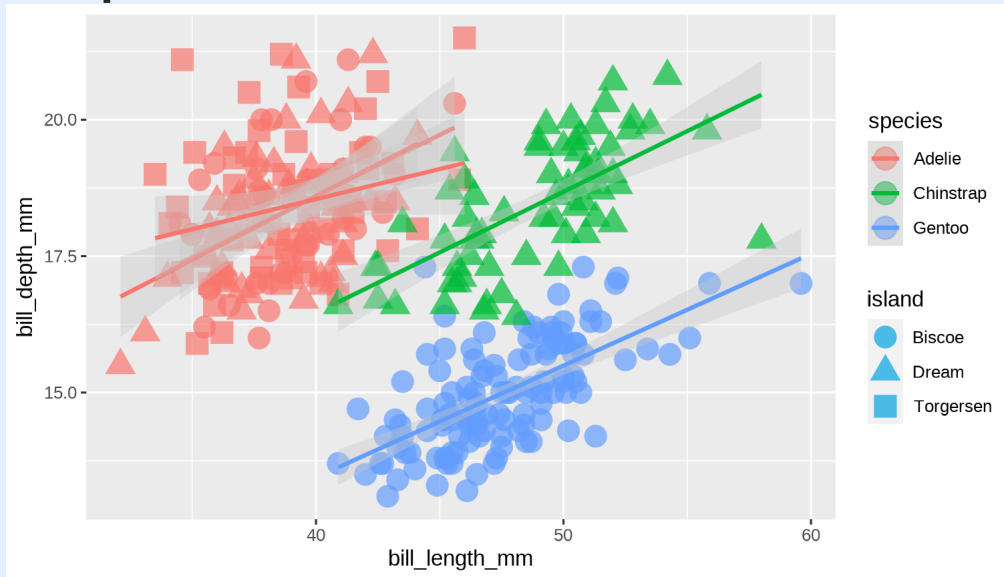


- **aesthetics** in `ggplot()` are **passed on** to **all** geometries.
- **aesthetics** in `geom_*()` are **specific** (and can **overwrite** inherited)

Try it

- map the **island** variable to a **shape** aesthetics for **both** dots and linear models
- **all** dots (circles / triangles / squares) with:
 - a size of **5**
 - a transparency of 30% (**alpha** = **0.7**)

Aim plot:



Answer:

```
ggplot(penguins,  
  aes(x = bill_length_mm,  
      y = bill_depth_mm,  
      shape = island,  
      colour = species)) +  
  geom_point(size = 5, alpha = 0.7) +  
  geom_smooth(method = "lm",  
              formula = 'y ~ x')
```

05:00

Joining observations

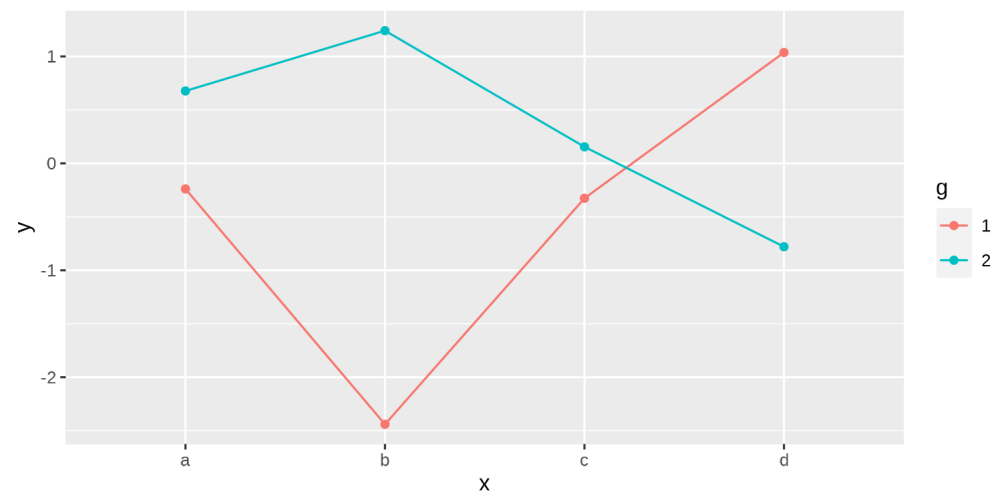
```
set.seed(212) # tidyr::crossing generate combinations
tib <- tibble(crossing(x = letters[1:4],
                      g = factor(1:2)),
             y = rnorm(8))
```

Suppose we want to **connect** dots by colors

Should be the job of `geom_line()`

tib

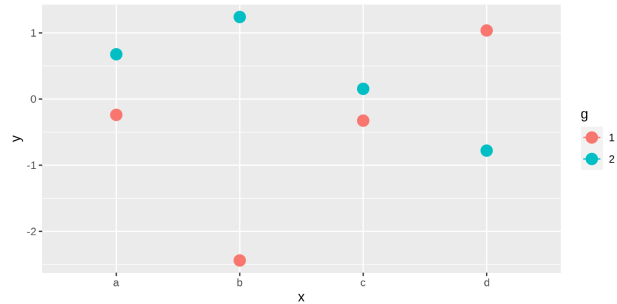
```
# A tibble: 8 × 3
  x     g     y
<chr> <fct> <dbl>
1 a     1 -0.239
2 a     2  0.677
3 b     1 -2.44
4 b     2  1.24
5 c     1 -0.327
6 c     2  0.154
7 d     1  1.04
8 d     2 -0.780
```



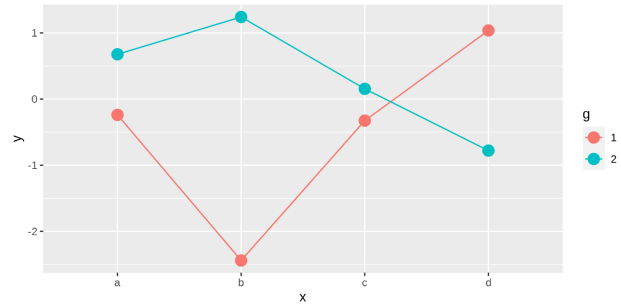
Invisible aesthetic: grouping

```
ggplot(tib, aes(x, y, colour = g)) +  
  geom_line() +  
  geom_point(size = 4)
```

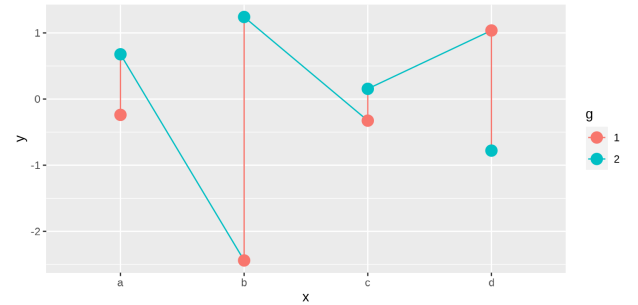
geom_path: Each group consists of only the group aesthetic?



```
ggplot(tib, aes(x, y, colour = g)) +  
  geom_line(aes(group = g)) +  
  geom_point(size = 4)
```



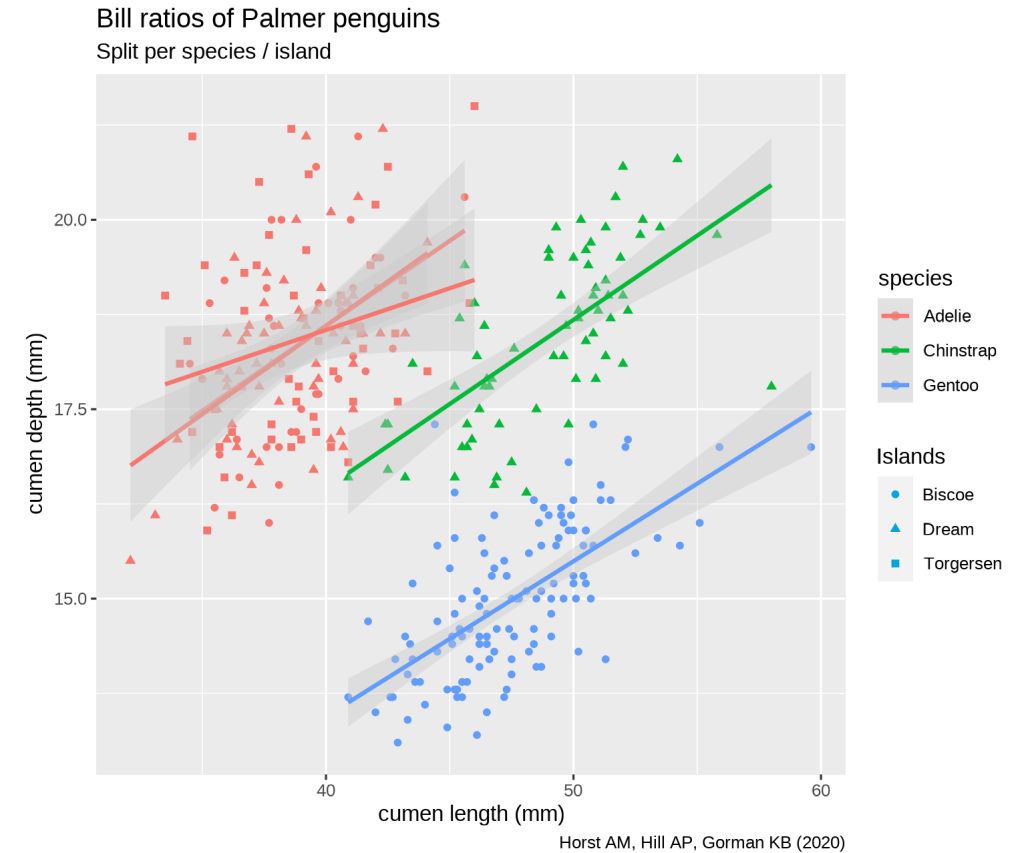
```
ggplot(tib, aes(x, y, colour = g)) +  
  geom_line(aes(group = 1)) +  
  geom_point(size = 4)
```



Source: koshske blog: [geom_line\(\) doesn't draw lines](#)

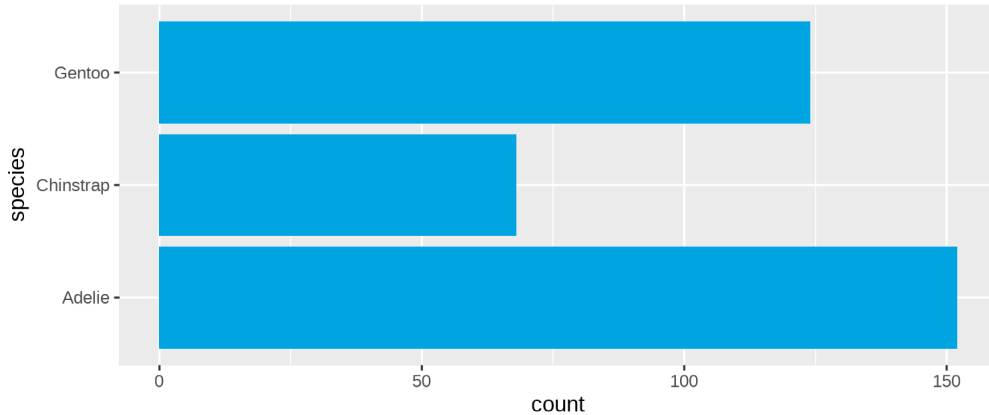
Labels

```
ggplot(penguins,  
  aes(x = bill_length_mm,  
    y = bill_depth_mm,  
    shape = island,  
    colour = species)) +  
  geom_point() +  
  geom_smooth(method = "lm",  
    formula = 'y ~ x') +  
  labs(title = "Bill ratios of Palmer penguins",  
    caption = "Horst AM, Hill AP, Gorman KB (2020)",  
    subtitle = "Split per species / island",  
    shape = "Islands",  
    x = "cumen length (mm)",  
    y = "cumen depth (mm)")
```

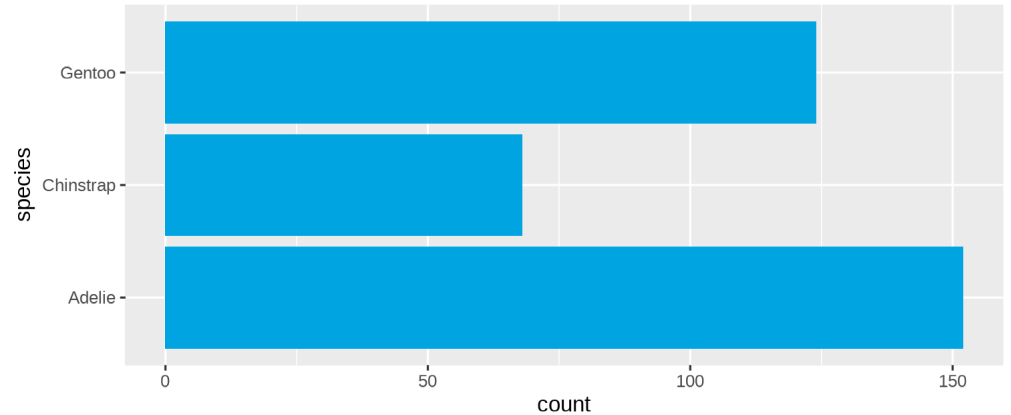


Statistics / geometries are interchangeable

```
ggplot(penguins) +  
  geom_bar(aes(y = species))
```



```
ggplot(penguins) +  
  stat_count(aes(y = species))
```

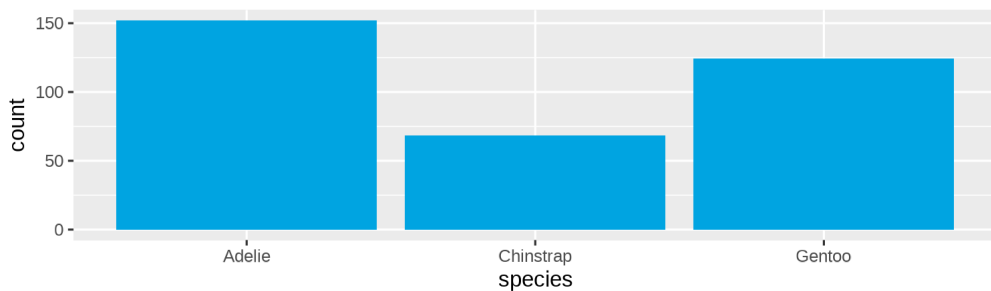


- Feels more natural since visual
- But just a preference
- Most code in the wild use `geom`

Let ggplot2 doing the stat for you

`stat_count` could be omitted since default

```
ggplot(penguins, aes(x = species)) +  
  geom_bar(stat = "count")
```



`stat_count` acts on the mapped var like `dplyr::count()`

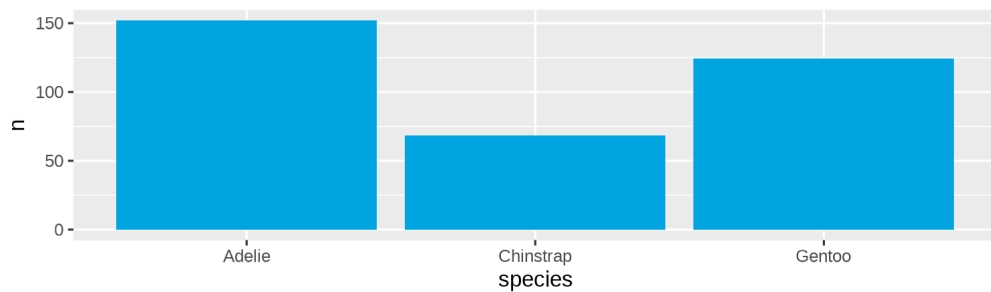
```
count(penguins, species)
```

```
# A tibble: 3 × 2  
  species     n  
  <fct>   <int>  
1 Adelie   152  
2 Chinstrap 68  
3 Gentoo   124
```

Or do it yourself, but with `geom_col()`

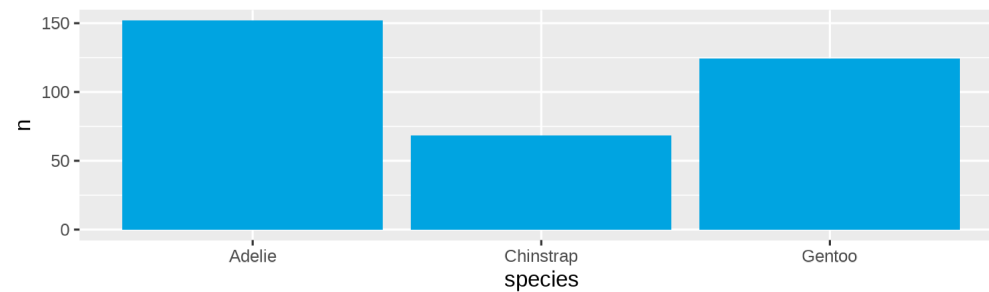
If you give counts, change the stat

```
count(penguins, species) %>%  
  ggplot(aes(x = species, y = n)) +  
  geom_bar(stat = "identity")
```



