Stat7350: Plotting with ggplot2 (lesson 1)

AC Gerstein

2019-03-05

Learning Objectives

- Produce scatter plots, boxplots, and timeseries plot using ggplot.
- Describe what faceting is and apply faceting in ggplot.
- Modify the aesthetics of an existing ggplot plot (including axis labels and color).
- Build complex and customized plots from data in a data frame.

Pre-analysis workflow: packages & data prep

Folders

We are going to use a consistent workflow throughout this course. Part of that will involve organizing files in a consistent manner. Before we begin, you should make sure you have the following folders in the project directory: * data * data_output * scripts * fig_output * other

Packages

Note that `ggplot2` is include in the metapackage `tidyverse`. If you have not previously loaded these libraries you will need to use `install.packages()` to do so before running this code.

```
library(tidyverse)
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

Load Data

Today we are going to use data from the Portal Project Teaching Database. The data is a time-series for a small mammal community in southern Arizona. This is part of a project studying the effects of rodents and ants on the plant community that has been running for almost 40 years. The rodents are sampled on a series of 24 plots, with different experimental manipulations controlling which rodents are allowed to access which plots. For more information see http://esapubs.org/archive/ecol/E090/118/ and https://datacarpentry.org/ecology-workshop/data/.

The dataset is stored as a comma separated value (CSV) file. Each row holds information for a single animal, and the columns represent:

Table 1: Data dictionary.

Column	Description
record_id	Unique id for the observation
month	month of observation
day	day of observation
year	year of observation
plot_id	ID of a particular plot
species_id	2-letter code
sex	sex of animal ("M", "F")
hindfoot_length	length of the hindfoot in mm
weight	weight of the animal in grams
genus	genus of animal
species	species of animal
taxon	e.g. Rodent, Reptile, Bird, Rabbit
plot_type	type of plot

We are going to use the R function download.file() to download the CSV file that contains the survey data from figshare, and we will use `read_csv()` to load into memory the content of the CSV file as an object of class data.frame. Inside the download.file command, the first entry is a character string with the source URL ("https://ndownloader.figshare.com/files/2292169"). This source URL downloads a CSV file from figshare. The text after the comma ("data/portal_data_joined.csv") is the destination of the file on your local machine. You'll need to have a folder on your machine called "data" where you'll download the file. So this command downloads a file from figshare, names it "portal_data_joined.csv," and adds it to a preexisting folder named "data."

```
download.file(url="https://ndownloader.figshare.com/files/2292169",
              destfile = "data/portal_data_joined.csv")
```

Then load the data and save it as `surveys`. Remember that `read_csv` will save as a tibble.

```
surveys <- read_csv("data/portal_data_joined.csv")
```

```
## Parsed with column specification:
## cols(
##   record_id = col_double(),
##   month = col_double(),
##   day = col_double(),
##   year = col_double(),
##   plot_id = col_double(),
##   species_id = col_character(),
##   sex = col_character(),
##   hindfoot_length = col_double(),
##   weight = col_double(),
##   genus = col_character(),
##   species = col_character(),
##   taxa = col_character(),
##   plot_type = col_character()
## )
```

```
surveys
```

```
## # A tibble: 34,786 x 13
##    record_id month   day  year plot_id species_id sex   hindfoot_length
##        <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>
```

```
##  1         1     7    16  1977       2 NL         M                 32
##  2        72     8    19  1977       2 NL         M                 31
##  3       224     9    13  1977       2 NL         <NA>              NA
##  4       266    10    16  1977       2 NL         <NA>              NA
##  5       349    11    12  1977       2 NL         <NA>              NA
##  6       363    11    12  1977       2 NL         <NA>              NA
##  7       435    12    10  1977       2 NL         <NA>              NA
##  8       506     1     8  1978       2 NL         <NA>              NA
##  9       588     2    18  1978       2 NL         M                 NA
## 10       661     3    11  1978       2 NL         <NA>              NA
## # ... with 34,776 more rows, and 5 more variables: weight <dbl>,
## #   genus <chr>, species <chr>, taxa <chr>, plot_type <chr>
```

genus are unique; but species may not be unique, some species share the same speci name
taxa

Tidy the data

In preparation for plotting, we are going to prepare a cleaned up version of the data set that doesn't include any missing data.

Let's start by removing observations of animals for which weight and hindfoot_length are missing, or the sex has not been determined:

```
surveys_complete <- surveys %>%
  filter(!is.na(weight),           # remove missing weight
         !is.na(hindfoot_length),  # remove missing hindfoot_length
         !is.na(sex))                # remove missing sex
```

CHALLENGE

How many observations were removed above?

Because we are interested in plotting how species abundances have changed through time, we are also going to remove observations for rare species (i.e., that have been observed less than 50 times). We will do this in two steps: first we are going to create a data set that counts how often each species has been observed, and filter out the rare species; then, we will extract only the observations for these more common species:

```
## Extract the most common species_id
species_counts <- surveys_complete %>%
    count(species_id) %>%
    filter(n >= 50)
```

biological decision pointed out by someone

```
## Only keep the most common species
surveys_complete <- surveys_complete %>%
  filter(species_id %in% species_counts$species_id)
```

Surveys_complete should have 30463 rows and 13 columns.

Now that our data set is ready, we can save it as a CSV file in our data_output folder.

