University of Antwerp

ico

ICO Workshop R & RStudio

Part 4

Powerful visualisations with `ggplot2`

Sven De Maeyer & Tine van Daal

23th - 25th November, 2022

Overview

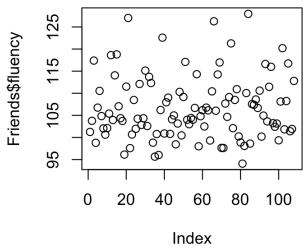
1. Simple plots in R

Plots in base

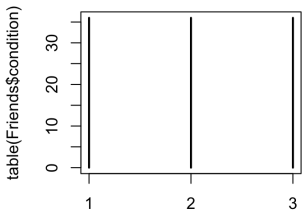
plot() boxplot() hist()

The generic function `plot()` "knows" what to do (plot) with the input it receives.

```
# one quantitatieve variable
plot(Friends$fluency)
```

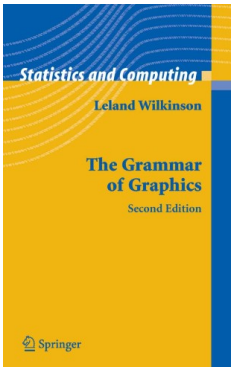



```
# one qualitative variables
plot(table(Friends$condition))
```



2. Grammar of Graphics

A more theoretical approach to visualisation



- Theoretical 'breakdown' of a visualisation into components (called layers)
- One system to create different visualisations
- At the heart of several modern graphical applications:
  - ggplot2
  - Tableau (Polaris)
  - Vega-Lite

Slide taken from slide show by Thomas Lin Pedersen

Key idea behind the Grammar of Graphics

Layers of a visualisation:
data
aesthetics
geometries
facets
statistics
coordinates
themes

Animation by Thomas de Beus

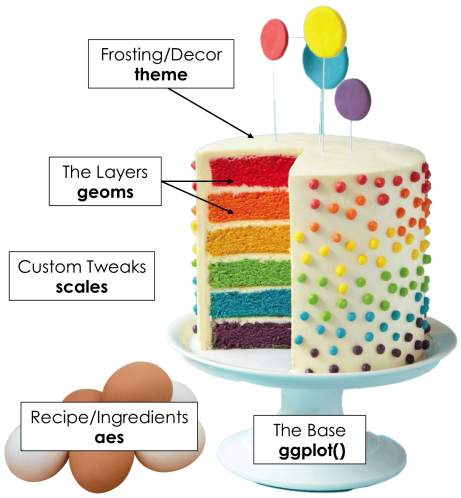
Grammar of Graphics is a bit like cake

Start by setting up the *foundation* with `ggplot()`

Specify *ingredients* (variables) with `aes()` and a *flavour* with `scales`

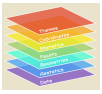
Create *layers* to plot with `geoms`

Style the cake with `theme`



Slide by Tanya Shapiro

Grammar of Graphics: data

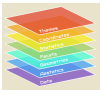


- Data is not only raw data, but can also be the results from an analysis

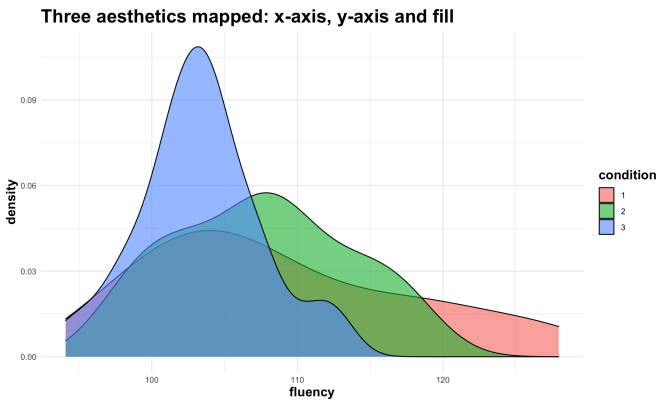
group	country	gender	mean_age	SD_age	CI_lower	CI_upper
BE Male	BE	Male	39	11.0	17.44	60.56
BE Female	BE	Female	41	13.2	15.13	66.87
BE Other	BE	Other	36	8.2	19.93	52.07
NL Male	NL	Male	37	12.0	13.48	60.52
NL Female	NL	Female	36	14.0	8.56	63.44
NL Other	NL	Other	31	7.2	16.89	45.11

- Data has to be 'tidy'

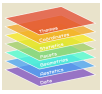
Grammar of Graphics: aesthetics



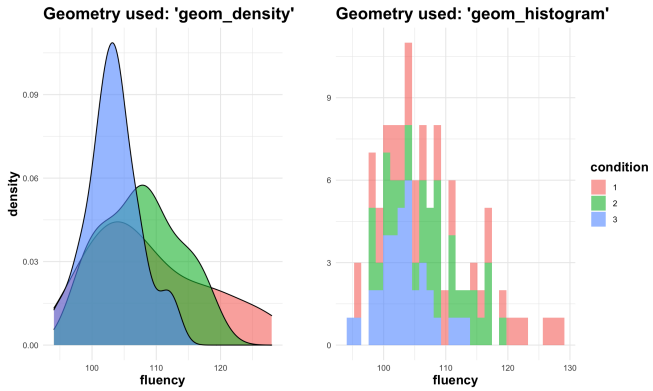
- Describe how variables in data are *mapped* to visual properties. For example:
  - variables mapped on x- and y-axis
  - variable that defines color or size of points
  - ...
- Change appearances of aesthetics using `scales`



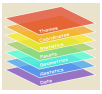
Grammar of Graphics: geometries



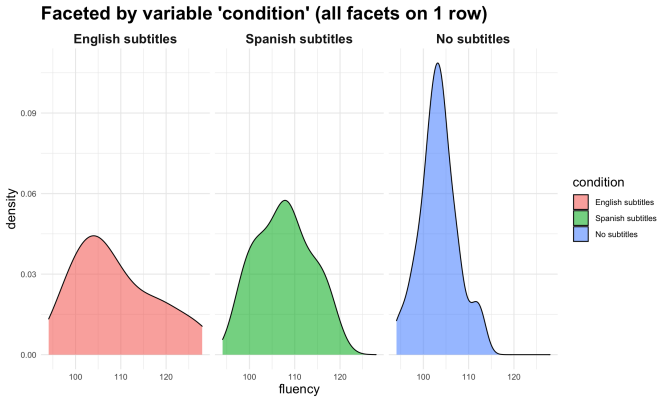
- Geometrical shapes at the heart of visualisation. For example:
  - boxplot
  - line
  - ...



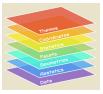
Grammar of Graphics: facets



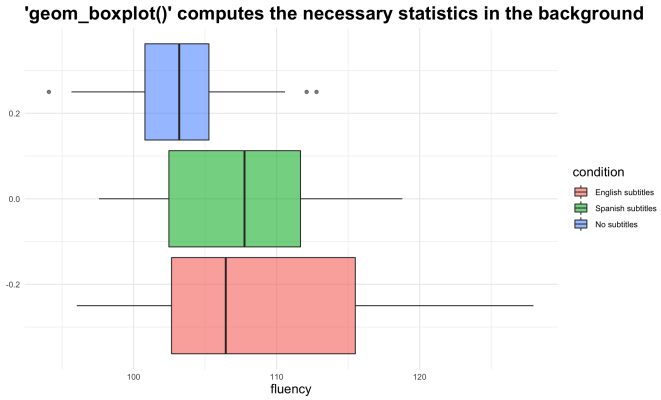
- Also called 'small multiples'
- Define how much panels are shown and how they are arranged



Grammar of Graphics: statistics



- Data might be `tidy`, but still in need of some statistical calculations. For example:
    - Calculate descriptive statistics to create a boxplot
    - Estimate a linear (or other) model to draw a regression line in a scatter plot
- Often implicit done by `ggplot2`

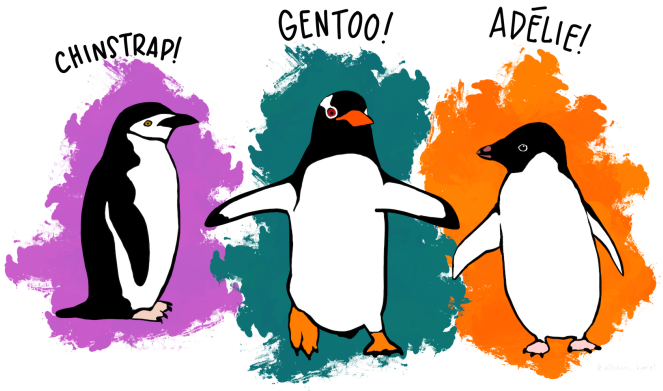


3. How `ggplot2` works (in a nutshell)

Time to get the penguins in...

Nice data set that can be used within R (Source: https://allisonhorst.github.io/palmerpenguins/articles/intro.html)

```
install.packages("palmerpenguins")
library(palmerpenguins)
data("penguins")
```



Artwork by @allison_horst

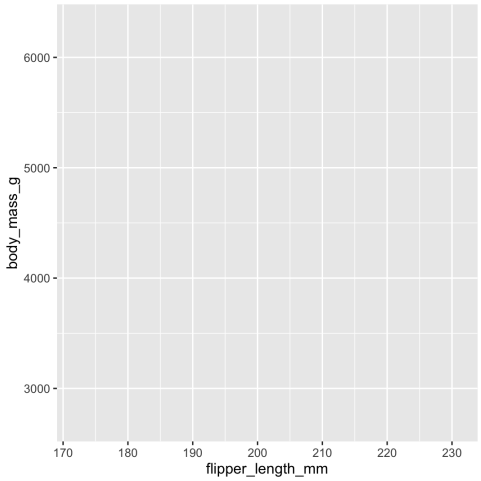
Time to get the penguins in...