`geom_col()` has the default identity

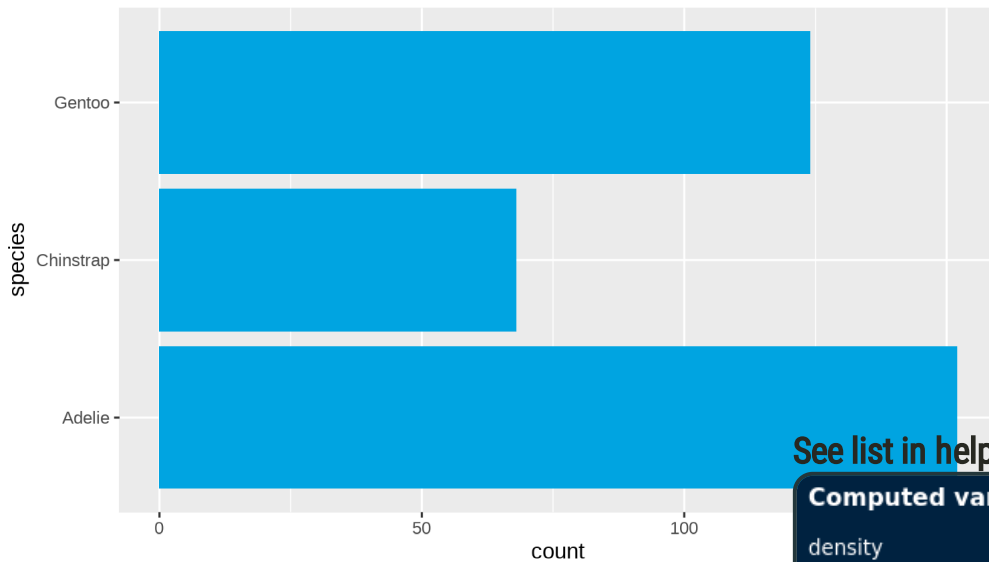
```
count(penguins, species) %>%  
  ggplot(aes(x = species, y = n)) +  
  geom_col()
```



The stat function allows computation, like proportions

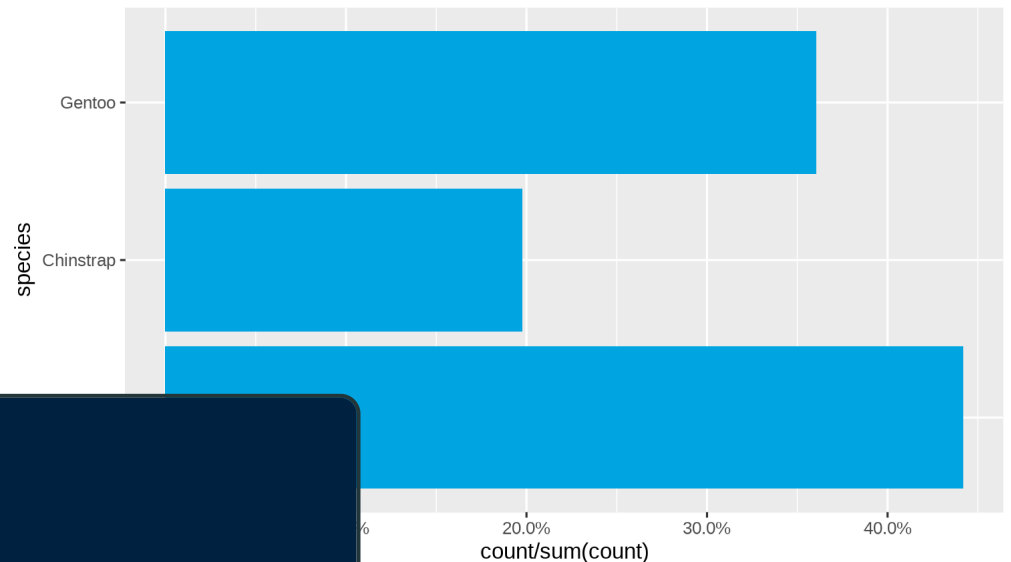
Classic counting

```
ggplot(penguins, aes(y = species)) +  
  geom_bar(aes(x = stat(count)))
```



- Now compute proportions
- **Bonus:** get **x** scale in **%** using [scales](#)

```
ggplot(penguins, aes(y = species)) +  
  geom_bar(aes(x = stat(count) / sum(count))) +  
  scale_x_continuous(labels = scales::percent)
```



See list in help pages

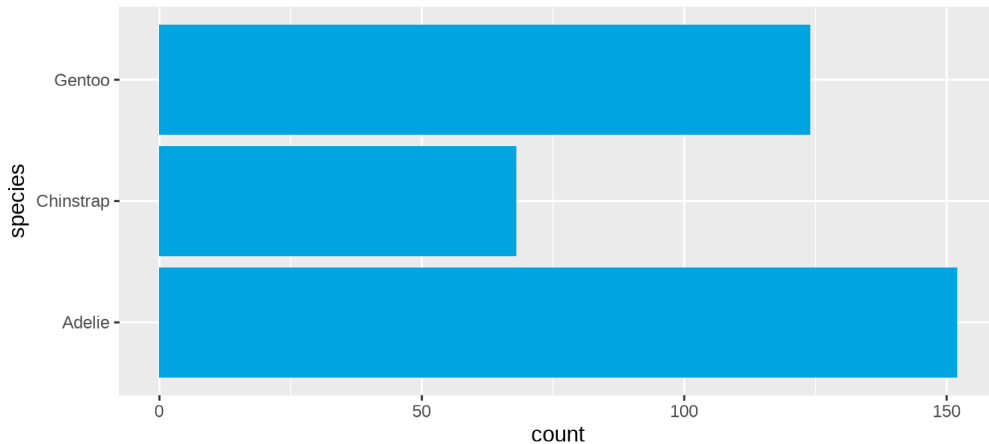
Computed variables

density
density estimate
count
density * number of points - useful for stacked density plots
scaled
density estimate, scaled to maximum of 1

Flexibility in the aesthetics for flipping axes (> v3.3.0)

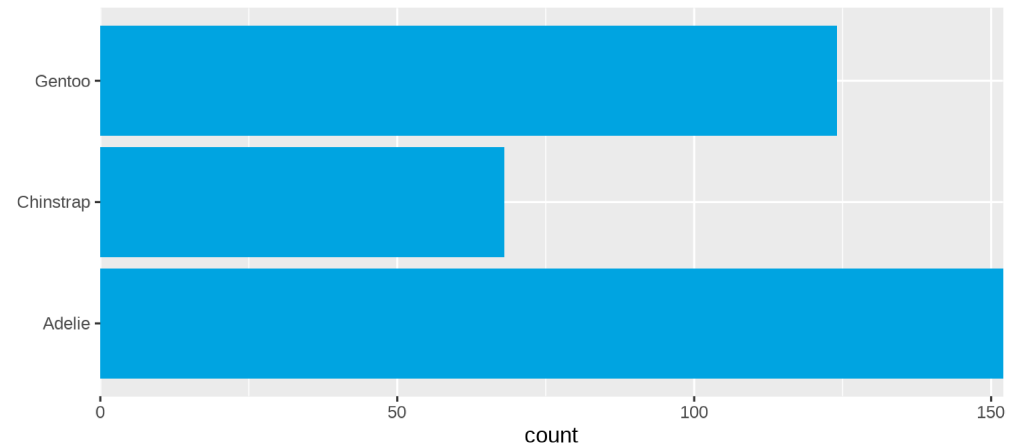
`geom_bar()` requires **x** OR **y**

```
penguins %>%  
  # horizontal brings readability  
  ggplot(aes(y = species)) +  
  geom_bar()
```



Cleanup plot

```
penguins %>%  
  ggplot(aes(y = species)) +  
  geom_bar() +  
  labs(y = NULL) +  
  scale_x_continuous(expand = c(0, NA))
```



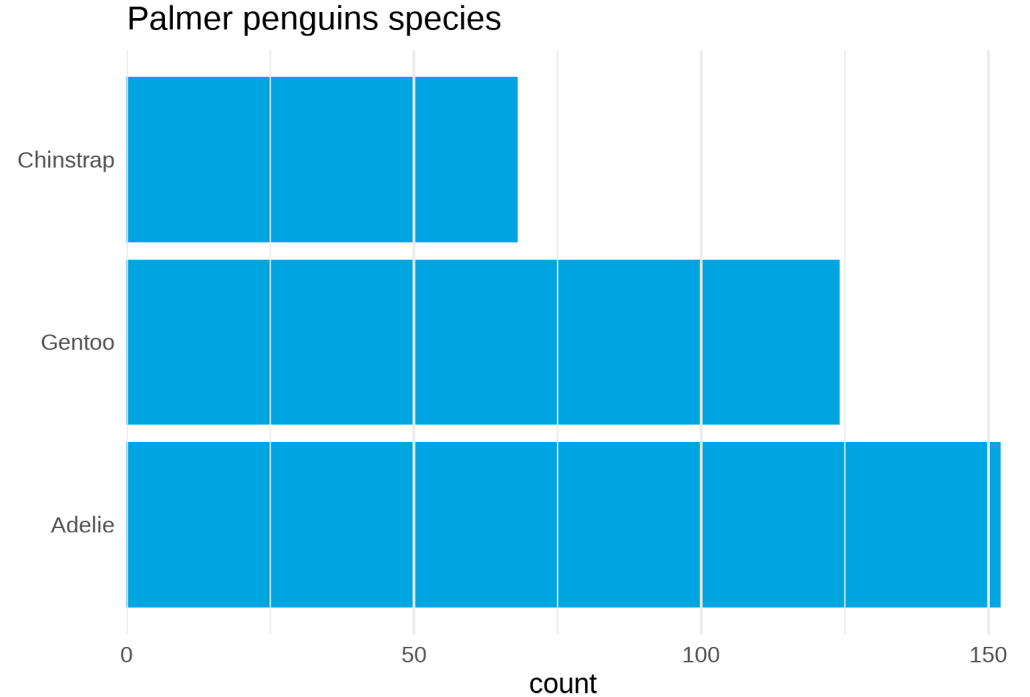
Annoying to see those 3 bars in disorder

Reorder the categorical variable (forcats)

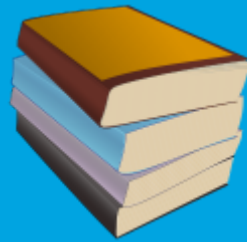
Using the function `fct_infreq()`

```
library(forcats)
penguins %>%
  ggplot(aes(y = fct_infreq(species))) +
  geom_bar() +
  scale_x_continuous(expand = c(0, NA)) +
  labs(title = "Palmer penguins species",
       y = NULL) +
  theme_minimal(14) +
  # nice trick from T. Pedersen
  theme(panel.ontop = TRUE,
        # better to hide the horizontal grid lines
        panel.grid.major.y = element_blank())
```

- See the new [article FAQ](#) about reordering.

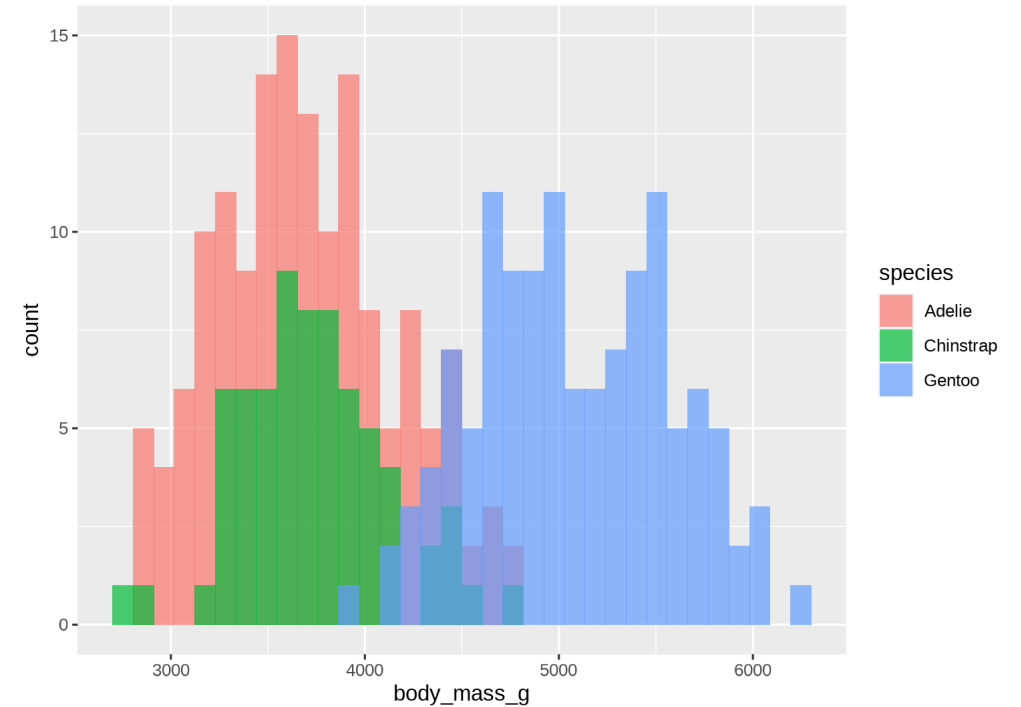


Geometries catalogue



Histograms

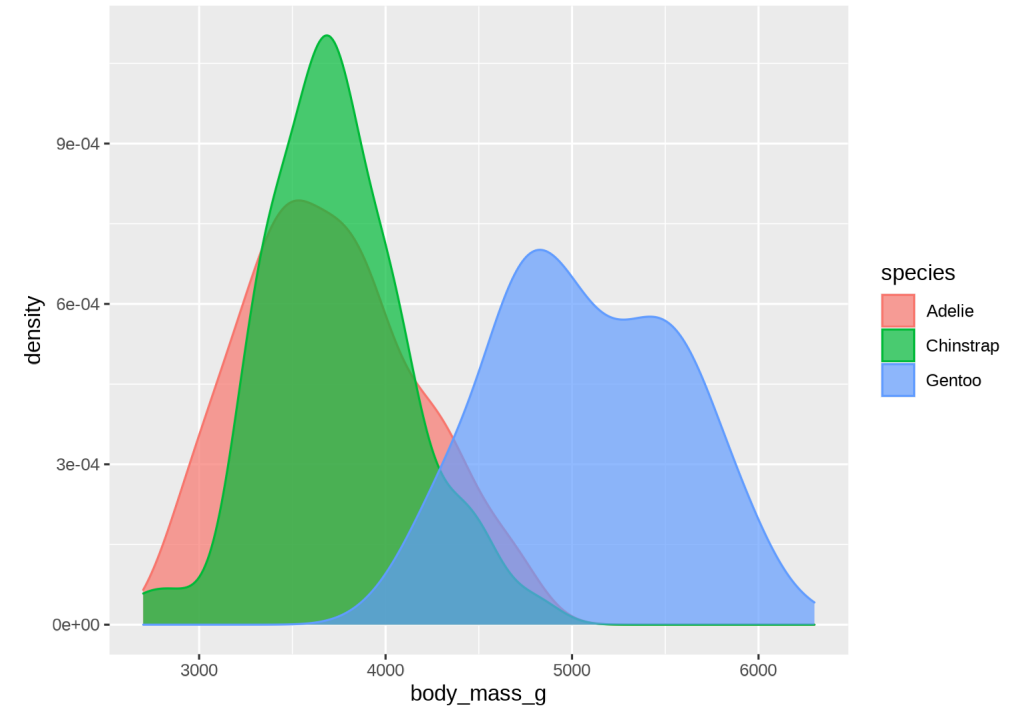
```
penguins %>%  
  ggplot(aes(x = body_mass_g,  
             fill = species)) +  
  geom_histogram(bins = 35,  
                alpha = 0.7,  
                position = "identity")
```



- default **bin** value is **30** and will be printed out as a message
- default is **stack** for the **position**. Here we overlay and use transparency

Density plots

```
penguins %>%  
  ggplot(aes(x = body_mass_g,  
             fill = species,  
             colour = species)) +  
  geom_density(alpha = 0.7)
```