```
write_csv(surveys_complete, path = "data_output/surveys_complete.csv")
```

Plotting with ggplot2

ggplot2 is a plotting package that makes it simple to create complex plots from data in a data frame. It provides a more programmatic interface for specifying what variables to plot, how they are displayed, and general visual properties. Therefore, we only need minimal changes if the underlying data change or if we decide to change from a bar plot to a scatterplot. This helps in creating publication quality plots with minimal amounts of adjustments and tweaking.

ggplot2 functions like data in the 'long' format, i.e., a column for every dimension, and a row for every observation. Well-structured data will save you lots of time when making figures with ggplot2.

ggplot graphics are built step by step by adding new elements. Adding layers in this fashion allows for extensive flexibility and customization of plots.

To build a ggplot, we will use the following basic template that can be used for different types of plots:

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

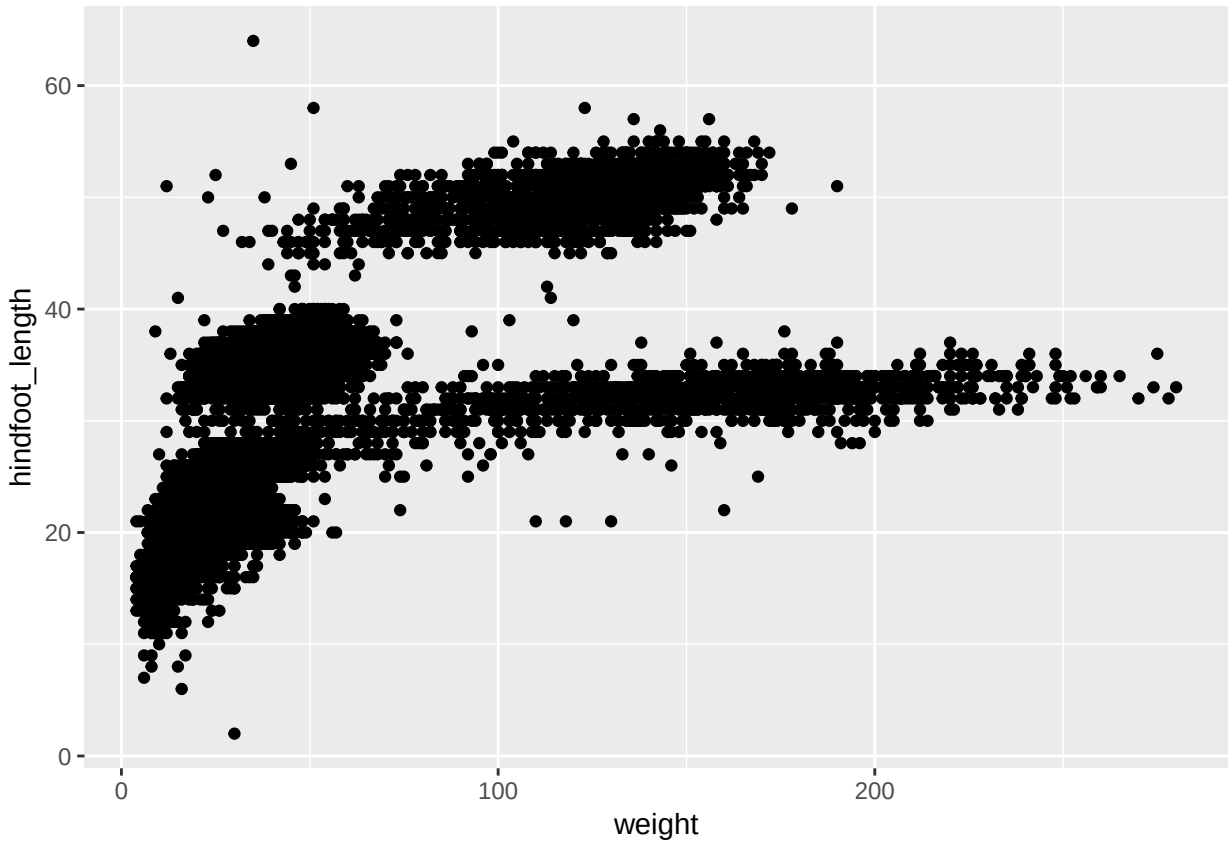
Use the ggplot() function and bind the plot to a specific data frame using the data argument `ggplot(data = surveys_complete)`

Define a mapping (using the aesthetic (aes) function), by selecting the variables to be plotted and specifying how to present them in the graph, e.g. as x/y positions or characteristics such as size, shape, color, etc. `ggplot(data = surveys_complete, mapping = aes(x = weight, y = hindfoot_length))`

Need to add 'geoms' – graphical representations of the data in the plot (points, lines, bars). ggplot2 offers many different geoms; we will use some common ones today, including: - `geom_point()` for scatter plots, dot plots, etc. - `geom_boxplot()` for, well, boxplots! - `geom_line()` for trend lines, time series, etc.

Let's compare weight and hindfoot length. Since these are both continuous variables, use the `geom_point`

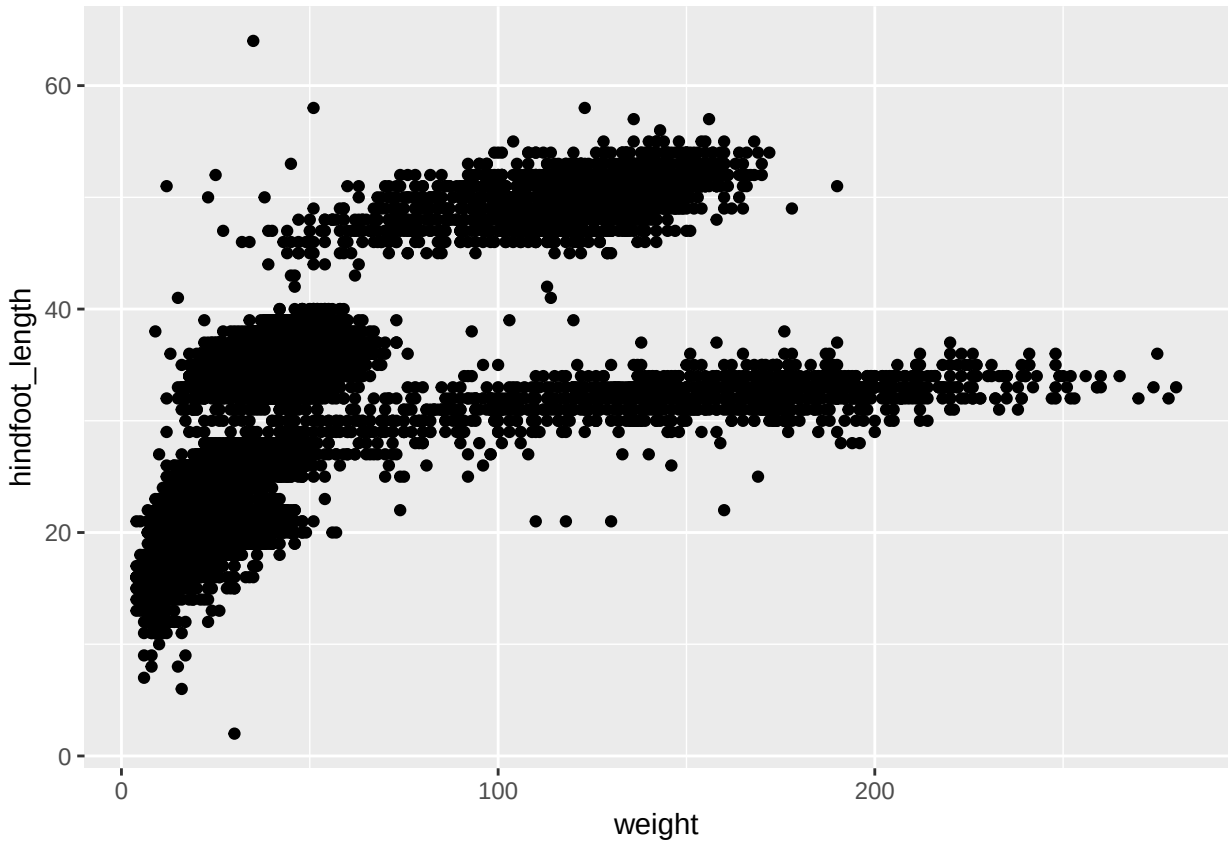
```
ggplot(data = surveys_complete, mapping = aes(x = weight, y = hindfoot_length)) +
  geom_point()
```



Build plots iteratively

Save the base plot as variable p that we can 'build up' as we go

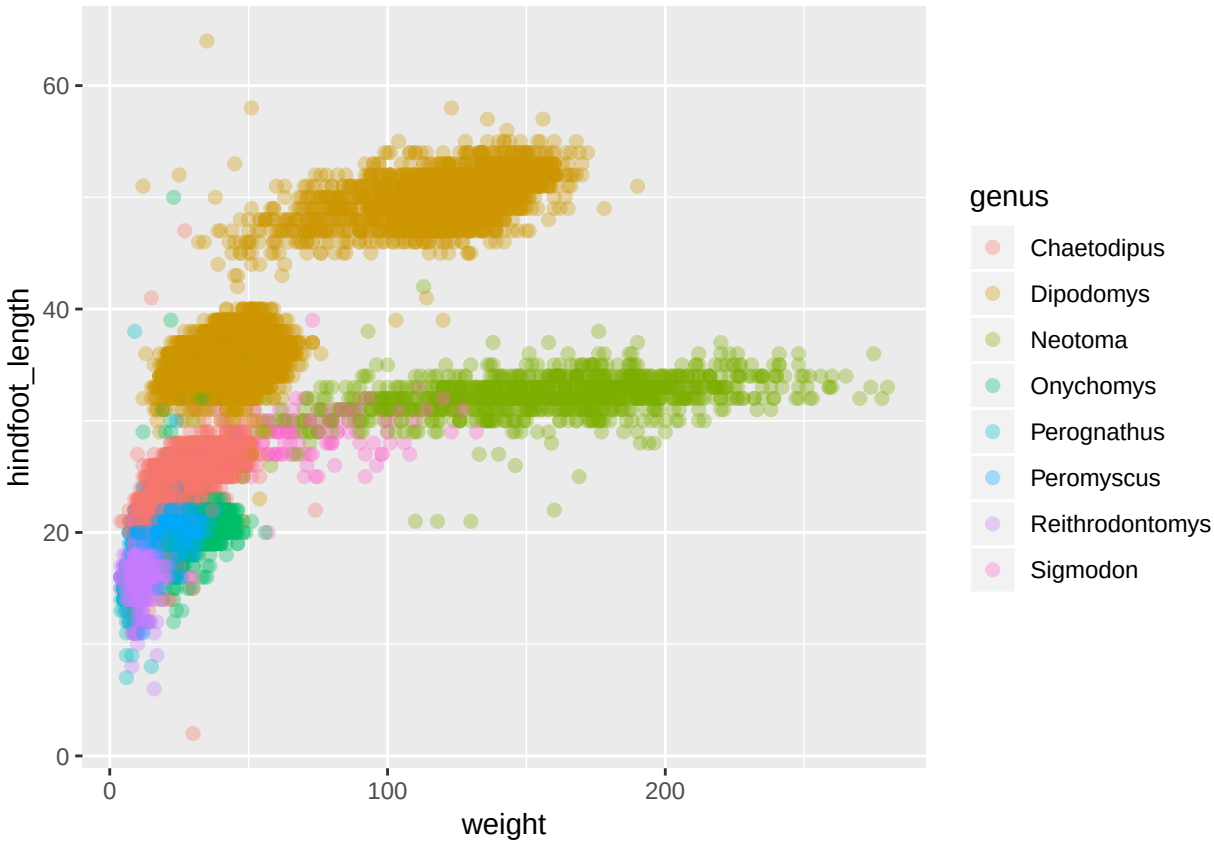
```
p <- ggplot(data = surveys_complete, mapping = aes(x = weight, y = hindfoot_length))
p + geom_point()
```



A common workflow is to gradually build up the plot you want and re-define the object 'p' as you develop "keeper" commands.

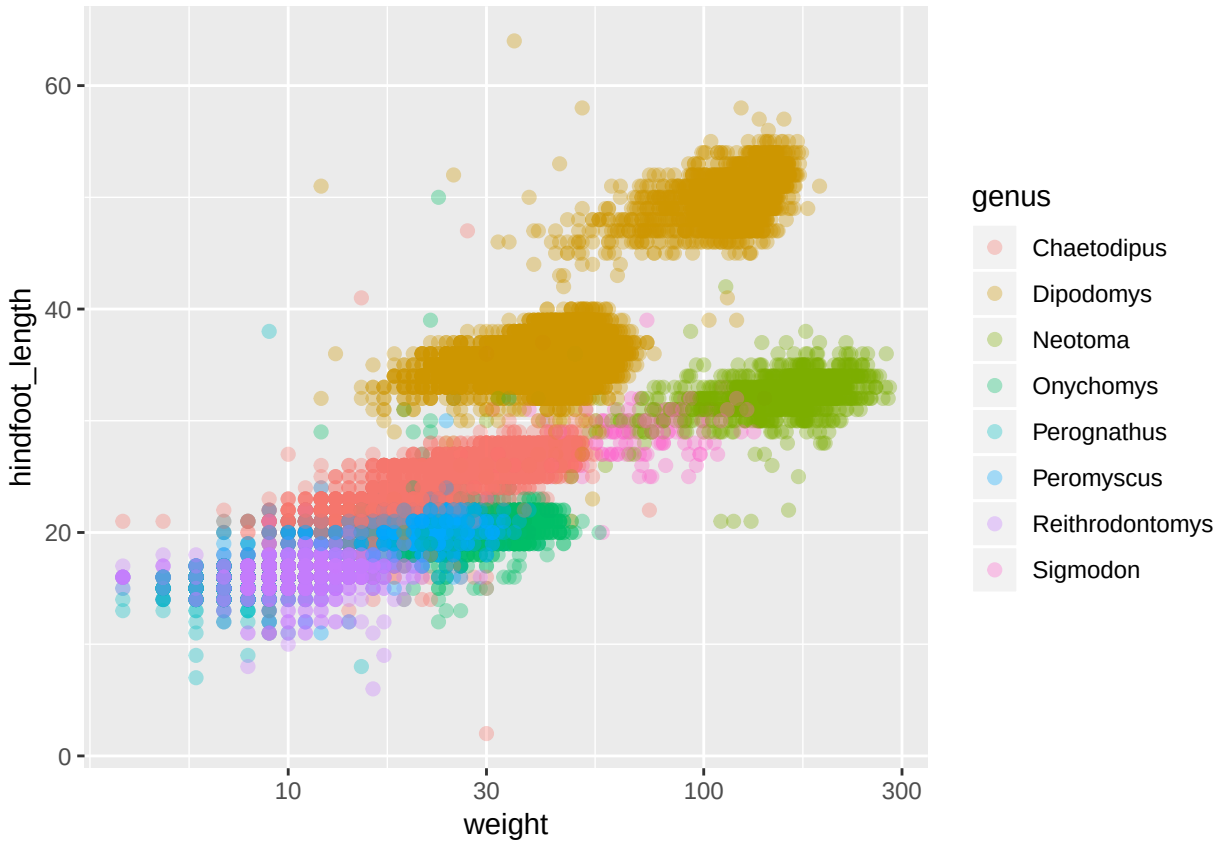
Convey species by color: MAP species_id variable to aesthetic color, add transarency (alpha) and change the point size

