

# Data visualization with ggplot2

Day 2 - Introduction to Data Analysis with R

Selina Baldauf

Freie Universität Berlin - Theoretical Ecology

March 10, 2025

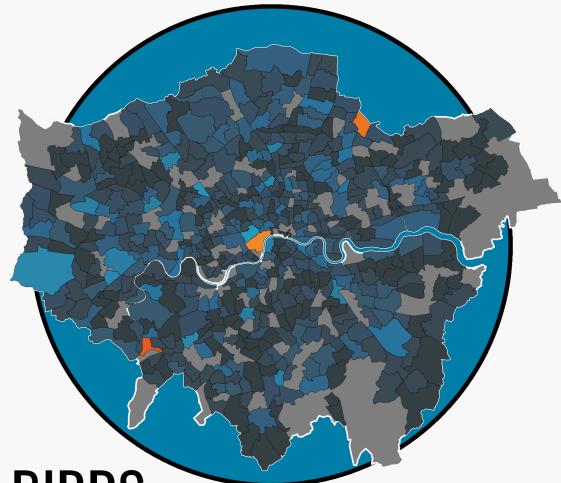
# A ggplot showcase

Example plots you can create with ggplot

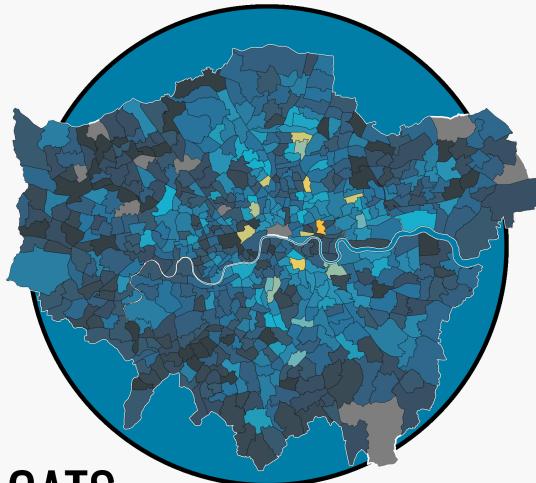
# A ggplot showcase

## Frequency of Rescues of Birds, Cats and Dogs in London from 2009-2021

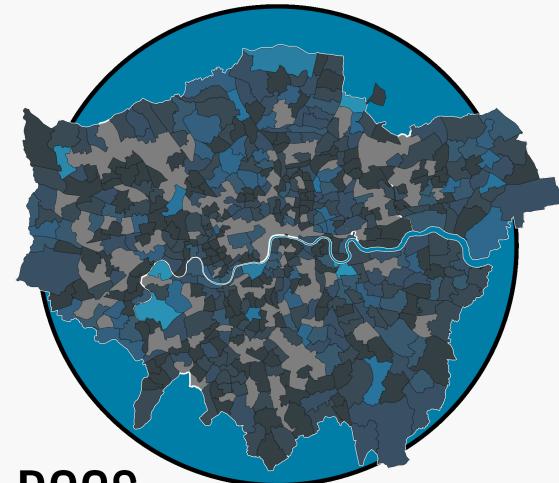
Illustrated below in three choropleth maps are rescues of birds, cats and dogs in London wards. Darker colors indicate lower rescue numbers while brighter colors indicate a greater number of rescues in that ward.



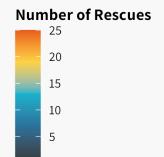
BIRDS



CATS



DOGS



Data: London.gov | Graphic: @jakekaupp

Visualization by [Jake Kaupp](#), code available on [Github](#)

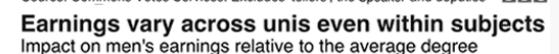
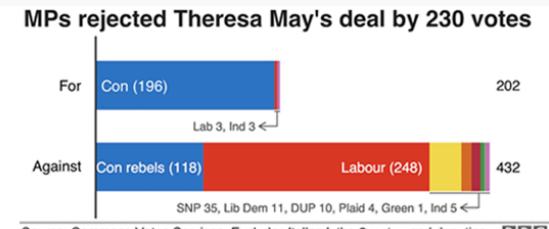
# A ggplot showcase



Source: AP, 19:01 ET



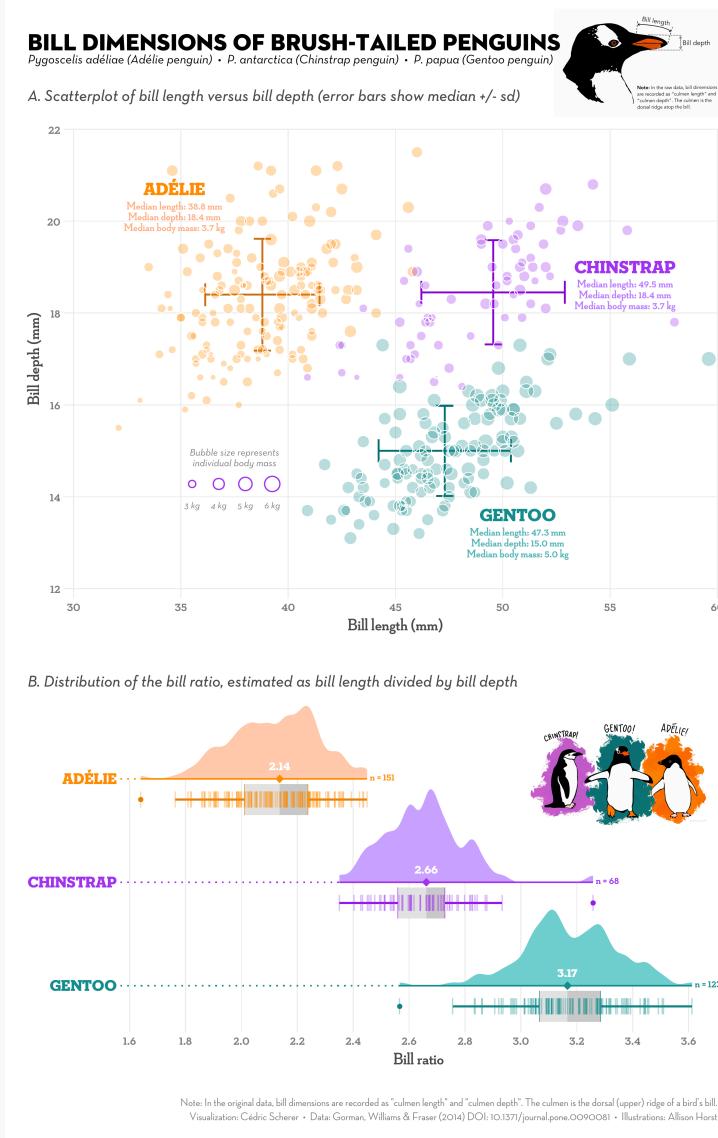
Source: Verisk Maplecroft. Circle size represents current population.



Source: Institute for Fiscal Studies

Visualizations produced by the BBC News data team

# A ggplot showcase



Visualization by Cédric Scherer, code available on [Github](#)  
Selina Baldauf // Data visualization with ggplot2

# Advantages of ggplot

- **Consistent** grammar/structure
- **Flexible** structure allows you to produce any type of plots
- Highly customizable appearance (themes)
- Many **extension packages** that provide additional plot types, themes, colors, animation, ...
  - See [here](#) for a list of ggplot extensions
- Active community that provides help and inspiration
- Perfect package for **exploratory data analysis** and **beautiful plots**

# The data

Data set `and_vertebrates` with measurements of a trout and 2 salamander species in different forest sections.

- `year`: observation year
- `section`: CC (clear cut forest) or OG (old growth forest)
- `unittype`: channel classification (C = Cascade, P = Pool, ...)
- `species`: Species measured
- `length_1_mm`: body length [mm]
- `weight_g`: body weight [g]



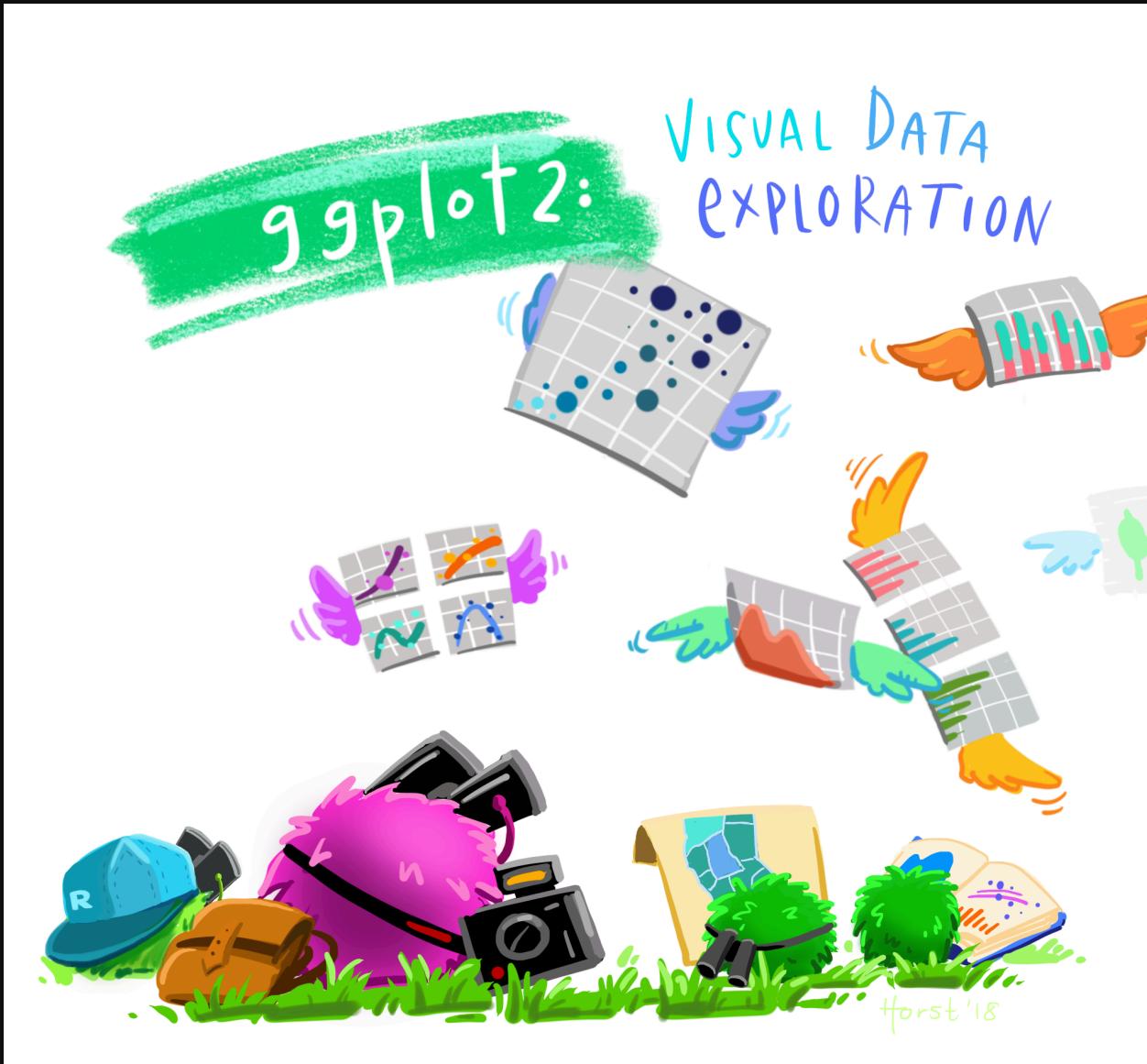
References: Kaylor, M.J. and D.R. Warren. (2017) and  
Gregory, S.V. and I. Arismendi. (2020) as provided in the  
Selina Baldauf // Data visualization with ggplot2

Coastal giant salamander (terrestrial form)  
Andrews Forest Program by Lina DiGregorio  
via CC-BY from  
<https://andrewsforest.oregonstate.edu>

# The data

Data set `and_vertebrates` with measurements of a trout and 2 salamander species in different forest sections.

```
library(lterdatasampler)
vertebrates <- and_vertebrates |>
  select(year, section, unittype, species, length_1_mm, weight_g) |>
  filter(species != "Cascade torrent salamander")
vertebrates
#> # A tibble: 32,191 × 6
#>   year section unittype species      length_1_mm weight_g
#>   <dbl> <chr>   <chr>    <chr>           <dbl>     <dbl>
#> 1 1987 CC       R        Cutthroat trout      58      1.75
#> 2 1987 CC       R        Cutthroat trout      61      1.95
#> 3 1987 CC       R        Cutthroat trout      89      5.6
#> 4 1987 CC       R        Cutthroat trout      58      2.15
#> 5 1987 CC       R        Cutthroat trout      93      6.9
#> 6 1987 CC       R        Cutthroat trout      86      5.9
#> 7 1987 CC       R        Cutthroat trout     107     10.5
#> 8 1987 CC       R        Cutthroat trout     131     20.6
#> 9 1987 CC       R        Cutthroat trout     103     9.55
#> 10 1987 CC      R        Cutthroat trout     117     13
#> # i 32,181 more rows
```



Artwork by Allison Horst

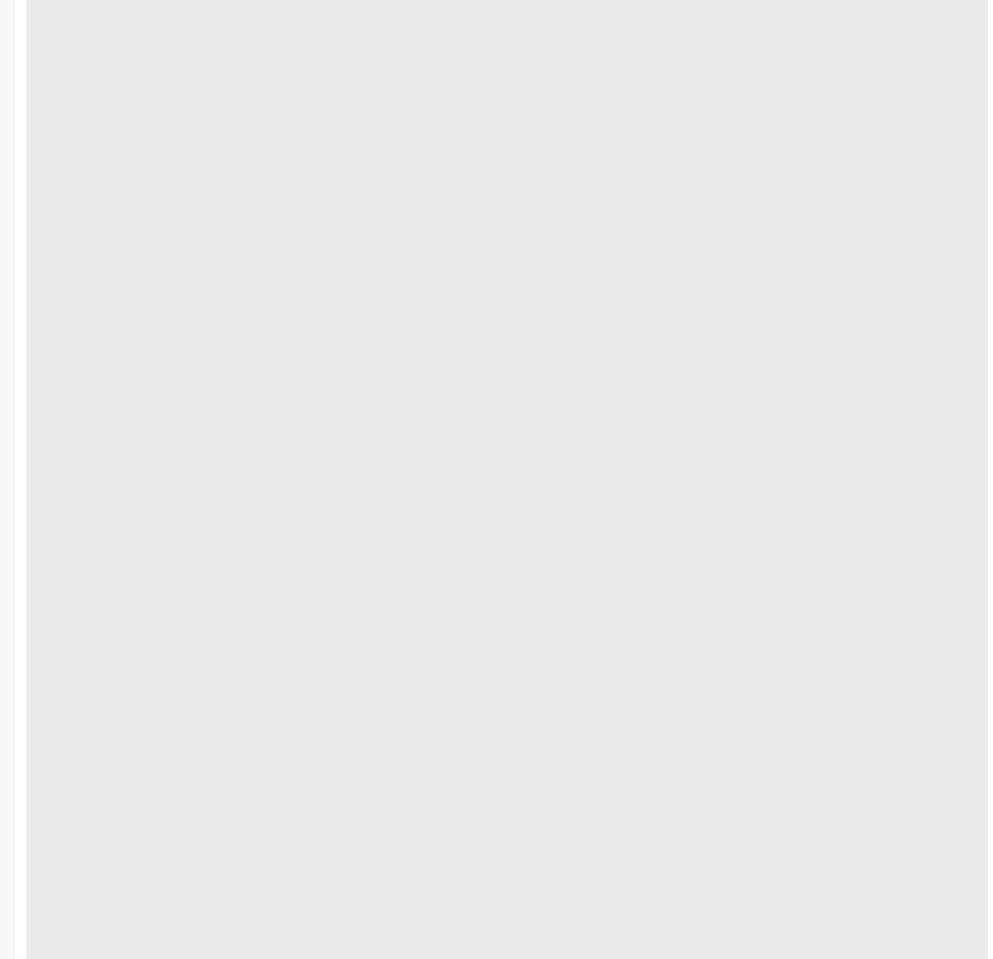
# ggplot(data)