Table 1. Random sample of 10 observations from the Palmer Pinguins dataset

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
Adelie	Dream	36.0	18.5	186	3100	female	2007
Adelie	Torgersen	41.4	18.5	202	3875	male	2009
Gentoo	Biscoe	42.6	13.7	213	4950	female	2008
Adelie	Torgersen	38.7	19.0	195	3450	female	2007
Gentoo	Biscoe	48.8	16.2	222	6000	male	2009
Adelie	Torgersen	37.8	17.1	186	3300	NA	2007
Adelie	Dream	36.6	18.4	184	3475	female	2009
Chinstrap	Dream	51.3	18.2	197	3750	male	2007
Gentoo	Biscoe	50.5	15.9	225	5400	male	2008
Adelie	Biscoe	36.5	16.6	181	2850	female	2008

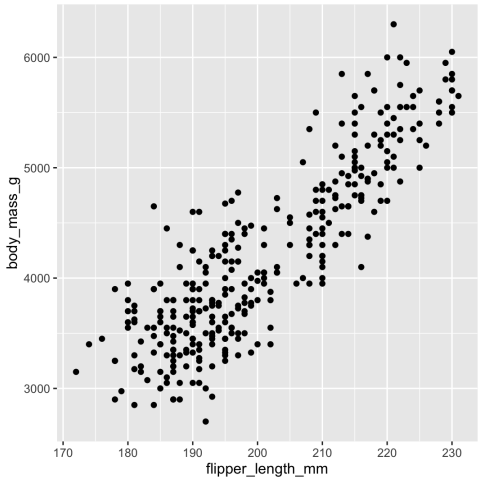
The basics of ggplot2: *data & aesthetics*

```
Plot <- ggplot(

  ## Step 1: data
  data = penguins,

  ## Step 2: specify aesthetics (mapping)
  aes(
    x = flipper_length_mm,
    y = body_mass_g)
)

Plot
```



The basics of ggplot2: *geometry*

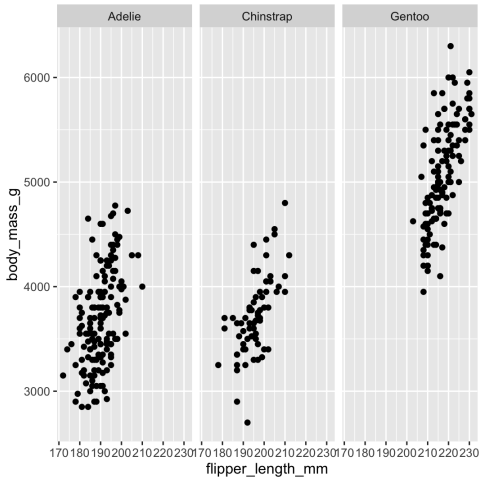
```
Plot <- ggplot(

  ## Step 1: data
  data = penguins,

  ## Step 2: specify aesthetics (mapping)
  aes(
    x = flipper_length_mm,
    y = body_mass_g)
) +
  ## Step 3: add geometry
  geom_point()

Plot
```



Not every component of the *Grammar of Graphics* needs to be defined. The other components have *default* values that are automatically applied.

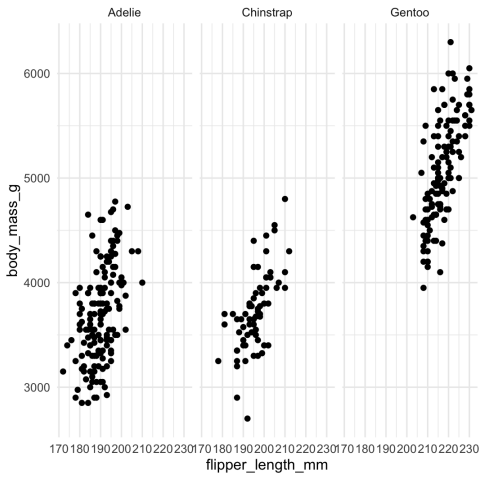
The basics of ggplot2: *facets*

```
Plot <- ggplot(

  ## Step 1: data
  data = penguins,

  ## Step 2: specify aesthetics (mapping)
  aes(
    x = flipper_length_mm,
    y = body_mass_g)
) +
  ## Step 3: add geometry
  geom_point() +

  ## Step 4: define facets
  facet_wrap(~species)


Plot
```

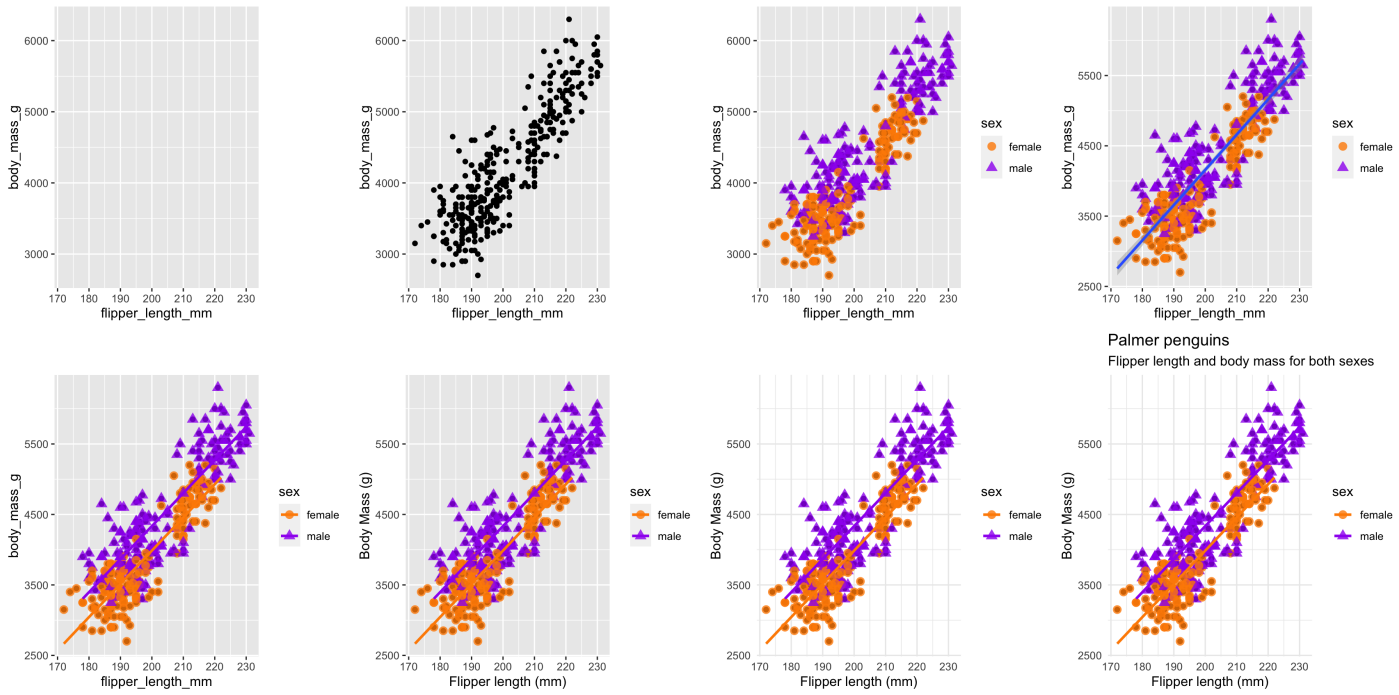


The basics of ggplot2: *theme*

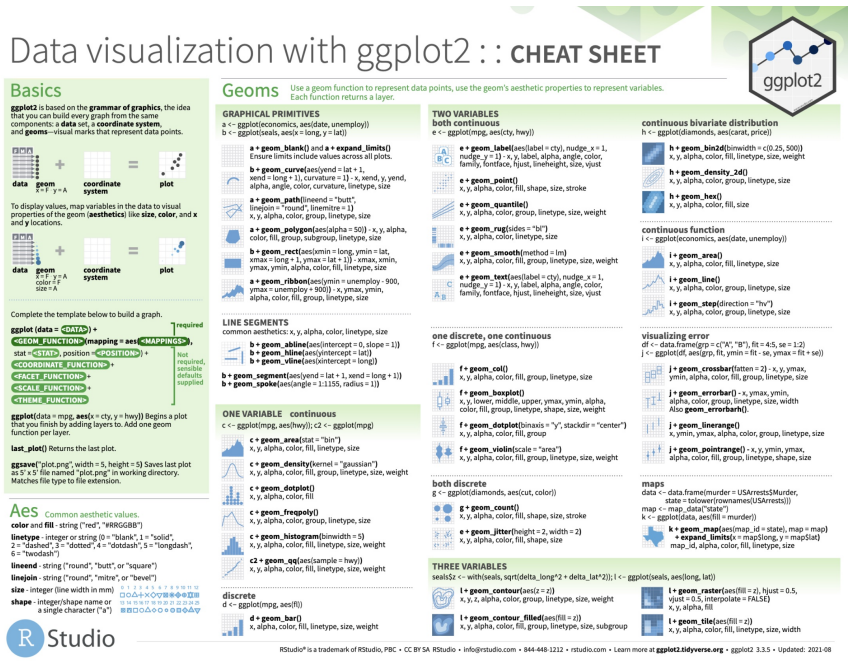
```
Plot <- ggplot(

  ## Step 1: data
  data = penguins,

  ## Step 2: specify aesthetics (mapping)
  aes(
    x = flipper_length_mm,
    y = body_mass_g)
) +
  ## Step 3: add geometry
  geom_point() +

  ## Step 4: define facets
  facet_wrap(~species) +

  ## Step 5: set theme
  theme_minimal()


Plot
```



Building a visualisation by adding layers ...