```
p +
  geom_point(alpha= (1/3), size=2, aes(color = genus))
```



CHALLENGE

Can you log10 transform weight?

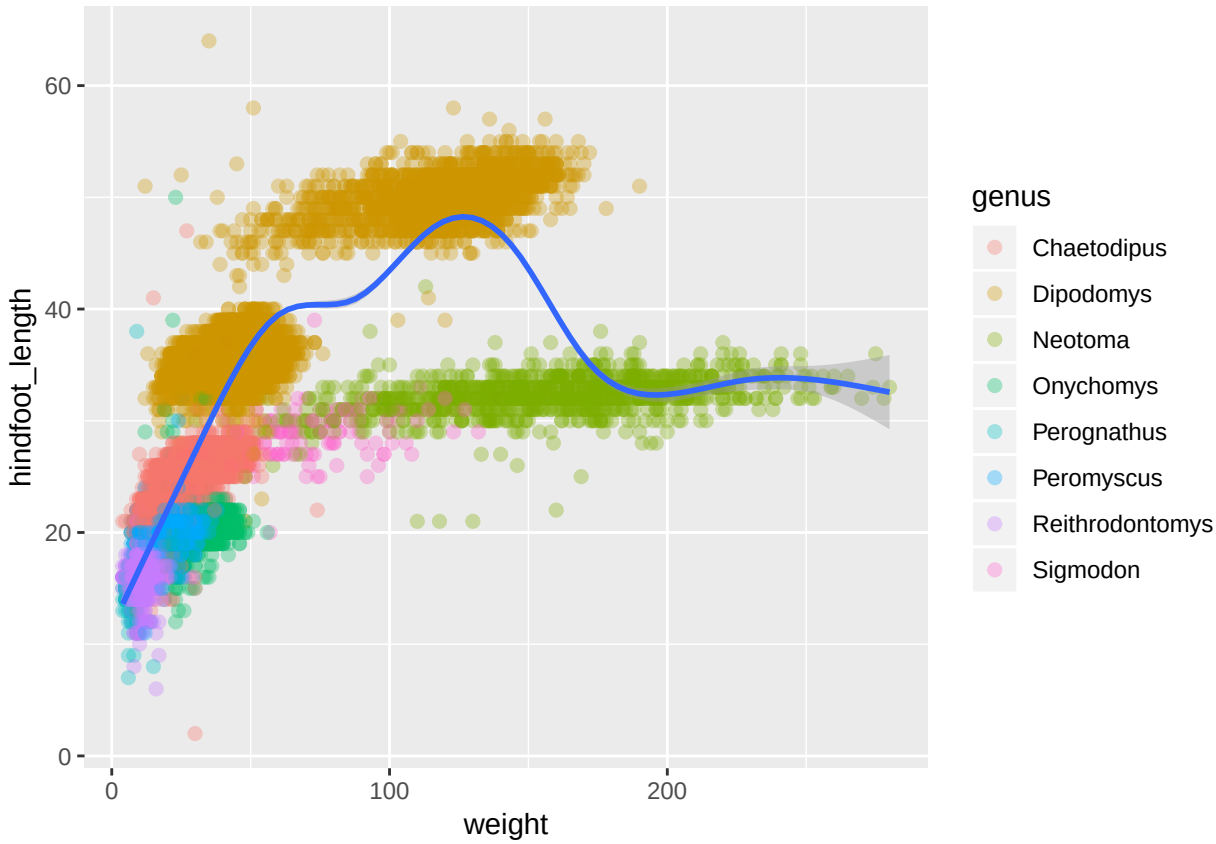


Add fitted curves or lines

add a fitted curve or line

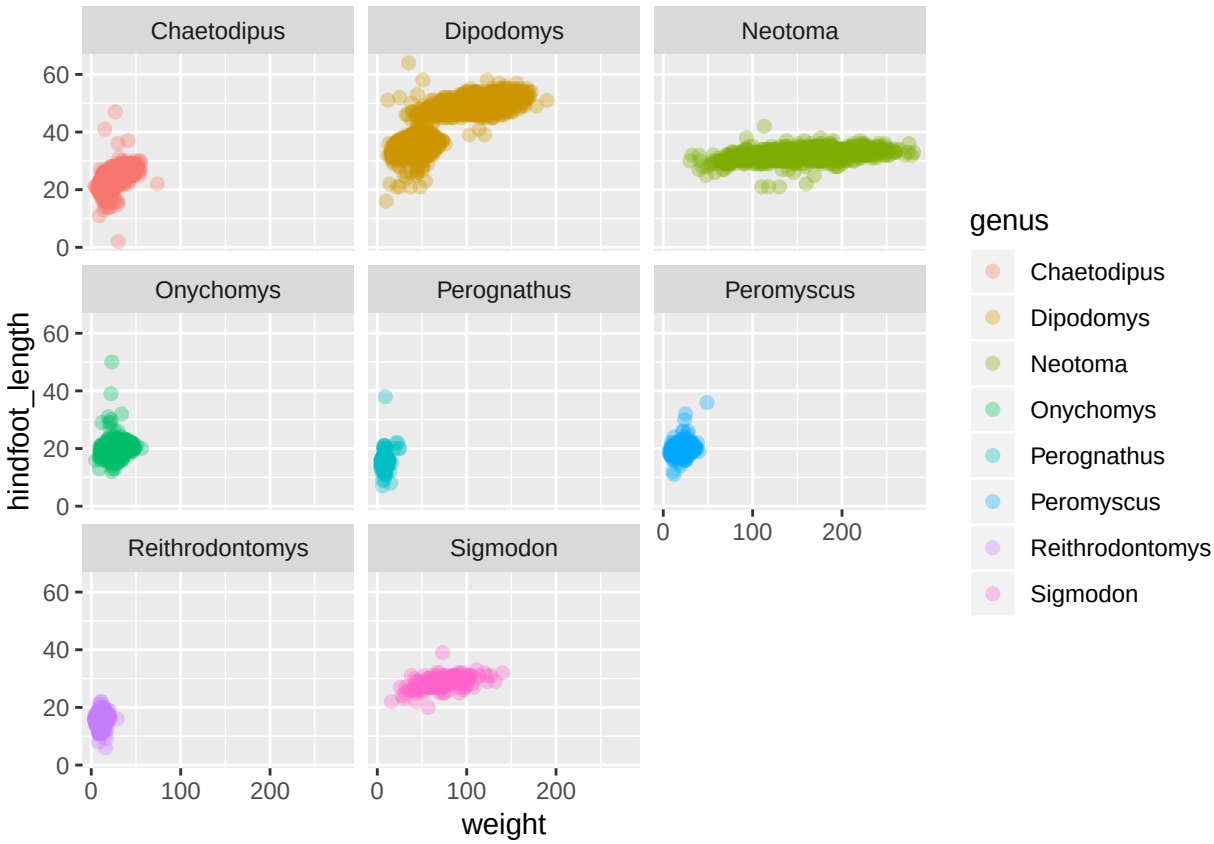
```
p + geom_point(alpha= (1/3), size=2, aes(color = genus)) +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



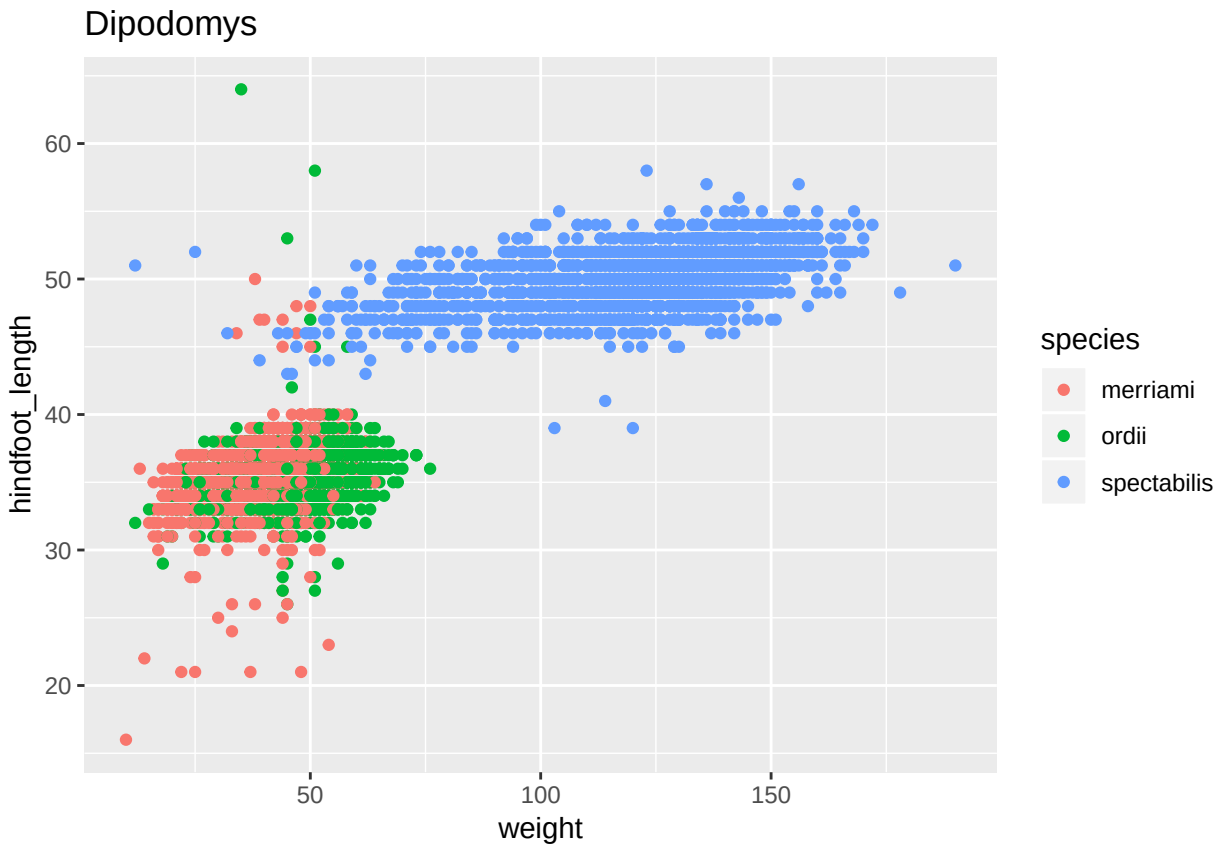
Facetting: another way to exploit a factor

```
p + geom_point(alpha= (1/3), size=2, aes(color = genus)) +
  facet_wrap(~ genus)
```



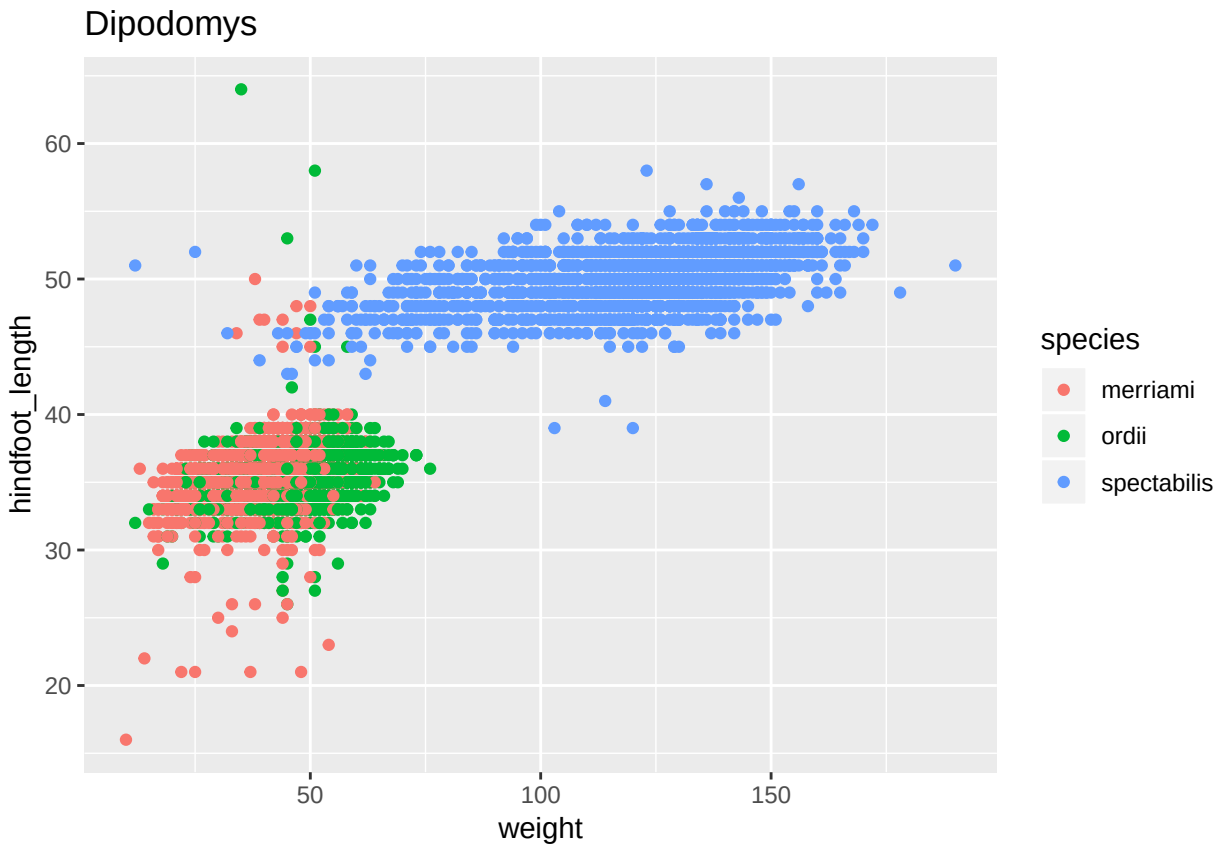
Use `dplyr::filter()` to plot for just one genus

```
Dp <- "Dipodomys"
surveys_complete %>%
  filter(genus == Dp) %>%
  ggplot(aes(x = weight, y = hindfoot_length)) +
  labs(title = Dp) +
  geom_point(aes(colour=species))
```