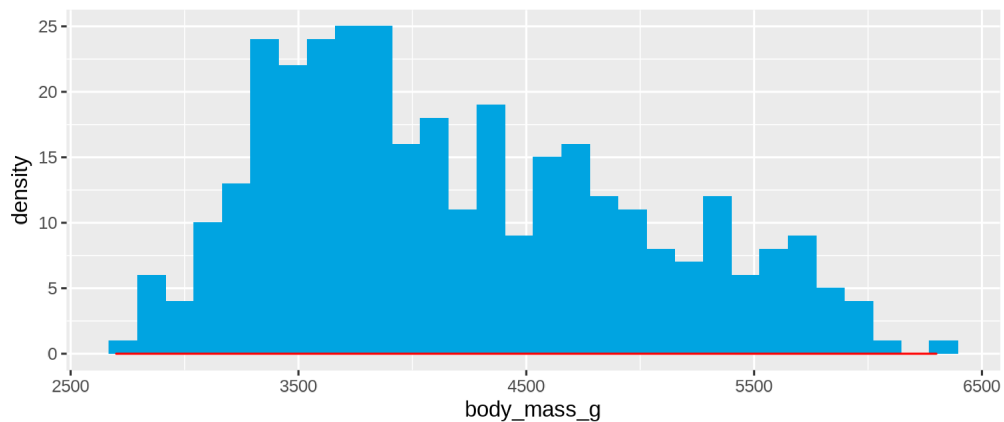


- use both `colour` and `fill` mapped to the same variable for cosmetic purposes

Overlay density and histogram

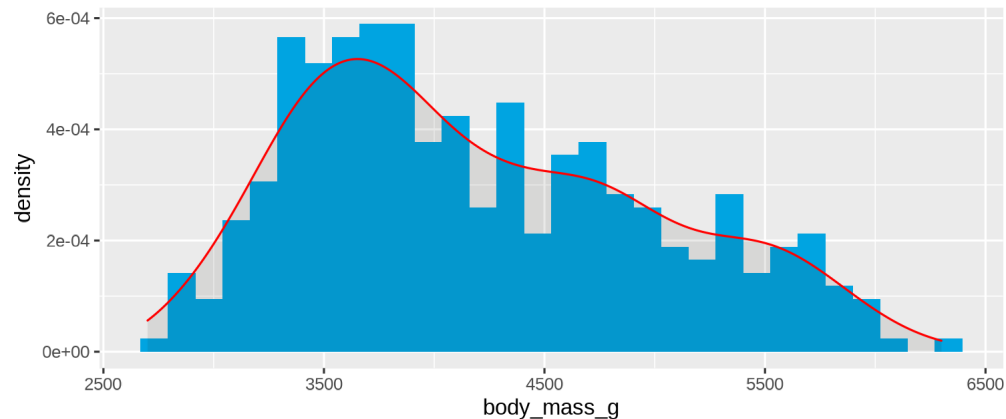
Naive approach: scale issue

```
ggplot(penguins, aes(x = body_mass_g)) +  
  geom_histogram(bins = 30) +  
  geom_density(colour = "red")
```



Solution: scale histogram to density one

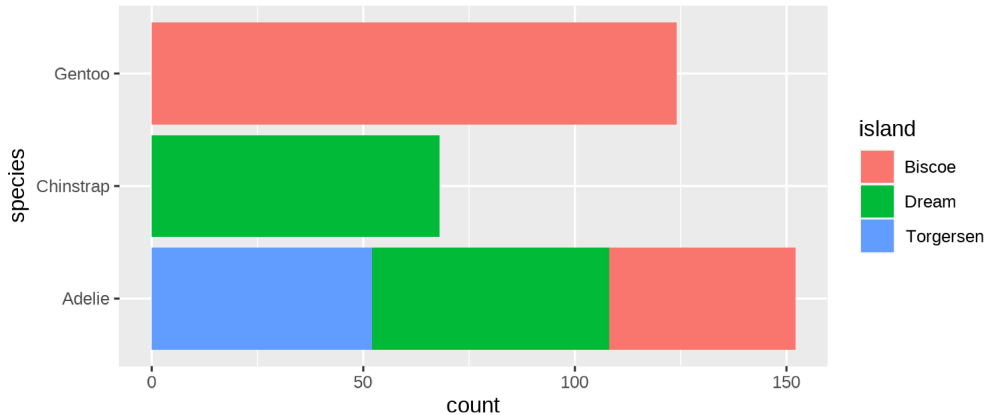
```
ggplot(penguins, aes(x = body_mass_g)) +  
  geom_histogram(bins = 30,  
    aes(y = stat(density))) +  
  geom_density(colour = "red")
```



Barplot, categories

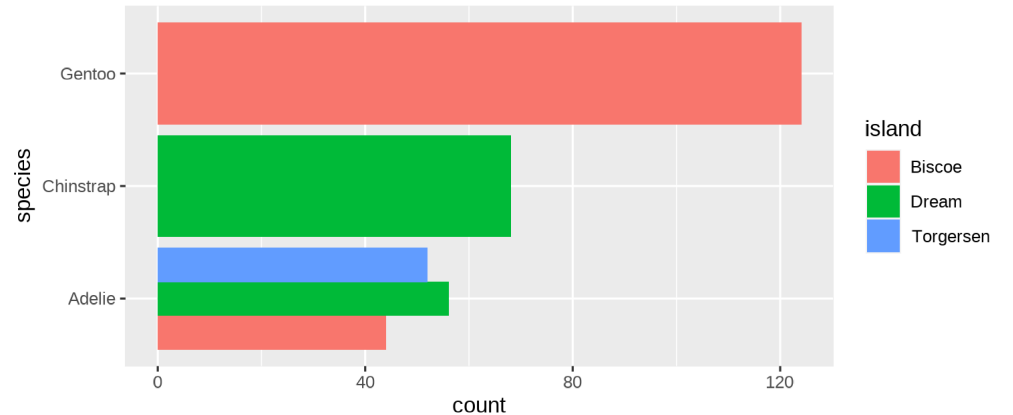
Get stacked by default (`position = "stack"`)

```
ggplot(penguins) +  
  geom_bar(aes(y = species,  
               fill = island))
```



Dodging `island`: side by side

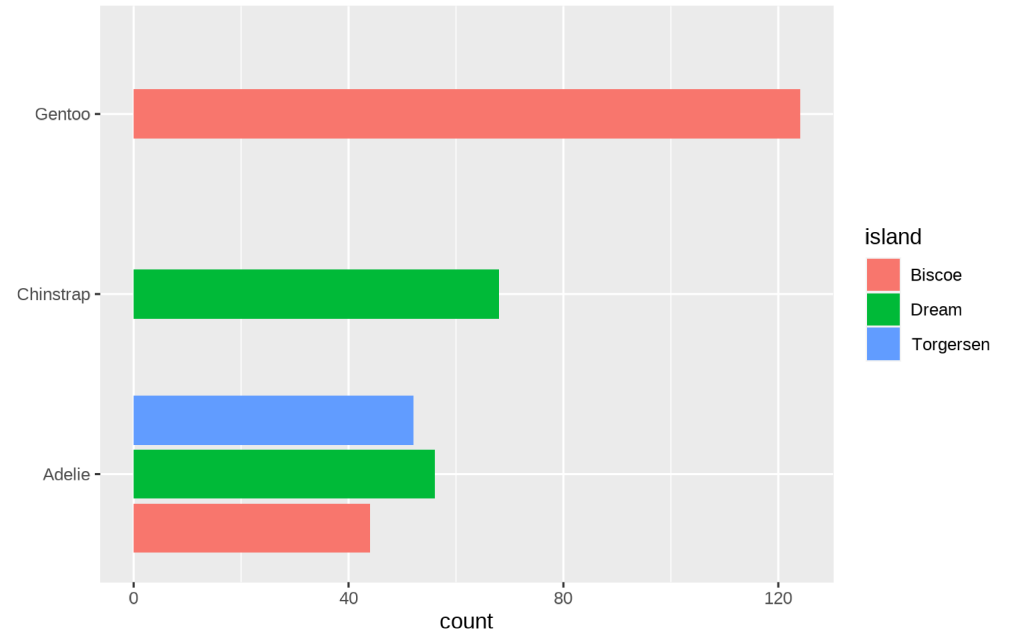
```
ggplot(penguins) +  
  geom_bar(aes(y = species, fill = island),  
           position = "dodge")
```



But global width per species is preserved

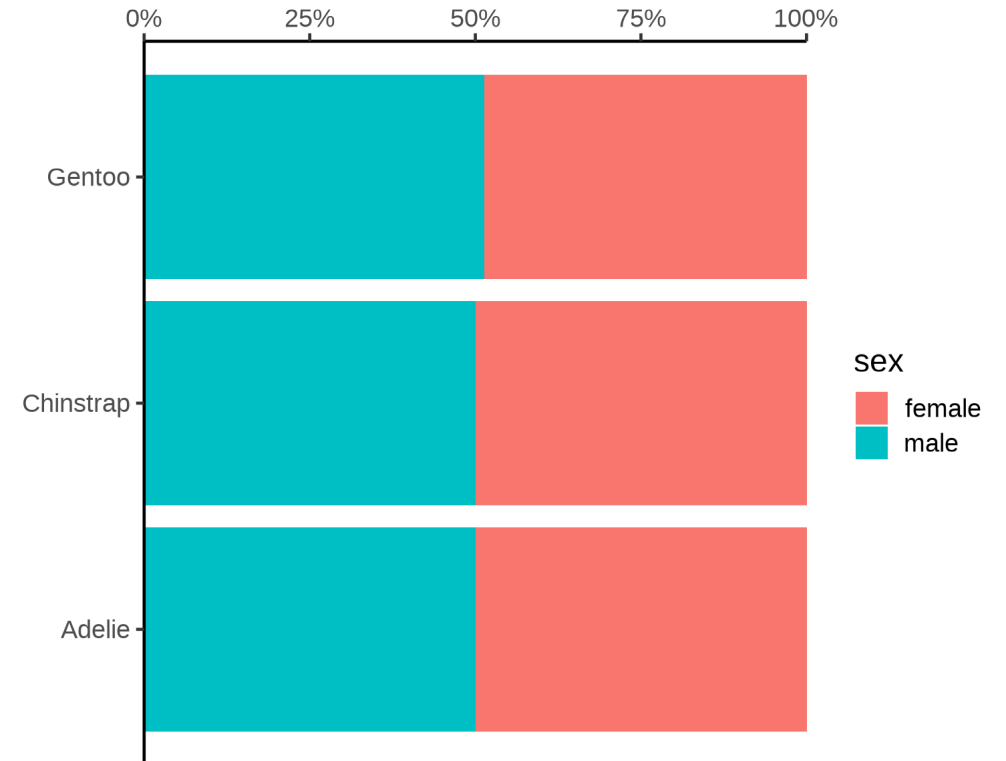
Preserve single bar (same width)

```
ggplot(penguins) +  
  geom_bar(aes(y = species,  
               fill = island),  
           position = position_dodge2(preserve = "single")) +  
  labs(y = NULL)
```



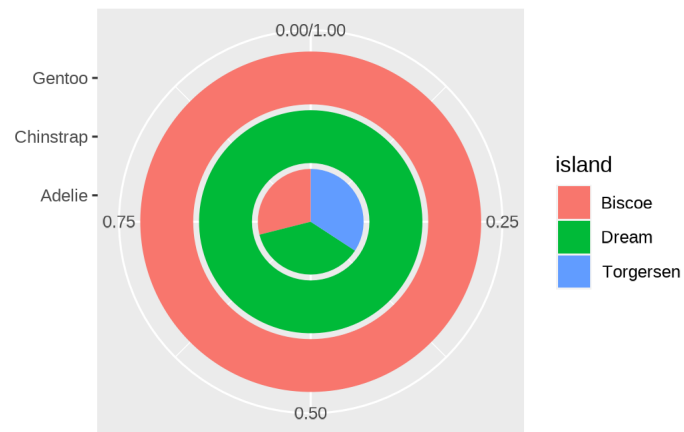
Stacked barchart for proportions

```
penguins %>%  
  drop_na(sex) %>% # from tidyr  
  ggplot() +  
  geom_bar(aes(y = species,  
               fill = sex),  
           position = "fill") +  
  scale_x_continuous(labels = scales::percent,  
                     position = "top",  
                     expand = c(0, 0)) +  
  labs(x = NULL, y = NULL) +  
  theme_classic(16)
```

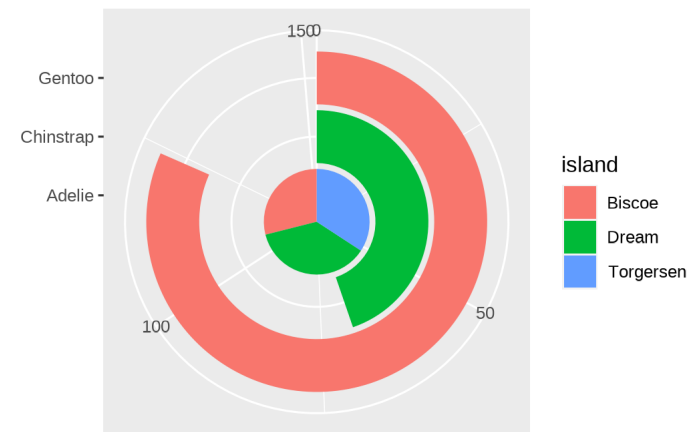


Pie charts involved another coordinate system

```
ggplot(penguins) +  
  geom_bar(aes(y = species,  
               fill = island),  
           position = "fill") +  
  labs(x = NULL, y = NULL) +  
  coord_polar()
```



```
penguins %>%  
  ggplot() +  
  geom_bar(aes(y = species,  
               fill = island)) +  
  labs(x = NULL, y = NULL) +  
  coord_polar()
```



Boxplot, a continuous y by a categorical x

```
ggplot(penguins) +  
  geom_boxplot(aes(y = body_mass_g,  
                  x = species))
```

`geom_boxplot()` is assessing that:

- `body_mass_g` is **continuous**
- `species` is **categorical/discrete**

CONTINUOUS

measured data, can have ∞ values within possible range.



I AM 3.1" TALL
I WEIGH 34.16 grams

DISCRETE

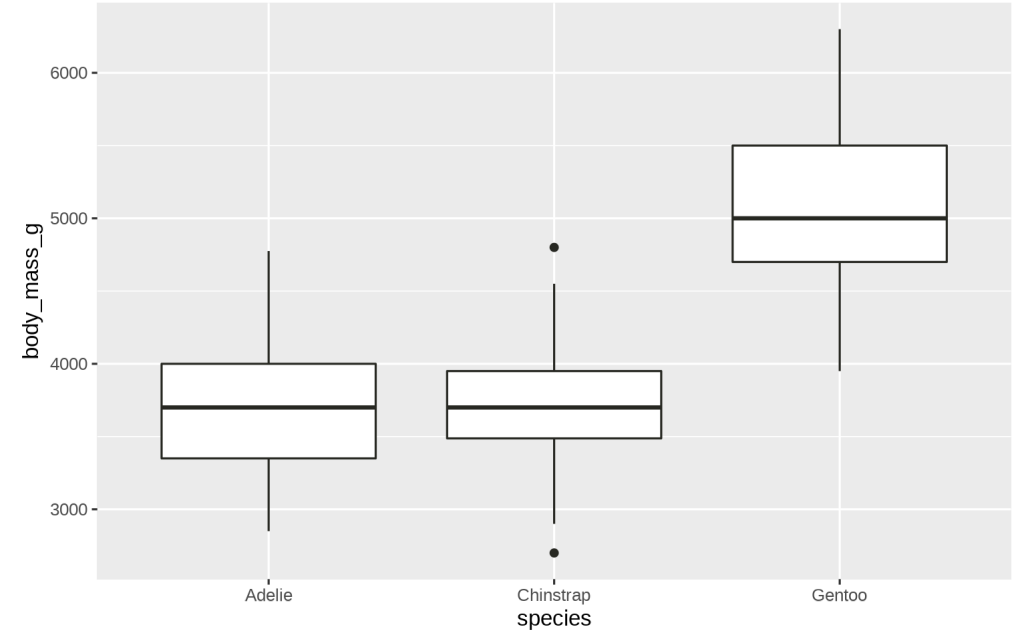
OBSERVATIONS CAN ONLY EXIST AT LIMITED VALUES, OFTEN COUNTS.



I HAVE 8 LEGS
and
4 SPOTS!