Several `geom_*` options...



The *cheatsheet*: https://github.com/rstudio/cheatsheets/blob/main/data-visualization-2.1.pdf

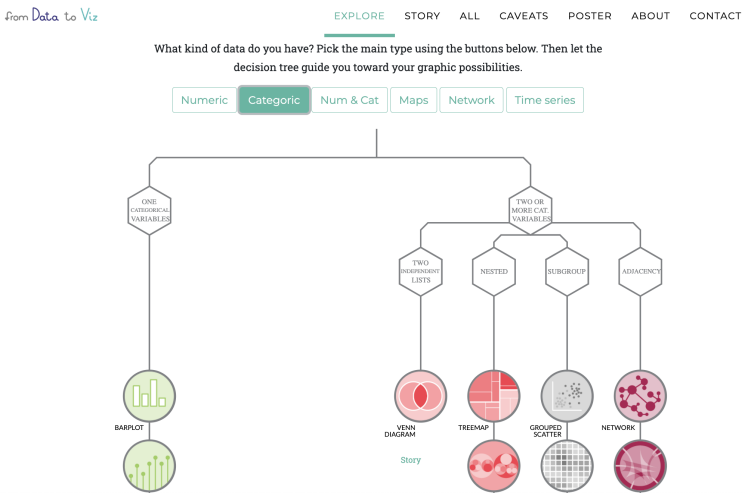
4. Visualising a categorical variable

"The bar is open... Let's have a lollipop?"

Visualising a categorical variable?

There are many ways to visualise a categorical variable...

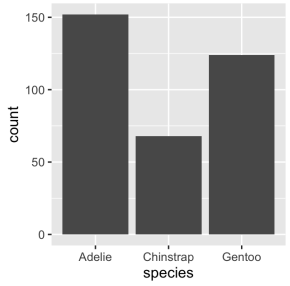
Can you think of some?



Have a look at data to viz.com

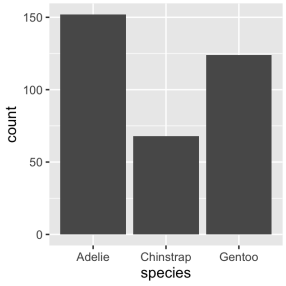
Creating a barplot with `geom_bar()` or `geom_col()`

`geom_bar()`




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

# stat_count() does the counting automatically
```

`geom_col()`

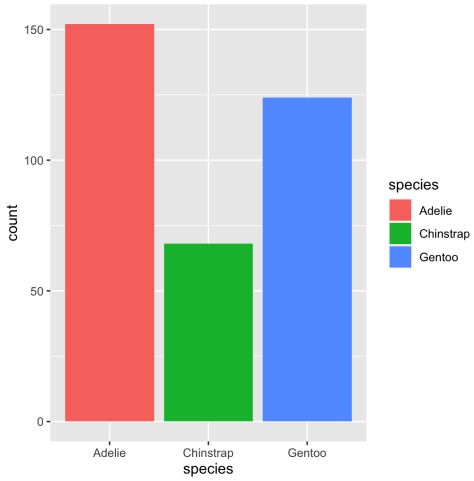



```
count_data <- penguins %>%
  count(species, name = 'count')

ggplot(count_data,
       aes(
         x = species,
         y = count
       )) +
  geom_col()
```


Creating a barplot with `geom_bar()`

- Add color to barplot by defining an additional *aesthetic*: **`fill`**

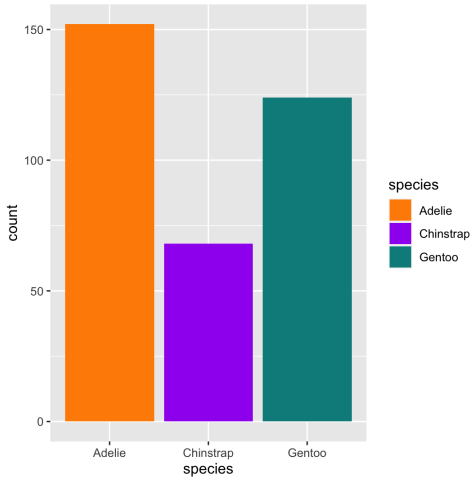
```
ggplot(penguins,
       aes(
         x = species
       )) +
  geom_bar(
    ## Additional aesthetic: "fill"-scale
    aes(fill = species)
    )
```



Creating a barplot with `geom_bar()`

- Determine "fill"-colors by adding **`scale_fill_manual()`**

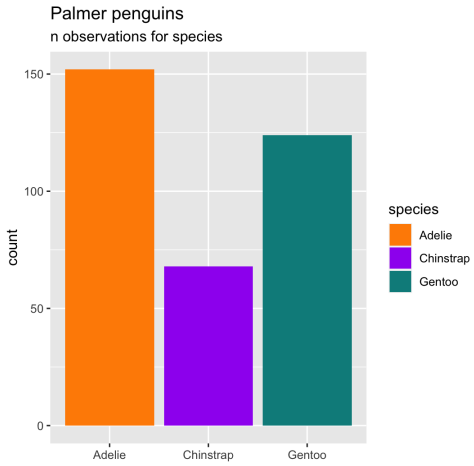
```
ggplot(penguins,
       aes(
         x = species
       )) +
  geom_bar(
    aes(fill = species)
    ) +
  ## Specify fill-colors
  scale_fill_manual(
    values = c("darkorange", "purple", "cyan4")
    )
```



Creating a barplot with `geom_bar()`

- Add title, subtitle and change label of x-axis using **`labs()`**

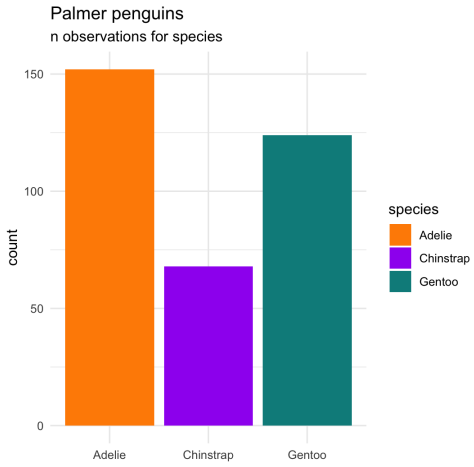
```
ggplot(penguins,
       aes(
         x = species
       )) +
  geom_bar(
    aes(fill = species)
    ) +
  scale_fill_manual(
    values = c("darkorange","purple","cyan4")
    ) +
  ## Add title, subtitle and label x-axis
  labs(
    title = "Palmer penguins",
    subtitle = "n observations for species",
    x = ""
  )
```



Creating a barplot with `geom_bar()`

- Add another *theme* using **`theme_minimal()`**

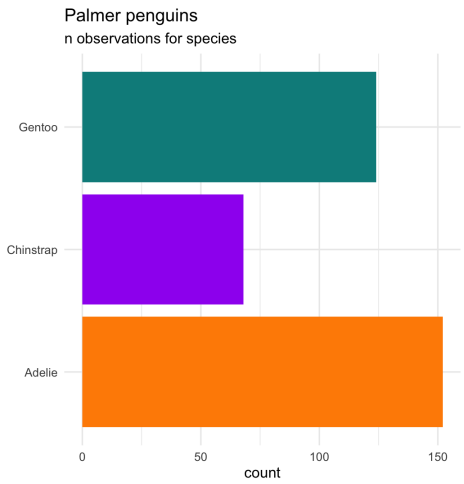
```
ggplot(penguins,
       aes(
         x = species
       )) +
  geom_bar(
    aes(fill = species)
    ) +
  scale_fill_manual(
    values = c("darkorange","purple","cyan4")
    ) +
  labs(
    title = "Palmer penguins",
    subtitle = "n observations for species",
    x = ""
  ) +
  ## Choose theme
  theme_minimal()
```



Creating a barplot with `geom_bar()`

- Flip x- and y-axis using **`coord_flip()`**
- Remove legend using **`theme(legend.position = "none")`**

```
ggplot(penguins,
       aes(
         x = species
       )) +
  geom_bar(
    aes(fill = species)
    ) +
  scale_fill_manual(
    values = c("darkorange","purple","cyan4")
    ) +
  labs(
    title = "Palmer penguins",
    subtitle = "n observations for species",
    x = ""
  ) +
  ## Flip x- and y-axis
  coord_flip( ) +
  theme_minimal( ) +
  ## Remove legend
  theme(
    legend.position = "none"
  )
```



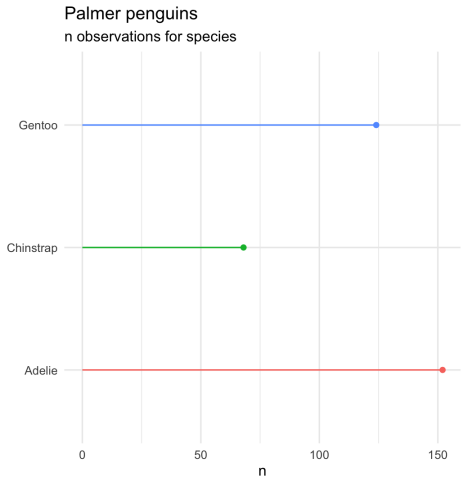
Creating a lollipop plot

Use two *geoms*: `geom_point()` and `geom_segment()`