Another way:

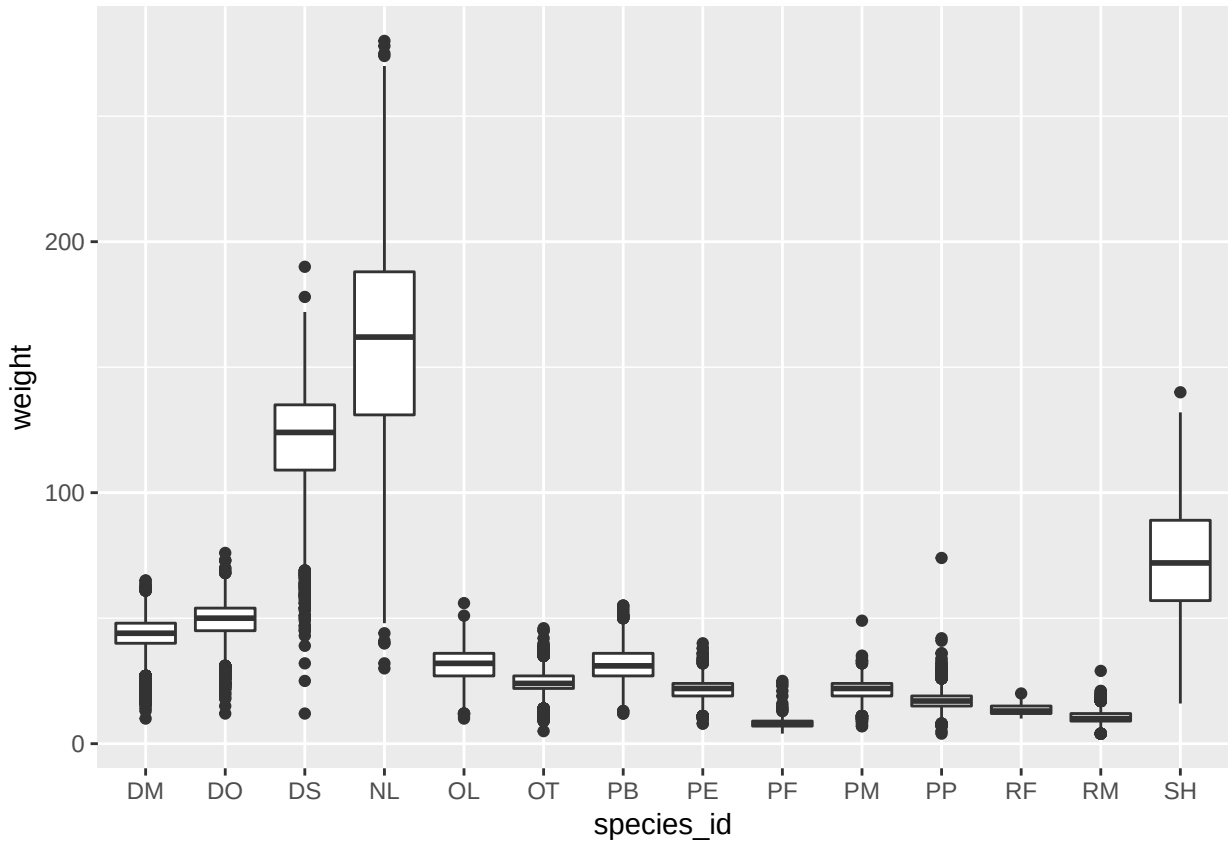
```
ggplot(surveys_complete %>% filter(genus == Dp),
   aes(x = weight, y = hindfoot_length)) +
  labs(title = Dp) +
  geom_point(aes(colour=species))
```



Boxplot

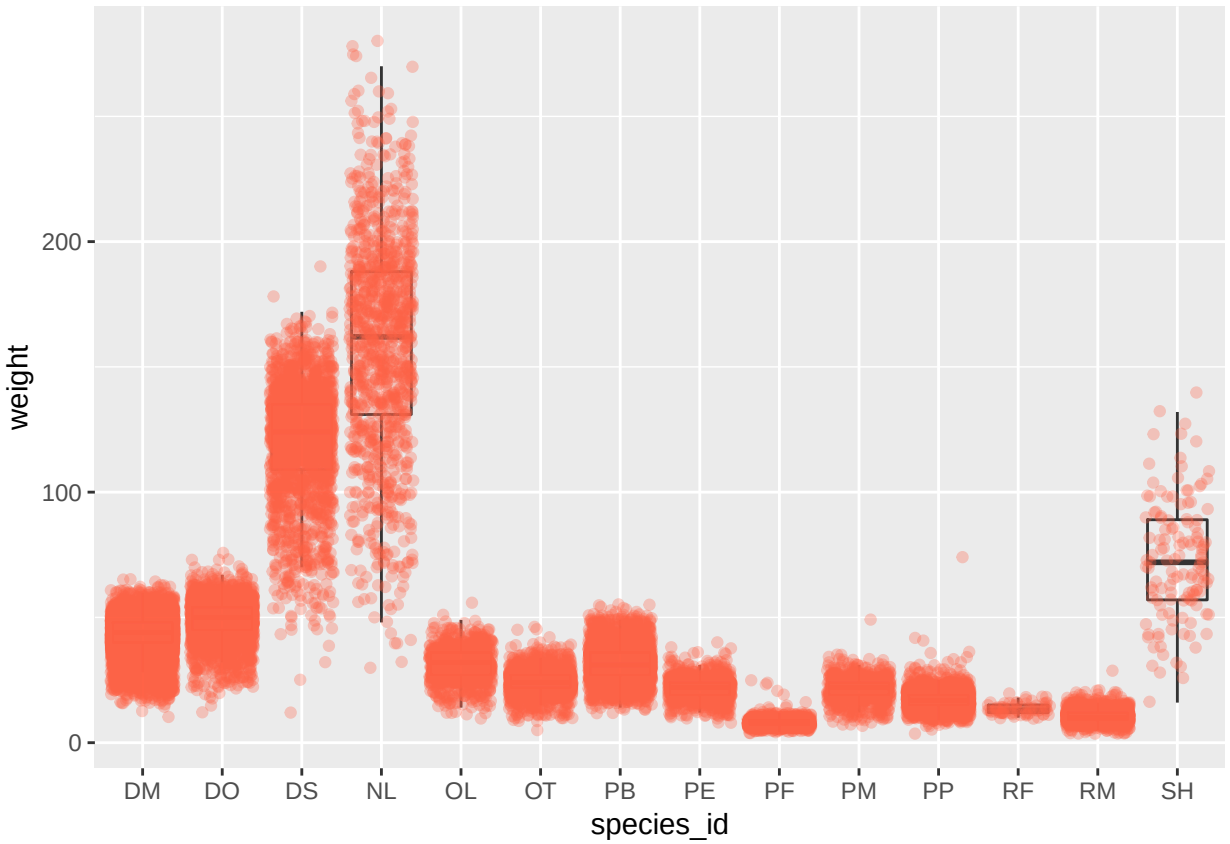
We can use boxplots to visualize the distribution of weight within each species:

```
ggplot(data = surveys_complete, mapping = aes(x = species_id, y = weight)) +
    geom_boxplot()
```



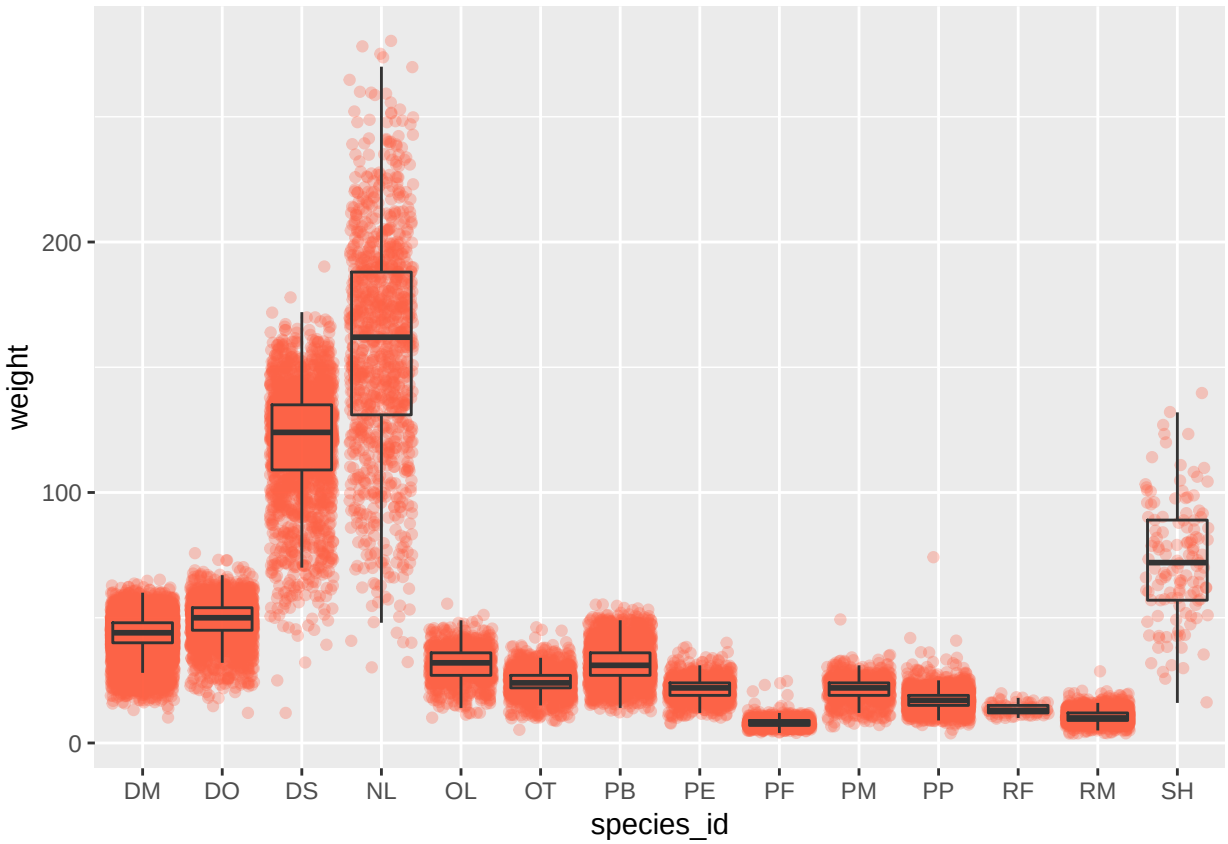
By adding points to boxplot, we can have a better idea of the number of measurements and of their distribution:

```
ggplot(data = surveys_complete, mapping = aes(x = species_id, y = weight)) +
    geom_boxplot(alpha = 0) +
    geom_jitter(alpha = 0.3, color = "tomato")
```