The `ggplot()` function initializes a ggplot object. Every ggplot needs this function.

```
library(ggplot2)
# or library(tidyverse)

ggplot(data = vertebrates)
```

- Empty plot because we did not specify the mapping of data variables

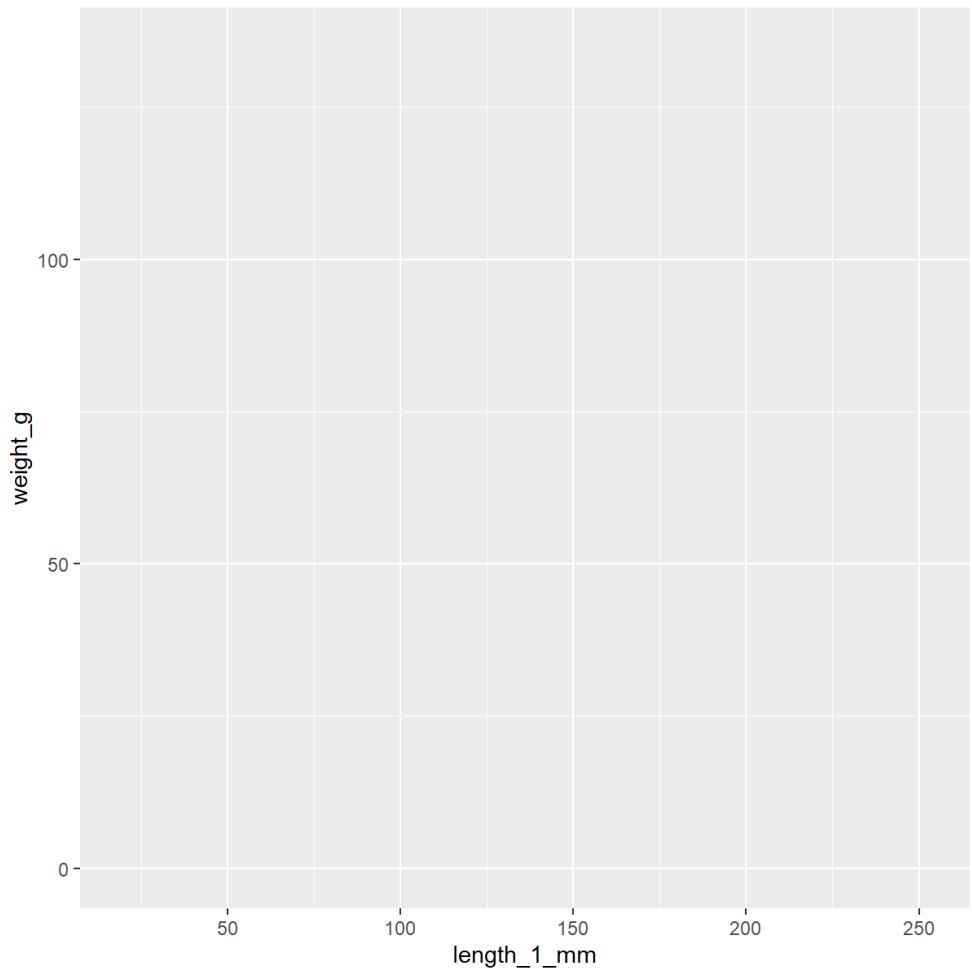


# aes(x, y)

The **aesthetics** define how data variables are mapped plot properties.

```
ggplot(  
  data = vertebrates,  
  mapping = aes(  
    x = length_1_mm,  
    y = weight_g  
  )  
)
```

- Map variable `length_1_mm` to x-axis and `weight_g` to y-axis
- Default scales are automatically adapted to range of data



# aes(x, y)

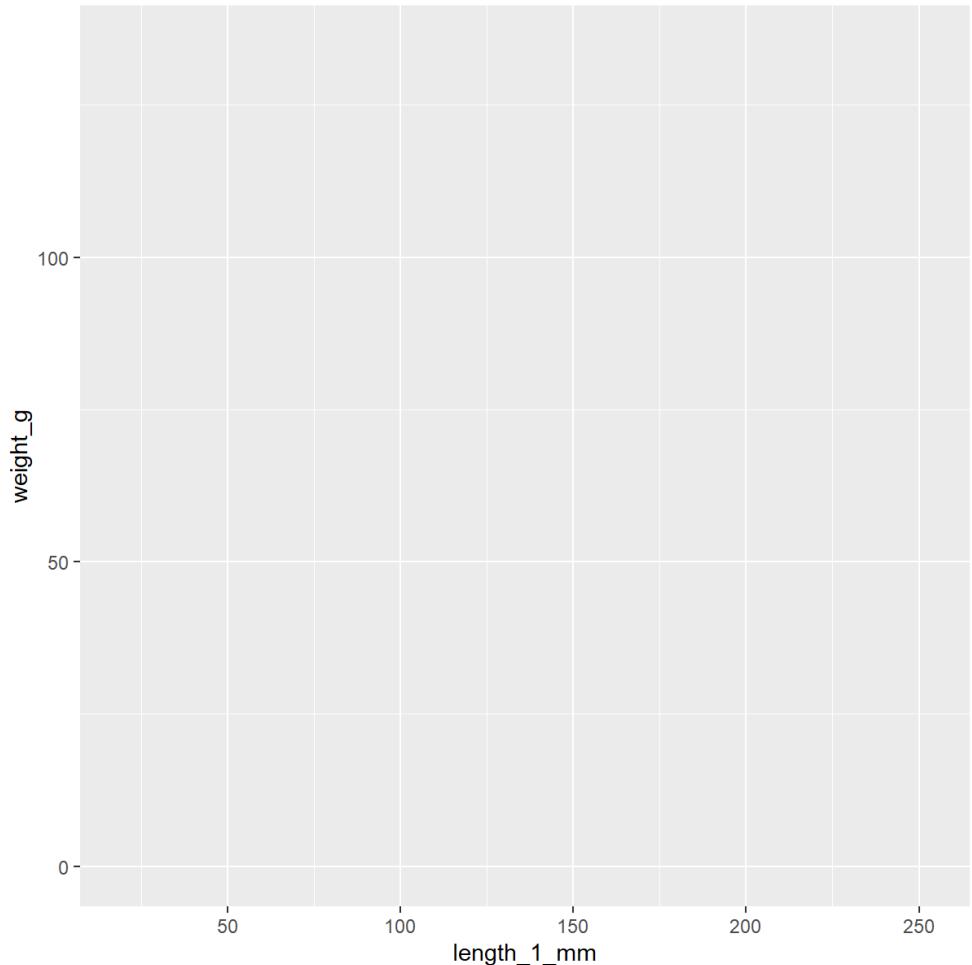
The **aesthetics** define how data variables are mapped plot properties.

```
ggplot(  
  data = vertebrates,  
  mapping = aes(  
    x = length_1_mm,  
    y = weight_g  
  )  
)
```

This is the same but shorter:

```
ggplot(  
  vertebrates,  
  aes(  
    x = length_1_mm,  
    y = weight_g  
)
```

Remember argument matching by position?



# geom\_\*

geoms define how data points are represented. There are many different geoms to chose from

The image is a detailed ggplot2 geom cheat sheet. It is organized into several sections:

- a + geom\_blank()**: (Useful for expanding limits)
- b + geom\_curve(aes(yend = lat + 1, xend = long + 1), curvature = 1) - x, yend, alpha, angle, color, curvature, linetype, size**
- a + geom\_path(lineend = "butt", linejoin = "round", linemitre = 1)**: x, y, alpha, color, group, linetype, size
- a + geom\_polygon(aes(group = group))**: x, y, alpha, color, fill, group, linetype, size
- b + geom\_rect(aes(xmin = long, ymin = lat, xmax = long + 1, ymax = lat + 1)) - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size**
- a + geom\_ribbon(aes(ymin = unemploy - 900, ymax = unemploy + 900)) - x, ymax, ymin, alpha, color, fill, group, linetype, size**
- LINE SEGMENTS**  
common aesthetics: x, y, alpha, color, linetype, size
  - b + geom\_abline(aes(intercept = 0, slope = 1))**
  - b + geom\_hline(aes(intercept = lat))**
  - b + geom\_vline(aes(xintercept = long))**
  - b + geom\_segment(aes(yend = lat + 1, xend = long + 1))**
  - b + geom\_spoke(aes(angle = 1:155, radius = 1))**
- ONE VARIABLE continuous**  
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)
  - c + geom\_area(stat = "bin")**: x, y, alpha, color, fill, linetype, size
  - c + geom\_density(kernel = "gaussian")**: x, y, alpha, color, fill, group, linetype, size, weight
  - c + geom\_dotplot()**: x, y, alpha, color, fill
  - c + geom\_freqpoly()**: x, y, alpha, color, group, linetype, size
  - c + geom\_histogram(binwidth = 5)**: x, y, alpha, color, fill, linetype, size, weight
- e + geom\_label(aes(label = cty), nudge\_x = 1, nudge\_y = 1, check\_overlap = TRUE)**: x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust
- e + geom\_jitter(height = 2, width = 2)**: x, y, alpha, color, fill, shape, size
- e + geom\_point()**: x, y, alpha, color, fill, shape, size, stroke
- e + geom\_quantile()**: x, y, alpha, color, group, linetype, size, weight
- e + geom\_rug(sides = "bl")**: x, y, alpha, color, linetype, size
- e + geom\_smooth(method = lm)**: x, y, alpha, color, fill, group, linetype, size, weight
- e + geom\_text(aes(label = cty), nudge\_x = 1, nudge\_y = 1, check\_overlap = TRUE)**: x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust
- f + geom\_col()**: x, y, alpha, color, fill, group, linetype, size
- f + geom\_boxplot()**: x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight
- f + geom\_dotplot(binaxis = "Y", stackdir = "center")**: x, y, alpha, color, fill, group
- f + geom\_violin(scale = "area")**: x, y, alpha, color, fill, group, linetype, size, weight
- discrete x , continuous**  
f <- ggplot(mpg, aes(class, hwy))
  - f + geom\_col()**: x, y, alpha, color, fill, group, linetype, size
- discrete x , discrete y**  
g <- ggplot(diamonds, aes(cut, color))
  - g + geom\_count()**: x, y, alpha, color, fill, shape, size, stroke
- continuous function**  
i <- ggplot(economics, aes(date, unemploy))
  - i + geom\_area()**: x, y, alpha, color, fill, linetype, size
  - i + geom\_line()**: x, y, alpha, color, group, linetype, size
  - i + geom\_step(direction = "hv")**: x, y, alpha, color, group, linetype, size
- visualizing error**  
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)  
j <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))
  - j + geom\_crossbar(fatten = 2)**: x, y, ymax, ymin, alpha, color, fill, group, linetype, size
  - j + geom\_errorbar()**: x, ymax, ymin, alpha, color, group, linetype, size, width (also `geom_errorbarh()`)
  - j + geom\_linerange()**: x, ymin, ymax, alpha, color, group, linetype, size
  - j + geom\_pointrange()**: x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size
- maps**  
data <- data.frame(murder = USArrests\$Murder, state = tolower(rownames(USArrests)))  
map <- map\_data("state")  
k <- ggplot(data, aes(fill = murder))
  - k + geom\_map(aes(map\_id = state), map = map) + expand\_limits(x = map\$long, y = map\$lat)**: map\_id, alpha, color, fill, linetype, size

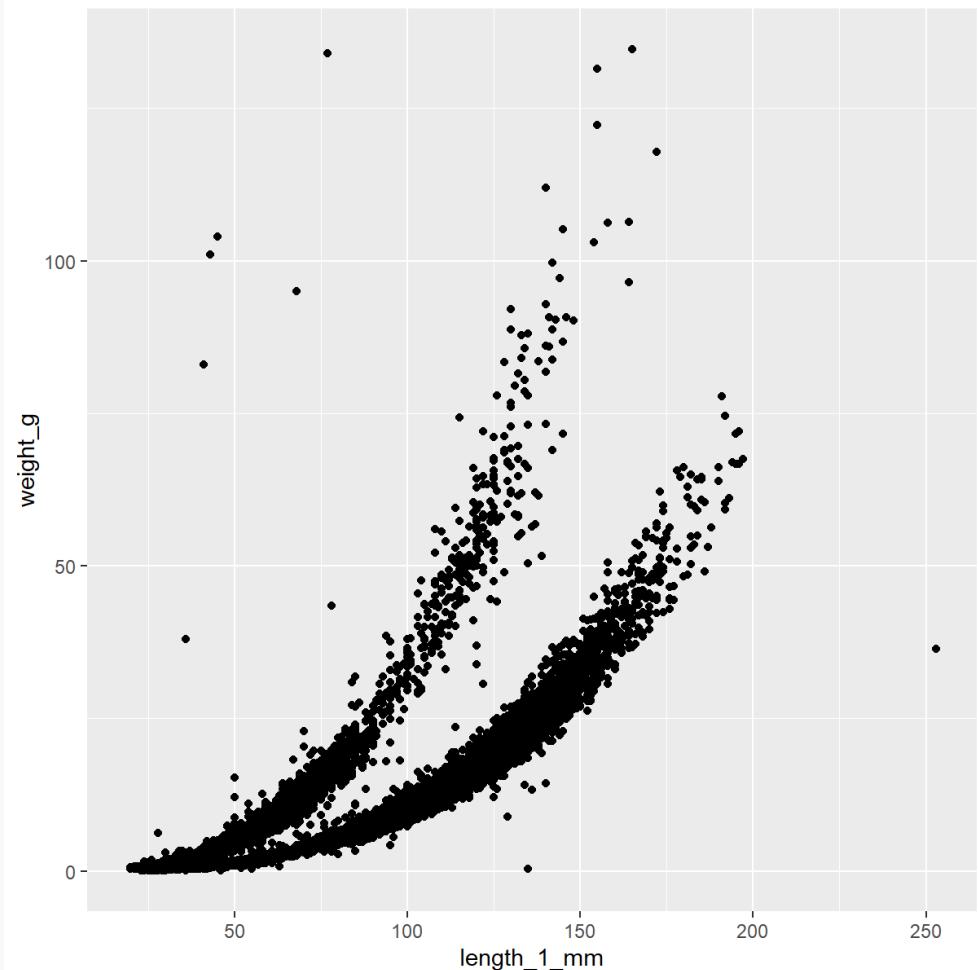
from [ggplot cheatsheet](#)

# geom\_point

```
ggplot(  
  data = vertebrates,  
  aes(  
    x = length_1_mm,  
    y = weight_g  
  )  
) +  
  geom_point()
```

- New plot layers added with `+`
- Warning that points could not be plotted due to missing values
- `data` and `aes` defined in `ggplot` call are inherited to all plot layers

```
#> Warning: Removed 13270 rows containing missing  
values or values outside the scale range  
#> (`geom_point()`).
```