```
penguins %>%
  count(species) %>%
  ggplot(aes(x = species, y = n)) +
  geom_point(
    aes(col = species)
  ) +
  geom_segment(
    aes(x = species, xend = species,
        y = 0, yend = n, col = species)
    ) +
```

Reuse remainder of the code!

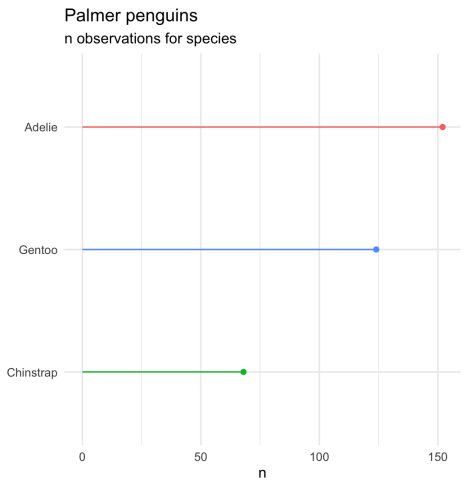
```
  scale_fill_manual(
    values = c("darkorange","purple","cyan4")
    ) +
  labs(
    title = "Palmer penguins",
    subtitle = "n observations for species",
    x = ""
  ) +
  coord_flip( ) +
  theme_minimal( ) +
  theme(
    legend.position = "none"
  )
```



Creating a lollipop plot

- Reorder 'bars' by creating a new variable using **`reorder()`**

```
penguins %>%
  count(species) %>%
  ggplot(
    aes(x = reorder(species, n), y = n) +
  geom_point(
    aes(col = species)
  ) +
  geom_segment(
    aes(x = species, xend = species,
        y = 0, yend = n, col = species)
    ) +
  scale_fill_manual(
    values = c("darkorange","purple","cyan4")
    ) +
  labs(
    title = "Palmer penguins",
    subtitle = "n observations for species",
    x = ""
  ) +
  coord_flip( ) +
  theme_minimal( ) +
  theme(legend.position = "none")
```



Chosing colors ...

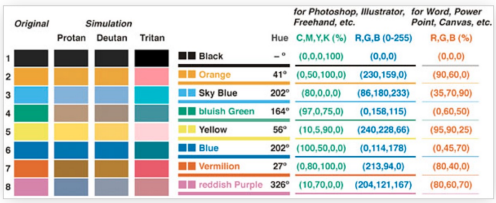
A colorblind-friendly palette

These are color-blind-friendly palettes, one with gray, and one with black.

To use with ggplot2, it is possible to store the palette in a variable, then use it later.

```
# The palette with grey:
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")

# The palette with black:
cbbPalette <- c("#000000", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")

# To use for fills, add
  scale_fill_manual(values=cbPalette)

# To use for line and point colors, add
  scale_colour_manual(values=cbPalette)
```

This palette is from http://jfly.iam.u-tokyo.ac.jp/color/:



	Original	Protan	Deutan	Tritan		Hue	for Photoshop, Illustrator, Freehand, etc. C,M,Y,K (%)	R,G,B (0-255)	for Word, Power Point, Canvas, etc. R,G,B (%)
1					Black	– °	(0,0,0,100)	(0,0,0)	(0,0,0)
2					Orange	41°	(0,50,100,0)	(230,159,0)	(90,60,0)
3					Sky Blue	202°	(80,0,0,0)	(86,180,233)	(35,70,90)
4					bluish Green	164°	(97,0,75,0)	(0,158,115)	(0,60,50)
5					Yellow	56°	(10,5,90,0)	(240,228,66)	(95,90,25)
6					Blue	202°	(100,50,0,0)	(0,114,178)	(0,45,70)
7					Vermilion	27°	(0,80,100,0)	(213,94,0)	(80,40,0)
8					reddish Purple	326°	(10,70,0,0)	(204,121,167)	(80,60,70)

http://www.cookbook-r.com/Graphs/Colors_(ggplot2)/

Exercises [ggplot2]: part 1

- You can find the qmd-file `Exercises_ggplot2.qmd` on the Open Science Framework (Exercises > Exercise3_ggplot2)
- Download this file to your laptop
- Open the file in `RStudio`
- The file contains a set of coding assignments with empty code blocks
- Now, we focus on part 1 of the exercises
- Write the code (and test it by running it)
- Stuck? No Worries!
  - We are there
  - Help each other
  - There is a solution key (`Exercises_ggplot2_solutions.qmd`)

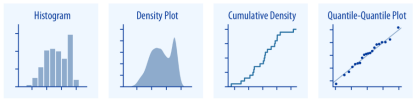
5. Visualising a quantitative variable

Visualising a quantitative variable?

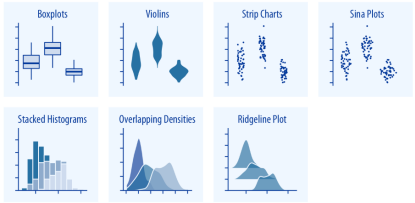
There are many ways to visualise a quantitative variable...

Can you think of some?

5.2 Distributions



Histograms and density plots (Chapter 7) provide the most intuitive visualizations of a distribution, but both require arbitrary parameter choices and can be misleading. Cumulative densities and quantile-quantile (q-q) plots (Chapter 8) always represent the data faithfully but can be more difficult to interpret.

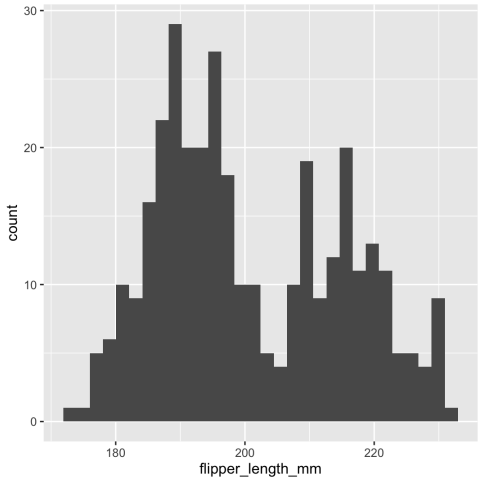


Boxplots, violins, strip charts, and sina plots are useful when we want to visualize many distributions at once and/or if we are primarily interested in overall shifts among the distributions (Chapter 9.1). Stacked histograms and overlapping densities allow a more in-depth comparison of a smaller number of distributions, though stacked histograms can be difficult to interpret and are best avoided (Chapter 7.2). Ridgeline plots can be a useful alternative to violin plots and are often useful when visualizing very large numbers of distributions or changes in distributions over time (Chapter 9.2).

Taken from Fundamentals of Data Visualization by Claus Wilke

Creating a histogram with `geom_histogram()`

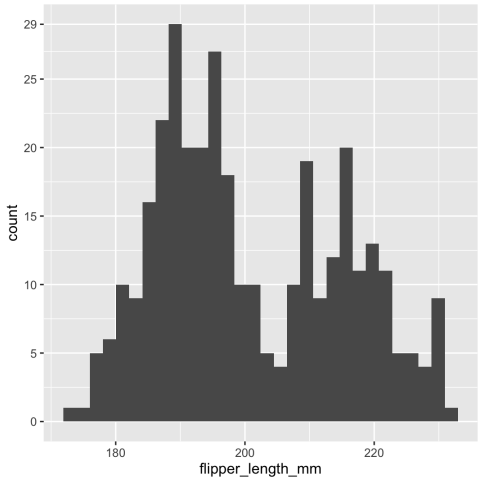
```
ggplot(penguins,
       aes(
         x = flipper_length_mm,
       )) +
  geom_histogram()
```



Creating a histogram with `geom_histogram()`

- Change breaks on y-axis using **`scale_y_continuous()`**

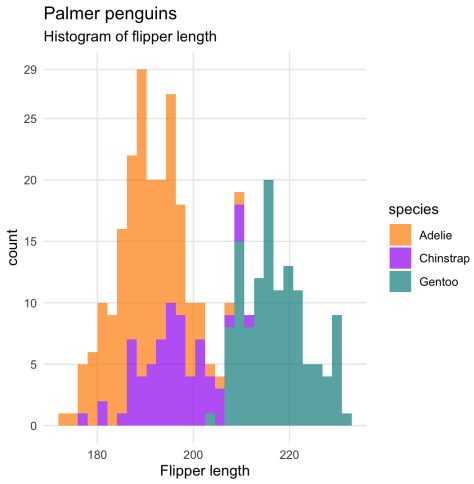
```
ggplot(penguins,
       aes(
         x = flipper_length_mm,
       )) +
  geom_histogram() +
  ## Specify breaks
  scale_y_continuous(
    breaks = c(seq(0, 25, 5), 29)
    )
```



Creating a histogram with `geom_histogram()`

- Change transparency using argument **`alpha`**
- Remove minor grid lines using argument **`panel.grid.minor = element_blank()`**

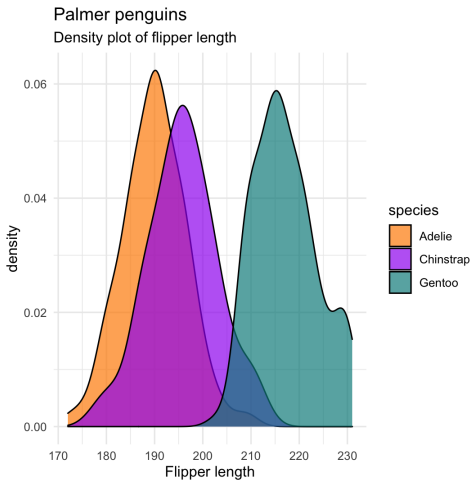
```
ggplot(penguins,
       aes(
         x = flipper_length_mm,
         fill = species
       )) +
  geom_histogram(
    ## Change transparency of points
    alpha = .7
  ) +
  ## Specify breaks
  scale_y_continuous(
    breaks = c(seq(0, 25, 5), 29)
    ) +
  scale_fill_manual(
    values = c("darkorange","purple","cyan4")
    ) +
  labs(
    title = "Palmer penguins",
    subtitle = "Histogram of flipper length",
    x = "Flipper length"
  ) +
  theme_minimal() +
  theme(
    ## Remove minor grid lines
    panel.grid.minor = element_blank()
  )
```



Creating a density plot° with `geom_density()`

- Replace `geom_histogram()` with **`geom_density()`** plot