Notice how the boxplot layer is behind the jitter layer? What do you need to change in the code to put the boxplot in front of the points such that it's not hidden?

by change the order of the codes



CHALLENGE

Boxplots are useful summaries, but hide the shape of the distribution. For example, if the distribution is bimodal, we would not see it in a boxplot. An alternative to the boxplot is the violin plot, where the shape (of the density of points) is drawn.

- Replace the box plot with a violin plot; see geom_violin().
- Try making a new plot to explore the distribution of another variable within each species:
  - Create a boxplot for hindfoot_length. Overlay the boxplot layer on a jitter layer to show actual measurements.
  - Add color to the data points on your boxplot according to the plot from which the sample was taken (plot_id). how to do this??

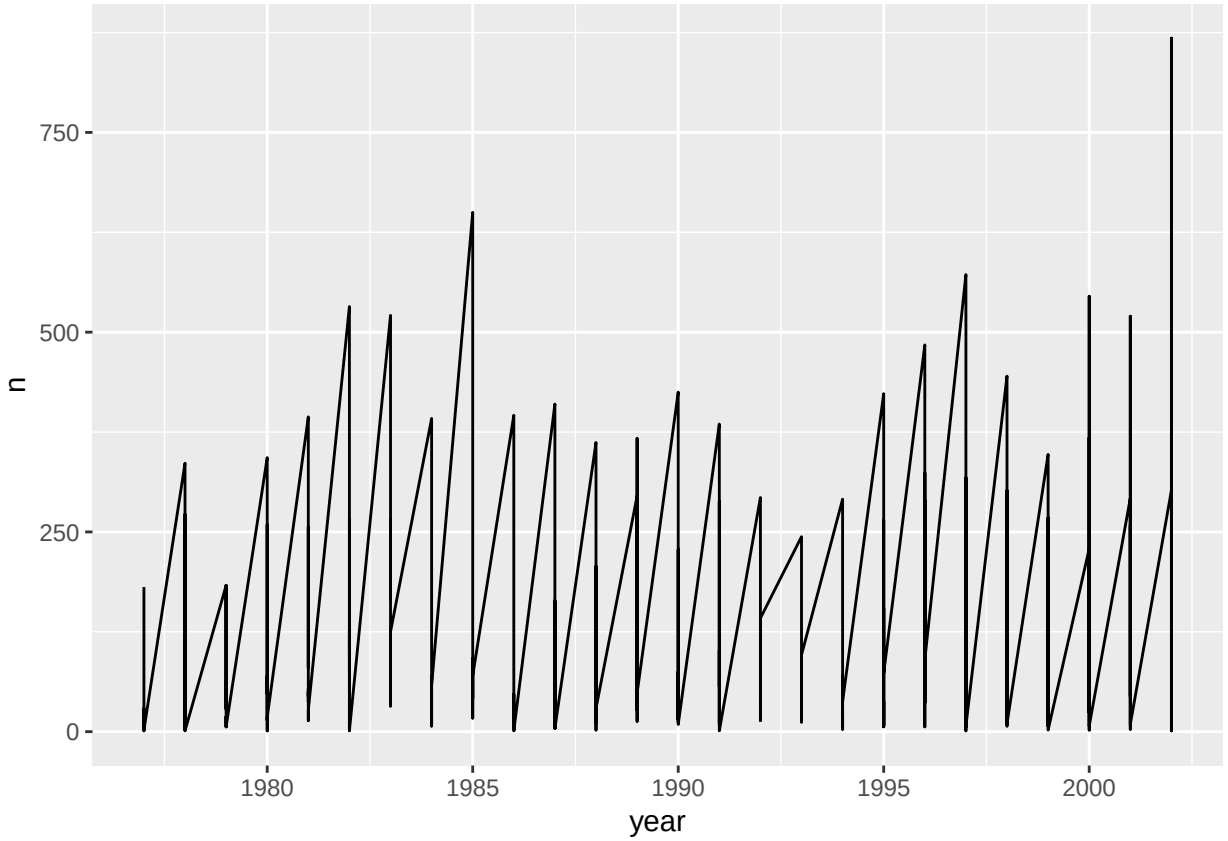
Plotting time series data

Let's calculate number of counts per year for each species. First we need to group the data and count records within each group:

```
yearly_counts <- surveys_complete %>%
                 count(year, species_id)
```

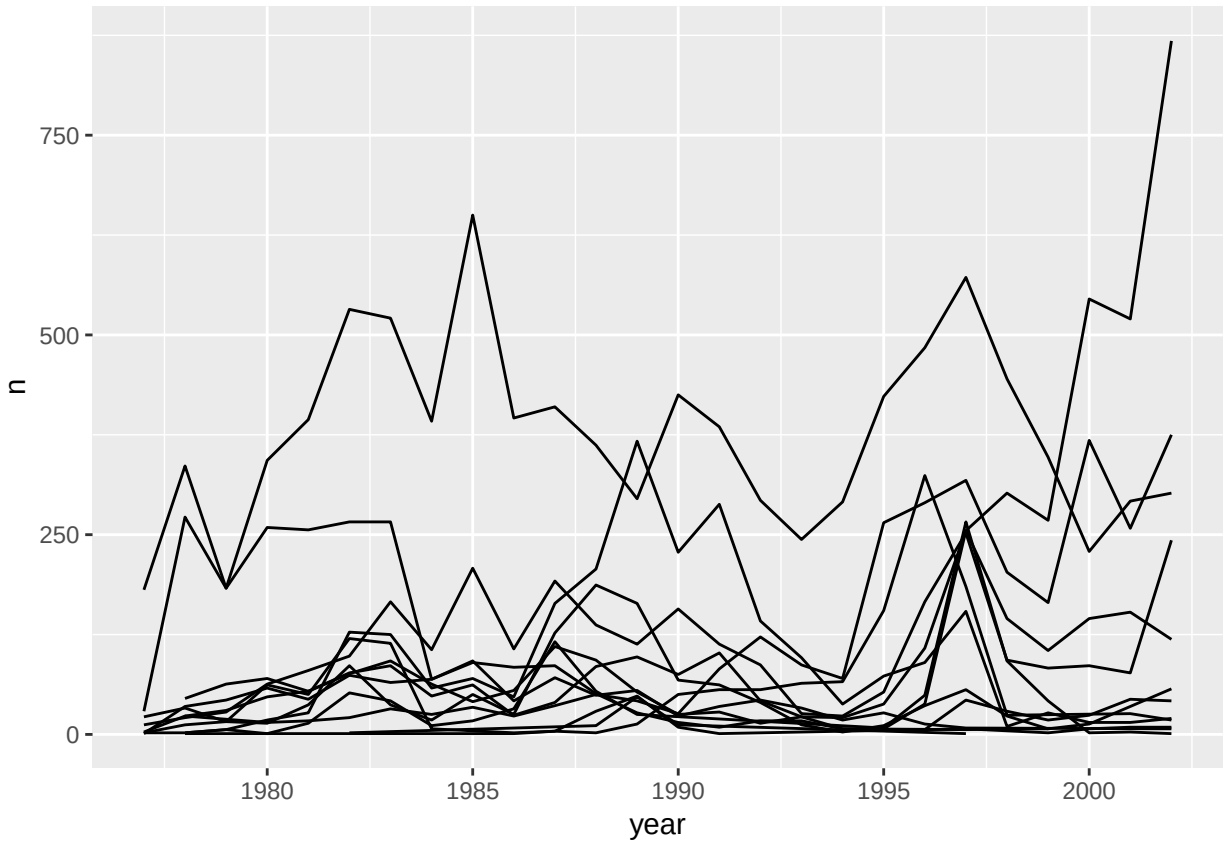
Time series data can be visualized as a line plot with years on the x axis and counts on the y axis:

```
ggplot(data = yearly_counts, mapping = aes(x = year, y = n)) +
    geom_line()
```



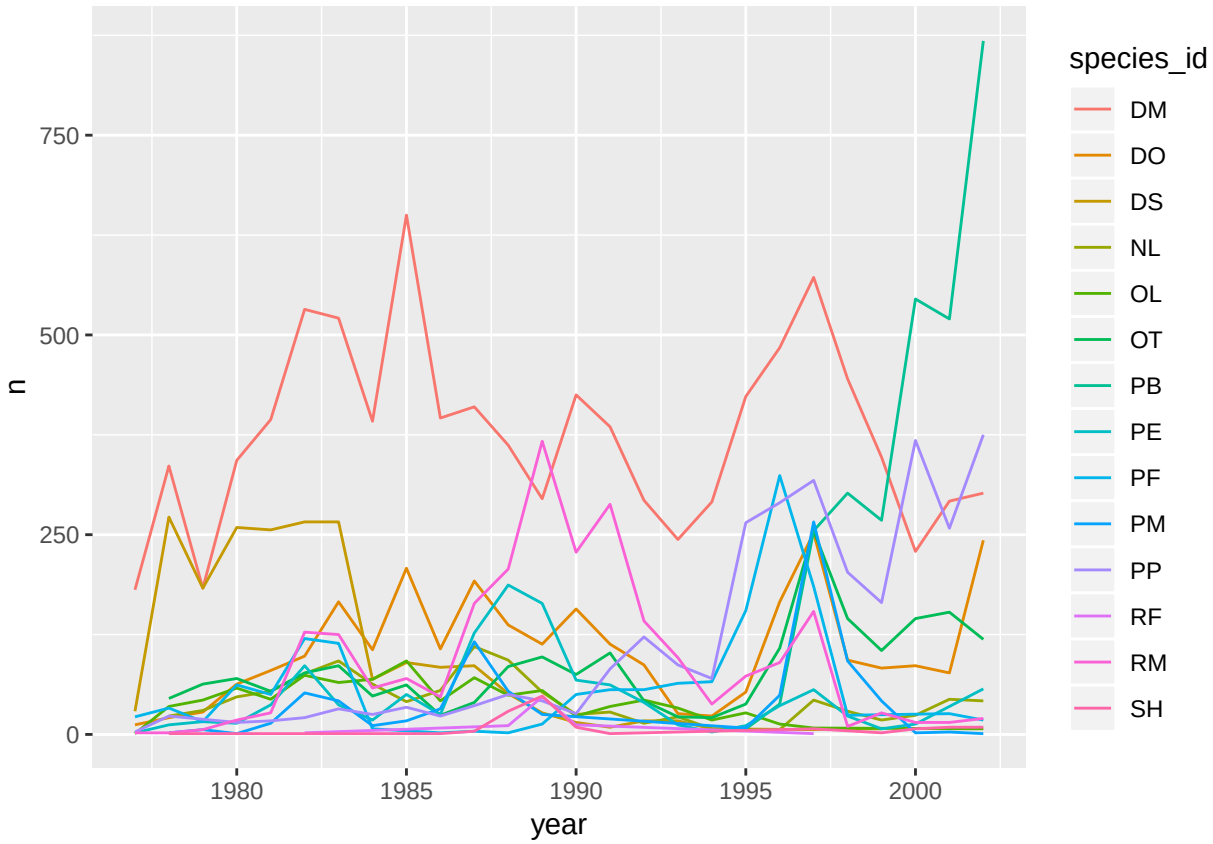
Unfortunately, this does not work because we plotted data for all the species together. We need to tell ggplot to draw a line for each species by modifying the aesthetic function to include group = species_id:

```
ggplot(data = yearly_counts, mapping = aes(x = year, y = n, group = species_id)) +
    geom_line()
```