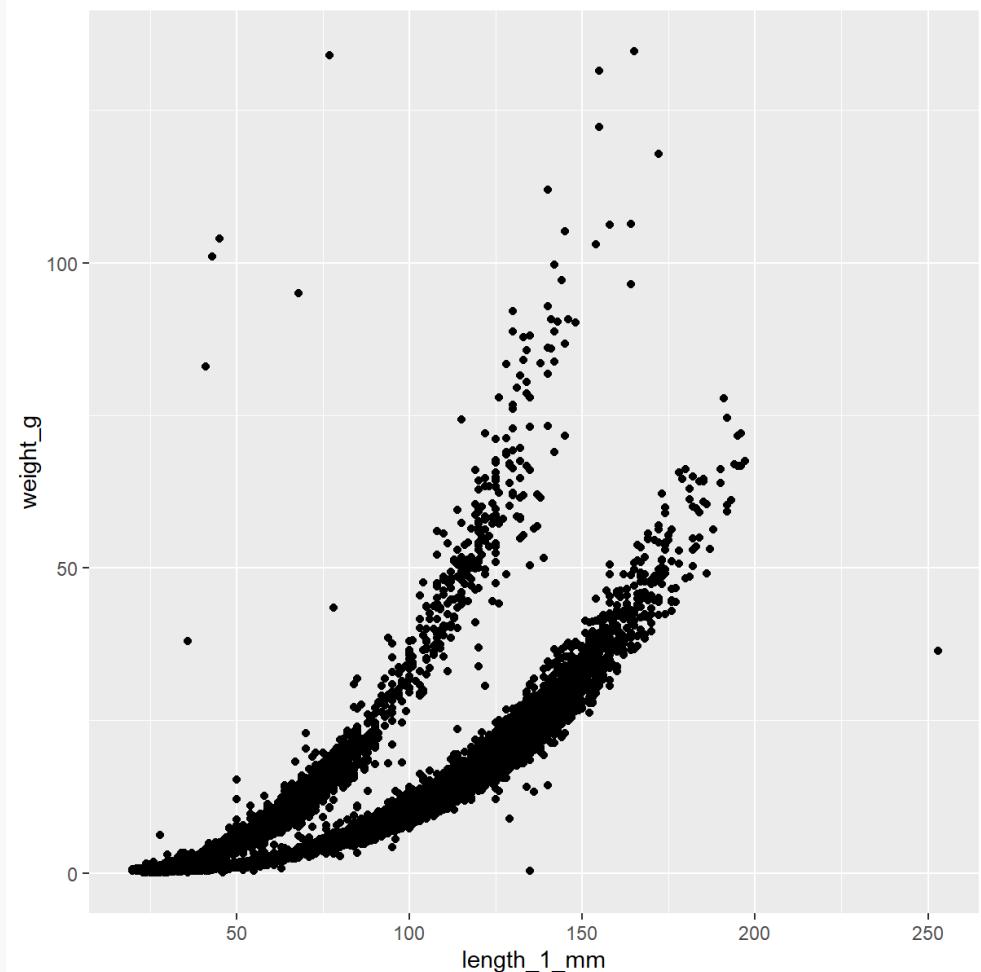
# geom\_point

```
ggplot() +  
  geom_point(  
    data = vertebrates,  
    aes(  
      x = length_1_mm,  
      y = weight_g  
    )  
  )
```

- **data** and **aes** can also be local to a layer:

Here, it does not make a difference in the result.

```
#> Warning: Removed 13270 rows containing missing  
values or values outside the scale range  
#> (`geom_point()`).
```

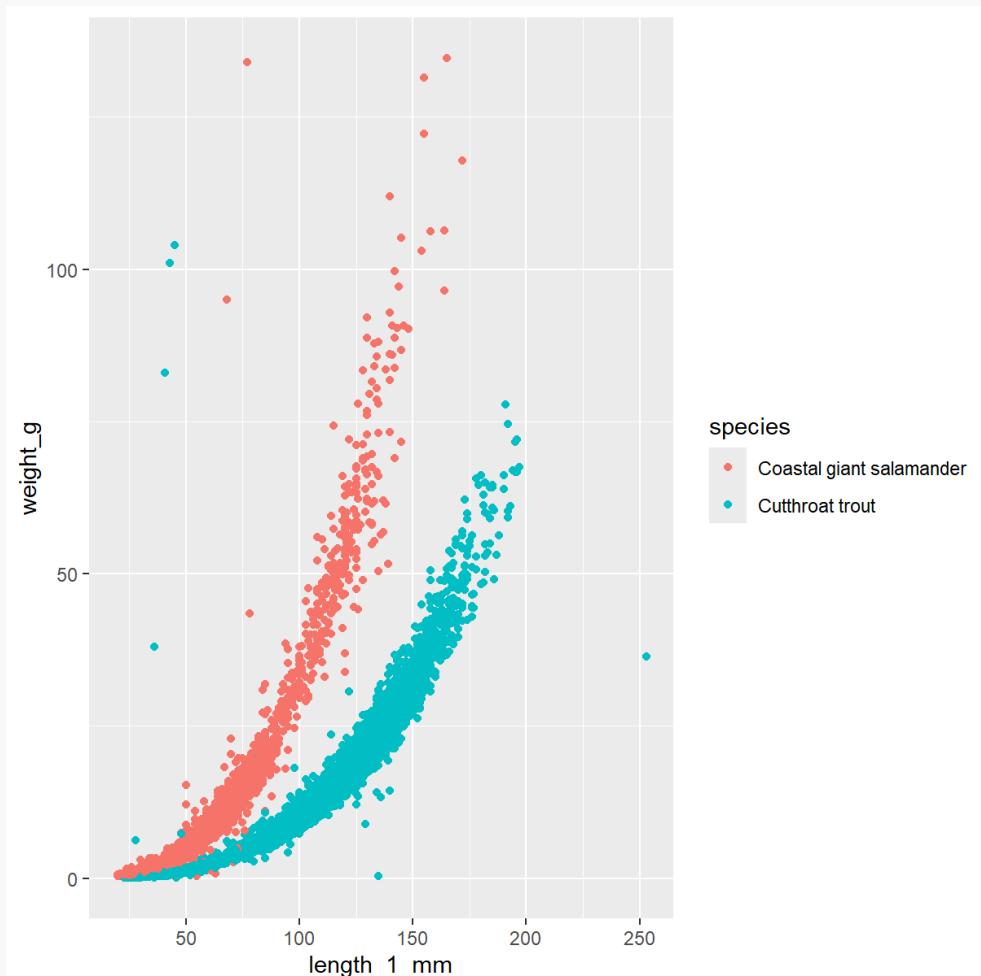


# aes(color): mapping color to a variable

Looks like there are two groups of data: **Map color of points to a variable** by adding it to aesthetics:

```
ggplot(  
  data = vertebrates,  
  aes(  
    x = length_1_mm,  
    y = weight_g,  
    color = species  
  )  
) +  
  geom_point()
```

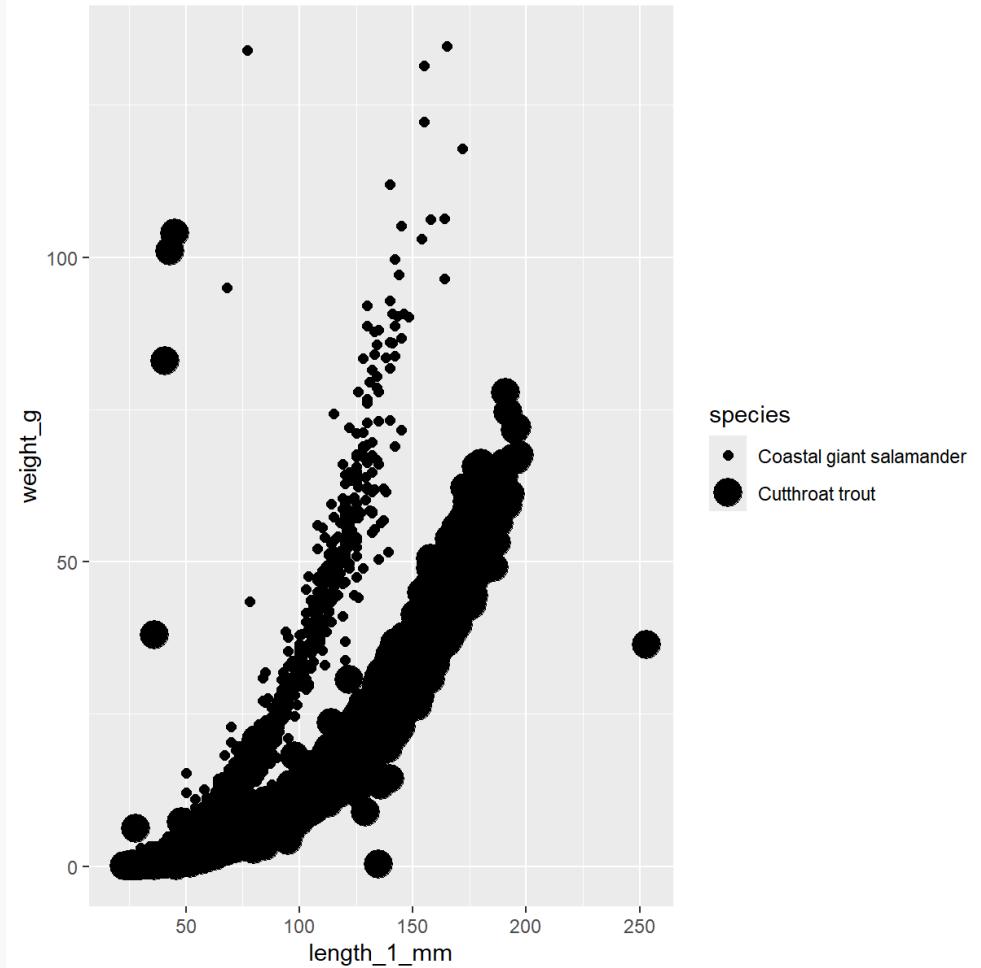
- Map the **species** variable to the color aesthetic of the plot



# aes(size): mapping size to a variable

We can do the same with size:

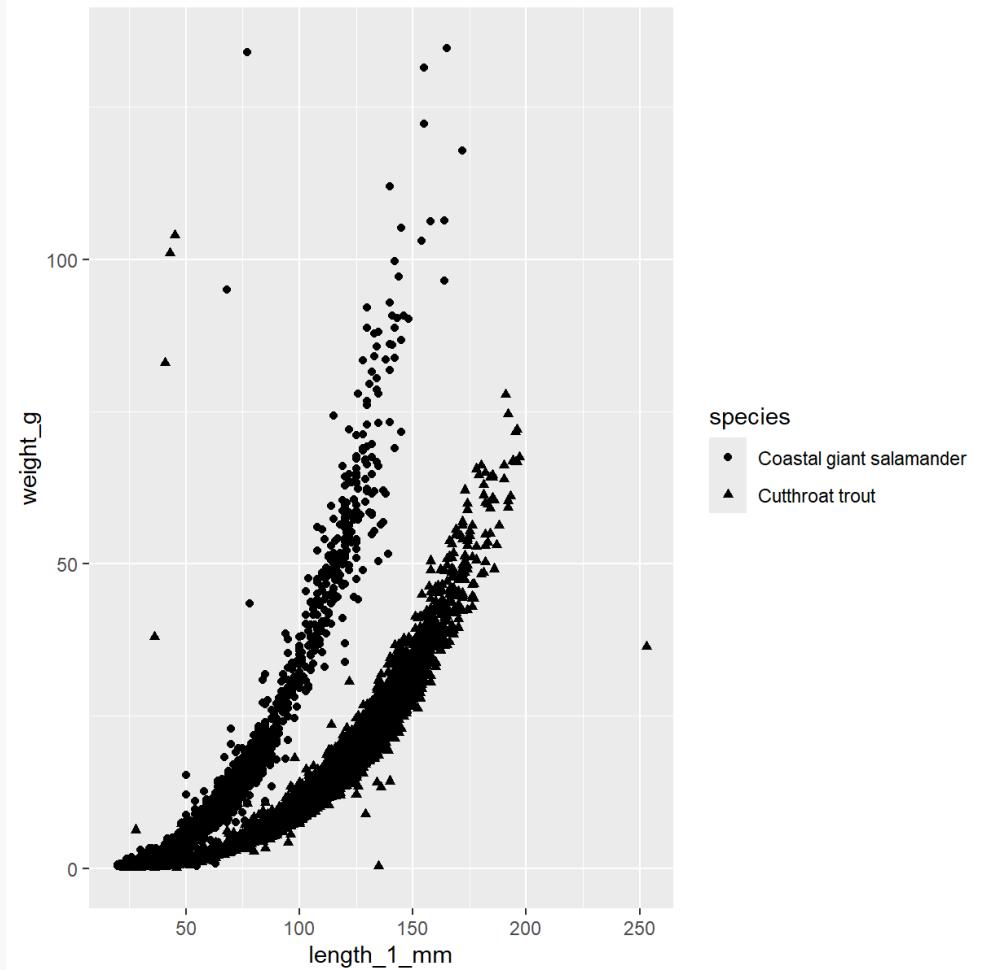
```
ggplot(  
  data = vertebrates,  
  aes(  
    x = length_1_mm,  
    y = weight_g,  
    size = species  
  )  
) +  
  geom_point()
```



# aes(shape): mapping shape to a variable

We can do the same with shape:

```
ggplot(  
  data = vertebrates,  
  aes(  
    x = length_1_mm,  
    y = weight_g,  
    shape = species  
  )  
) +  
  geom_point()
```

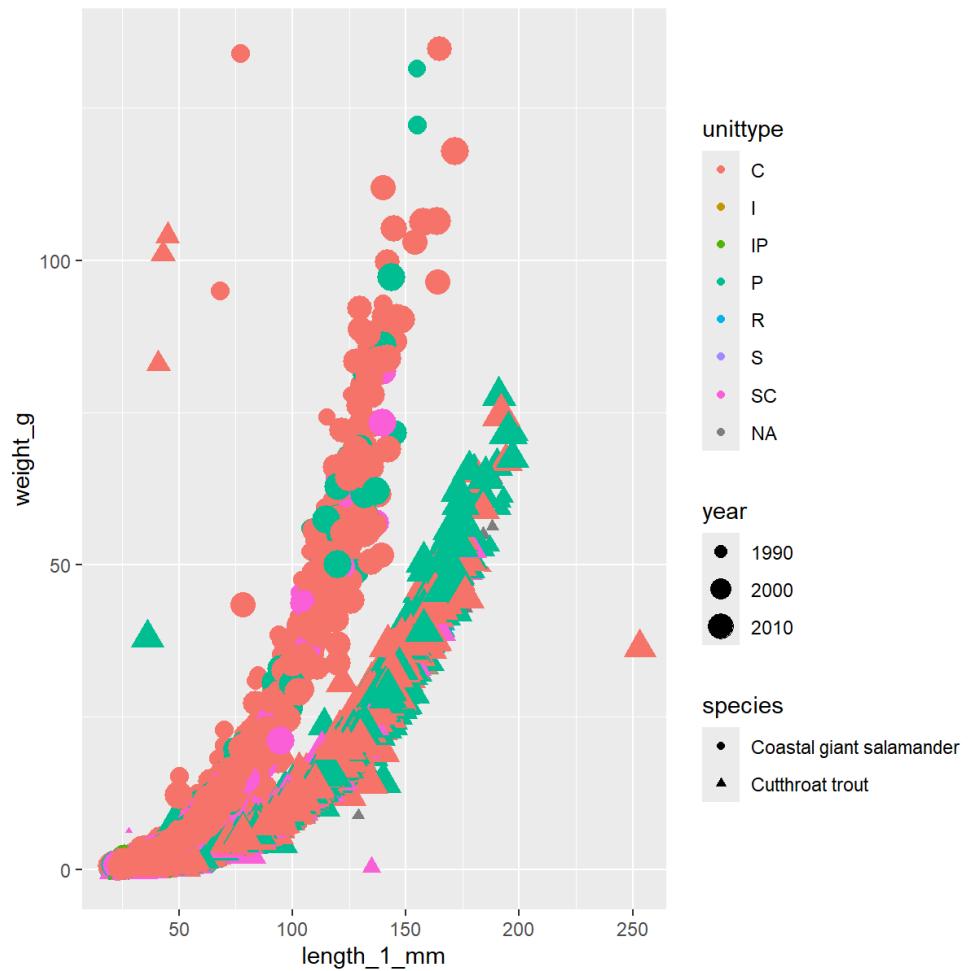


# Combine color, size and shape

We can also combine these aesthetics and map different variables

```
ggplot(  
  data = vertebrates,  
  aes(  
    x = length_1_mm,  
    y = weight_g,  
    color = unitype,  
    shape = species,  
    size = year  
  )  
) +  
  geom_point()
```