```
ggplot(penguins,
       aes(
         x = flipper_length_mm,
         fill = species
       )) +
  ## Use geom_density
  geom_density(
    alpha = .7
  ) +
  scale_fill_manual(
    values = c("darkorange","purple","cyan4")
    ) +
  labs(
    title = "Palmer penguins",
    subtitle = "Density plot of flipper length",
    x = "Flipper length"
  ) + theme_minimal()
```

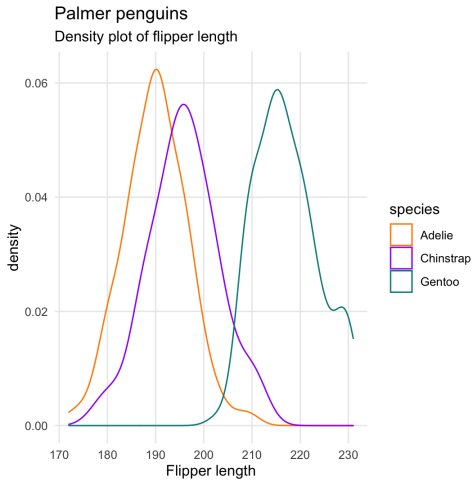


°*surface below curve = 100%*

Creating a density plot with `geom_density()`

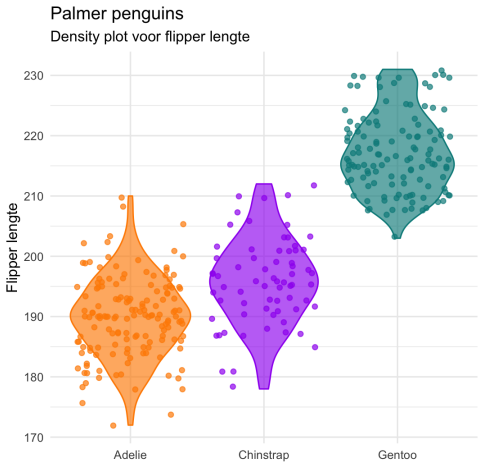
- Only colored lines by replacing argument **`fill`** with **`color`**

```
ggplot(penguins,
       aes(
         x = flipper_length_mm,
         color = species
       )) +
  geom_density() +
  ## Replace scale_color with scale_fill
  scale_color_manual(
    values = c("darkorange","purple","cyan4")
    ) +
  labs(
    title = "Palmer penguins",
    subtitle = "Density plot of flipper length",
    x = "Flipper length"
  ) +
  theme_minimal() +
  theme(panel.grid.minor = element_blank())
```



Or even better ... `geom_violin()` + `geom_jitter()`

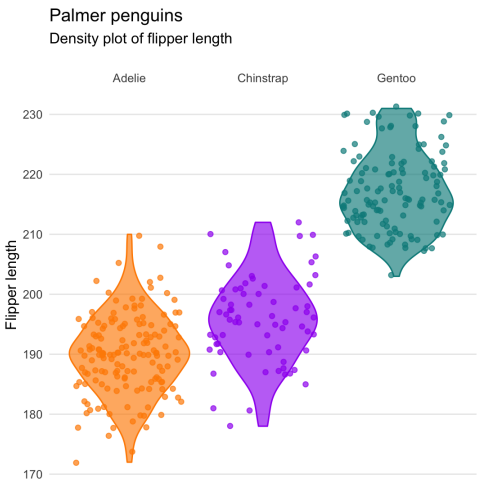
```
ggplot(penguins,
       aes(
         x = species,
         y = flipper_length_mm,
         ## Aesthetics fill and color applied to EVERY geom
         fill = species,
         color = species
       )) +
  ## Use geom_violin
  geom_violin(
    alpha = .65
  ) +
  ## Use geom_jitter
  geom_jitter(
    alpha = .7
  ) +
  scale_fill_manual(
    values = c("darkorange","purple","cyan4")
    ) +
  scale_color_manual(
    values = c("darkorange","purple","cyan4")
    ) +
  labs(
    title = "Palmer penguins",
    subtitle = "Density plot voor flipper lengte",
    y = "Flipper lengte",
    x = "",
  ) + theme_minimal() + theme(legend.position = "none")
```



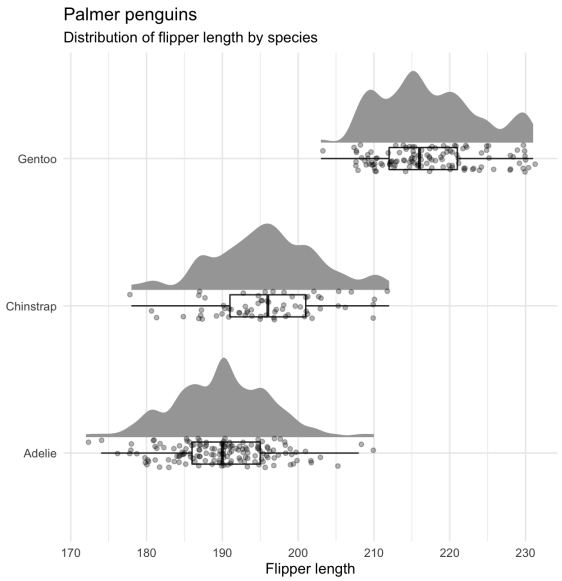
Or even better `geom_violin()+geom_jitter()`

- Put names of species on top using argument **`position="top"`**
- Remove minor grid lines using argument **`panel.grid.major = element_blank()`**

```
ggplot(penguins,
       aes(
         x = species,
         y = flipper_length_mm,
         fill = species,
         color = species
       )) +
  geom_violin(
    alpha = .65
  ) +
  geom_jitter(
    alpha = .7
  ) +
  scale_x_discrete(
    ## Put x-axis labels on top
    position = "top"
    ) +
  scale_fill_manual(
    values = c("darkorange","purple","cyan4")
    ) +
  scale_color_manual(
    values = c("darkorange","purple","cyan4")
    ) +
  labs(
    title = "Palmer penguins",
    subtitle = "Density plot of flipper length",
    y = "Flipper length",
    x = "",
  ) + theme_minimal() +
  theme(legend.position = "none",
        panel.grid.minor = element_blank(),
        ## Remove major grid lines
        panel.grid.major.x = element_blank())
```



Or a *rain cloud plot*



Based on the tutorial of Cédric Scherer: https://www.cedricscherer.com/

Or a *rain cloud plot* (Appendix 1: How to grow a rain cloud plot?)

Additional package required: `ggdist`