@allison_horst

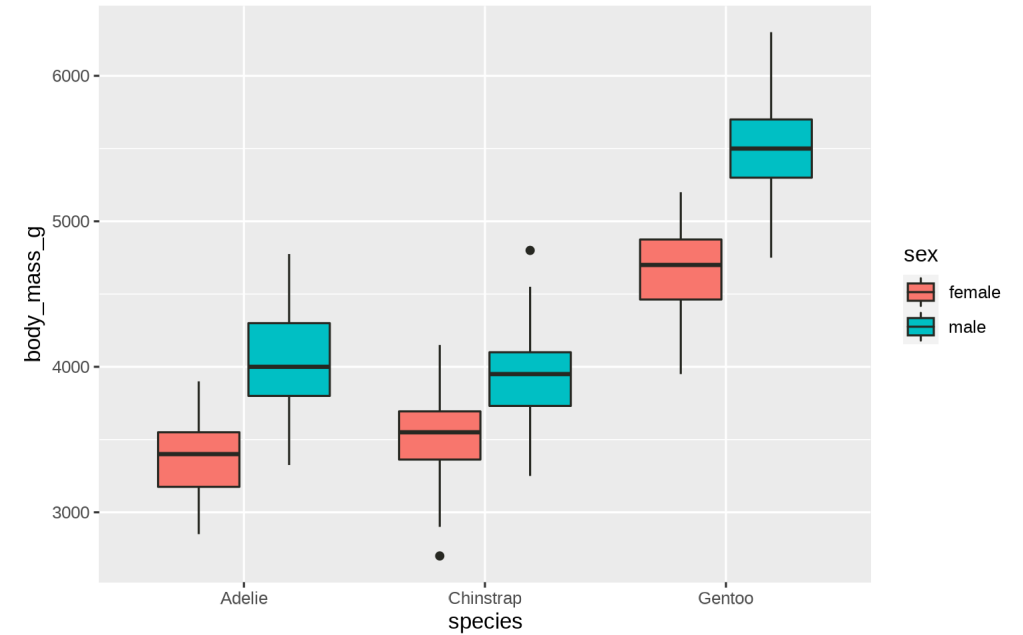
Artwork by [Allison Horst](#)



Boxplot, dodging by default

Filter out **NA** to avoid this category

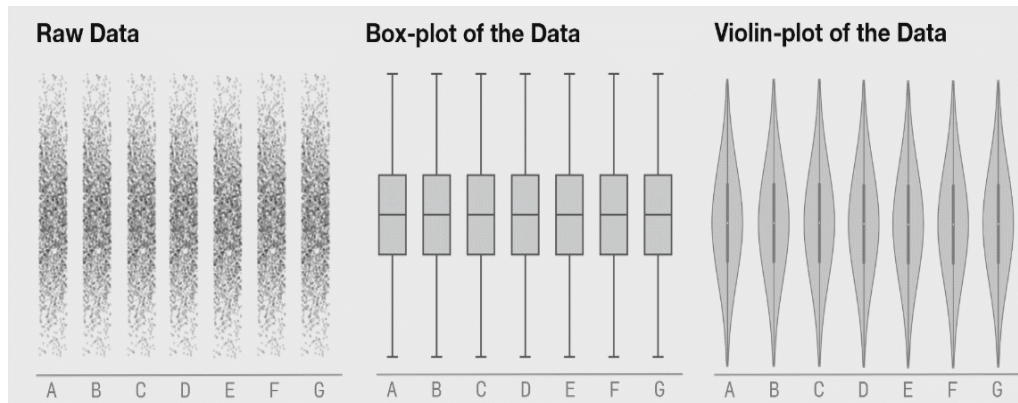
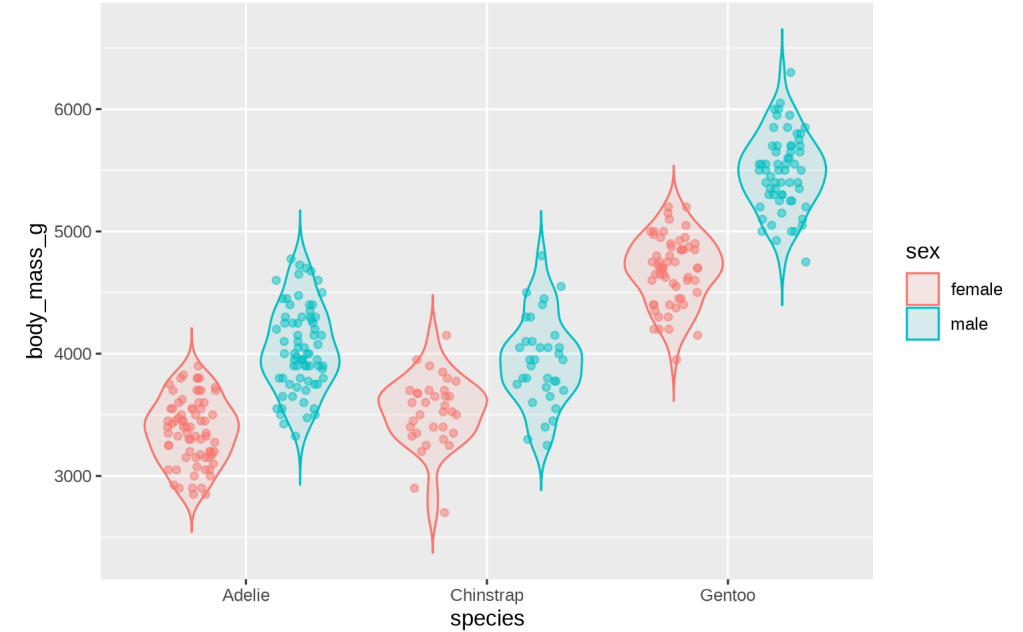
```
penguins %>% # alternative to tidyr::drop_na()
  filter(!is.na(sex)) %>%
  ggplot() +
  geom_boxplot(aes(y = body_mass_g,
                   x = species,
                   fill = sex))
```



Better: violin and jitter

Show the data

```
penguins %>%  
  filter(!is.na(sex)) %>%  
  # define aes here for both geometries  
  ggplot(aes(y = body_mass_g,  
             x = species,  
             fill = sex,  
             # for violin contours and dots  
             colour = sex  
           )) +  
    # very transparent filling  
    geom_violin(alpha = 0.1, trim = FALSE) +  
    geom_point(position = position_jitterdodge(dodge.width = 0.9)  
              alpha = 0.5,  
              # don't need dots in legend  
              show.legend = FALSE)
```

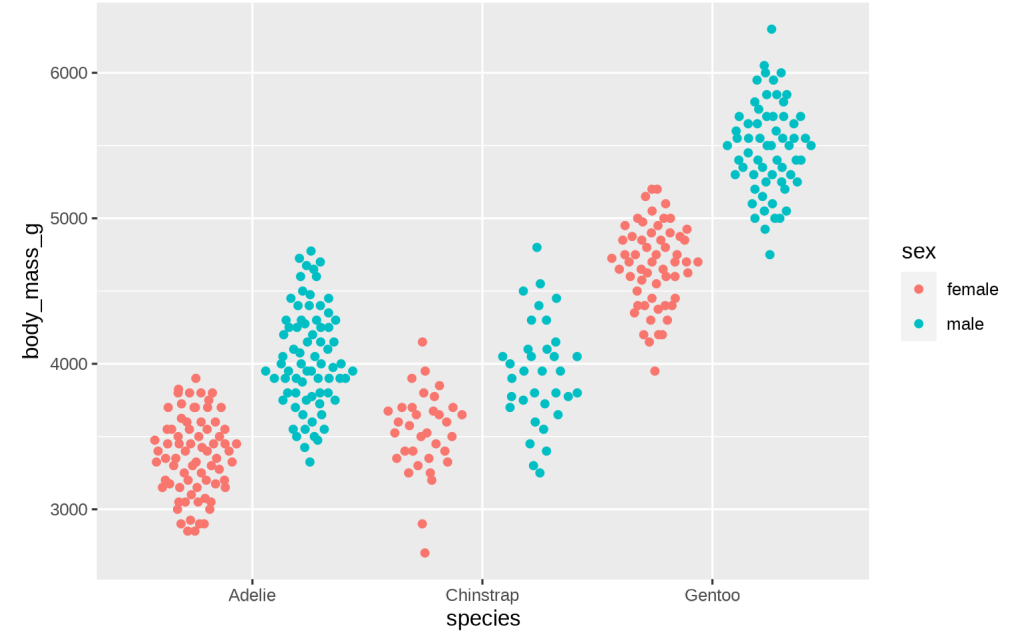


GIF source: [Linh Ngo @BioTuring](#)

Even better: beeswarm

ggplot extension [ggbeeswarm](#)

```
library(ggbeeswarm)
penguins %>%
  filter(!is.na(sex)) %>%
  ggplot(aes(y = body_mass_g,
             x = species,
             colour = sex)) +
  geom_quasirandom(dodge.width = 1)
```



Artwork by [@allison_horst](#)

Coding mistake

What is wrong with the above code?

(**Hint:** think about inherited aesthetics)

```
penguins %>%  
  ggplot() +  
  geom_point(aes(x = bill_length_mm,  
                 y = body_mass_g)) +  
  geom_smooth(method = "lm")
```

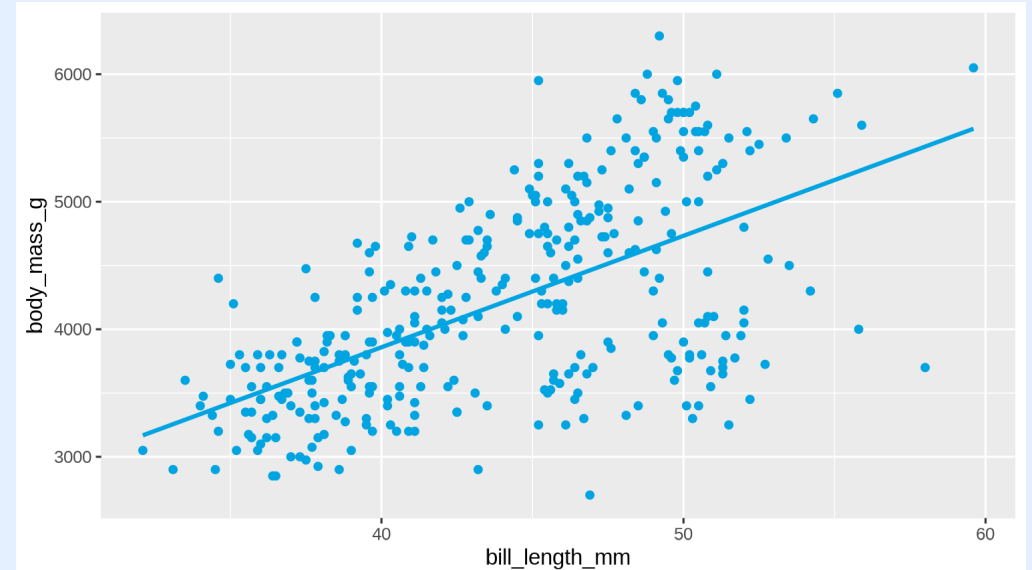
`geom_smooth()` using formula 'y ~ x'

Error: stat_smooth requires the following missing aesthetics: x

Inheritance of **aesthetics** in main **ggplot()**

```
penguins %>%  
  ggplot(aes(x = bill_length_mm,  
             y = body_mass_g)) +  
  geom_point() +  
  geom_smooth(method = "lm", se = FALSE)
```

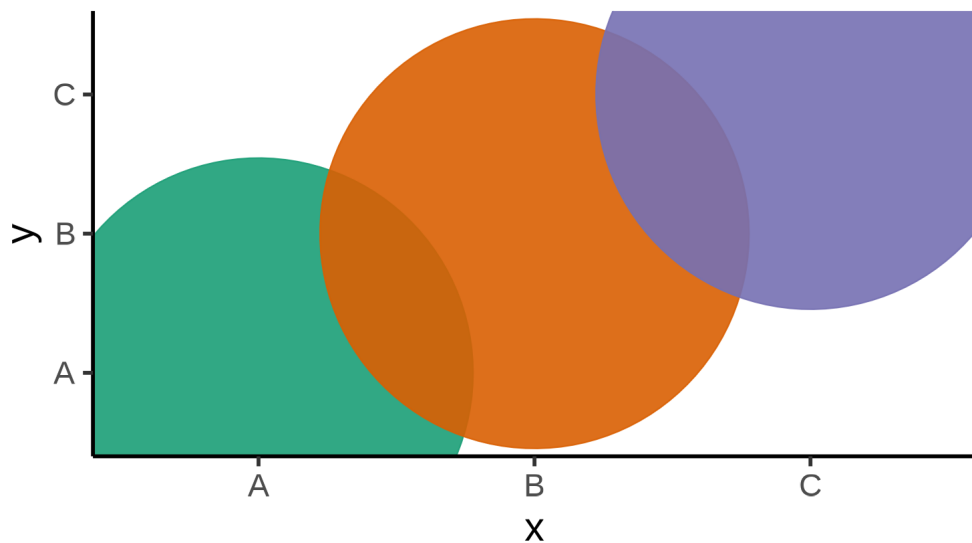
`geom_smooth()` using formula 'y ~ x'



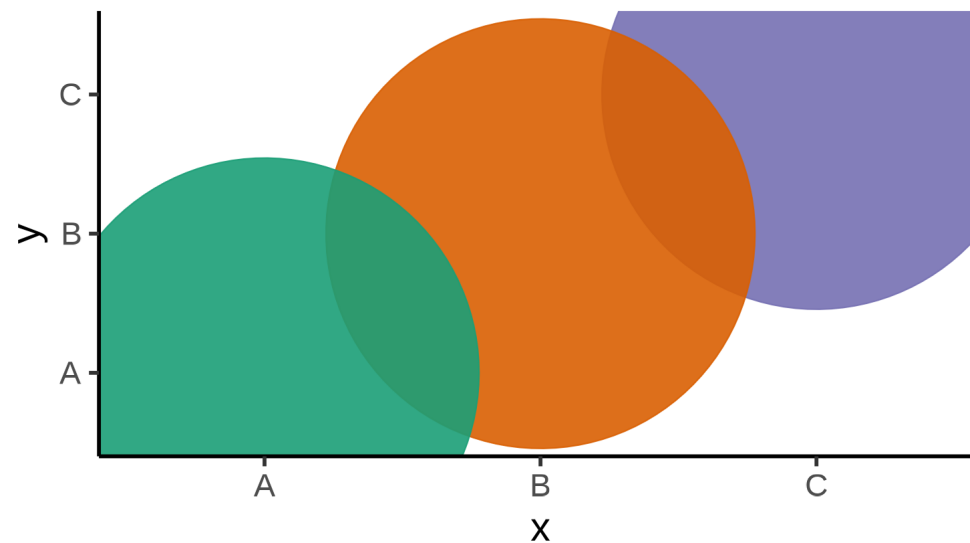
Control the dots plotting order

`ggplot2` outputs dots **as they appear** in the input data

```
tibble(x = LETTERS[1:3],  
       y = x) %>%  
  ggplot(aes(x, y)) +  
  geom_point(aes(colour = x),  
            show.legend = FALSE,  
            size = 100, alpha = 0.9) +  
  scale_color_brewer(palette = "Dark2") +  
  theme_classic(20)
```



```
tibble(x = LETTERS[1:3], y = x) %>%  
  arrange(desc(x)) %>%  
  ggplot(aes(x, y)) +  
  geom_point(aes(colour = x),  
            show.legend = FALSE,  
            size = 100, alpha = 0.9) +  
  scale_color_brewer(palette = "Dark2") +  
  theme_classic(20)
```



Scales catalogue

62 functions available!



Scales for all

Functions are based on this **scheme**:

```
scale_{aes}_{type}()
```

arguments to change default

- **breaks**, choose where are labels
- **labels**, usually with package [scales](#)
- **trans**, to change to *i.e* log

tidyr **crossing** + glue **glue_data** trick

to generate combinations

```
tidyr::crossing(aes = c("fill", "colour", "alpha", "x", "y"),  
               type = c("continuous", "discrete", "date")) %>%  
  glue::glue_data("scale_{aes}_{type}()")
```

```
scale_alpha_continuous()  
scale_alpha_date()  
scale_alpha_discrete()  
scale_colour_continuous()  
scale_colour_date()  
scale_colour_discrete()  
scale_fill_continuous()  
scale_fill_date()  
scale_fill_discrete()  
scale_x_continuous()  
scale_x_date()  
scale_x_discrete()  
scale_y_continuous()  
scale_y_date()  
scale_y_discrete()
```

```
scales::label_percent()(c(0.4, 0.6))
```

```
[1] "40%" "60%"
```

```
scales::label_comma()(20^(1:3))
```

```
[1] "20"      "400"     "8,000"
```

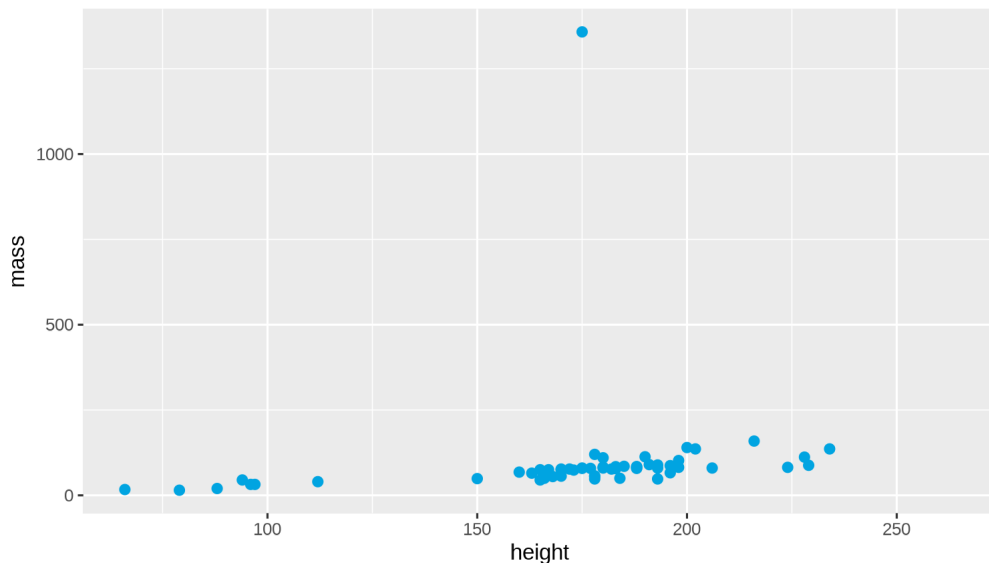
```
scales::breaks_log()(20^(1:3))
```

```
[1]    10    30   100   300  1000  3000 10000
```