- This is a bit too much for this plot, but sometimes can be useful

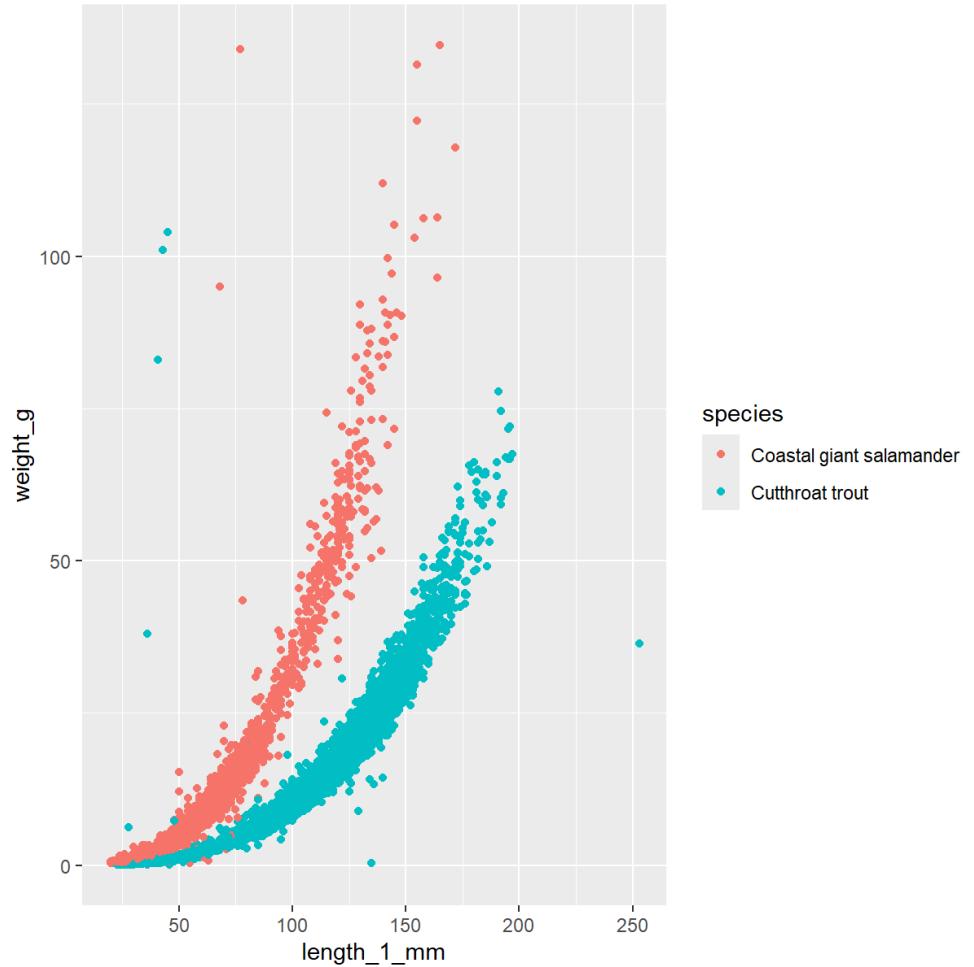


# Changing the scales of the aesthetics

The scales onto which the aesthetic elements are mapped can be changed.

```
ggplot(  
  data = vertebrates,  
  aes(  
    x = length_1_mm,  
    y = weight_g,  
    color = species  
  )  
) +  
  geom_point()
```

- Exponential relationship?
- How does it look like on the log scale?



# scale\_x\_log10

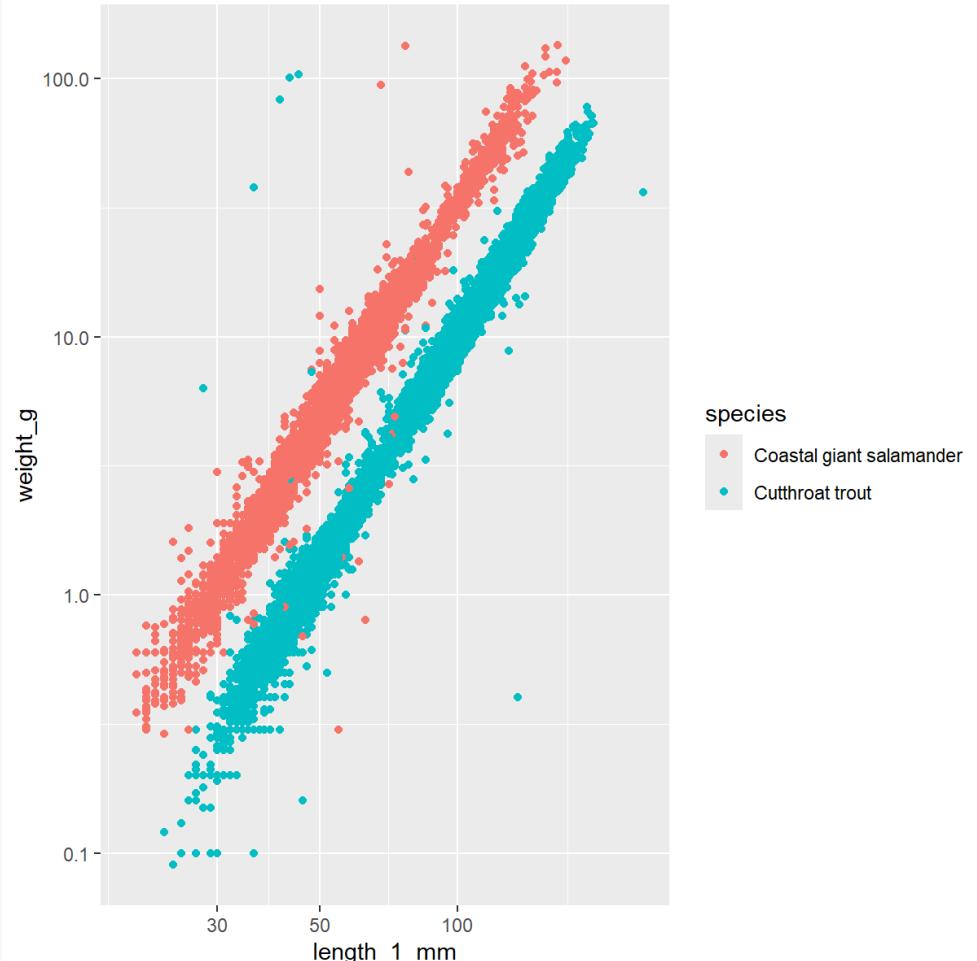
The scales onto which the aesthetic elements are mapped can be changed.

```
ggplot(  
  data = vertebrates,  
  aes(  
    x = length_1_mm,  
    y = weight_g,  
    color = species  
  )  
) +  
  geom_point() +  
  scale_x_log10() +  
  scale_y_log10()
```

- Scales can be changed for all elements of `aes`:

`scale_aes-name_scale-type`

In this example we scale the `x` and the `y` aesthetic to `log10`.

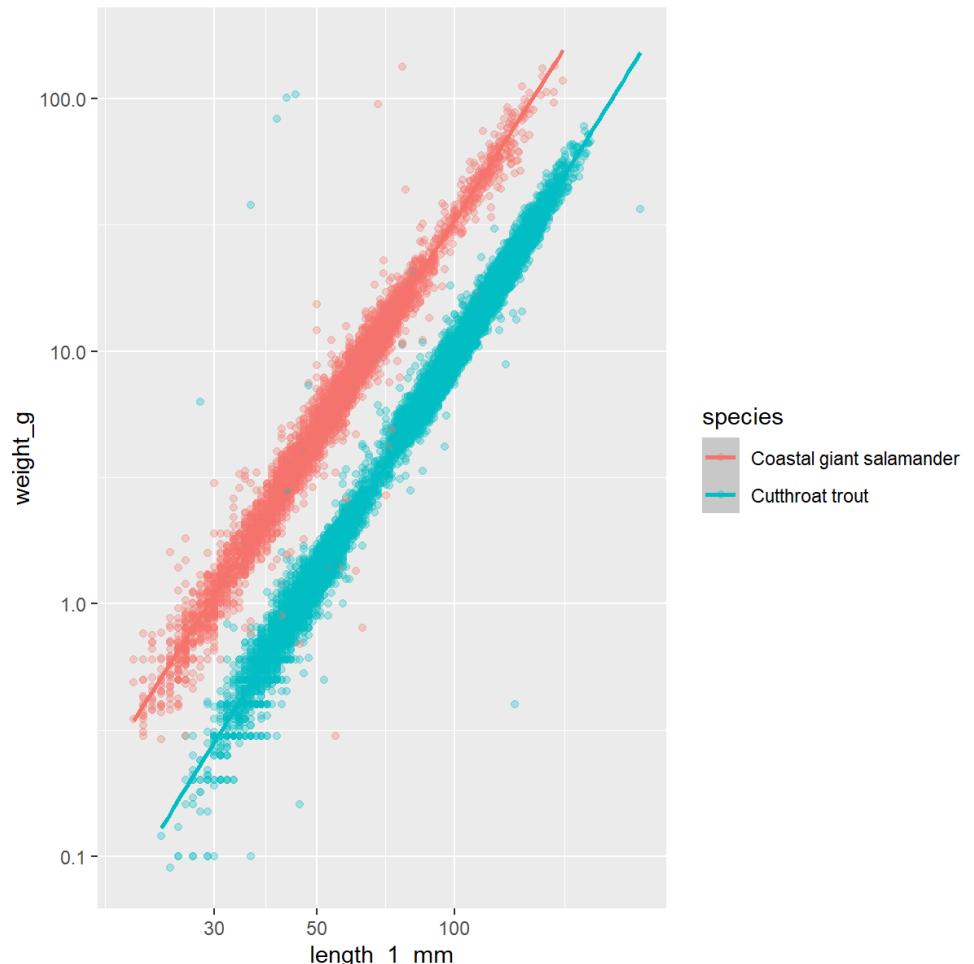


# geom\_smooth

Add a smoothing line that helps see patterns in the data

```
ggplot(  
  data = vertebrates,  
  aes(  
    x = length_1_mm,  
    y = weight_g,  
    color = species  
  )  
) +  
  geom_point(alpha = 0.3) +  
  geom_smooth(method = "lm") +  
  scale_x_log10() +  
  scale_y_log10()
```

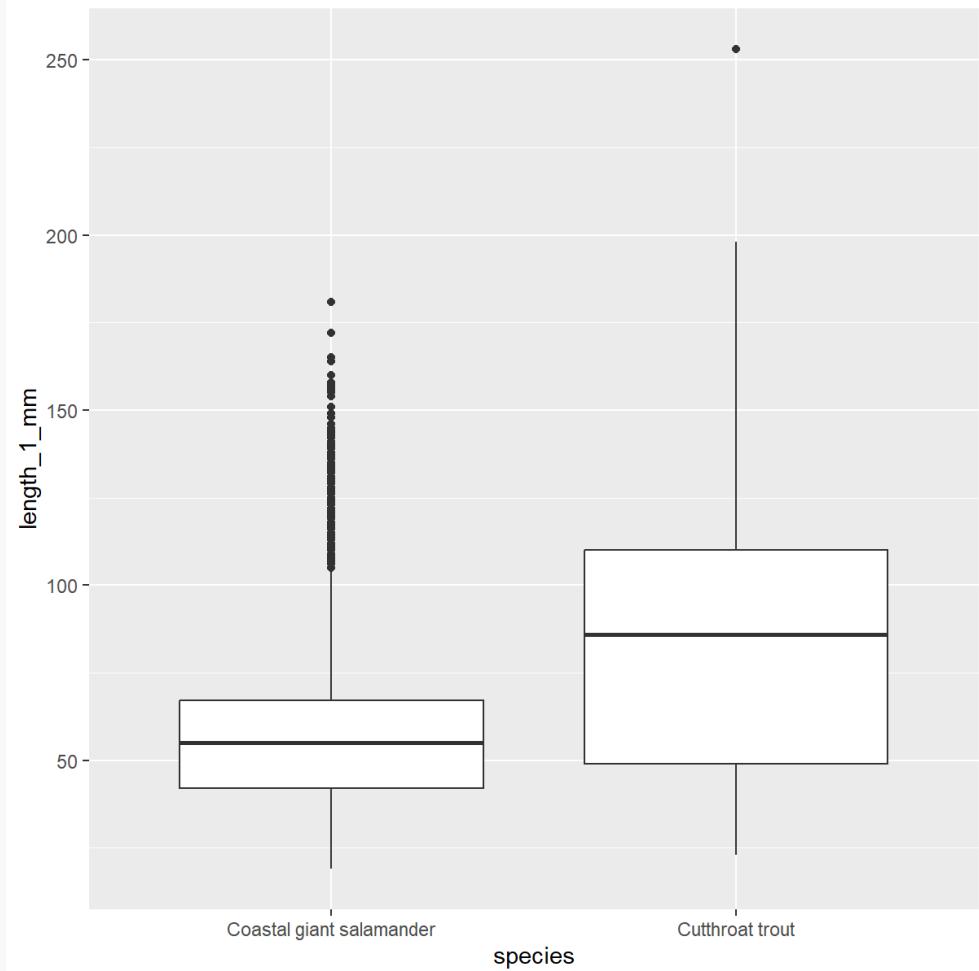
- With **method = "lm"**, a linear regression line is added
- All geoms done separately for species because color is defined globally
- Alpha makes points transparent (0-1)



# geom\_boxplot

Compare groups using a boxplot

```
ggplot(  
  vertebrates,  
  aes(  
    x = species,  
    y = length_1_mm  
  ))  
  +  
  geom_boxplot()
```