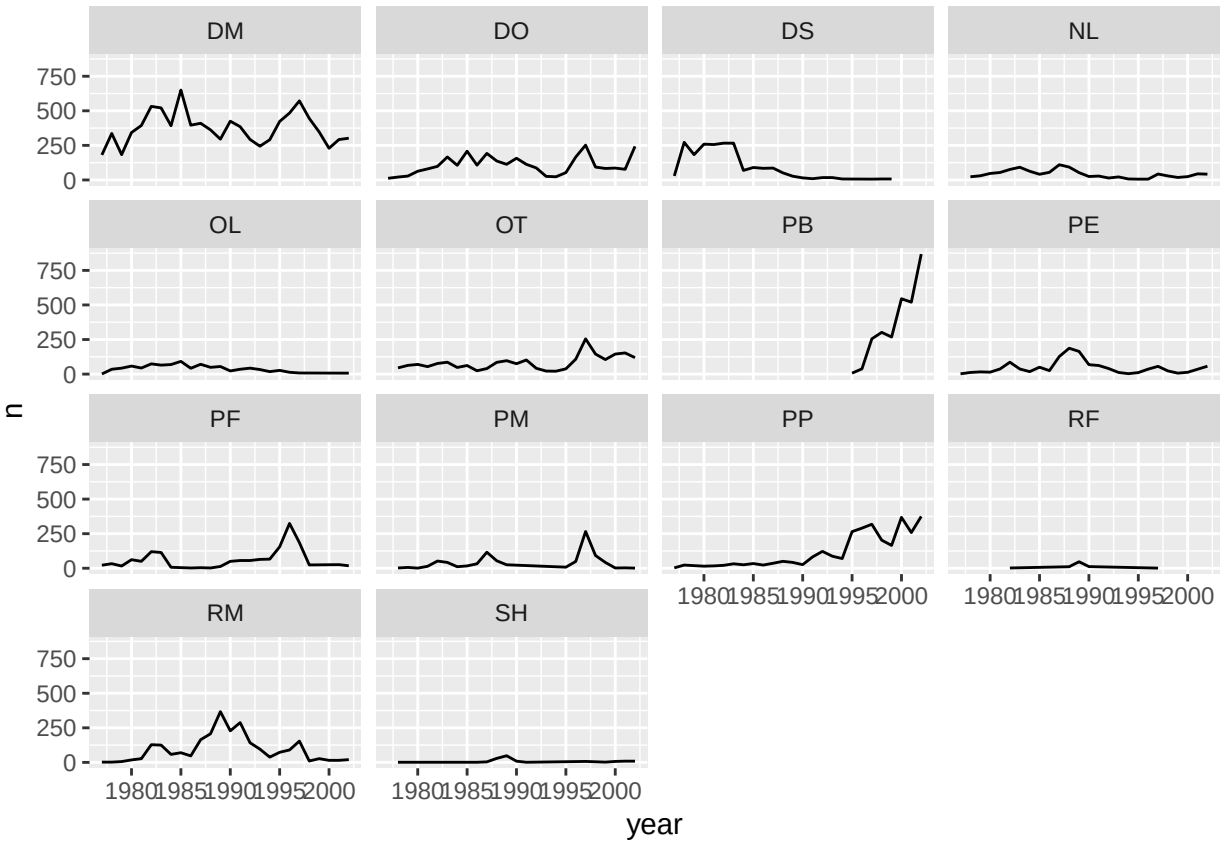
We will be able to distinguish species in the plot if we add colors (using color also automatically groups the data):

```
ggplot(data = yearly_counts, mapping = aes(x = year, y = n, color = species_id)) +
    geom_line()
```



Look for differences between sexes by species over time. First, facet by sepcies:

```
ggplot(data = yearly_counts, mapping = aes(x = year, y = n)) +
    geom_line() +
    facet_wrap(~ species_id)
```

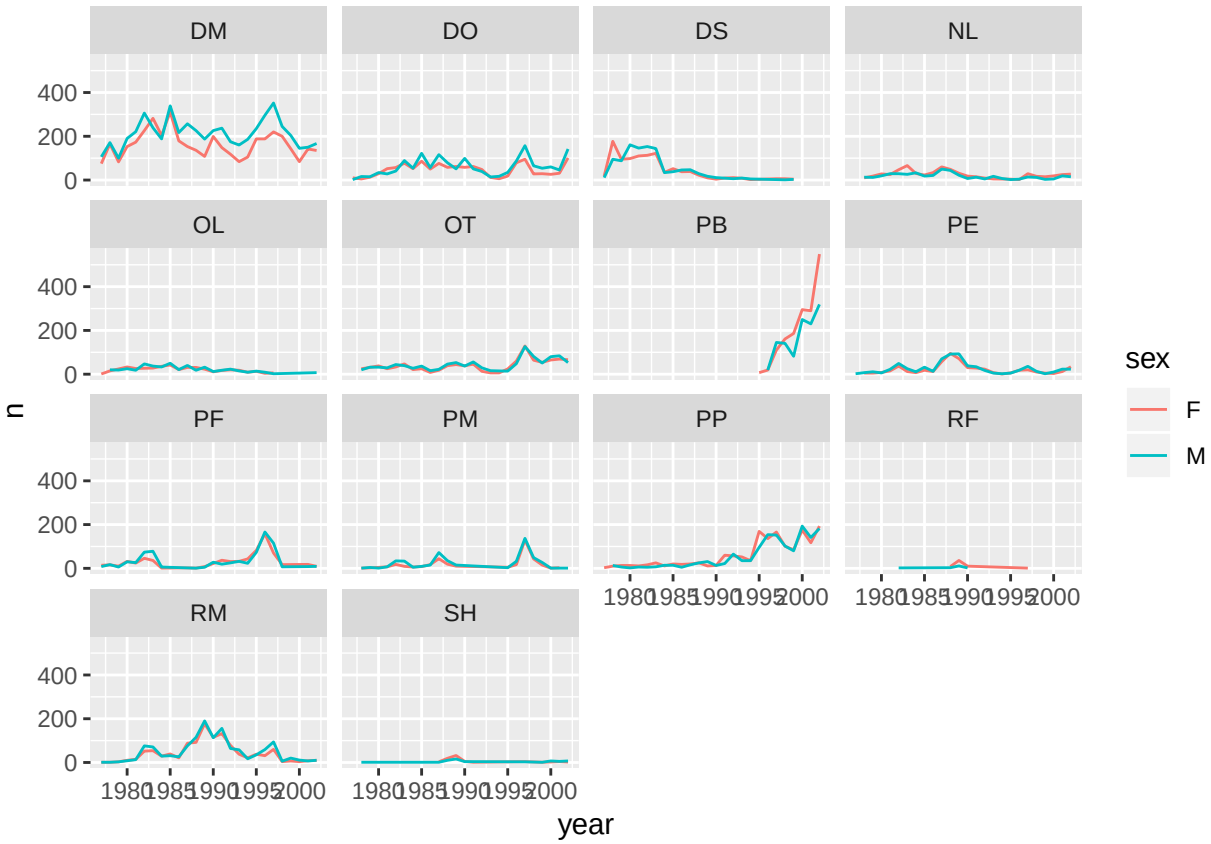


Then split each plot by the sex of each individual measured. To do that we need to make counts in the data frame grouped by year, species_id, and sex:

```
yearly_sex_counts <- surveys_complete %>%
                      count(year, species_id, sex)
```

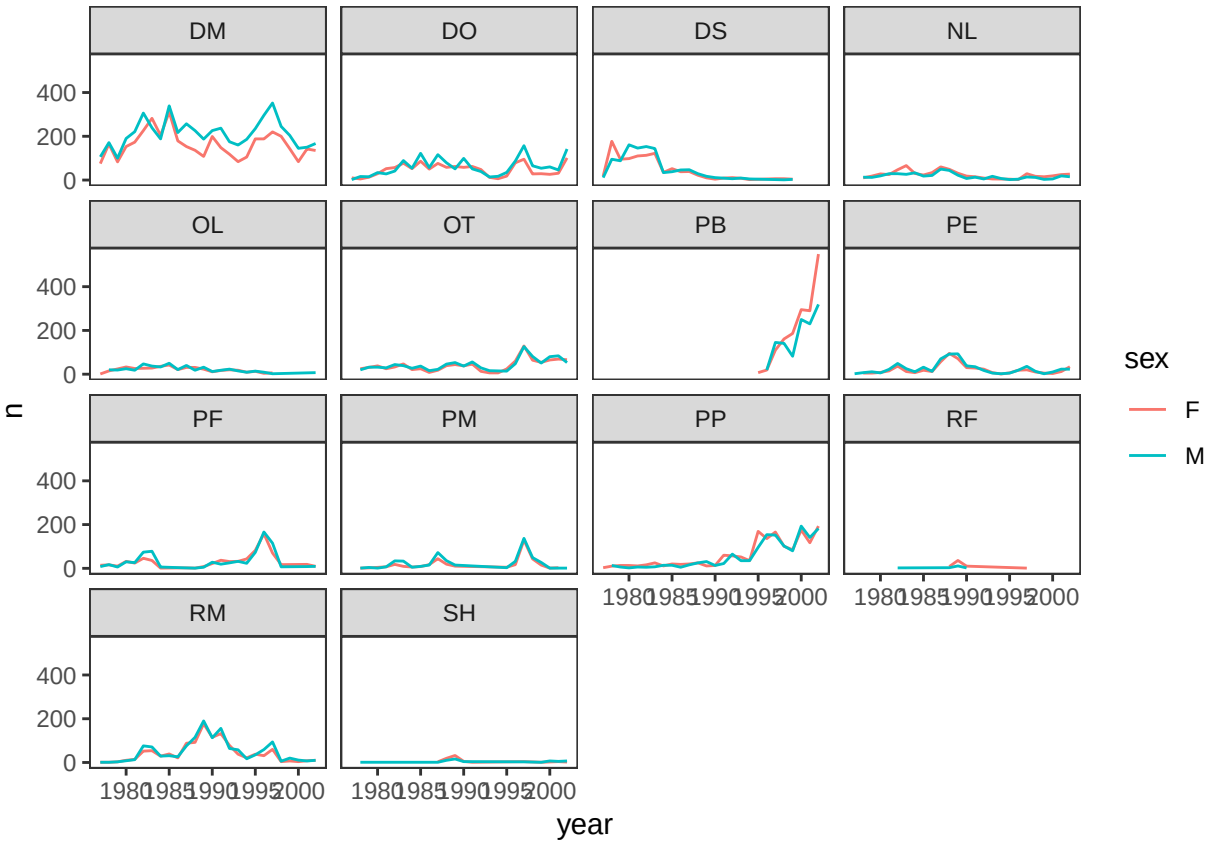
Now make the faceted plot by splitting further by sex using color (within a single plot):

```
ggplot(data = yearly_sex_counts, mapping = aes(x = year, y = n, color = sex)) +
     geom_line() +
     facet_wrap(~ species_id)
```