Scales transformation

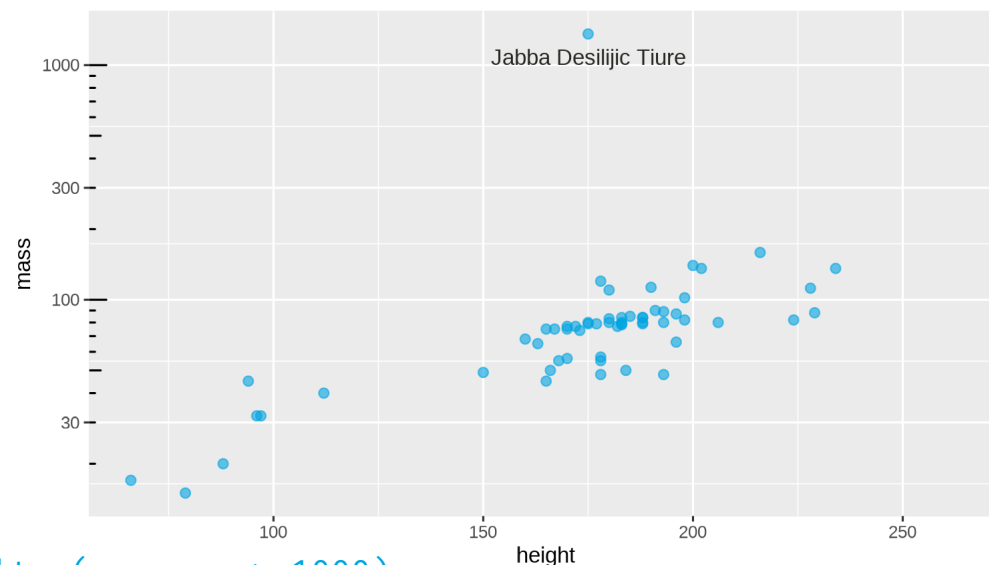
Starwars (dplyr dataset): who is the massive guy?

```
starwars %>%  
  ggplot(aes(x = height,  
             y = mass)) +  
  geom_point(size = 2)
```



Log scaling the y axis

```
starwars %>%  
  ggplot(aes(x = height, y = mass)) +  
  geom_point(alpha = 0.6, size = 2) +  
  # all data arg can take either a df or a function  
  geom_text(data = \(x) filter(x, mass > 1000),  
            aes(label = name), nudge_y = -0.1) +  
  annotation_logticks(sides = "l") +  
  scale_y_log10() # same as trans = "log10"
```



using the new `lambda` for anonymous functions, `R` < 4.1 would be `function(x) filter(x, mass > 1000)`

Custom colors

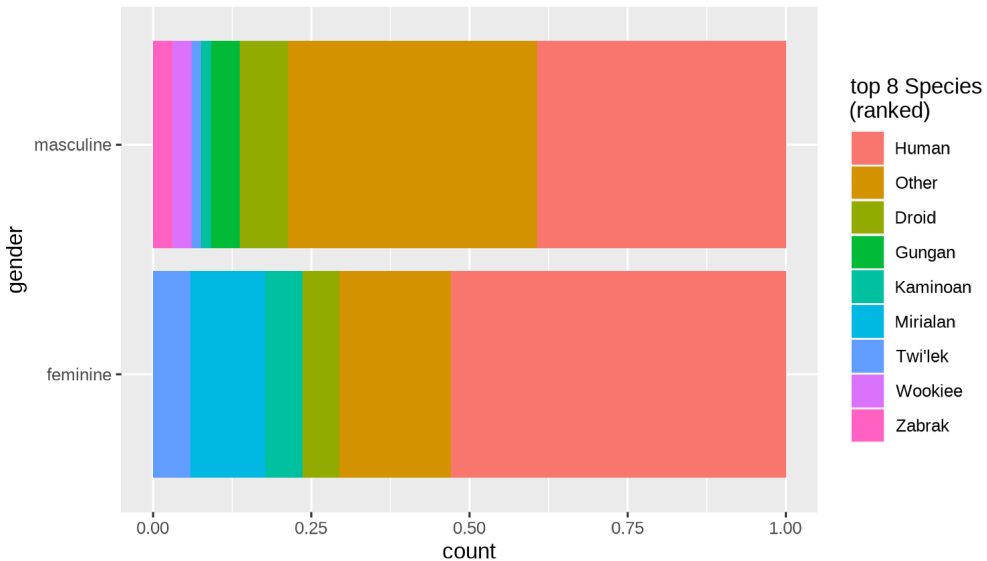


Better: see [Emil Hvitfeldt repo](#)

Changing colours for categories

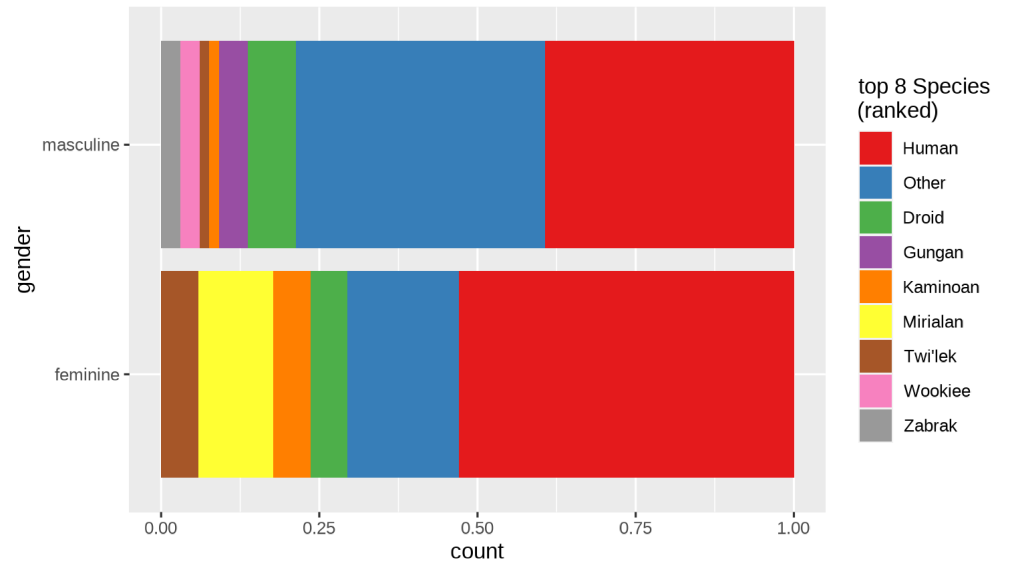
Default is `scale_fill_hue()`

```
filter(starwars, !is.na(gender)) %>%  
ggplot(aes(y = gender,  
           fill = fct_lump_n(species, 8) %>%  
             fct_infreq())) +  
geom_bar(position = "fill") +  
scale_fill_hue() +  
labs(fill = "top 8 Species\n(ranked)")
```



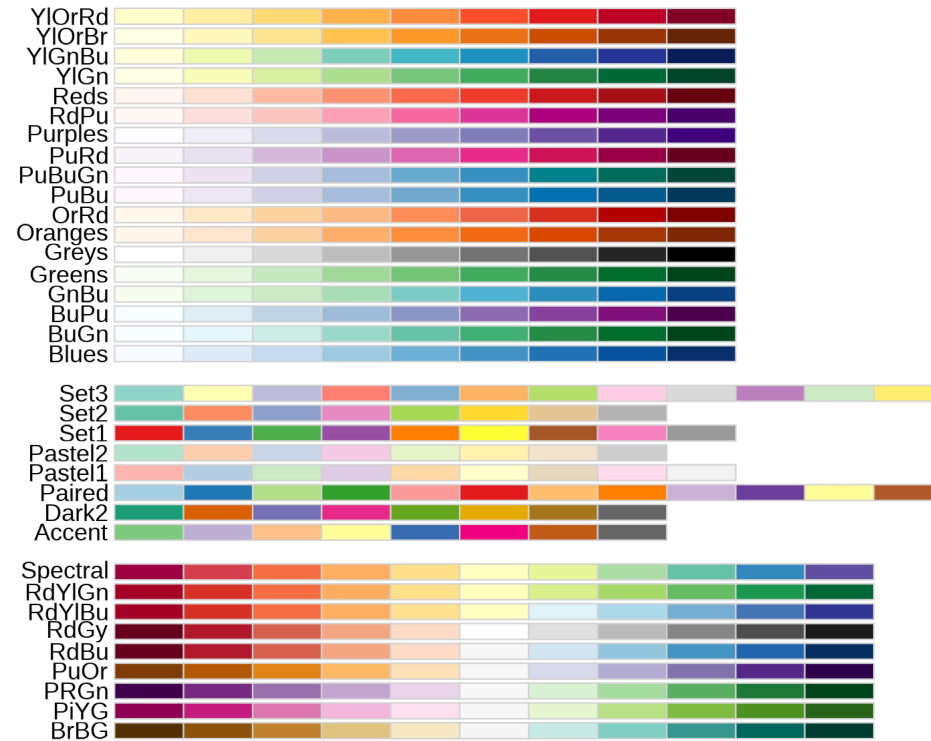
Rbrewer is a safe alternative

```
filter(starwars, !is.na(gender)) %>%  
ggplot(aes(y = gender,  
           fill = fct_lump_n(species, 8) %>%  
             fct_infreq())) +  
geom_bar(position = "fill") +  
scale_fill_brewer(palette = "Set1") +  
labs(fill = "top 8 Species\n(ranked)")
```



Predefined colour palettes

```
library(RColorBrewer)
par(mar = c(0, 4,
           0, 0))
display.brewer.all()
```

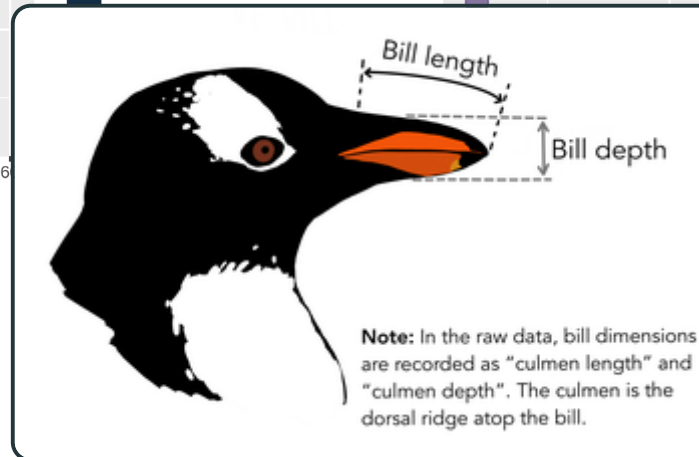
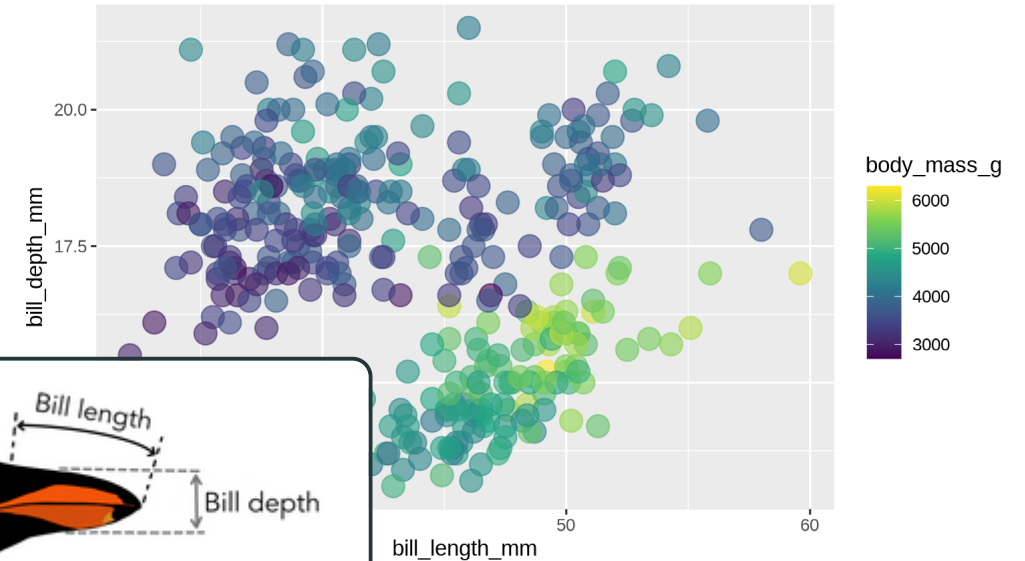
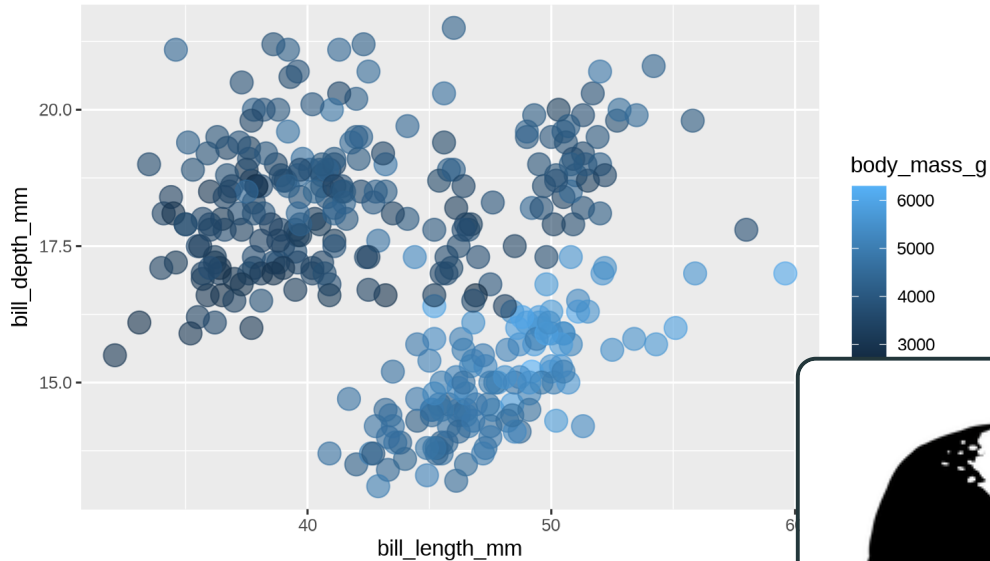


Colour gradient, for continuous variables

The default gradient generated by `ggplot2` is not very good... Better off using `viridis` (`scale_color_viridis_c()` for continuous)

```
penguins %>%  
  ggplot(aes(x = bill_length_mm,  
             y = bill_depth_mm,  
             colour = body_mass_g)) +  
  geom_point(alpha = 0.6, size = 5)
```

```
penguins %>%  
  ggplot(aes(x = bill_length_mm, y = bill_depth_mm,  
             colour = body_mass_g)) +  
  geom_point(alpha = 0.6, size = 5) +  
  scale_color_viridis_c()
```

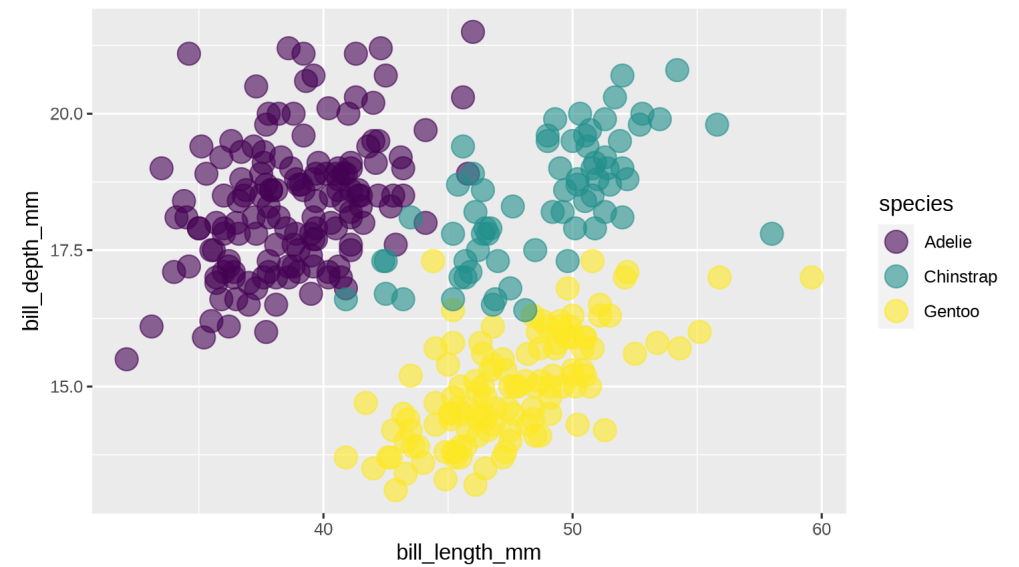


Viridis palettes

- 5 different scales
- Also for discrete variables
- [viridis](#) is colour blind friendly and nice in b&w
- In [ggplot2](#) since v3.0 but not the default



```
penguins %>%  
  ggplot(aes(x = bill_length_mm, y = bill_depth_mm,  
             colour = species)) +  
  geom_point(alpha = 0.6, size = 5) +  
  scale_color_viridis_d()
```