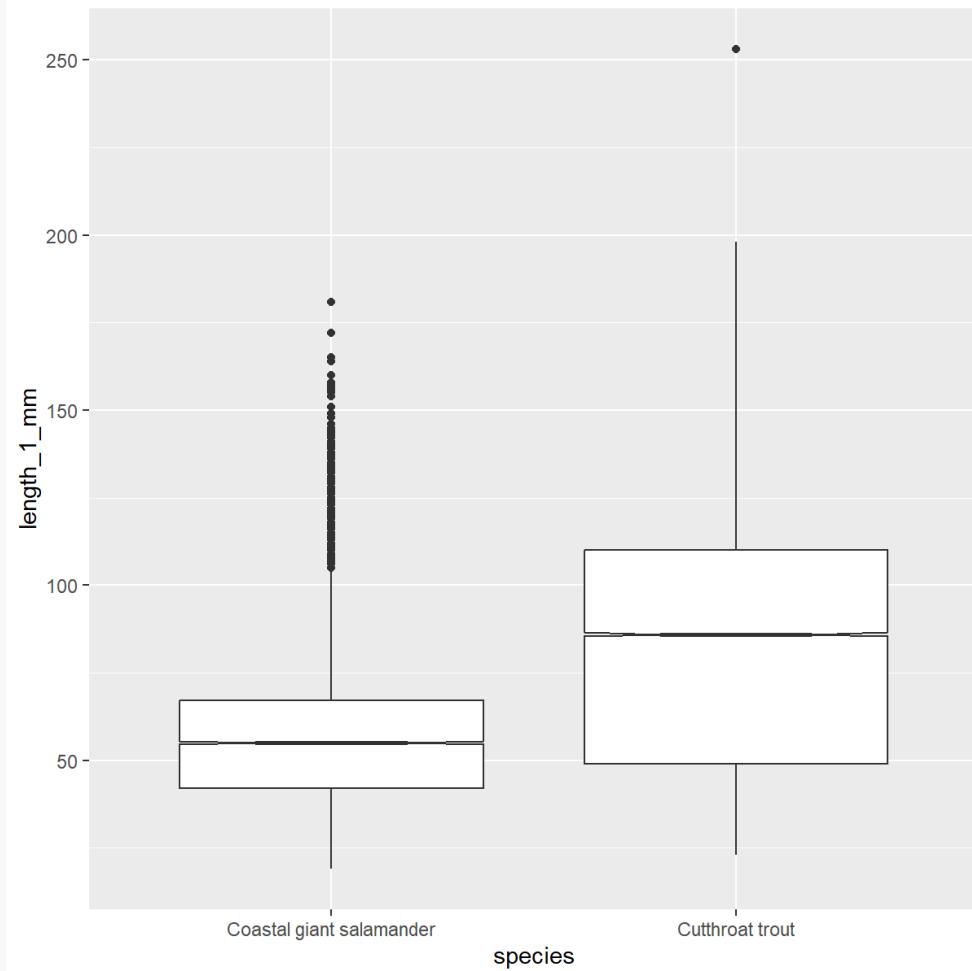


# geom\_boxplot

Compare groups using a boxplot

```
ggplot(  
  vertebrates,  
  aes(  
    x = species,  
    y = length_1_mm  
)  
) +  
  geom_boxplot(notch = TRUE)
```

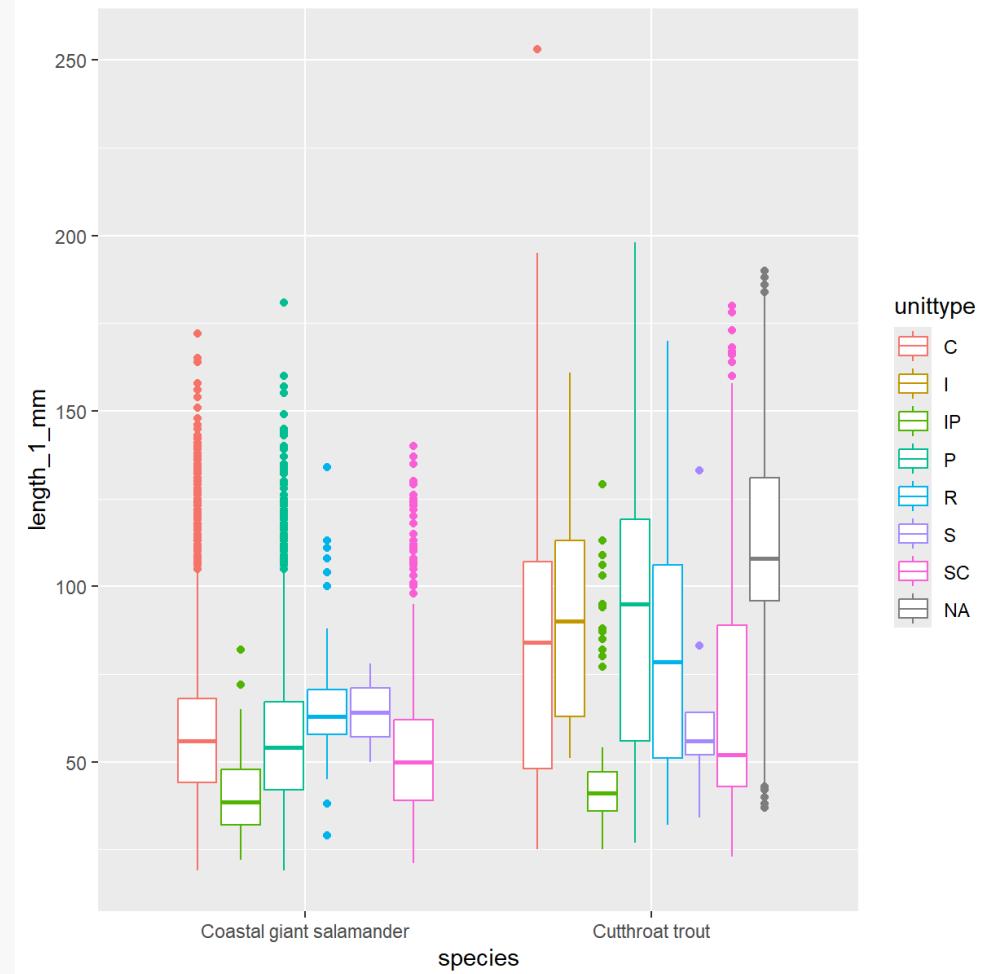
- If notches don't overlap, the medians of the groups are likely different



# geom\_boxplot

Map the `unittype` to the `color` aesthetic of the boxplot

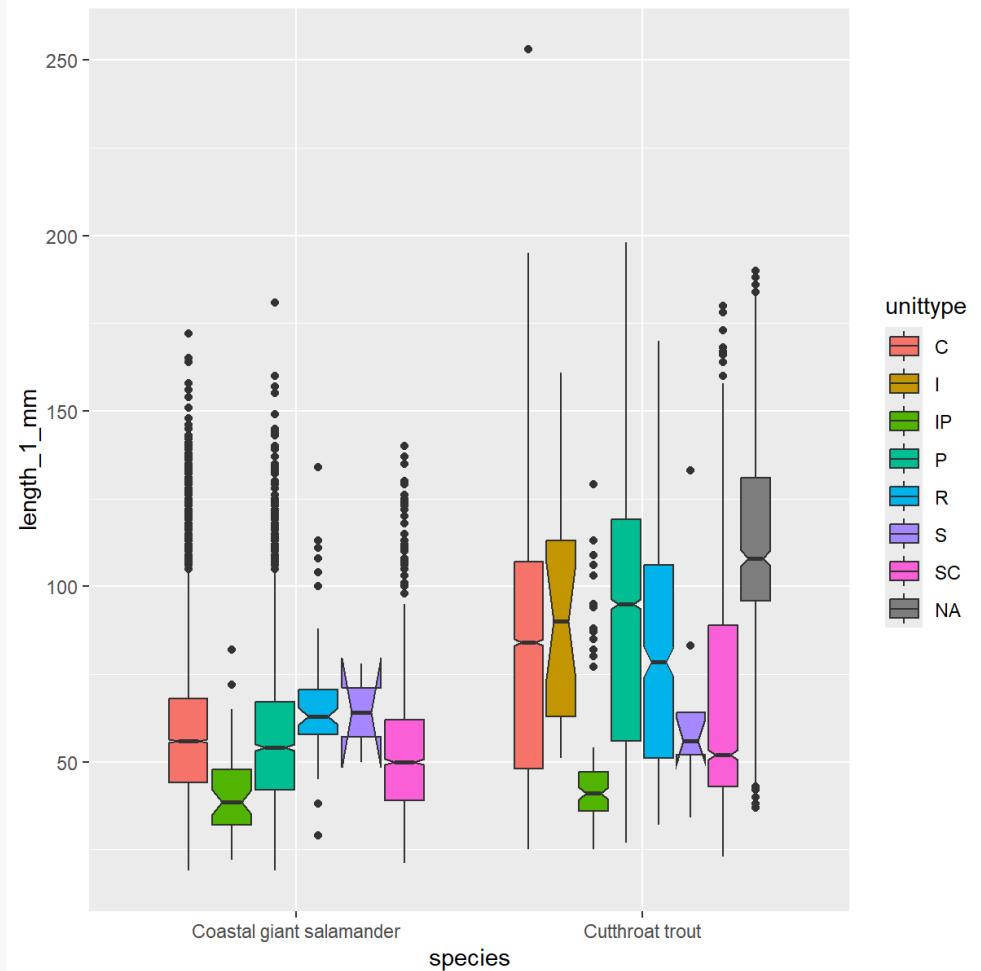
```
ggplot(  
  vertebrates,  
  aes(  
    x = species,  
    y = length_1_mm,  
    color = unittype  
  ))  
  +  
  geom_boxplot()
```



# geom\_boxplot

Map the `unittype` to the `fill` aesthetic of the box

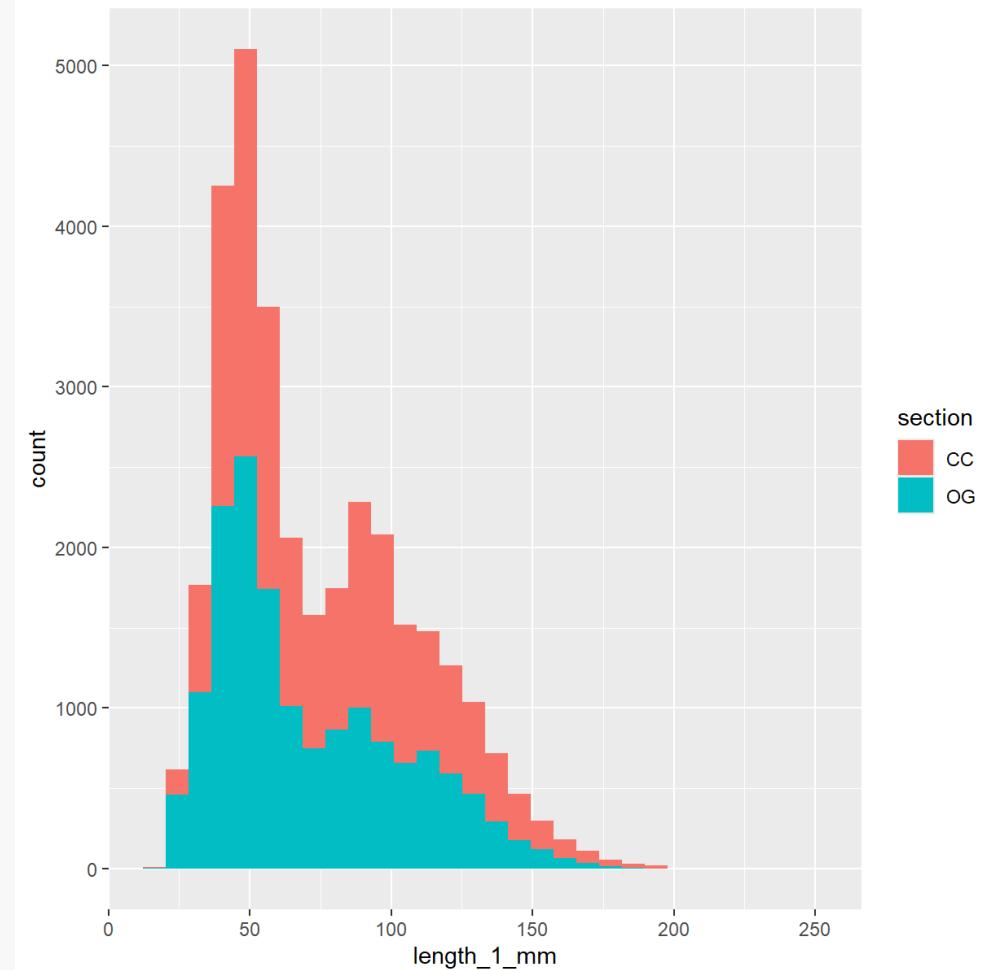
```
ggplot(  
  vertebrates,  
  aes(  
    x = species,  
    y = length_1_mm,  
    fill = unittype  
  )  
) +  
  geom_boxplot(notch = TRUE)
```



# geom\_histogram

```
ggplot(  
  vertebrates,  
  aes(  
    x = length_1_mm,  
    fill = section  
  )  
) +  
  geom_histogram()
```

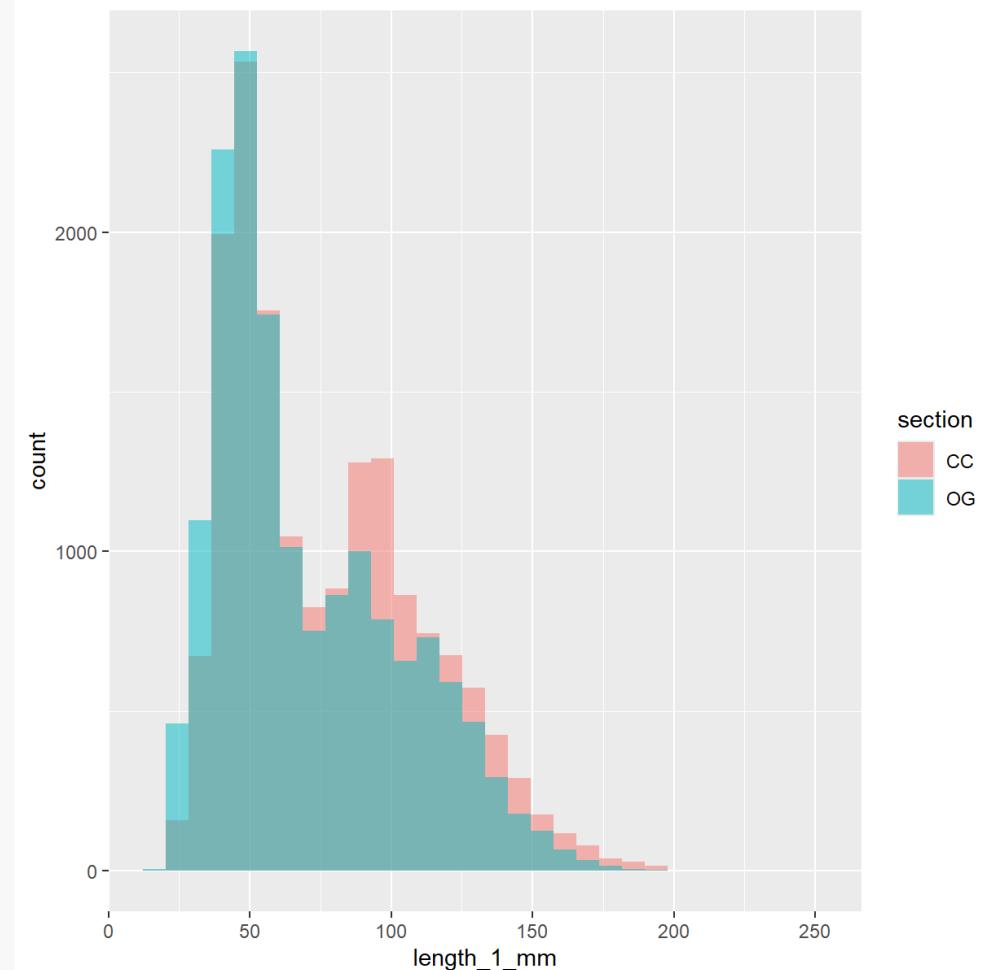
- Careful: By default the histogram is stacked for the different groups!