Let's change the theme to make the background white and remove the gridlines.

```
ggplot(data = yearly_sex_counts, mapping = aes(x = year, y = n, color = sex)) +
    geom_line() +
    facet_wrap(~ species_id) +
    theme_bw() +
    theme(panel.grid = element_blank())
```



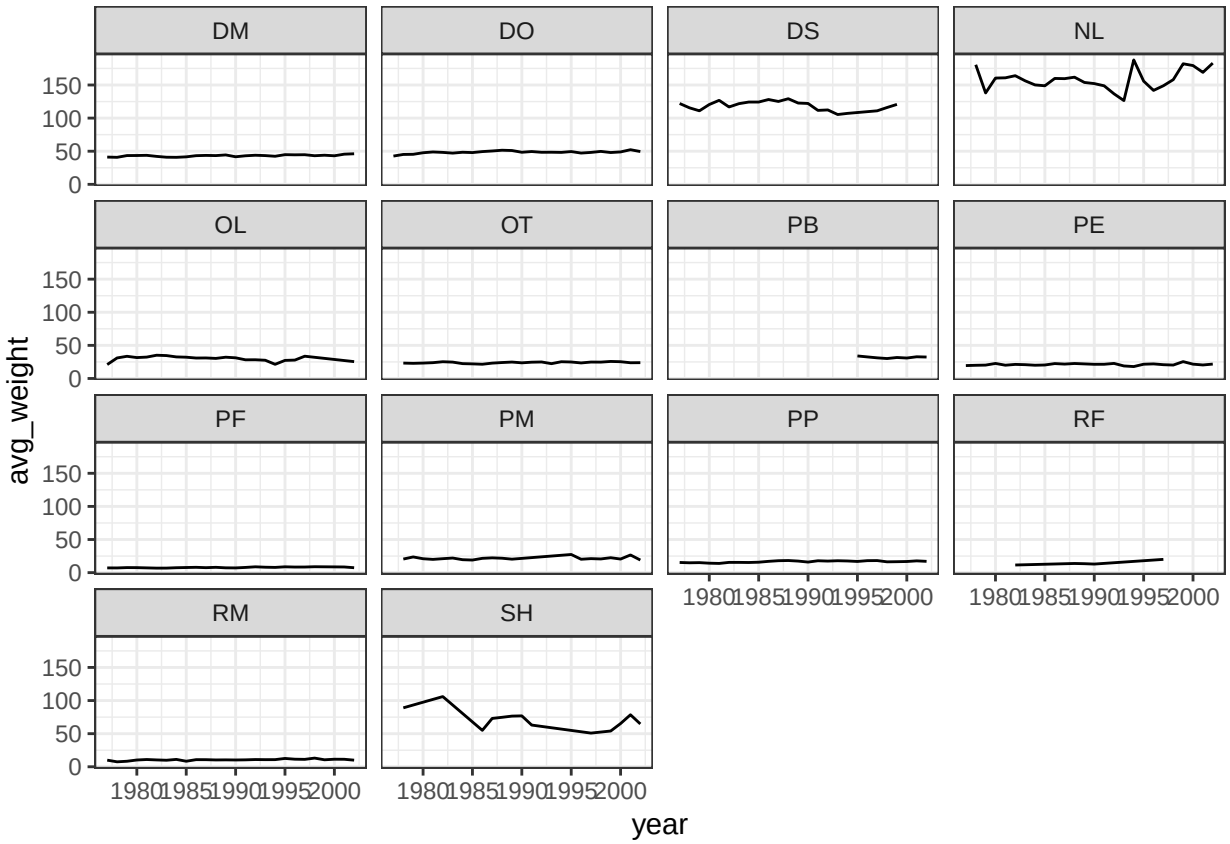
CHALLENGE

Use what you just learned to create a plot that depicts how the average weight of each species changes through the years.

---

use dyplr to do group data and take the average: https://datacarpentry.org/R-genomics/04-dplyr.html

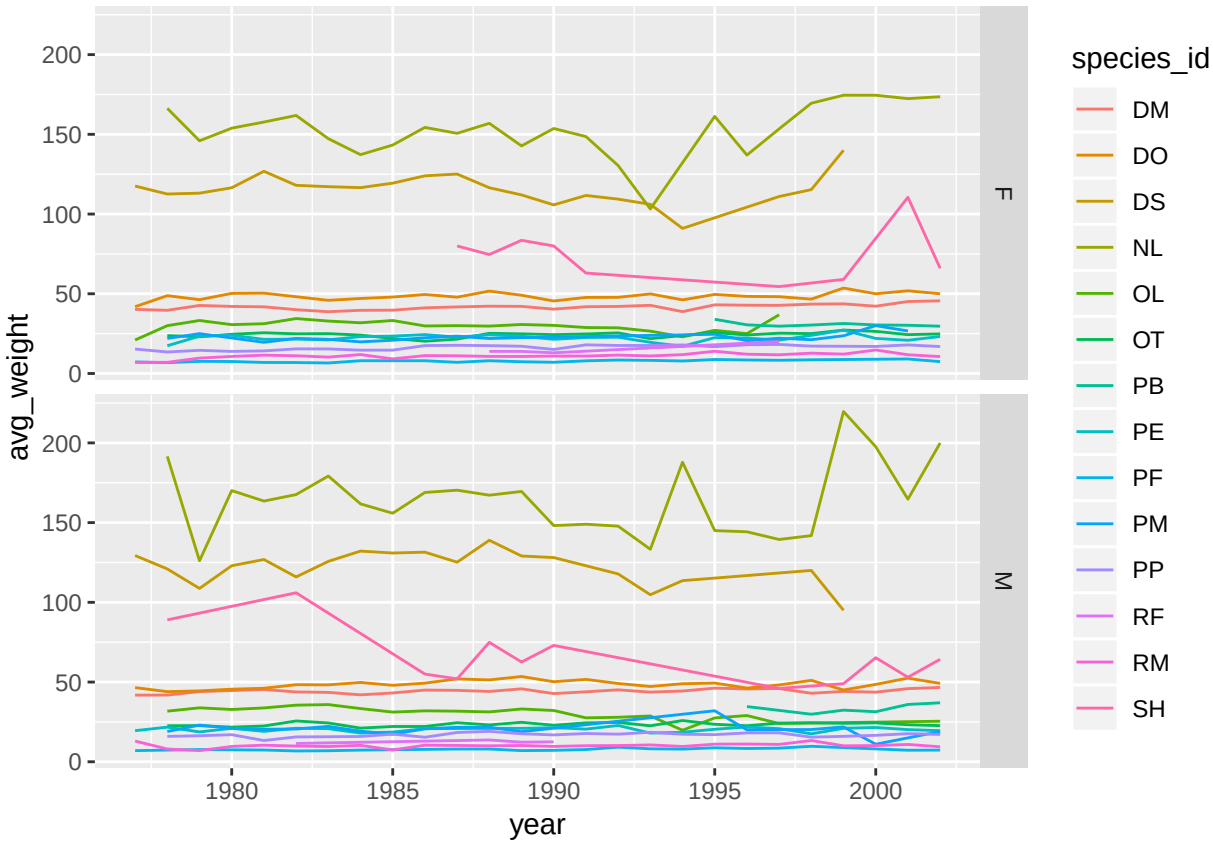
use sql to do the task: sqldf("
SELECT ...
FROM ...
GROUP BY ...
")



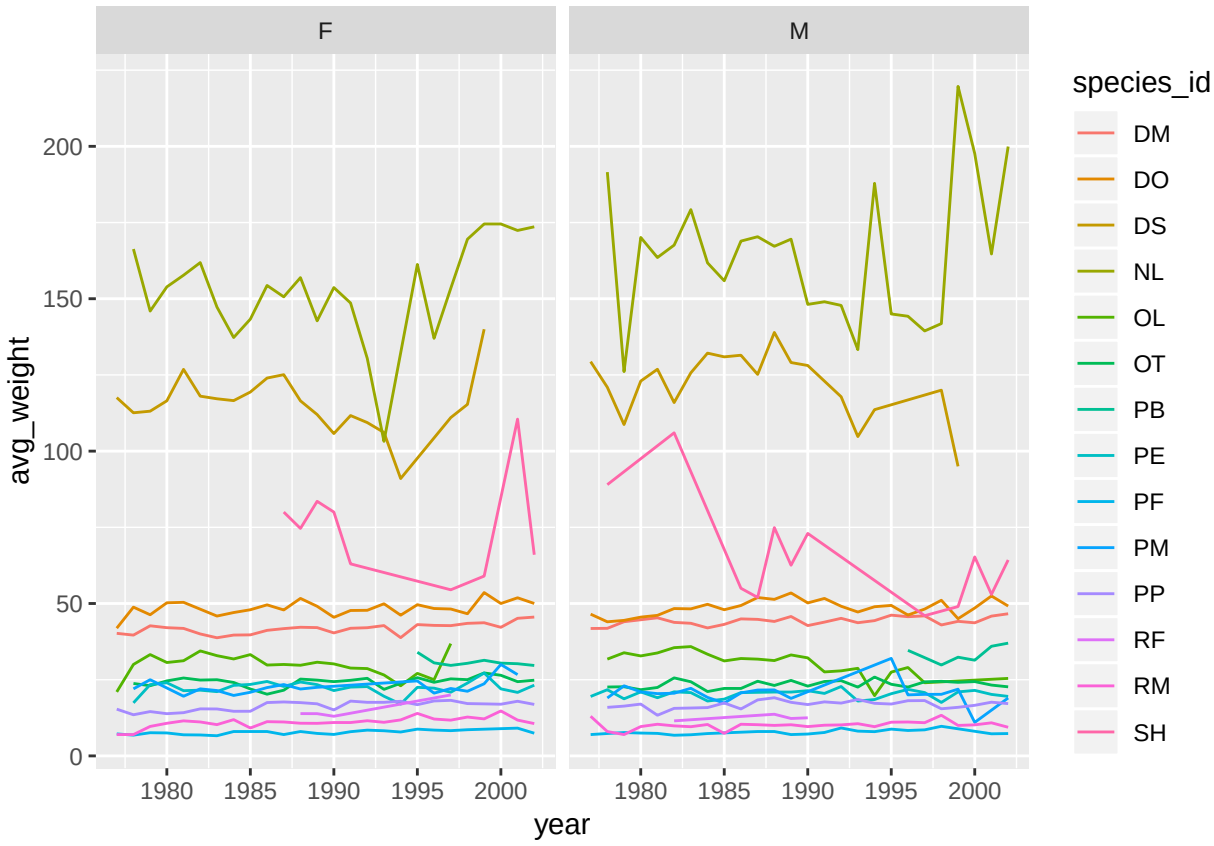
The `facet_wrap` geometry extracts plots into an arbitrary number of dimensions to allow them to cleanly fit on one page. On the other hand, the `facet_grid` geometry allows you to explicitly specify how you want your plots to be arranged via formula notation (`rows ~ columns`; a `.` can be used as a placeholder that indicates only one row or column). manually set how you want your data panel/facet to be displayed