```
library(ggdist)
ggplot(penguins, aes(x = species, y = flipper_length_mm)) +
  stat_halfeye(
    adjust = .5,
    width = .6,
    .width = 0,
    justification = -.2,
    point_colour = NA
  ) +
  geom_boxplot(
    width = .15,
    outlier.shape = NA
  ) +
  geom_point(
    size = 1.3,
    alpha = .3,
    position = position_jitter(
      seed = 1, width = .1
  )) +
  labs(
    title = "Palmer penguins",
    subtitle = "Distribution of flipper length by species",
    x = "",
    y = "Flipper length"
  ) +
  coord_cartesian(xlim = c(1.2, NA), clip = "off") +
  coord_flip() +
  theme_minimal()
```


Exercises [ggplot2] : part 2

- You can find the qmd-file `Exercises_ggplot2.qmd` on the Open Science Framework (Exercises > Exercise3_ggplot2)
- Download this file to your laptop
- Open the file in `RStudio`
- The file contains a set of coding assignments with empty code blocks
- Now, we focus on part 2 of the exercises
- Write the code (and test it by running it)
- Stuck? No Worries!
  - We are there
  - Help each other
  - There is a solution key (`Exercises_ggplot2_solutions.qmd`)

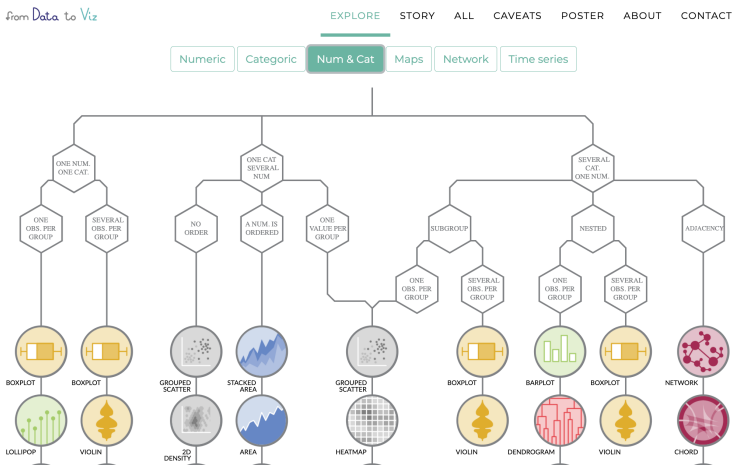
6. Visualising more than one variable

Visualising more than one variable?

There are many ways to visualise more than one variable... The choice depends (among other things) on the type of variables you want to plot:

- only quantitative variables;
- only qualitative variables;
- or a combination of both

Can you think of an example of each?



Have a look at data to viz.com

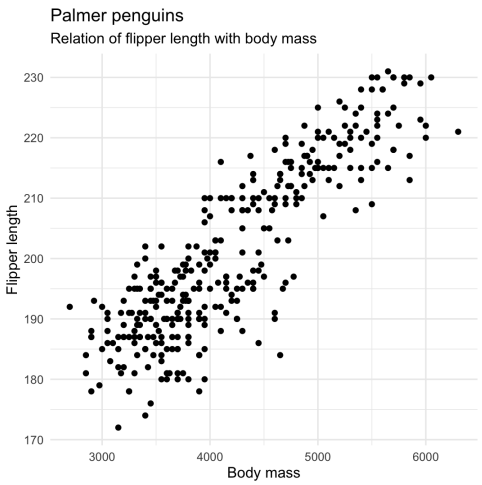
Visualising more than one variable

We focus today upon:

- **scatterplots** (two numeric variables, two numeric variables and one categorical variable)
- **grouped barplots** (two categorical variables)

Creating a scatterplot with `geom_point()`

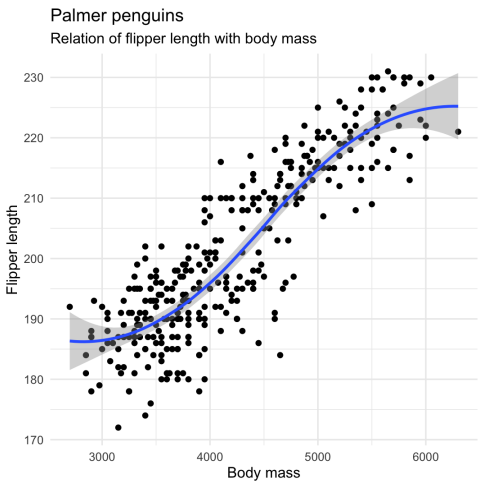
```
ggplot(penguins,
       aes(
         x = body_mass_g,
         y = flipper_length_mm
       )) +
  geom_point() +
  labs(
    title = "Palmer penguins",
    subtitle = "Relation of flipper length with body mass",
    y = "Flipper length",
    x = "Body mass",
  ) +
  theme_minimal() +
  theme(legend.position = "none")
```



Creating a scatterplot with `geom_point()` and `geom_smooth()`

- Add trend line by adding **`geom_smooth()`**

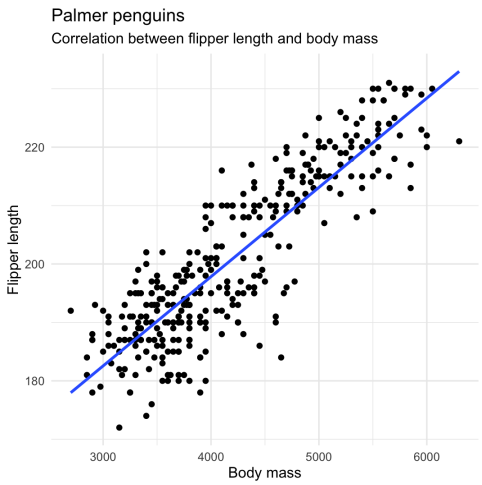
```
ggplot(penguins,
       aes(
         x = body_mass_g,
         y = flipper_length_mm
       )) +
  geom_point() +
  ## Add a trendline
  geom_smooth() +
  labs(
    title = "Palmer penguins",
    subtitle = "Relation of flipper length with body mass",
    y = "Flipper length",
    x = "Body mass"
  ) +
  theme_minimal() +
  theme(legend.position = "none")
```



Creating a scatterplot with `geom_point()` and `geom_smooth()`

- Linear trend line by adding argument **`method = "lm"`**
- Remove confidence intervals using argument **`se = FALSE`**

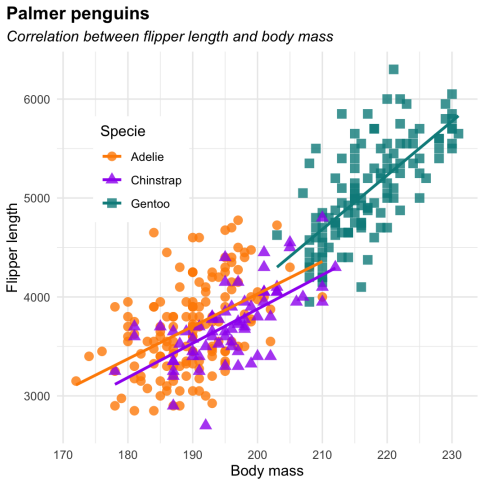
```
ggplot(penguins,
       aes(
         x = body_mass_g,
         y = flipper_length_mm
       )) +
  geom_point() +
  geom_smooth(
    ## Add a linear trend line
    method = "lm",
    ## Remove confidence band
    se = FALSE
  ) +
  labs(
    title = "Palmer penguins",
    subtitle = "Correlation between flipper length and body mass
    y = "Flipper length",
    x = "Body mass",
  ) +
  theme_minimal() +
  theme(legend.position = "none")
```



Creating a scatterplot with `geom_point()` and `geom_smooth()`

- Add shape and color by specifying additional *aesthetics*: **`shape`** and **`color`**
- Change legend position and background, change position of plot title, and layout (sub)title

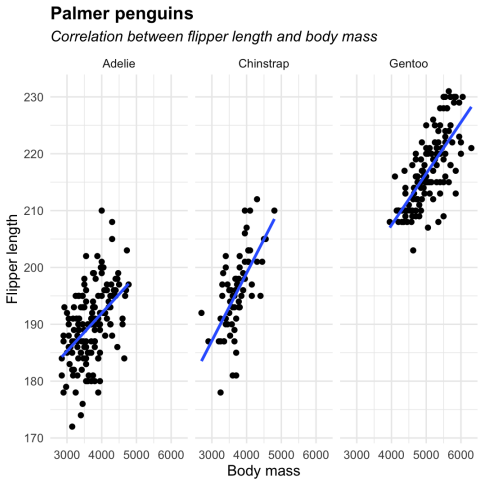
```
ggplot(data = penguins,
       aes(x = flipper_length_mm,
           y = body_mass_g)) +
  geom_point(aes(color = species,
                 ## Additional aesthetic: "shape"
                 shape = species),
             size = 3,
             alpha = 0.8) +
  geom_smooth(
    ## Additional aesthetic: "color"
    aes(color = species),
    se = F, method = "lm"
    ) +
  theme_minimal() +
  scale_color_manual(
    values = c("darkorange","purple","cyan4")) +
  labs(
    title = "Palmer penguins",
    subtitle = "Correlation between flipper length and body mass",
    y = "Flipper length",
    x = "Body mass",
    color = "Species",
    shape = "Species") +
  theme(## Change position legend
        legend.position = c(0.2, 0.7),
        ## Change background legend
        legend.background = element_rect(fill = "white", color = NA),
        ## Change position of plot title
        plot.title.position = "plot",
        ## Change layout of plot (sub)title
        plot.title = element_text(hjust = 0, face= "bold"),
        plot.subtitle = element_text(hjust = 0, face= "italic"))
```



Creating a scatterplot with `geom_point()` and `geom_smooth()`

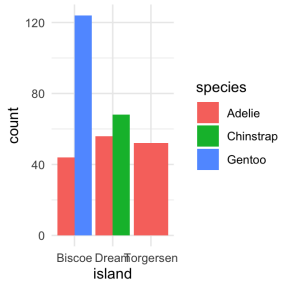
- Adds facets using **`facet_wrap()`**

```
ggplot(penguins,
       aes(
         x = body_mass_g,
         y = flipper_length_mm
       )) +
  geom_point( ) +
  geom_smooth(
    method = "lm",
    se = FALSE
  ) +
  ## Add facets
  facet_wrap(~species) +
  labs(
    title = "Palmer penguins",
    subtitle = "Correlation between flipper length and body mass",
    y = "Flipper length",
    x = "Body mass",
  ) +
  theme_minimal() +
  theme(legend.position = "none",
        plot.title = element_text(hjust = 0, face= "bold"),
        plot.subtitle = element_text(hjust = 0, face= "italic"))
```



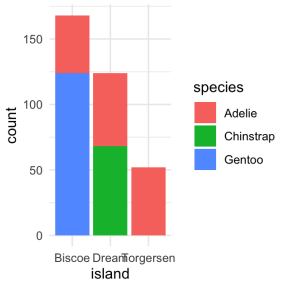
Creating a barplot of two variables (counts)

Grouped barplot of counts using **`position_dodge()`**




```
penguins %>%
  ggplot(aes(x = island,
             fill = species)) +
  geom_bar(
    position = position_dodge()
    ) +
  theme_minimal()
```

Stacked barplot of counts using argument **`position_stack()`**

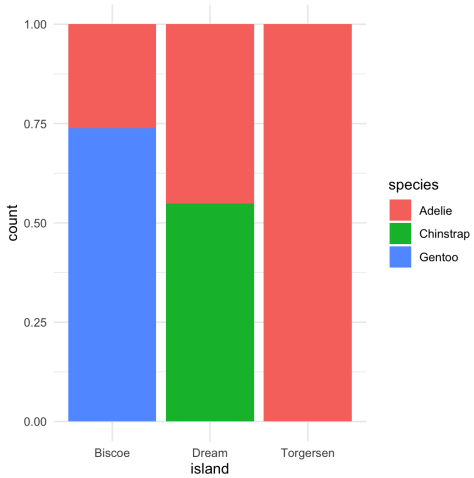