# geom\_histogram

```
ggplot(  
  vertebrates,  
  aes(  
    x = length_1_mm,  
    fill = section  
  )  
) +  
  geom_histogram(  
    position = "identity",  
    alpha = 0.5  
  )
```

- Change the position of the histogram to `"identity"`, if you don't want it stacked
- `alpha` makes sure that you see overlapping areas

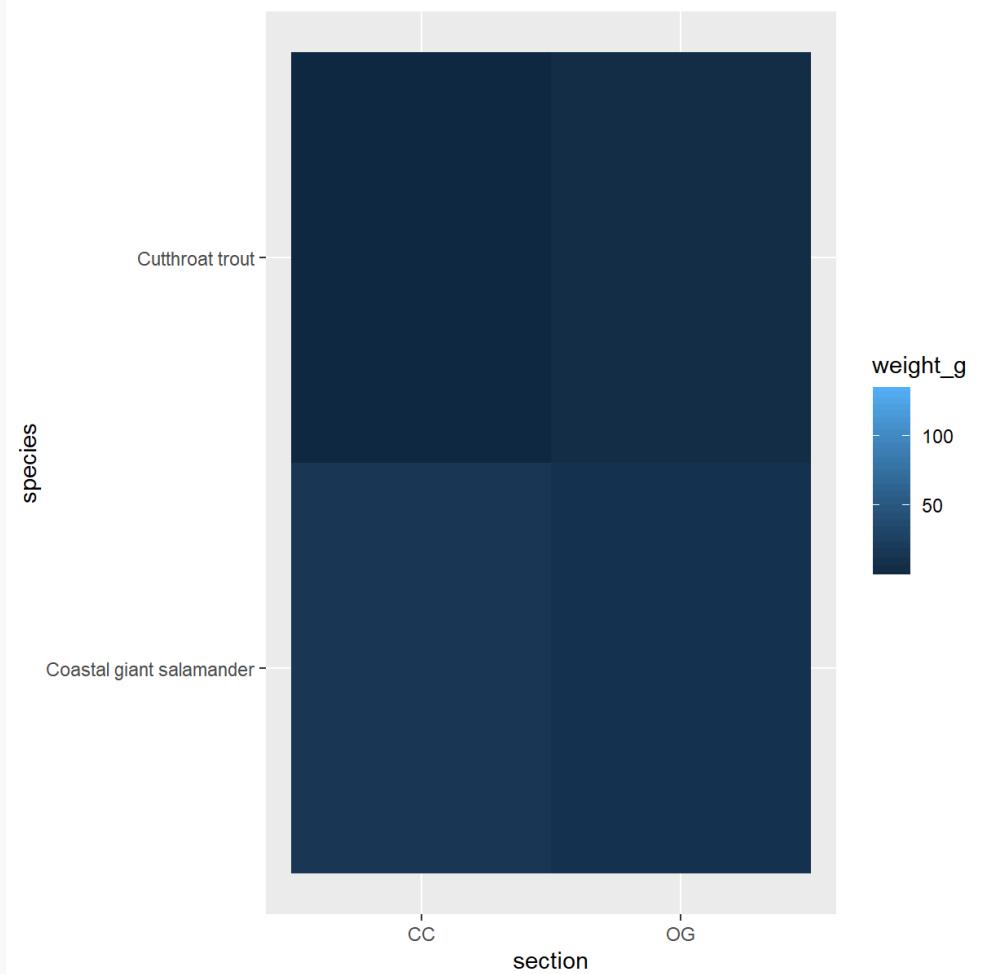


# geom\_tile

You can create a simple heatmap with `geom_tile`

```
ggplot(  
  vertebrates,  
  aes(  
    x = section,  
    y = species,  
    fill = weight_g  
  )  
) +  
  geom_tile()
```

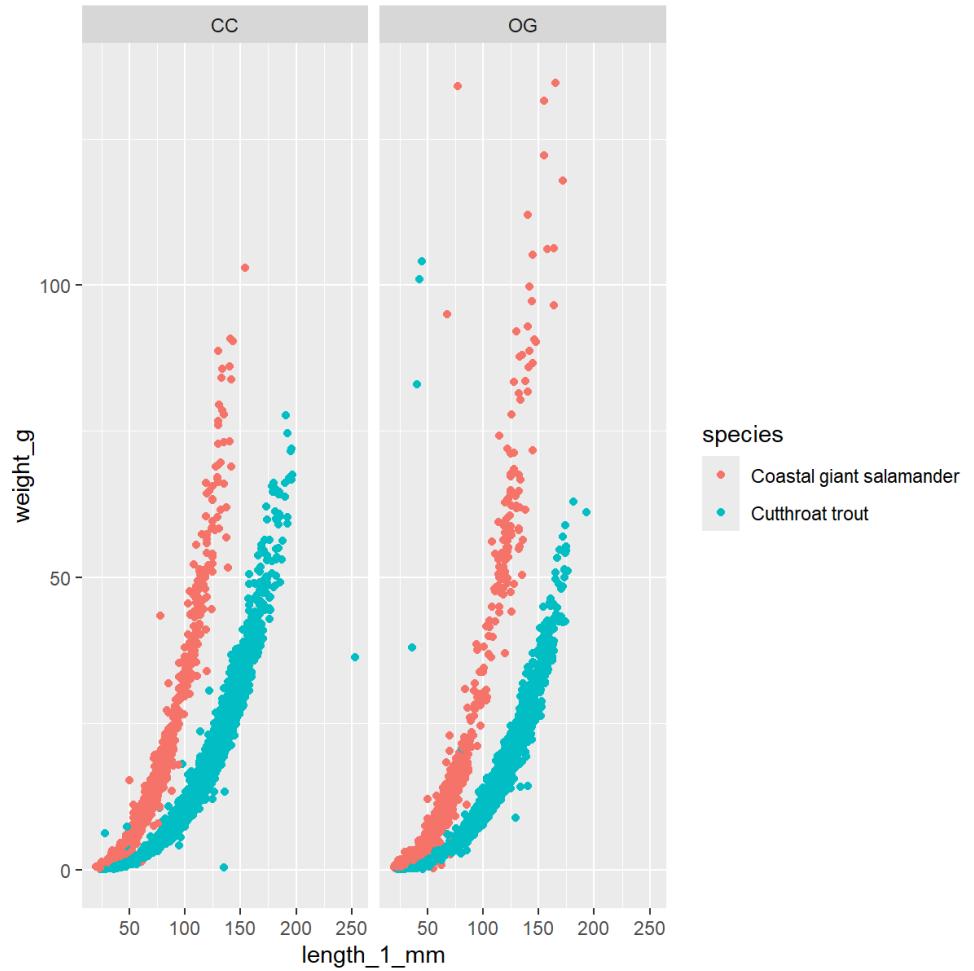
- Here we would have to choose a different color scheme to see differences



# Small multiples with `facet_wrap`

Split your plots along one variable with `facet_wrap`

```
ggplot(  
  data = vertebrates,  
  aes(  
    x = length_1_mm,  
    y = weight_g,  
    color = species  
  )  
) +  
  geom_point() +  
  facet_wrap(~section)
```

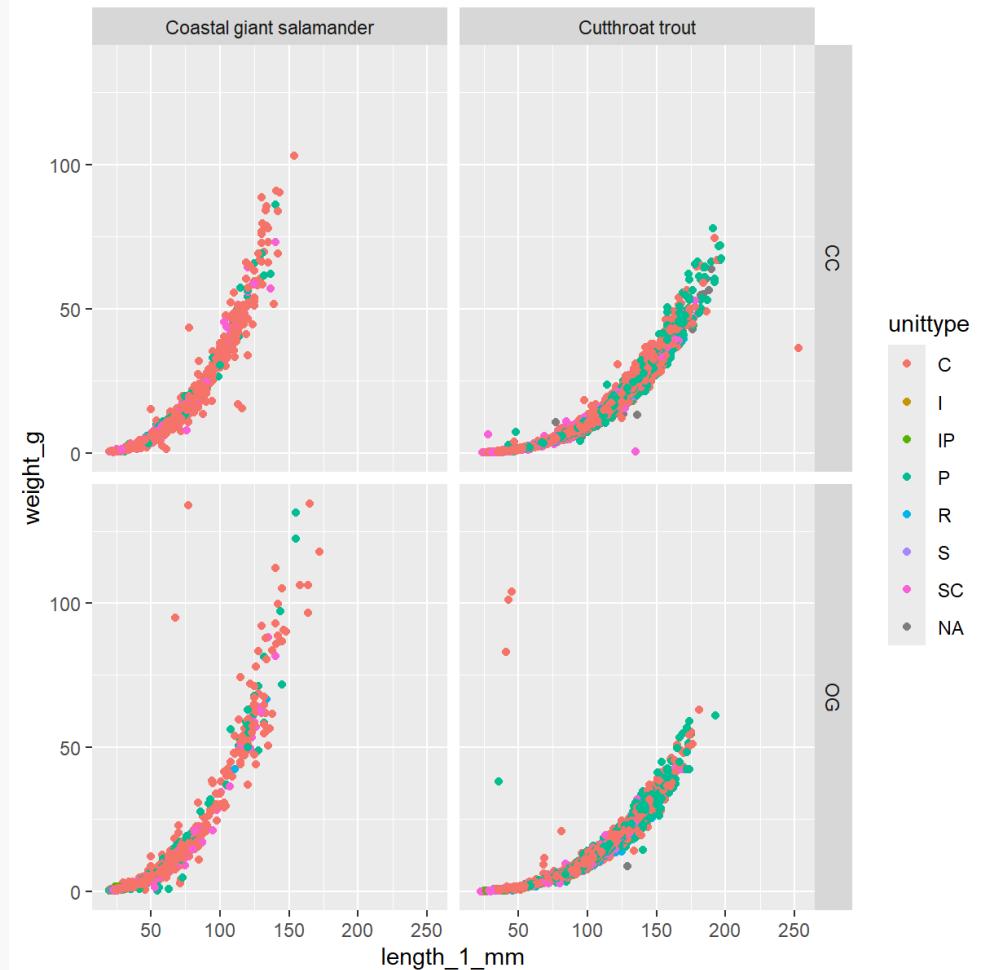


# Small multiples with `facet_grid`

Split your plots along two variables with `facet_grid`

```
ggplot(  
  data = vertebrates,  
  aes(  
    x = length_1_mm,  
    y = weight_g,  
    color = unittype  
  )  
) +  
  geom_point() +  
  facet_grid(section ~ species)
```

- `facet_grid(rows ~ columns)`



# Now you

**Task 1.1 - 1.2 (45 min)**

Exploratory data analysis with the penguin data set

Find the task description [here](#)

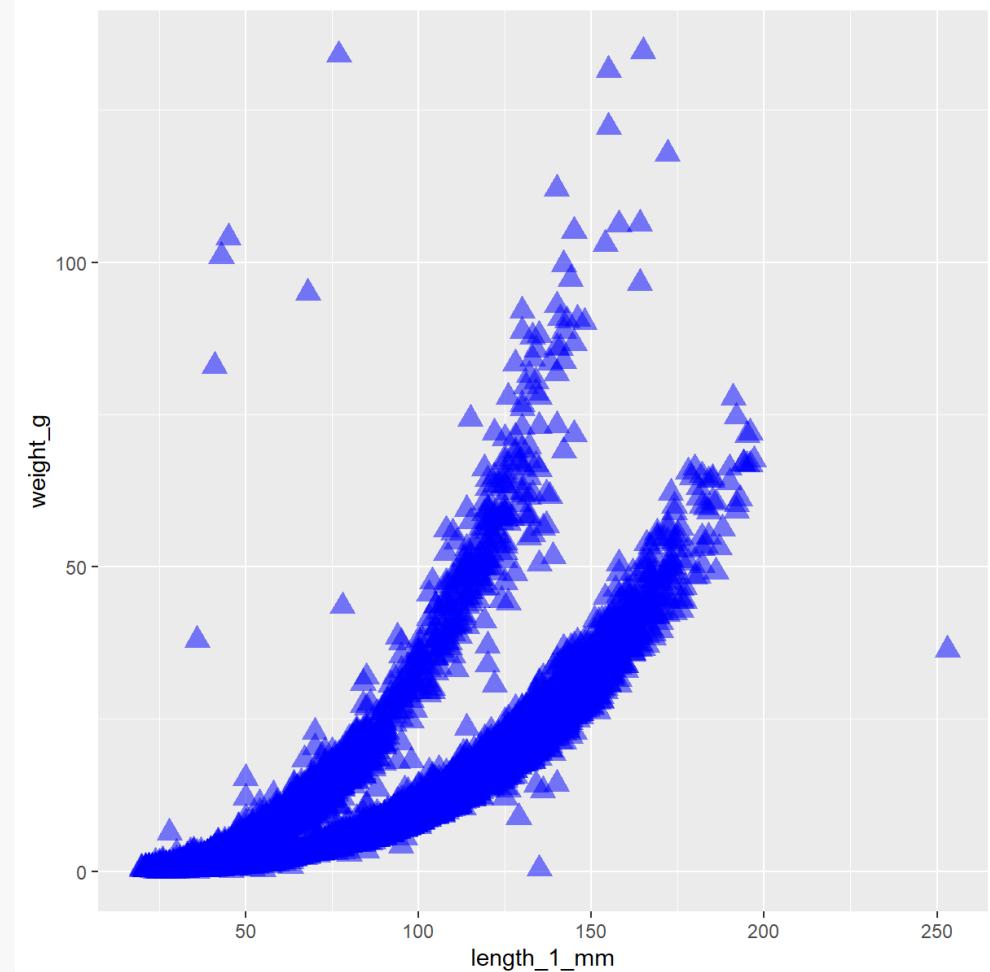
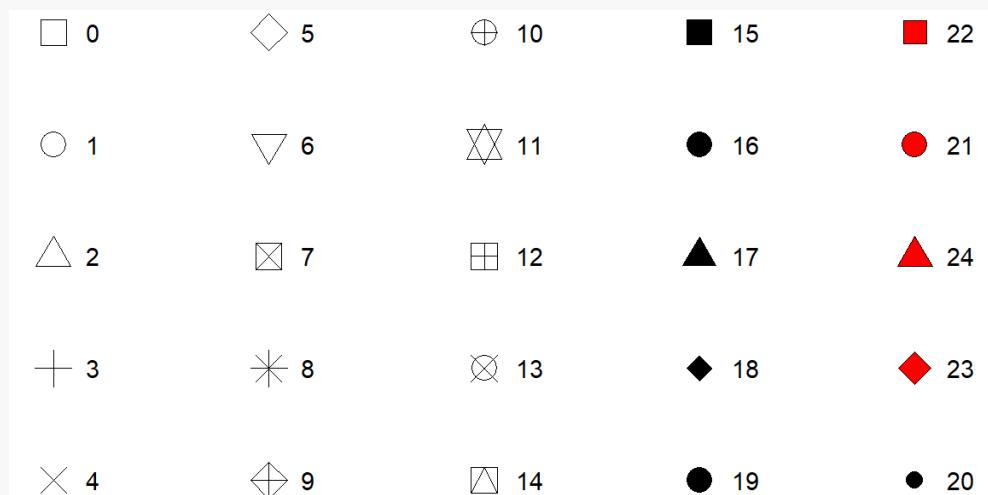
(Don't do the "Beautify" task)



Artwork by Allison Horst

# Change appearance of points

```
ggplot(vertebrates, aes(  
  x = length_1_mm,  
  y = weight_g  
) +  
  geom_point(  
    size = 4,  
    shape = 17,  
    color = "blue",  
    alpha = 0.5  
)
```