Let's modify the previous plot to compare how the weights of males and females have changed through time:

```
# One column, facet by rows
yearly_sex_weight <- surveys_complete %>%
    group_by(year, sex, species_id) %>%
    summarize(avg_weight = mean(weight))
ggplot(data = yearly_sex_weight,
       mapping = aes(x = year, y = avg_weight, color = species_id)) +
    geom_line() +
    facet_grid(sex ~ .)
```




```
# One row, facet by column
ggplot(data = yearly_sex_weight,
       mapping = aes(x = year, y = avg_weight, color = species_id)) +
    geom_line() +
    facet_grid(. ~ sex)
```



CHALLENG

Pick the plot you think is most informative and improve it: change the axes labels, add a title, change the font size or orientation on the x axis, change the theme.

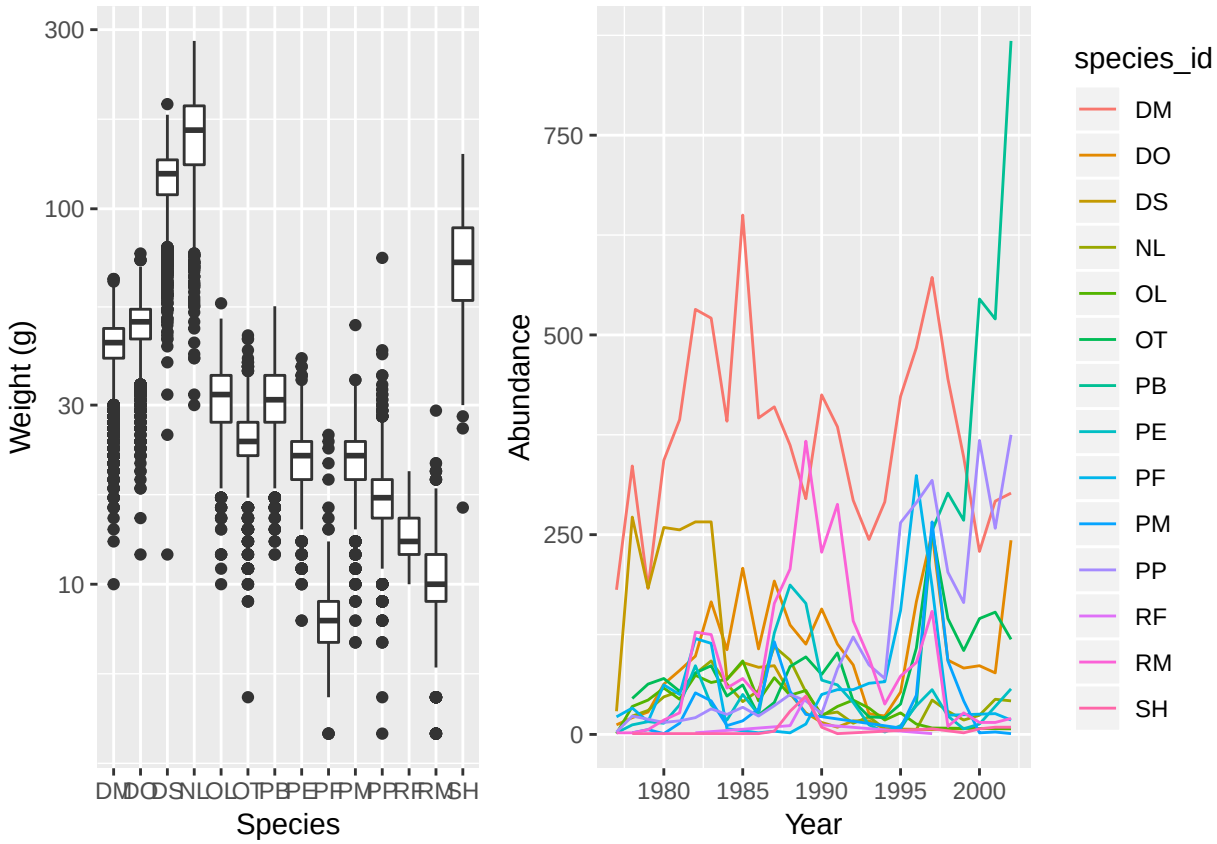
Arranging and exporting plots

The `gridExtra` package allows us to combine separate ggplots into a single figure using `grid.arrange()`:

```
spp_weight_boxplot <- ggplot(data = surveys_complete,
                             mapping = aes(x = species_id, y = weight)) +
  geom_boxplot() +
  xlab("Species") + ylab("Weight (g)") +
  scale_y_log10() question: what is this scale_y_log10 function doing?

spp_count_plot <- ggplot(data = yearly_counts,
                         mapping = aes(x = year, y = n, color = species_id)) +
  geom_line() +
  xlab("Year") + ylab("Abundance")

grid.arrange(spp_weight_boxplot, spp_count_plot, ncol = 2, widths = c(4, 6))
```



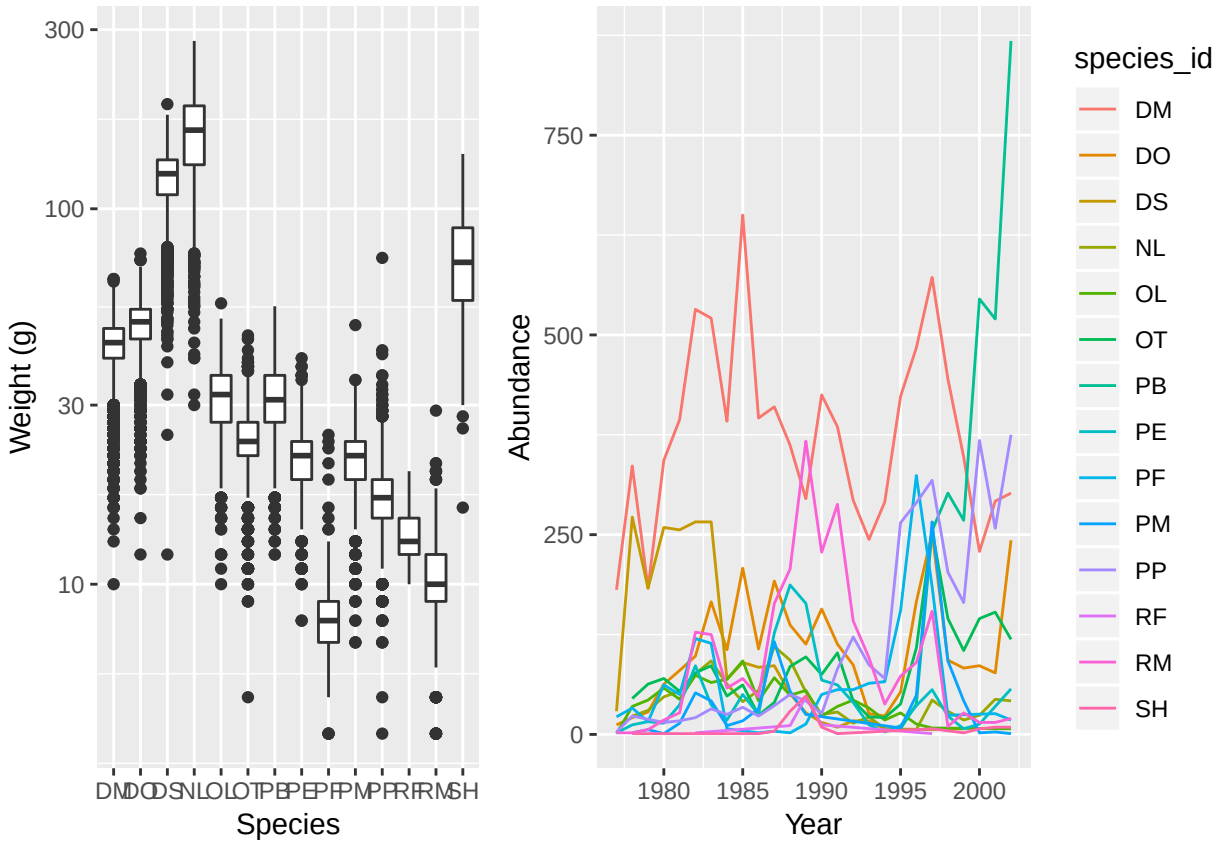
After creating your plot, you can save it to a file in your favorite format. The Export tab in the Plot pane in RStudio will save your plots at low resolution, which will not be accepted by many journals and will not scale well for posters.

Instead, use the `ggsave()` function, which allows you easily change the dimension and resolution of your plot by adjusting the appropriate arguments (width, height and dpi).

Make sure you have the `fig_output/` folder in your working directory.

```
my_plot <- ggplot(data = yearly_sex_counts,
                  mapping = aes(x = year, y = n, color = sex)) +
    geom_line() +
    facet_wrap(~ species_id) +
    labs(title = "Observed species in time",
         x = "Year of observation",
         y = "Number of individuals") +
    theme_bw() +
    theme(axis.text.x = element_text(colour = "grey20", size = 12, angle = 90, hjust = 0.5, vjust = 0.5
                        axis.text.y = element_text(colour = "grey20", size = 12),
          text=element_text(size = 16))
ggsave("fig_output/yearly_sex_counts.png", my_plot, width = 15, height = 10)

# This also works for grid.arrange() plots
combo_plot <- grid.arrange(spp_weight_boxplot, spp_count_plot, ncol = 2, widths = c(4, 6))
```




```
ggsave("fig_output/combo_plot_abun_weight.png", combo_plot, width = 10, dpi = 300)
```

```
## Saving 10 x 4.5 in image
```