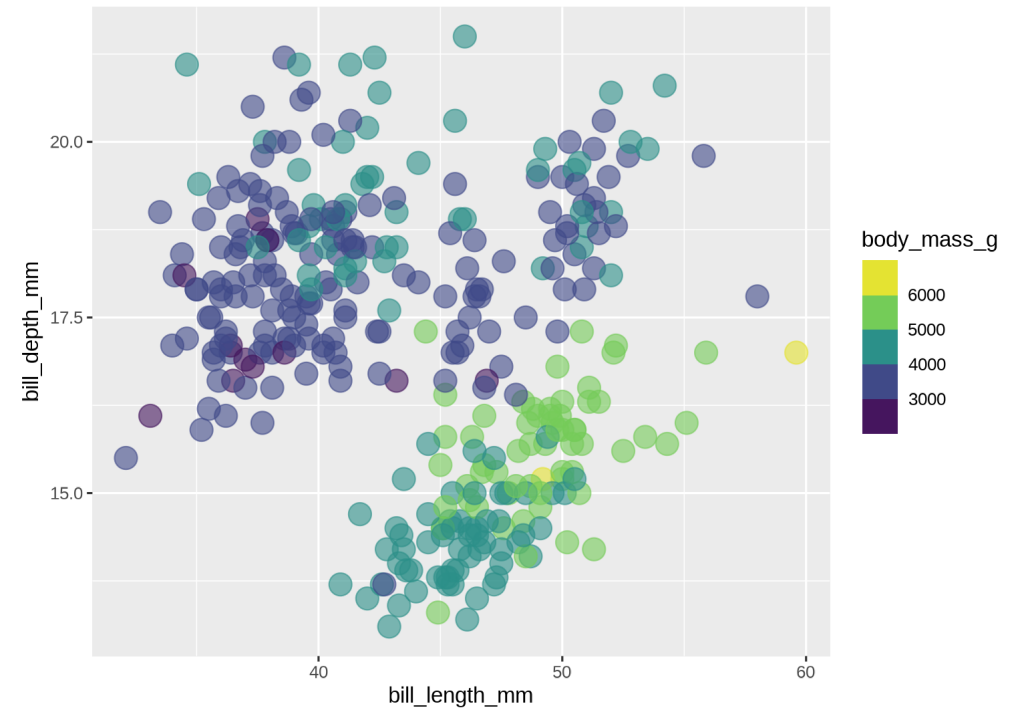


Binning instead of gradient (new v3.3.0)

Binning help grouping observations

- Default blue gradient, **viridis** option
- Number of bins, limits can be changed

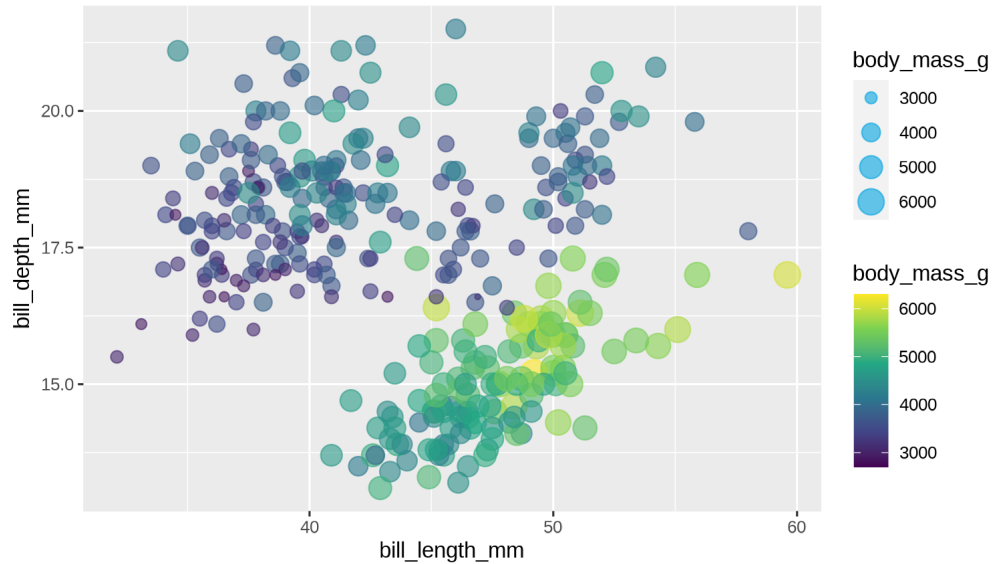
```
penguins %>%  
  ggplot(aes(x = bill_length_mm, y = bill_depth_mm,  
             colour = body_mass_g)) +  
  geom_point(alpha = 0.6, size = 5) +  
  scale_color_binned(type = "viridis")
```



Merge similar guides (new v3.3.0)

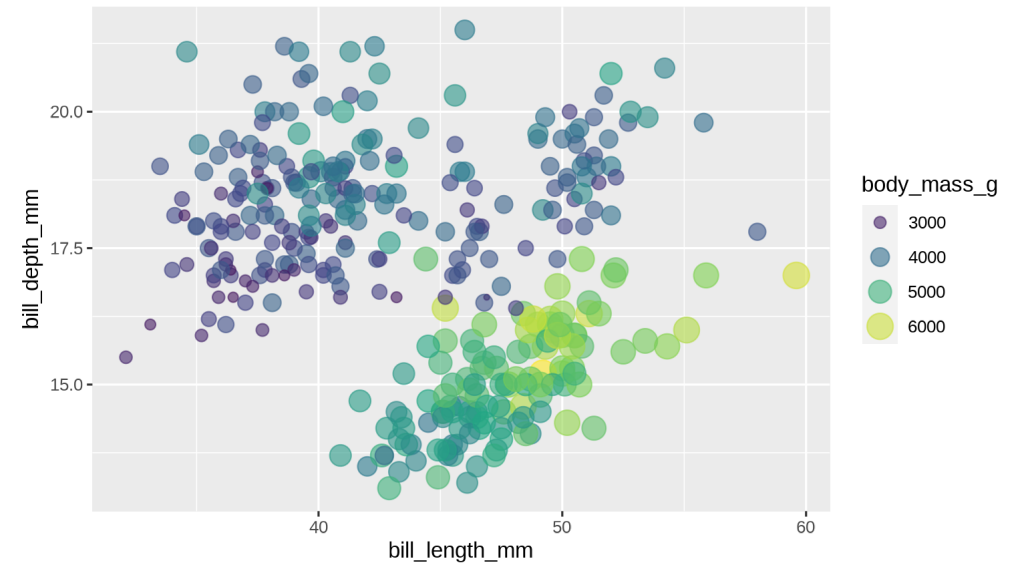
Size & colour on same variable → 2 guides

```
penguins %>%  
  ggplot(aes(x = bill_length_mm, y = bill_depth_mm,  
             colour = body_mass_g,  
             size = body_mass_g)) +  
  geom_point(alpha = 0.6) +  
  scale_color_viridis_c()
```

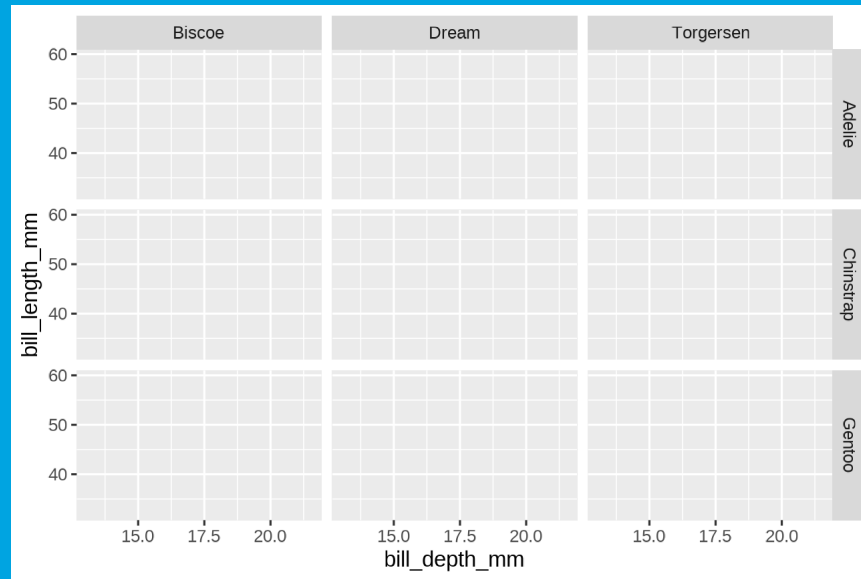


`guides()` merge both aesthetics

```
penguins %>%  
  ggplot(aes(x = bill_length_mm, y = bill_depth_mm,  
             colour = body_mass_g, size = body_mass_g)) +  
  geom_point(alpha = 0.6) +  
  scale_color_viridis_c() +  
  guides(colour = "legend")
```



Facets

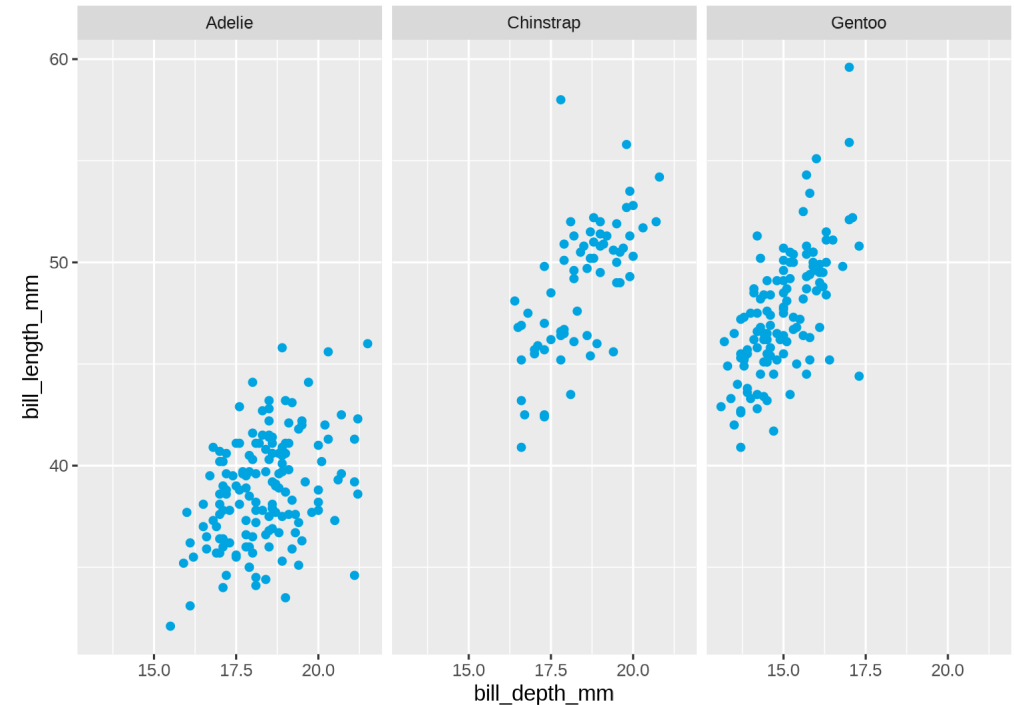


Facets: facet_wrap()

Creating facets

- Easiest way: `facet_wrap()`
- Use a formula (in R `~`)
- `facet_wrap()` is for one var
- Or the `vars()` function

```
ggplot(penguins,  
       aes(bill_depth_mm,  
           bill_length_mm)) +  
  geom_point() +  
  facet_wrap(~ species)
```

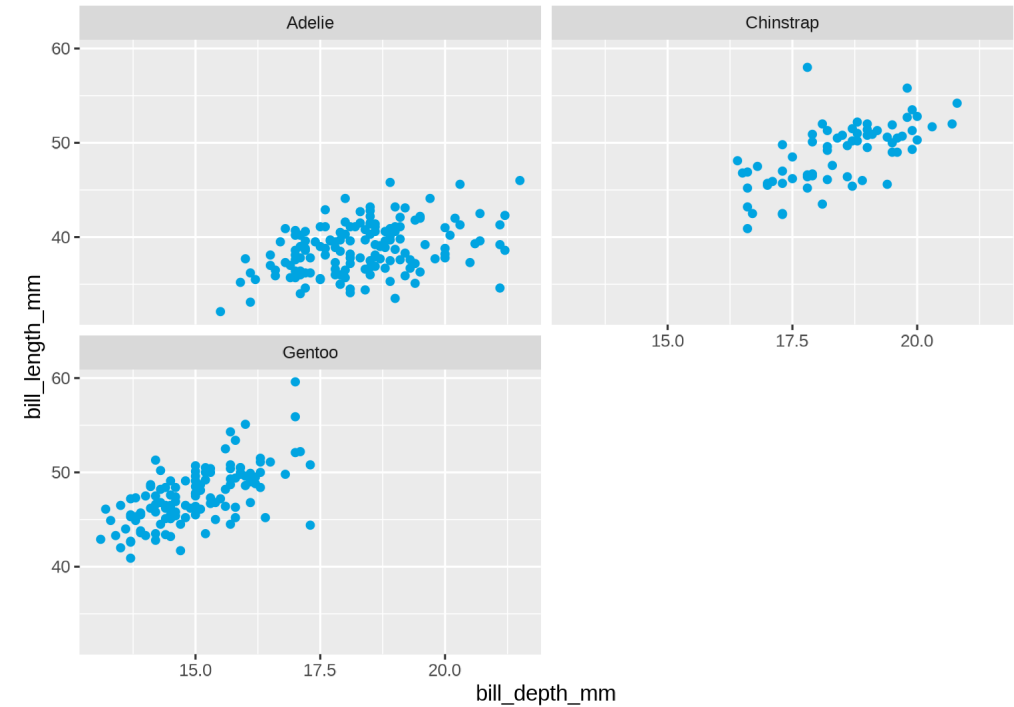


Facets layout

Specify the number of rows/columns:

- `ncol = integer`
- `nrow = integer`

```
fc <- ggplot(penguins,  
  aes(bill_depth_mm,  
      bill_length_mm)) +  
  geom_point()  
fc + facet_wrap(~ species, ncol = 2)
```

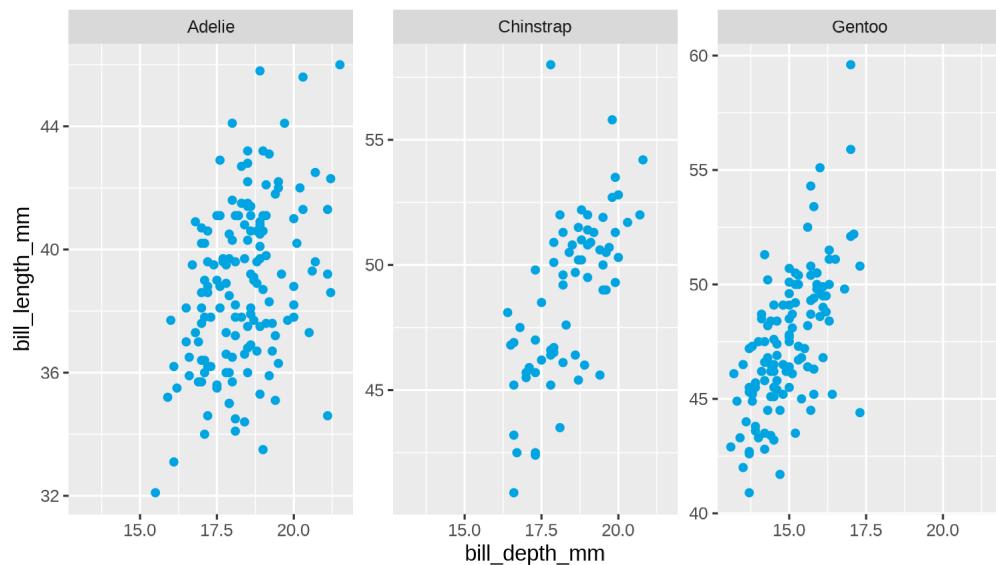


Facets, free scales

□ Make comparison harder

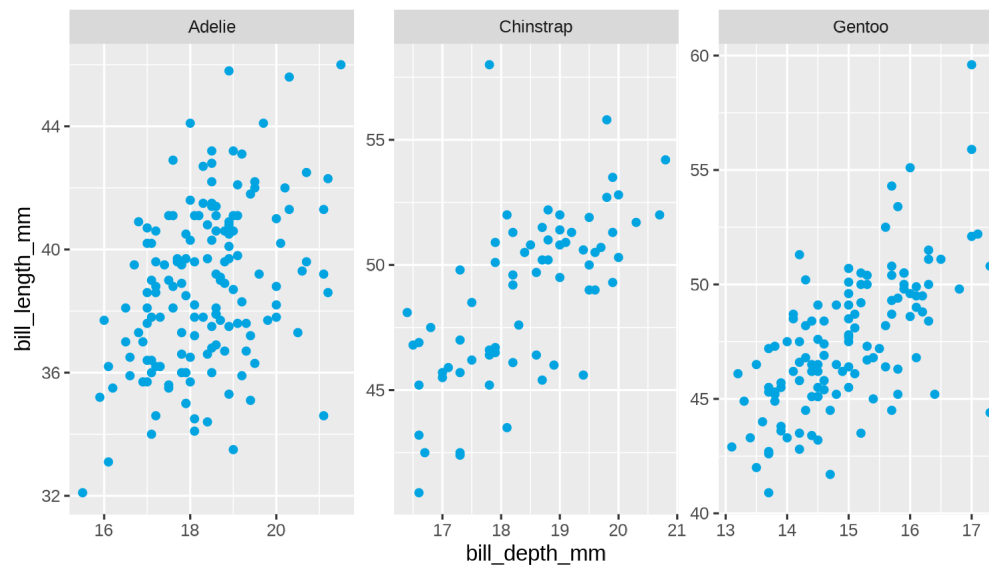
- `x` or `y` (`free_x` / `free_y`)

```
fc + facet_wrap(~ species, scales = "free_y")
```