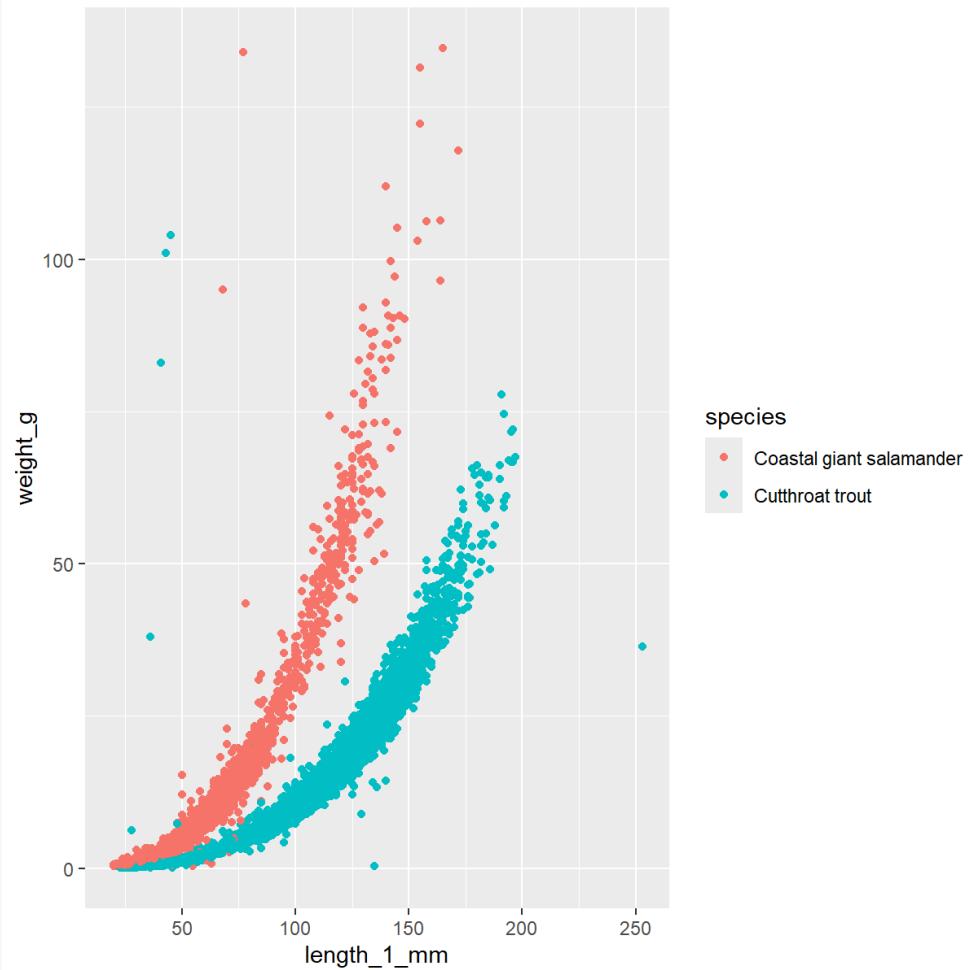
Shape and color codes

# Change color scale

We can also save a plot in a variable

```
g <- ggplot(vertebrates, aes(  
  x = length_1_mm,  
  y = weight_g,  
  color = species  
) +  
  geom_point()  
  
g
```

- Other plot layers can still be added to `g`

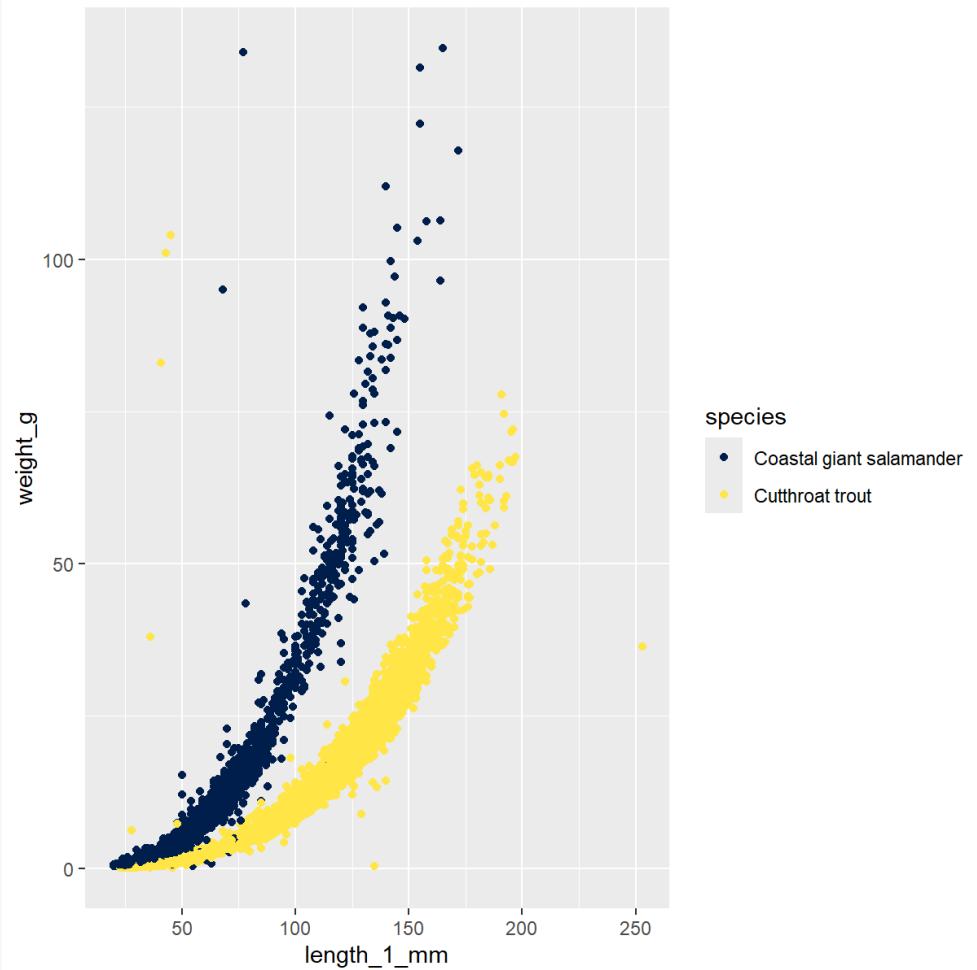


# scale\_color\_viridis\_d

Change the colors of the color aesthetic:

```
g +
  scale_color_viridis_d(
    option = "cividis"
  )
```

- The viridis color palette is designed for viewers with common forms of color blindness
- Different options of viridis color palettes: "magma", "inferno", "plasma", "viridis", "cividis"

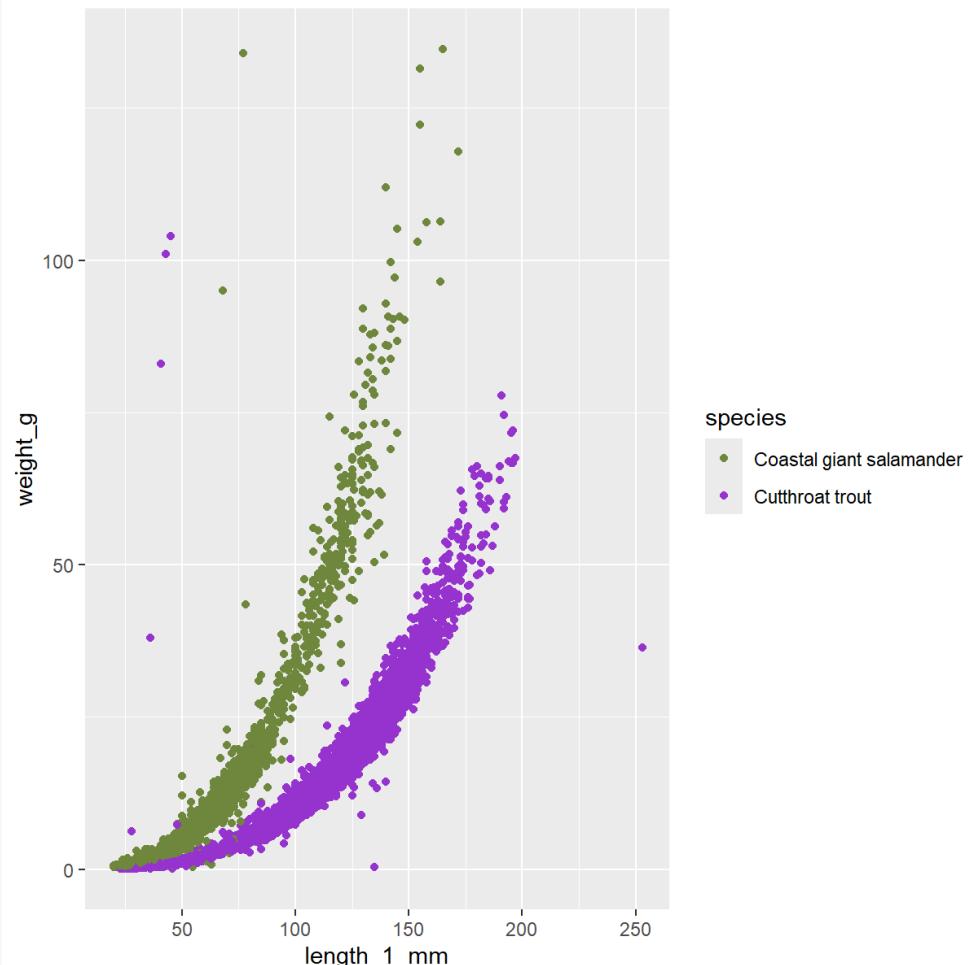


# scale\_color\_manual

We can also manually specify colors:

```
g +
  scale_color_manual(
    values = c(
      "darkolivegreen4",
      "darkorchid3"
    )
  )
```

- Length of color vector has to match number of levels in your aesthetic
- Specify colors
  - Via their [name](#)
  - Via their Hex color codes (use websites to generate your own color palettes, e.g. [here](#))

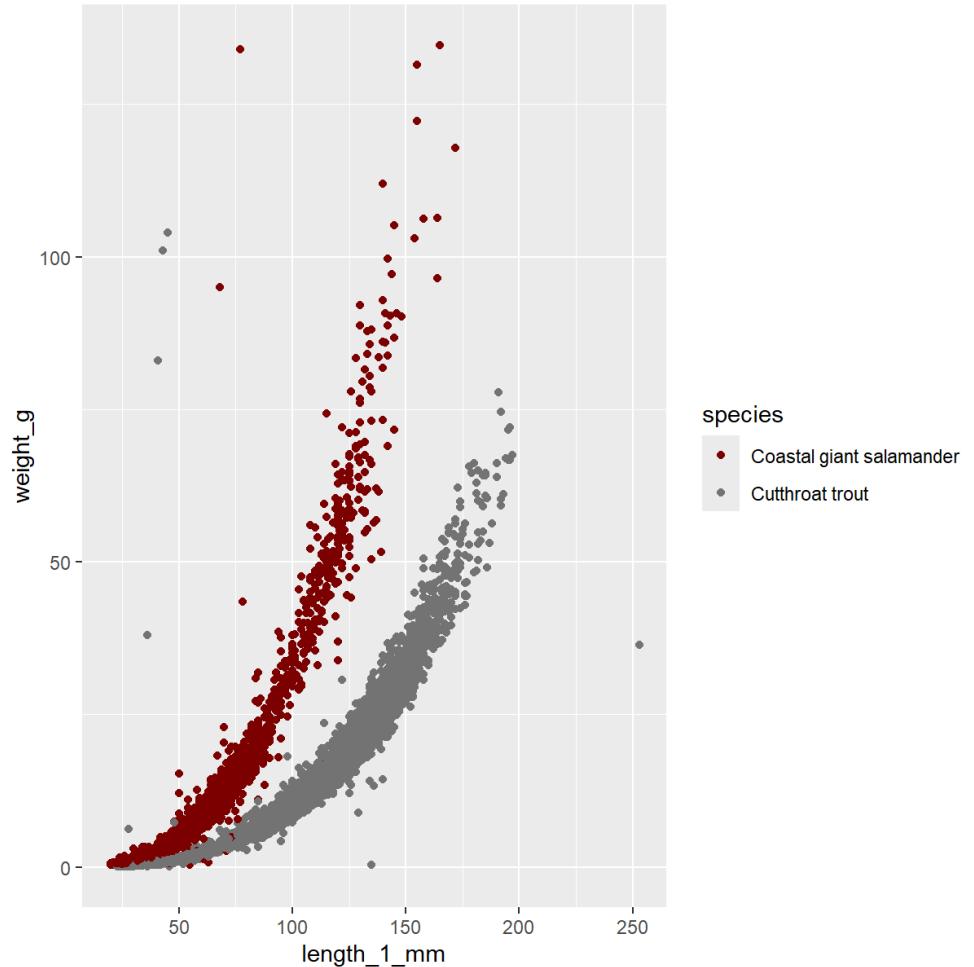


# Other color scales

You can use the `paletteer` package to access color scales from many packages.

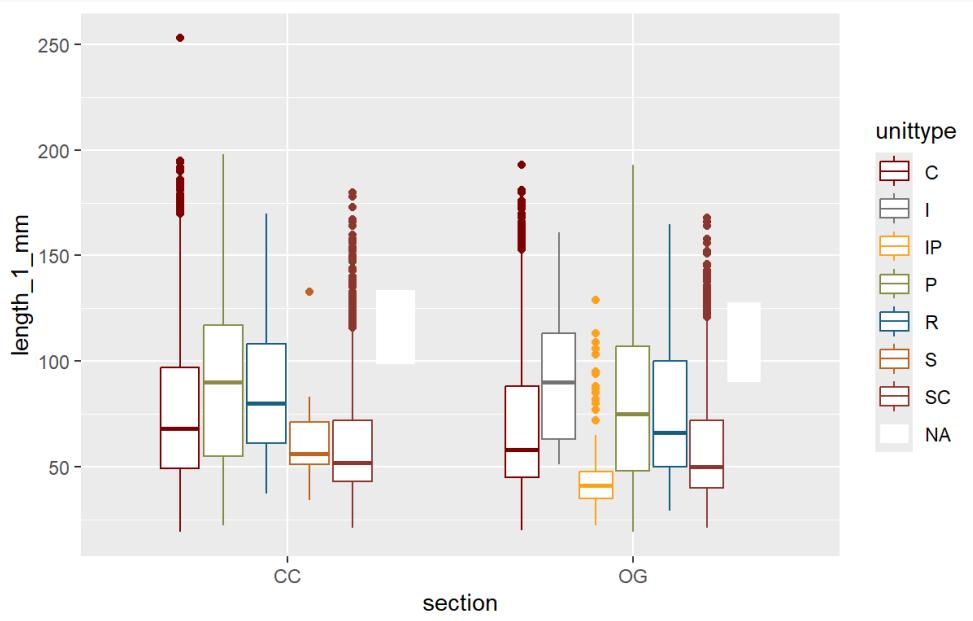
```
# install.packages("paletteer")
library(paletteer)
g <- g +
  scale_color_paletteer_d(
    palette = "ggsci::default_uchicago"
  )
g
```

- Use `scale_color_paletteer_d` for discrete and `scale_color_paletteer_c` for continuous color scales
- Check out all palettes available [here](#)