```
penguins %>%
  ggplot(aes(x = island,
             fill = species)) +
  geom_bar(
    position = position_stack()
    ) +
  theme_minimal()
```


Creating a barplot of two variables (percentage)

- Switch to barplot with relative frequencies using **`position_fill()`**

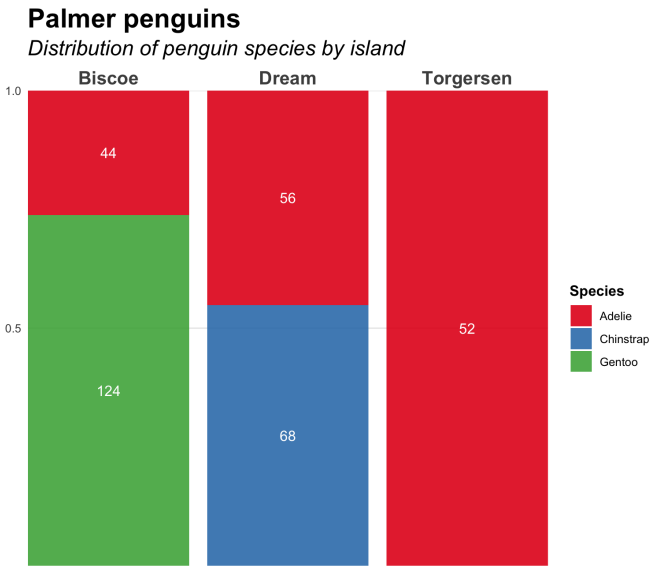
```
penguins %>%
  ggplot(aes(x = island,
             fill = species)) +
  geom_bar(
    position = position_fill()
    ) +
  theme_minimal()
```



Creating a barplot of two variables (percentage)

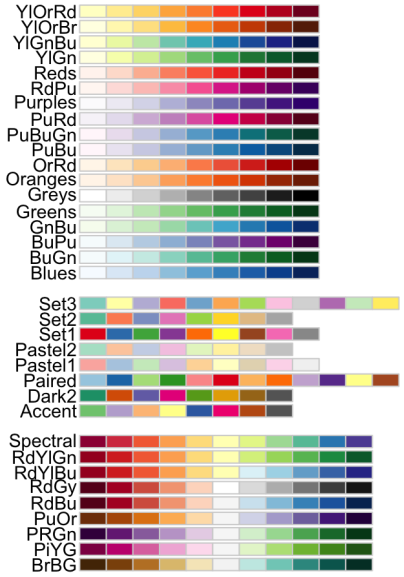
- Add texts to bars using **`geom_text()`**
- Add "fill" colors using **`scale_fill_brewer()`**
- Remove additional space using **`coord_cartesian(expand=FALSE)`**

```
penguins %>%
  ggplot(aes(x = island,
             fill = species)) +
  geom_bar(
    position = position_fill(),
    alpha = .9
    ) +
  ## Add text to bars
  geom_text(aes(label = ..count..),
            stat = "count",
            colour = "white",
            position = position_fill(vjust = 0.5)
            ) +
  ## Add "fill" colors
  scale_fill_brewer(
    type = "qual",
    palette = 6
  ) +
  scale_x_discrete(position = "top")+
  scale_y_continuous(breaks = c(0.5, 1)) +
  labs(
    title = "Palmer penguins",
    subtitle = "Distribution of penguin species by island",
    fill = "Species"
  ) +
  ## Remove additional space around plot
  coord_cartesian(expand = FALSE) +
  theme_minimal() +
  theme(plot.title = element_text(size = 20, face = "bold"),
        plot.subtitle = element_text(size = 16, face = "italic"),
        axis.title = element_blank(),
        axis.text.x = element_text(size = 14, face = "bold"),
        legend.title = element_text(face = "bold"),
        panel.grid.minor = element_blank(),
        panel.grid.major.x = element_blank())
```



Chosing colors ...

- Colors are not meaningless and ... they grab attention
- R has several built-in color palettes. The functions `scale_fill_brewer` and `scale_color_brewer` use palettes of RColorBrewer. These palettes come in three 'flavours':
  - **sequential**: for ordered data ((for example, scoring low/mediocre/high on a likert scale)
  - **qualitative**: for nominal or categorical information (for example, different islands)
  - **diverging**: for ordered data with meaningful mid values ((for example, temperature or change in score)



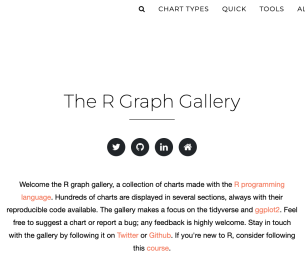
Exercises [ggplot2] : part 3

- You can find the qmd-file `Exercises_ggplot2.qmd` on the Open Science Framework (Exercises > Exercise3_ggplot2)
- Download this file to your laptop
- Open the file in `RStudio`
- The file contains a set of coding assignments with empty code blocks
- Now, we focus on part 3 of the exercises
- Write the code (and test it by running it)
- Stuck? No Worries!
    - We are there
    - Help each other
    - There is a solution key (`Exercises_ggplot2_solutions.qmd`)

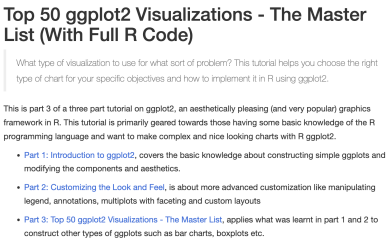
7. More about visualisation?

The tip of the icebeRg...

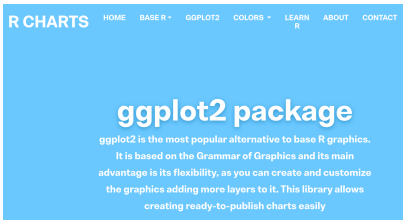
Gallery of different types of visualisations accompanied by ggplot-code



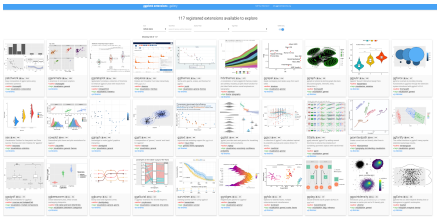
Top50 ggplot visualisations...



library of graphs with ggplot-code



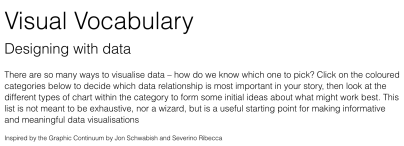
ggplot-extentions



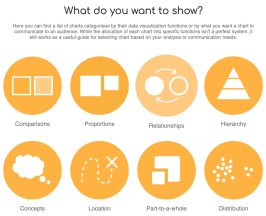
Data visualisation resources

Which visualisation to use?

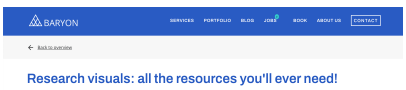
Visual vocabulary



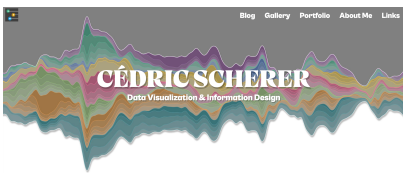
The Data Visualisation Catalogue



Research visuals: all the resources ...



Website of Cedric Scherer



In need of help?

Don't forget RStudio's help-function!

Files | Plots | Packages | Help | Viewer | Presentation

R: Points — Find in Topic

geom_point {ggplot2} — R Documentation

Points

Description

The point geom is used to create scatterplots. The scatterplot is most useful for displaying the relationship between two continuous variables. It can be used to compare one continuous and one categorical variable, or two categorical variables, but a variation like geom_jitter(), geom_count(), or geom_bin2d() is usually more appropriate. A bubblechart is a scatterplot with a third variable mapped to the size of points.

Usage

```
geom_point(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes() or aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer, as a string.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
...	Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat.
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders().

Overplotting

The biggest potential problem with a scatterplot is overplotting: whenever you have more than a few points, points may be plotted on top of one another. This can severely distort the visual appearance of the plot. There is no one solution to this problem, but there are some techniques that can help. You can add additional information with geom_smooth(), geom_quantile() or geom_density_2d(). If you have few unique x values, geom_boxplot() may also be useful.

Alternatively, you can summarise the number of points at each location and display that in some way, using geom_count(), geom_hex() or geom_density2d().

Another technique is to make the points transparent (e.g. geom_point(alpha = 0.05)) or very small (e.g. geom_point(shape = ".")).

Aesthetics

geom_point() understands the following aesthetics (required aesthetics are in bold):

- x
- y
- alpha
- colour
- fill
- group
- shape
- size
- stroke

Learn more about setting these aesthetics in vignette("ggplot2-specs").

Examples

```
p <- ggplot(mtcars, aes(wt, mpg))
```

Google is your best fRiend...

Just google your question and you'll find code, examples, ...