- Both axis

```
fc + facet_wrap(~ species, scales = "free")
```

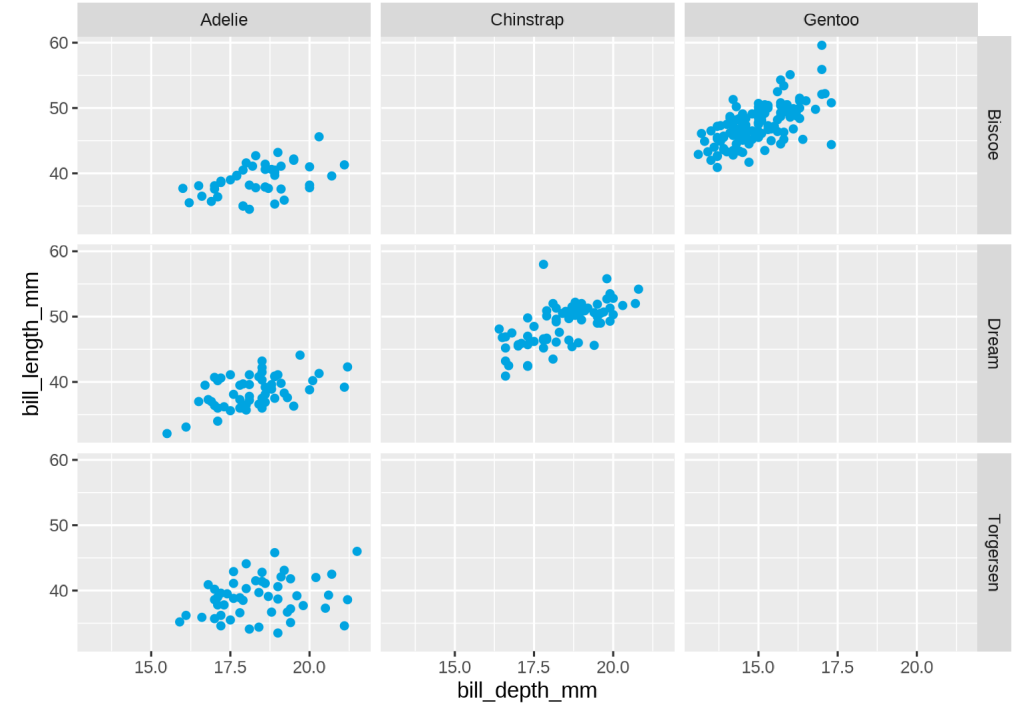


facet_grid() to lay out panels in a grid

Specify a 2 sides **formula**

rows on the left, **columns** on the right separated by a tilde ~

```
ggplot(penguins,  
       aes(bill_depth_mm,  
           bill_length_mm)) +  
  geom_point() +  
  facet_grid(island ~ species)
```

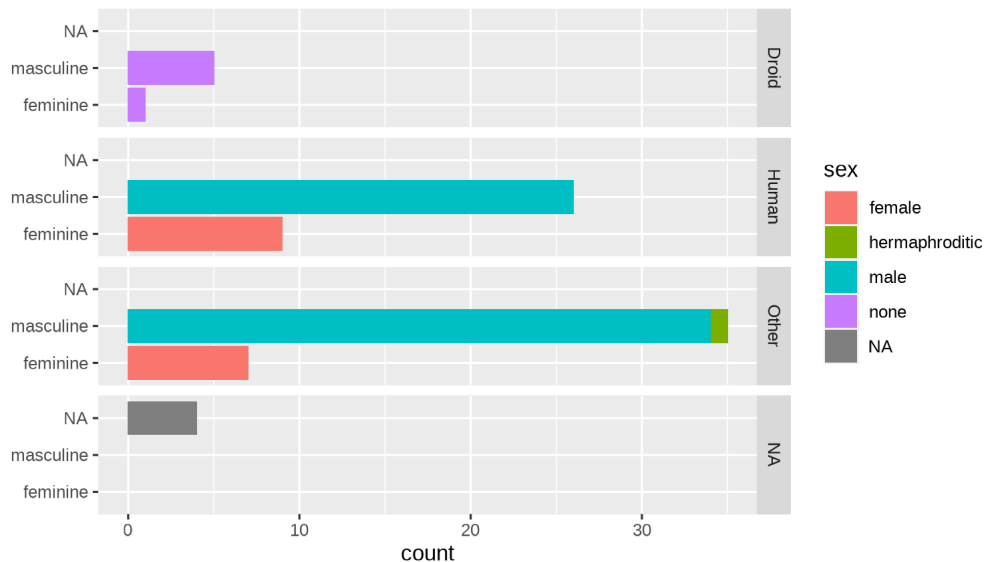


Barplots in facets

`facet_grid()` can also be used with **one** variable, complemented by a placeholder: `.`

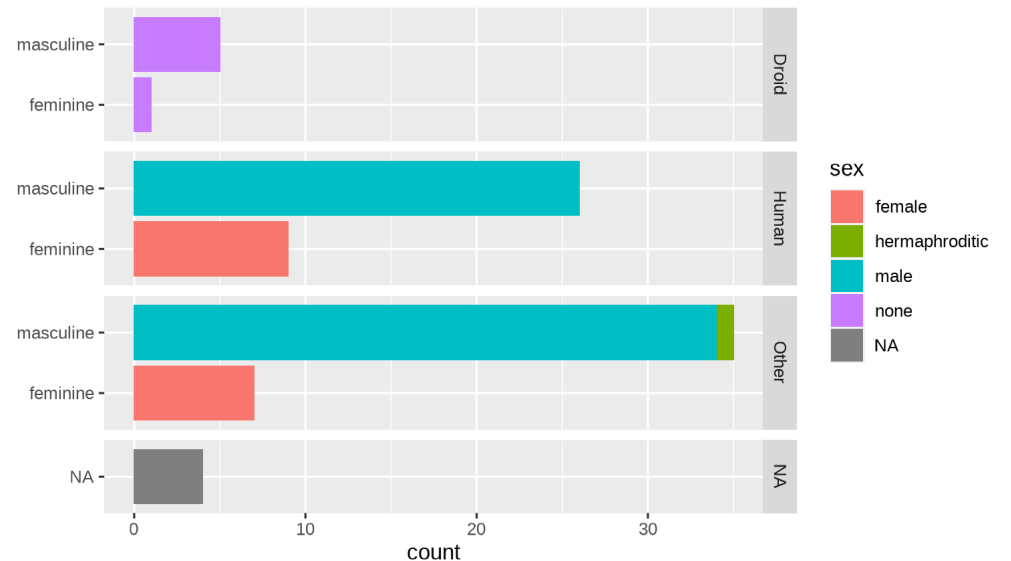
Waste of space

```
ggplot(starwars) +  
  geom_bar(aes(y = gender, fill = sex)) +  
  facet_grid(fct_lump_min(species, 4) ~ .) +  
  labs(y = NULL)
```



Optimize the space by removing empty slots

```
ggplot(starwars) +  
  geom_bar(aes(y = gender, fill = sex)) +  
  facet_grid(fct_lump_min(species, 4) ~ .,  
            space = "free", scales = "free_y") +  
  labs(y = NULL)
```

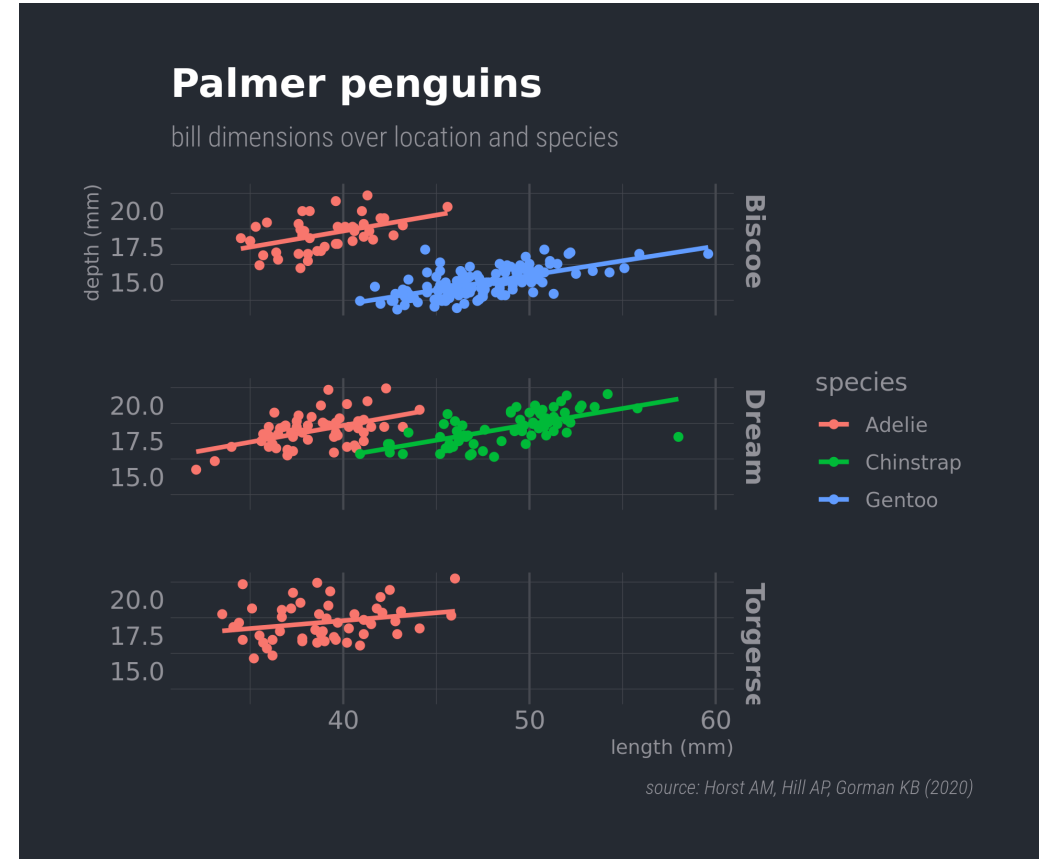


Cosmetic 
and helpers

Black theme

Using Bob Rudis [hrbrthemes](#) package

```
library(hrbrthemes)
ggplot(penguins,
      aes(x = bill_length_mm,
          y = bill_depth_mm,
          colour = species)) +
  geom_point() +
  geom_smooth(method = "lm", formula = 'y ~ x',
             # no standard error ribbon
             se = FALSE) +
  facet_grid(island ~ .) +
  labs(x = "length (mm)", y = "depth (mm)",
       title = "Palmer penguins",
       subtitle = "bill dimensions over location and species",
       caption = "source: Horst AM, Hill AP, Gorman KB (2020)")
# hrbrthemes specifications
scale_fill_ipsum() +
theme_ft_rc(14) +
# tweak the theme
theme(panel.grid.major.y = element_blank(),
      panel.grid.major.x = element_line(size = 0.5),
      plot.caption = element_text(face = "italic"),
      strip.text = element_text(face = "bold"),
      plot.caption.position = "plot")
```

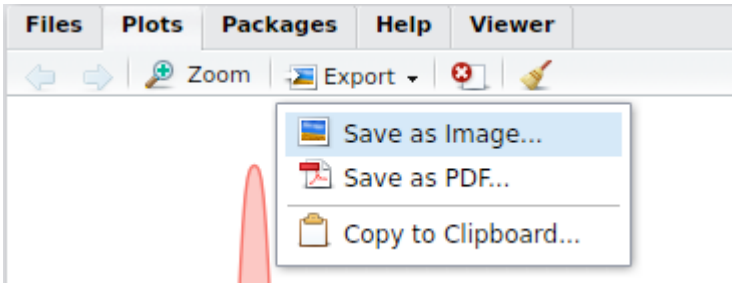


Exporting

Interactive or passive mode

Right panel

- using the Export button in the *Plots* panel



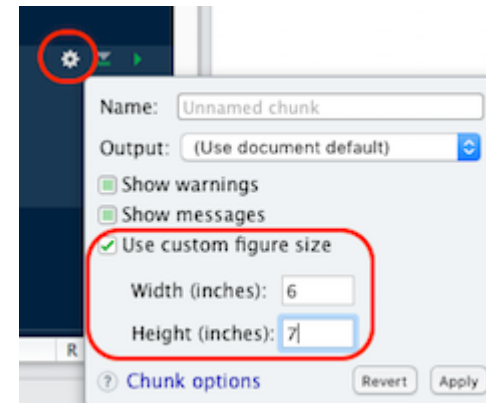
ggsave

- save the **ggplot object**, 2nd argument, here `p` or `last_plot()`
- guesses the type of desired output by the extension (jpg, png, pdf etc.)

```
ggsave("my_name.png", p, width = 60, height = 30, units = "mm")
ggsave("my_name.pdf", last_plot(), width = 50, height = 50, uni
```

Rmarkdown docs

- if needed, adjust the chunk options:
 - size: `fig.height`, `fig.width`
 - ratio: `fig.asp...`
 - others



Extensions

`ggplot2` introduced the possibility for the community to contribute and [create extensions](#).

They are referenced on a [dedicated site](#)

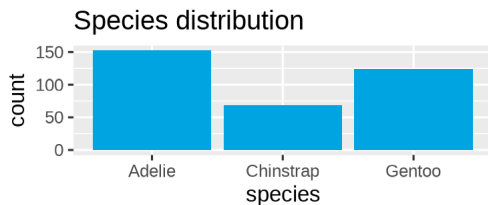
Compose plots with patchwork



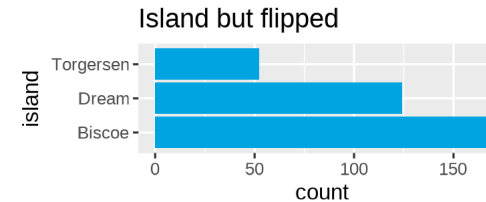
[Patchwork](#) is developed by [Thomas Lin Pedersen](#), main maintainer of [ggplot2](#).

Define 3 plots and assign them names

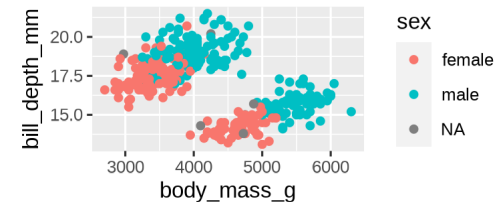
```
p1 <- ggplot(penguins,
             aes(x = species)) +
  geom_bar() +
  labs(title = "Species distribution")
p2 <- ggplot(penguins,
             aes(y = island)) +
  geom_bar() +
  labs(title = "Island but flipped")
p3 <- ggplot(penguins,
             aes(x = body_mass_g,
                 y = bill_depth_mm,
                 colour = sex)) +
  geom_point()
```



p2



p3



Now, compose them!