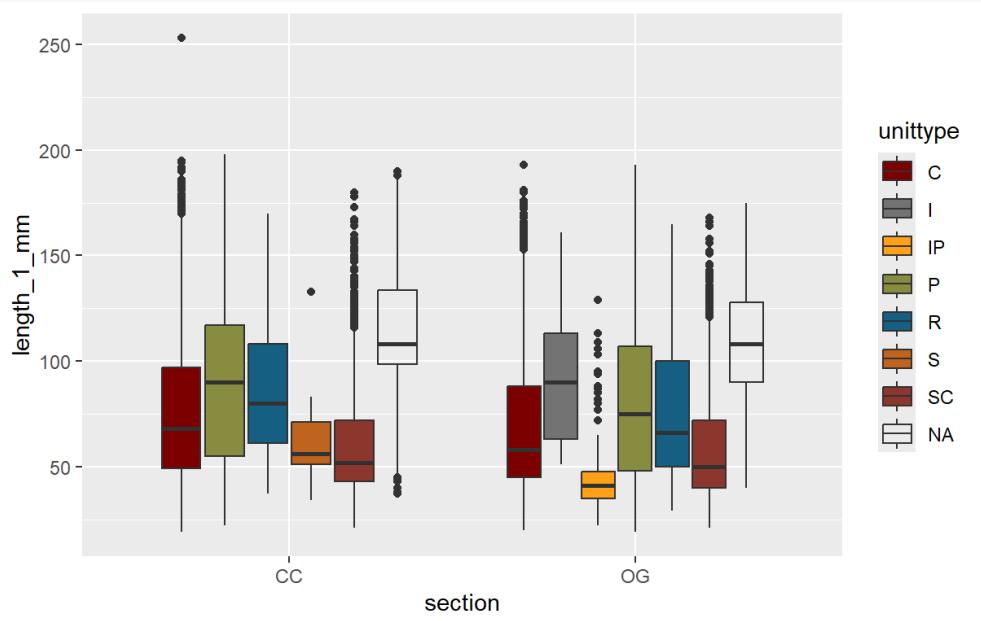


# scale\_fill\_\* vs. scale\_color\_\*

```
ggplot(  
  vertebrates,  
  aes(  
    x = section,  
    y = length_1_mm,  
    color = unitype)) +  
  geom_boxplot() +  
  scale_color_paletteer_d(  
    palette = "ggsci::default_uchicago"  
)
```



```
ggplot(  
  vertebrates,  
  aes(  
    x = section,  
    y = length_1_mm,  
    fill = unitype)) +  
  geom_boxplot() +  
  scale_fill_paletteer_d(  
    palette = "ggsci::default_uchicago"  
)
```



# labs: Change axis and legend titles and add plot title

```
g <- g +
  labs(
    x = "Length [mm]",
    y = "Weight [g]",
    color = "Species",
    title = "Length-Weight relationship",
    subtitle = "There seems to be an exponential relationship",
    caption = "Data from the `lterdatasampler` package"
  )
g
```

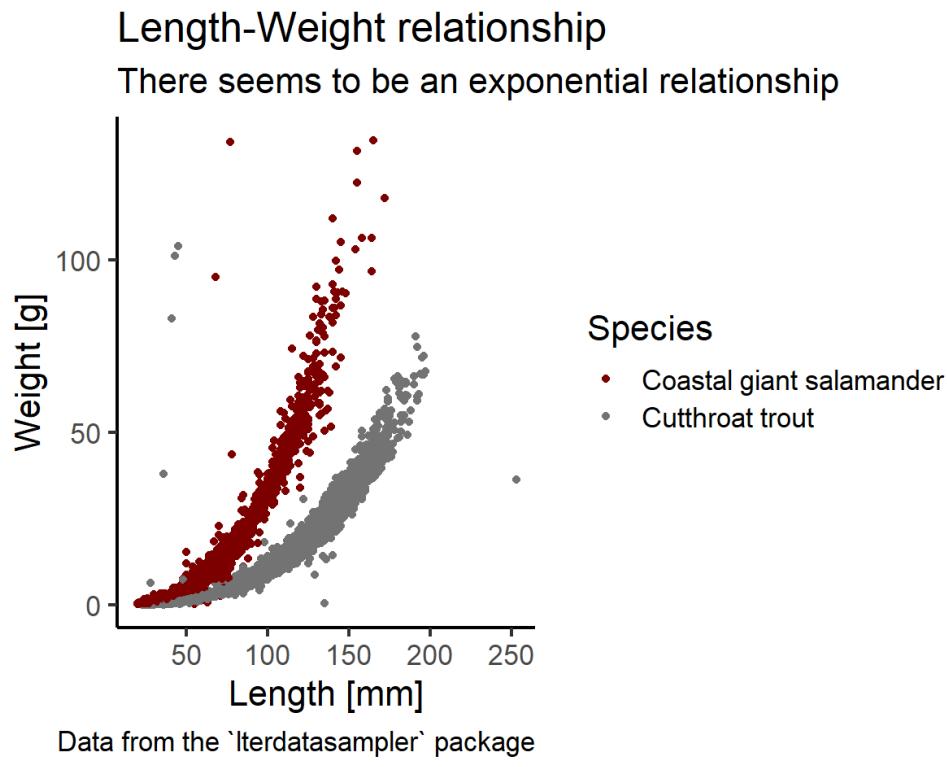
# labs: Change axis and legend titles and add plot title



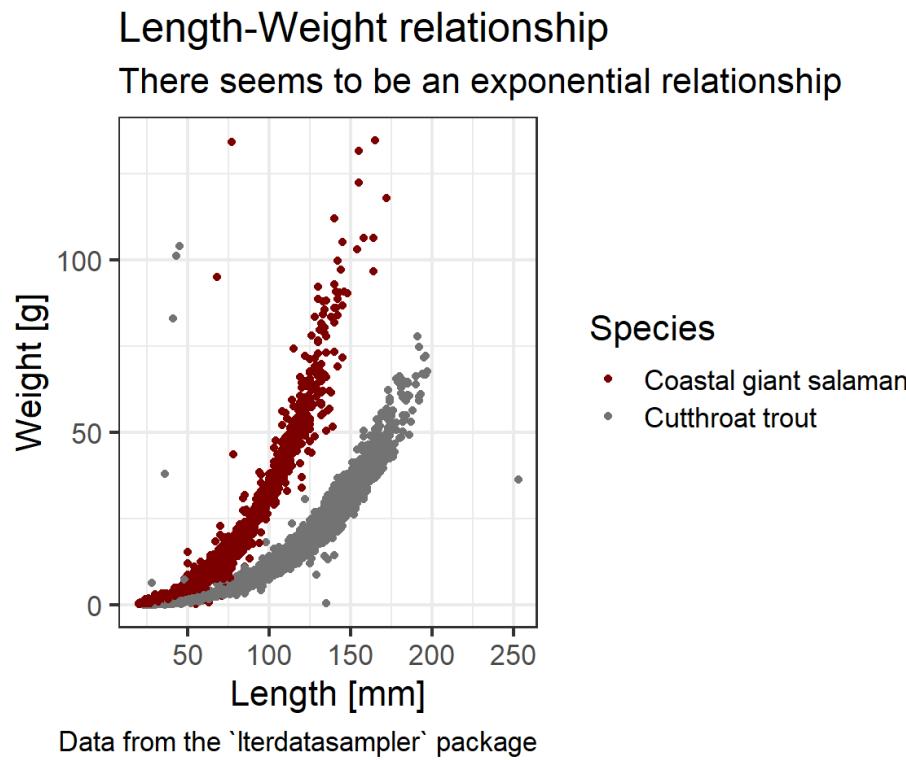
# theme\_\*: change appearance

`ggplot2` offers many pre-defined themes that we can apply to change the appearance of a plot.

```
g +  
  theme_classic()
```



```
g +  
  theme_bw()
```



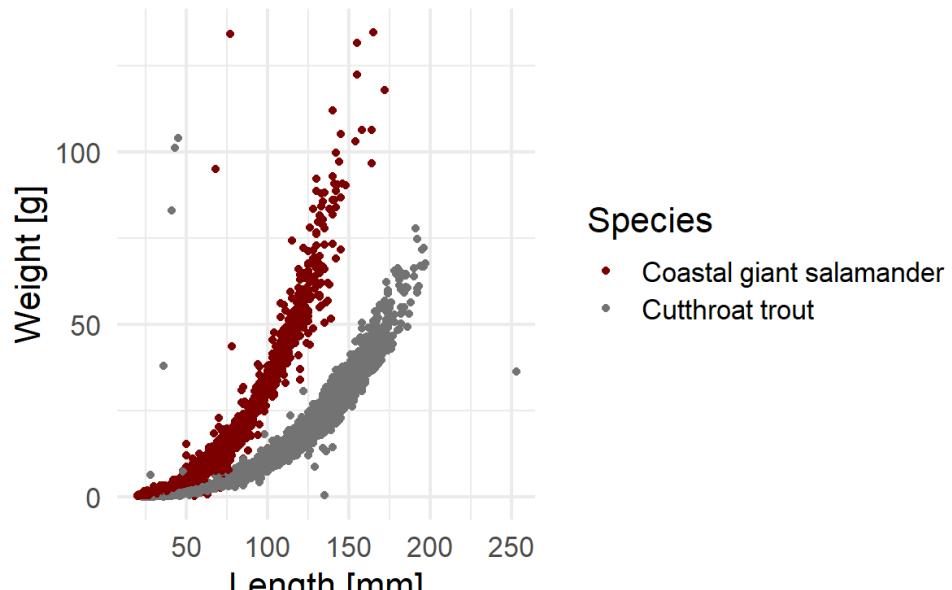
# theme\_\*: change appearance

`ggplot2` offers many pre-defined themes that we can apply to change the appearance of a plot.

```
g +  
  theme_minimal()
```

Length-Weight relationship

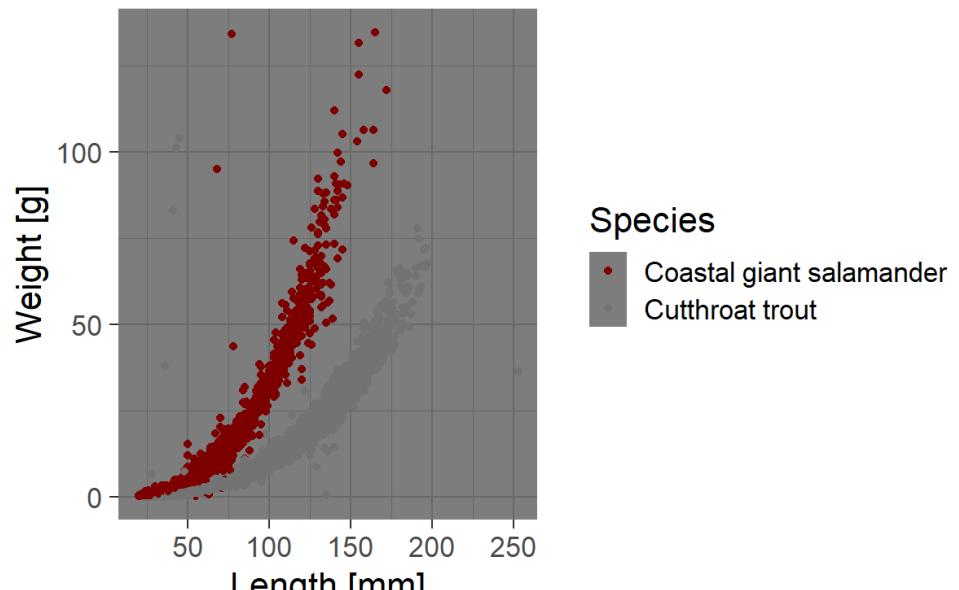
There seems to be an exponential relationship



```
g +  
  theme_dark()
```

Length-Weight relationship

There seems to be an exponential relationship



# theme(): customize theme

You can manually change a theme or even create an entire theme yourself. The elements you can control in the theme are:

- titles (plot, axis, legend, ...)
- labels
- background
- borders
- grid lines
- legends

If you want a full list of what you can customize, have a look at

```
?theme
```

- Look [here](#) for an overview of the elements that you can change and the corresponding functions

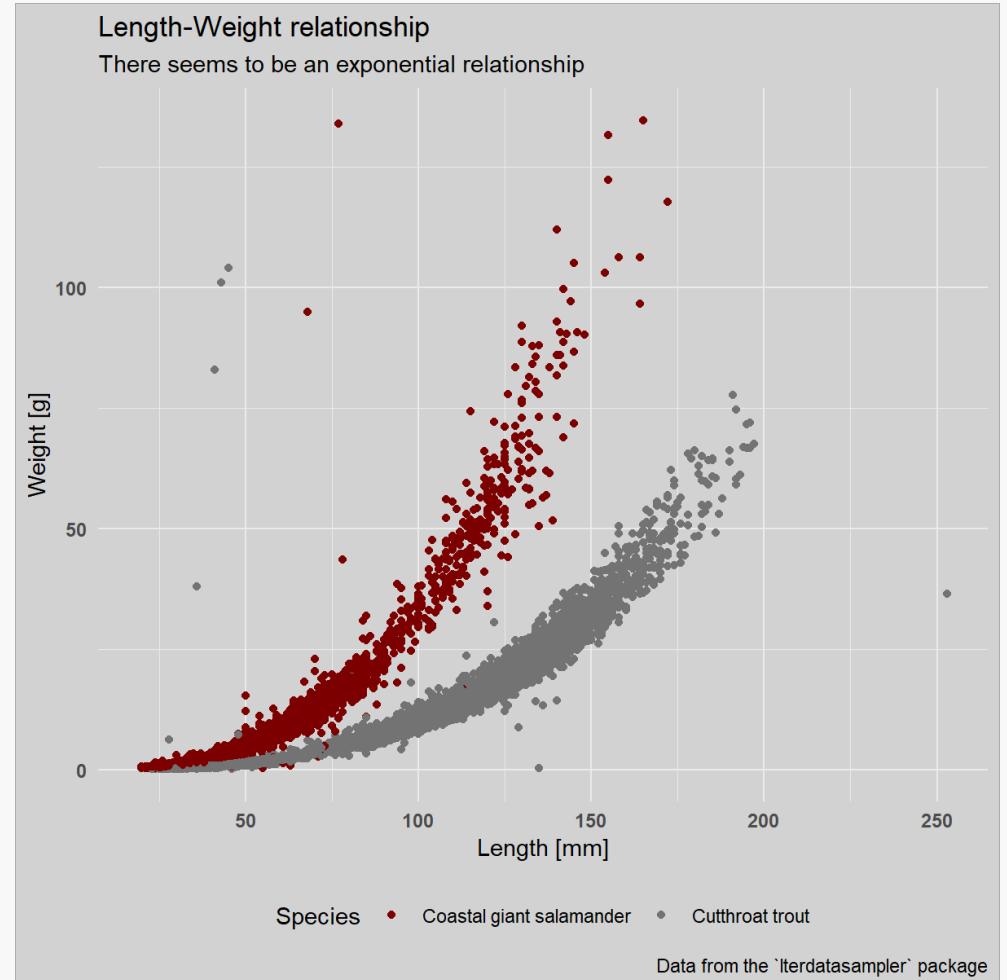
# theme(): customize theme

To edit a theme, just add another `theme()` layer to your plot.

```
g +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    axis.text = element_text(
      face = "bold"
    ),
    plot.background = element_rect(
      fill = "lightgrey",
      color = "darkred"
    )
  )
```

The basic functioning of theme elements  
is:

```
theme(
  element_name = element_function()
)
```



# theme\_set(): set global theme

You can set a global theme that will be applied to all ggplot objects in the current R session.

```
# Globally set theme_minimal as the default theme  
theme_set(theme_minimal())
```

Add this to the beginning of your script.

You can also specify some defaults, e.g. the text size:

```
theme_set(theme_minimal(base_size = 16))
```

This is very practical if you want to achieve a consistent look, e.g. for a scientific journal.

# ggsave()

A ggplot object can be saved on disk in different formats.

Without specifications:

```
# save plot g in img as my_plot.pdf  
ggsave(filename = "img/my_plot.pdf", plot = g)  
# save plot g in img as my_plot.png  
ggsave(filename = "img/my_plot.png", plot = g)
```

Or with specifications:

```
# save a plot named g in the img directory under the name my_plot.png  
# with width 16 cm and height 9 cm  
ggsave(  
  filename = "img/my_plot.png",  
  plot = g,  
  width = 16,  
  height = 9,  
  units = "cm"  
)
```

Have a look at `?ggsave` to see all options.

# Now you

Task 2 (30 min)

Make your penguin plots more beautiful

Find the task description [here](#)