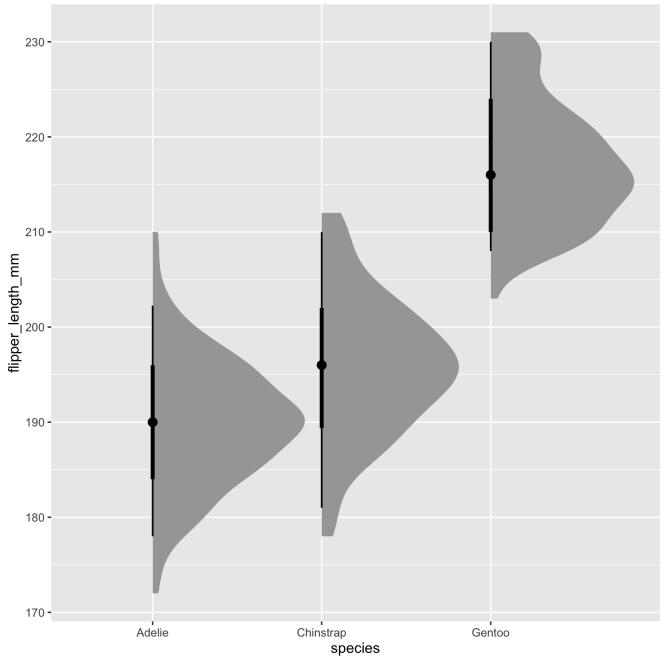
Appendix 1

How to grow a rain cloud plot?

Back to slide show...

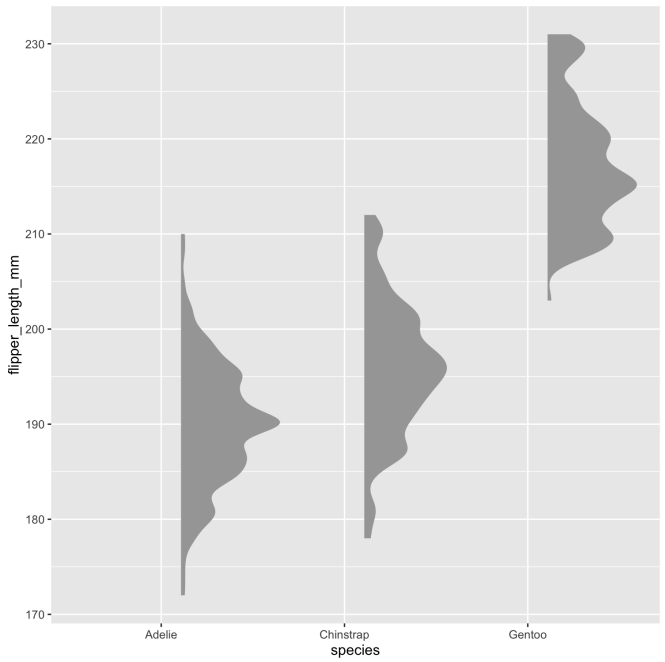
Step 1

```
library(ggdist)

P1 <-
  ggplot(penguins, aes(x = species, y = flipper_length_mm)) +
  stat_halfeye(
  )

P1
```



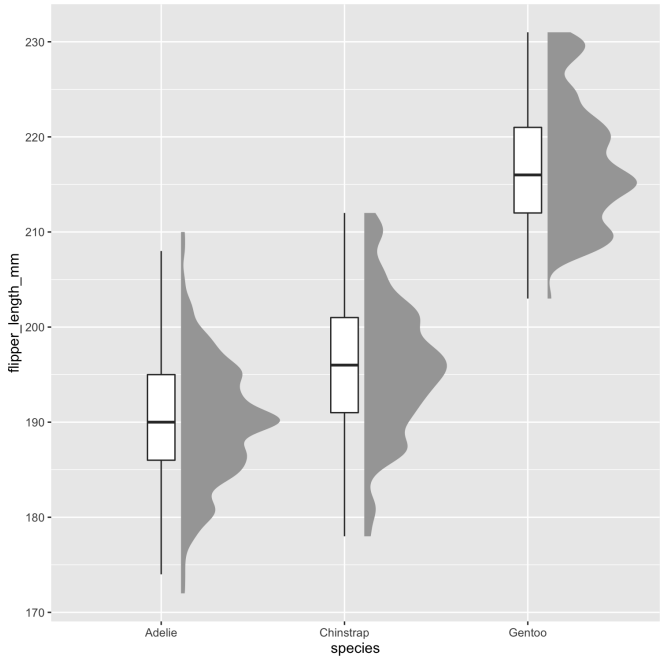
Step 2

```
P1 <-
  ggplot(penguins, aes(x = species, y = flipper_length_mm)) +
  stat_halfeye(
    adjust = .5,
    width = .6,
    .width = 0,
    justification = -.2,
    point_colour = NA
  )

P1
```



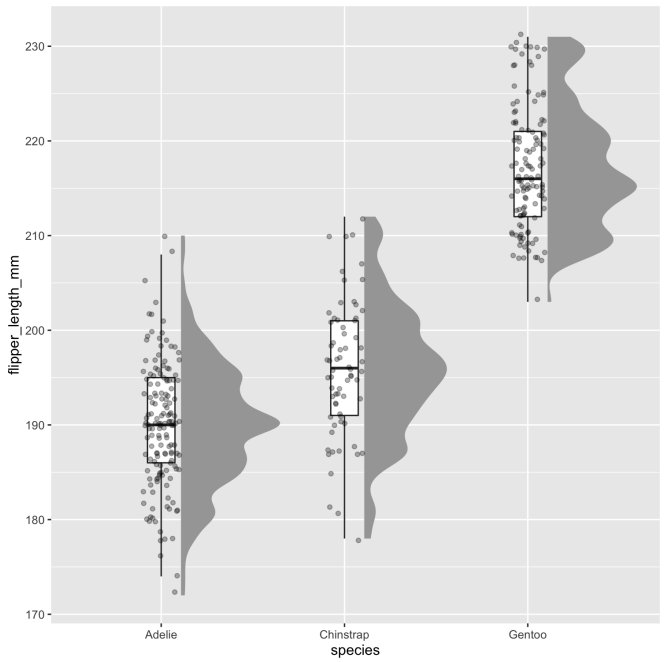
Step 3

```
P2 <- P1 +
  geom_boxplot(
    width = .15,
    outlier.shape = NA
  )

P2
```



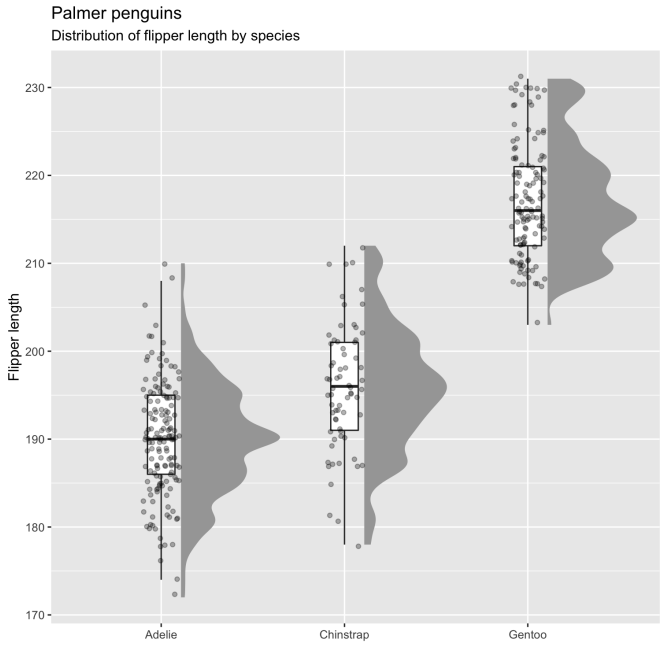
Step 4

```
P3 <- P2 +
  geom_point(
    size = 1.3,
    alpha = .3,
    position = position_jitter(
      seed = 1, width = .1
  ))

P3
```



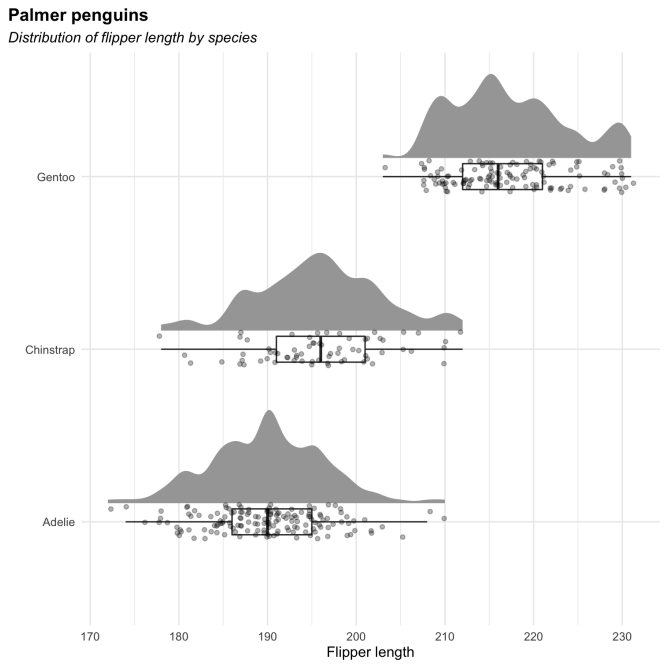
Step 5

```
P4 <- P3 +
  labs(
    title = "Palmer penguins",
    subtitle = "Distribution of flipper length by species",
    x = "",
    y = "Flipper length"
  )

P4
```



Step 6

```
P5 <- P4 +
  coord_cartesian(xlim = c(1.2, NA), clip = "off") +
  coord_flip() +
  theme_minimal() +
  theme(plot.title.position = "plot",
        plot.title = element_text(face = "bold"),
        plot.subtitle = element_text(face = "italic"))

P5
```



THE RAIN CLOUD PLOTS / Back to slide show...