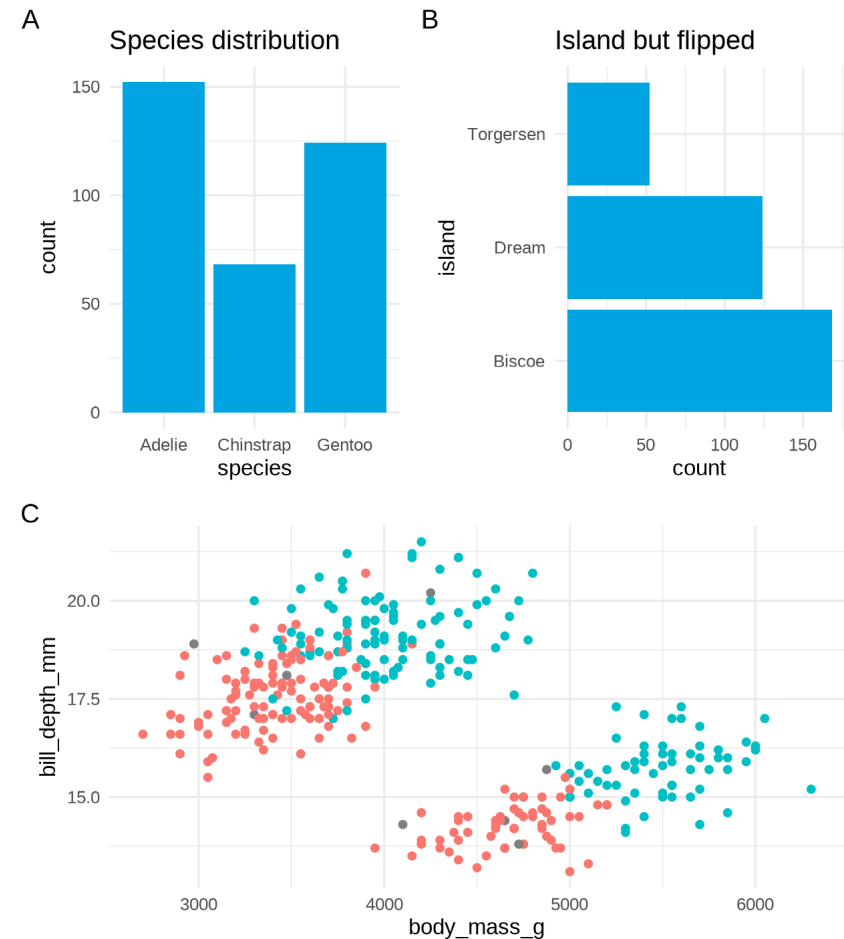
Compose plots with patchwork



patchwork provides an API using the classic arithmetic operators

```
library(patchwork)
(( p1 | p2 ) / p3) +
  # add tags and main title
  plot_annotation(tag_levels = 'A',
                  title = 'Plots about penguins') &
  # modify all plots recursively
  theme_minimal() +
  theme(text = element_text('Roboto'))
```

Plots about penguins



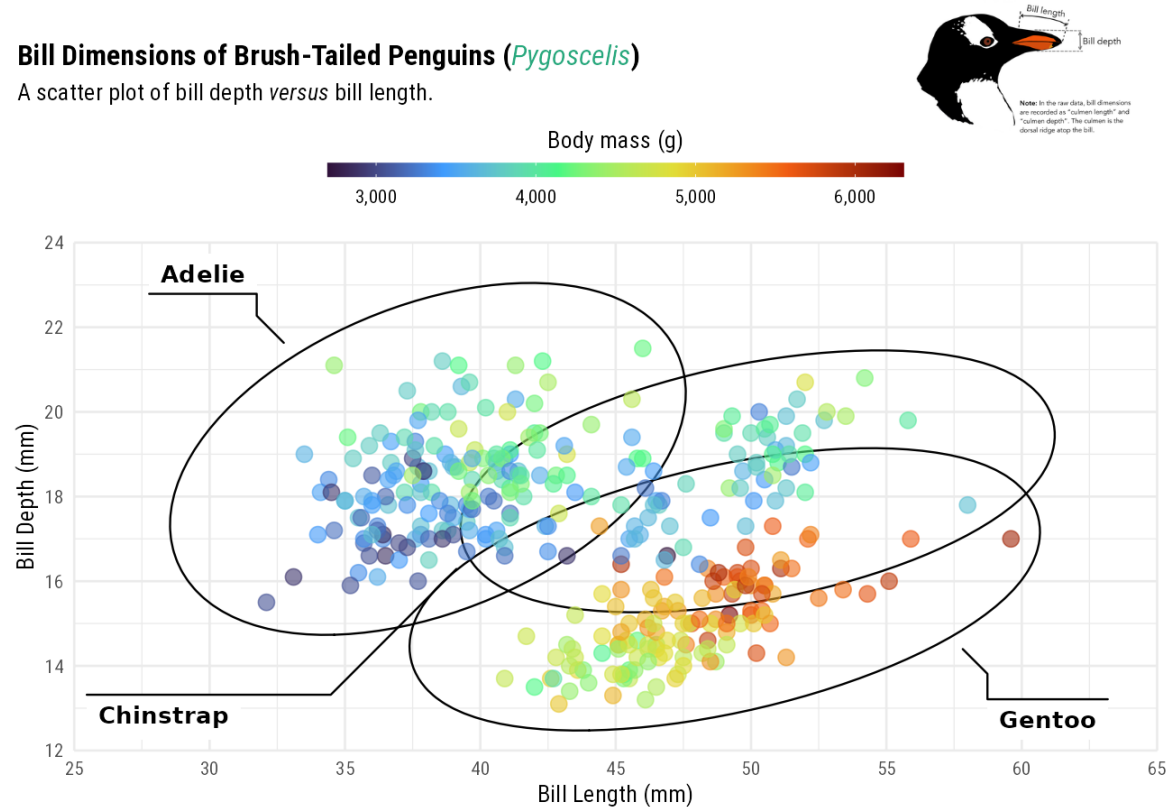
If you need to perfectly align axes, have a look at [cowplot](#) by **Claus O. Wilke**

Polished example by Cédric Scherer

Using [patchwork](#), [ggforce](#) (T. Pedersen) and [ggtext](#) (Claus O. Wilke)

Bill Dimensions of Brush-Tailed Penguins (*Pygoscelis*)

A scatter plot of bill depth versus bill length.



Data: Gorman, Williams & Fraser (2014) *PLoS ONE* • Illustration: Allison Horst

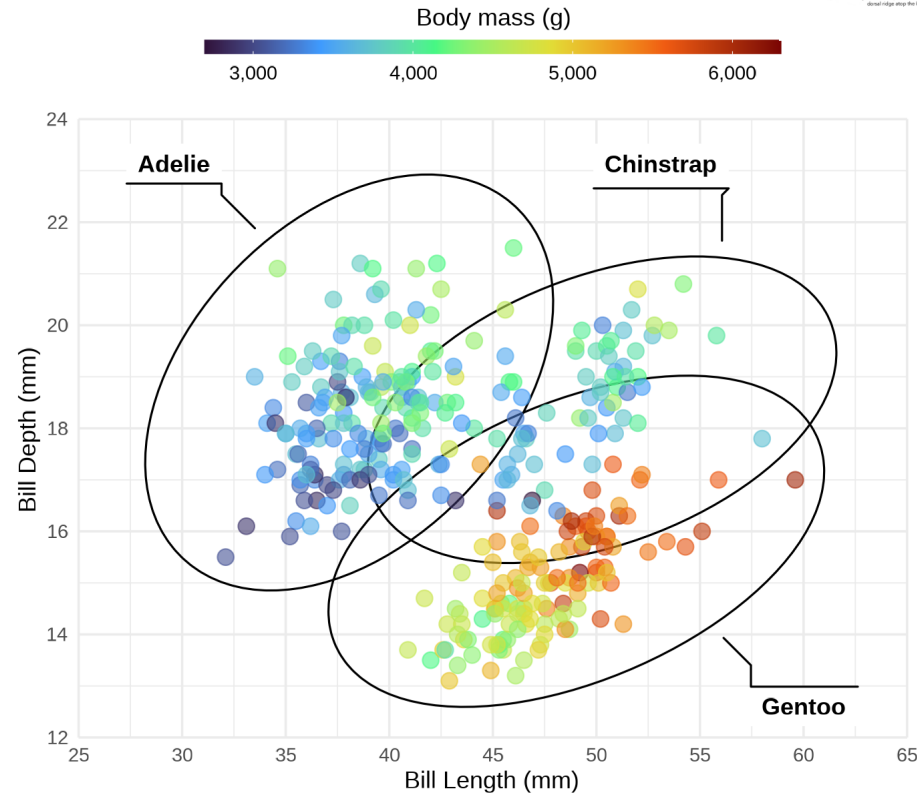
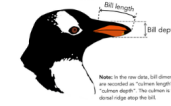
Source: [OutlierConf](#) by [Cédric Scherer](#), see also [code](#)



```
culmen <- png::readPNG(here::here("site/lectures/i
penguins %>%
  filter(across(ends_with("mm"), ~!is.na(.x))) %>%
  ggplot(aes(x = bill_length_mm, y = bill_depth_mm
ggforce::geom_mark_ellipse(
  aes(fill = species, label = species), alpha =
    show.legend = FALSE
) +
  geom_point(aes(color = body_mass_g), alpha = .6,
scale_x_continuous(breaks = seq(25, 65, by = 5),
scale_y_continuous(breaks = seq(12, 24, by = 2),
scale_color_viridis_c(option = "turbo", labels =
labs(
  title = "Bill Dimensions of Brush-Tailed Penguin
  subtitle = 'A scatter plot of bill depth *vers
  caption = "Data: Gorman, Williams & Fraser (20
  x = "Bill Length (mm)",
  y = "Bill Depth (mm)",
  color = "Body mass (g)") +
  theme_minimal(base_family = "RobotoCondensed", b
  theme(plot.title = element_markdown(face = "bold
    plot.subtitle = element_markdown(),
    plot.caption = element_markdown(margin = m
    axis.title.x = element_markdown(),
    axis.title.y = element_markdown()) +
  theme(plot.title.position = "plot") +
  theme(plot.caption.position = "plot") +
  theme(legend.position = "top") +
  guides(color = guide_colorbar(title.position = "
    title.hjust = .5,
    barwidth = unit(20
    barheight = unit(.
  coord_cartesian(expand = FALSE, clip = "off") +
  theme(plot.margin = margin(t = 25, r = 25, b = 1
  labs(caption = "Data: Gorman, Williams & Fraser
  patchwork::inset_element(culmen, left = 0.82, bo
```

Bill Dimensions of Brush-Tailed Penguins (*Pygoscelis*)

A scatter plot of bill depth *versus* bill length.

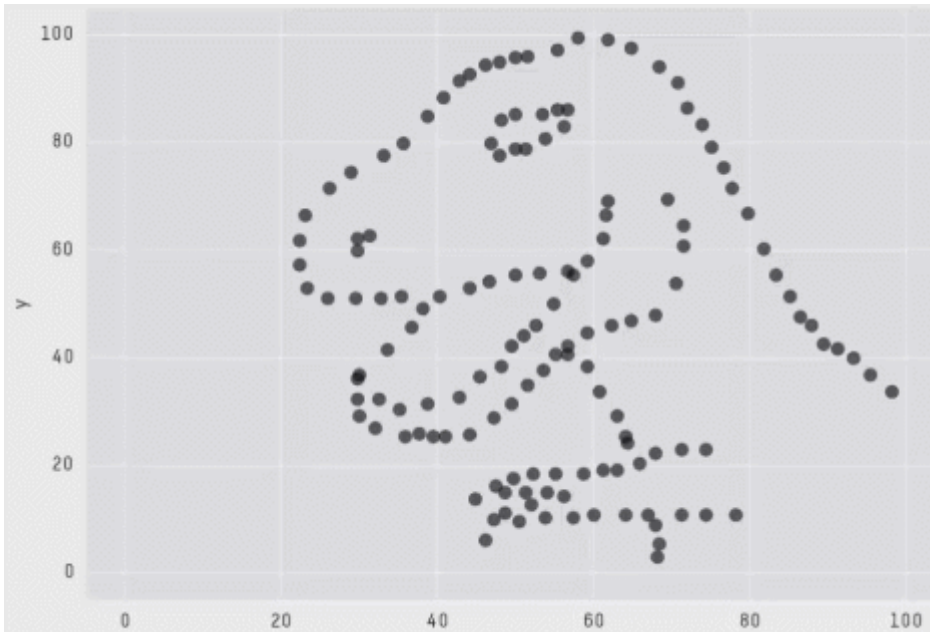


Data: Gorman, Williams & Fraser (2014) *PLoS ONE* • Illustration: Allison Horst

Plot your data!

Never trust summary statistics alone; always visualize your data

— *Alberto Cairo*



X Mean: 54.2659224
Y Mean: 47.8313999
X SD : 16.7649829
Y SD : 26.9342120
Corr. : -0.0642526

source: Justin Matejka, George Fitzmaurice [Same Stats, Different Graphs...](#)

Missing features

geoms list here

- `geom_tile()` heatmap
- `geom_bin2d()` 2D binning
- `geom_abline()` slope

stats list here

- `stat_ellipse()` (`ggforce` seen)
- `stat_summary()` easy mean, 95CI etc.

plot on multi-pages

- `ggforce::facet_grid_paginate()` facets
- `gridExtra::marrangeGrob()` plots

coordinate / transform

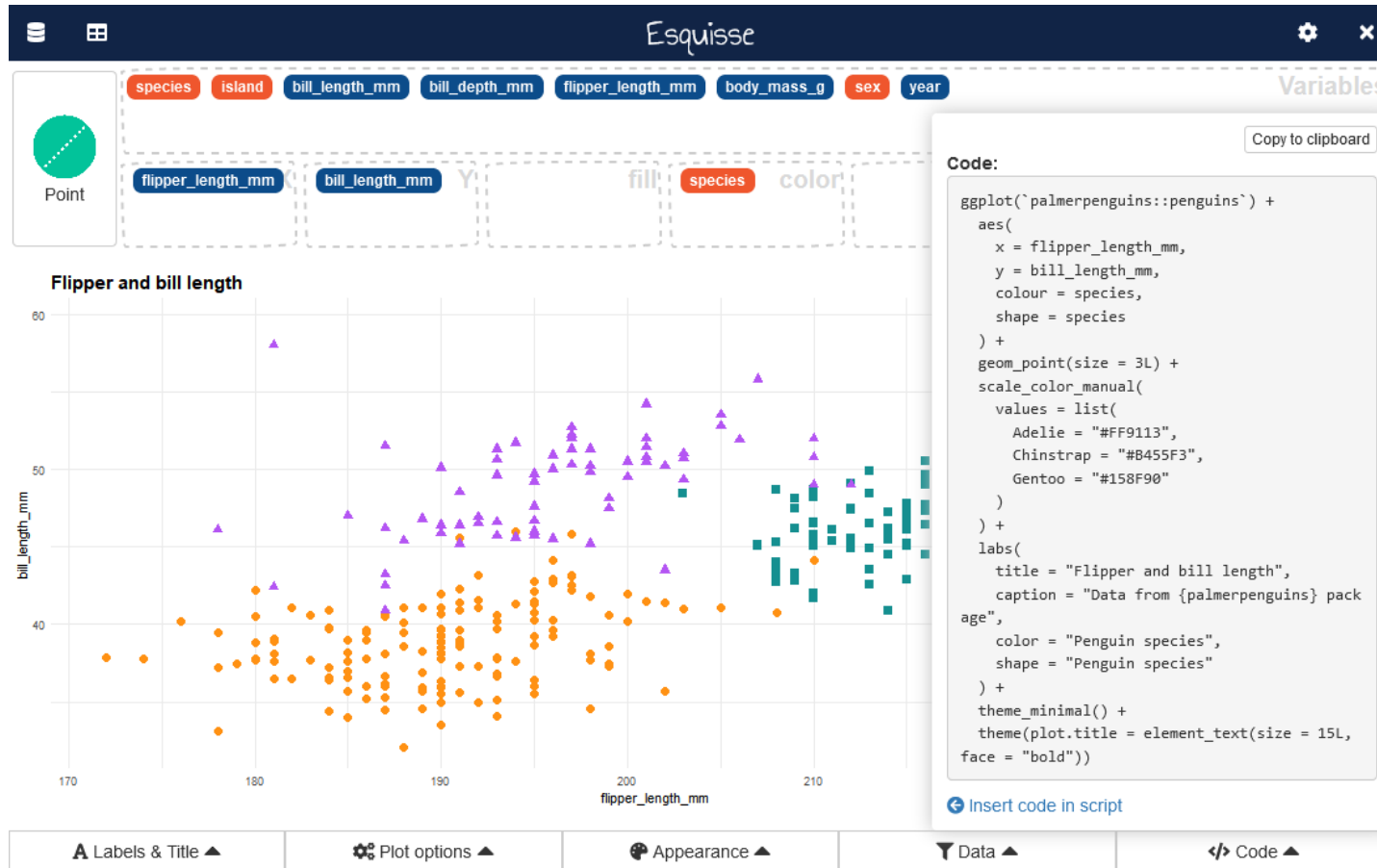
- `coord_cartesian()` for zooming in

customise theme elements

- legend & guide tweaks
- major/minor grids
- font, faces
- margins
- labels & ticks
- strip positions
- see [live examples](#) of pre-built themes

Build your plot with point and click

Using **esquisse** Rstudio addin by [dreamRs](#)



Before we stop

You learned to:

- Apprehend Graphics as a language
- Embrace the layer system
- Link data columns to aesthetics
- Discover geometries and scales
- The endless potential of cosmetic improvements

Acknowledgments ☐ ☐

- [Thomas Lin Pedersen](#), great webinar *Plotting everything with ggplot2*:
 - [part 1, ggplot2 API](#)
 - [part 2, extending ggplot2](#)
- [Thinkr](#) (Diane Beldame, Vincent Guyader, Colin Fay)
- [Allison M. Horst, Alison P. Hill and Kristen B. Gorman](#) *
- [DreamRs](#) (Victor Perrier, Fanny Meyer)
- [Romain Lesur](#) ([pagedown](#)), [Christophe Dervieux](#)
- [Hadley Wickham](#)
- [Cédric Scherer](#)

Further reading ☐

- [ggplot2 book](#) by H. Wickham (free access)
- [NEW official FAQ](#), axes, facets, custom, annot, reorder, bars
- [R for Data Science](#) by H. Wickham / G. Grolemund (free access)
- [TidyTuesday challenges](#)
- [Vizualisation gallery](#) by Cédric Scherer
- [plotting tutorial](#) by Cédric Scherer

Thank you for your attention!

*: Palmer penguins, data are available by CC-0 license and [Artwork](#) by Allison Horst