Manipulatind and analysing data with dplyr

Data analysis and visualization in R
UC Merced

Andrea Sánchez-Tapia
Rio de Janeiro Botanical Garden - ¡liibre! - RLadies+

2021-03-23

today

- dplyr and the "tidyverse"
- the pipe operator
- tidy data, pivoting data, joining tables
- group and summarize
- data visualization with ggplot2

manipulating and analyzing data with dplyr

the tidyverse: an "umbrella" package

- **ggplot2**: a "grammar of graphics" by Hadley Wickham. Divide the data and the aesthetics. Create and modify the plots layer by layer
- **dplyr**: a way to deal with data frames, sql external data bases, written in `C++`
- **readr**: read data
- **tidyr**: format data frames
- **stringr**: deals with strings
- **tibble**: a new data structure
- other packages: **lubridate** (dates), **forcats** (factors), and many more

Most of R is still **base**-based and both philosophies communicate well with each other

reading data with readr

```r
library(dplyr)
library(readr)
```

```r
surveys <- readr::read_csv("data/raw/portal_data_joined.csv")
```

```
##
## ── 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(),
```

the tibble

```r
surveys
vignette("tibble") #show what a vignette is
```

- modified data frames
- do not change input type (characters into factors)
- do not change the name of the columns and allows for non-standard names, such as `1999` and `total count` (it will require back ticks)
- no rownames
- subsetting always returns a tibble

some principal functions in dplyr

- **select** (columns)
- **filter** (rows)
- **rename** (columns)
- **mutate** (create new columns or modify existing columns)
- **arrange** to sort according to a column
- **count** cases of one or many columns

select columns

```r
select(surveys, plot_id, species_id, weight)
```

1. there is no need to put quotes
2. there is no need to put variables between `c()`

base R still works in a tibble

```r
surveys[, c("plot_id", "species_id", "weight")]
```

removing columns

```
select(surveys, -record_id, -species_id)
```

additional functions

```
select(surveys, -ends_with("id"))
```

`starts_with`, `contains`, `all_of`, `last_col`

filter rows

logical clauses!

```
surv_1995 <- filter(surveys, year == 1995)
```

No need to use $ or brackets

```
surveys$year == 1995
surveys[surveys$year == 1995 , ]
```

mutate creates or modifies columns

```
surveys <- mutate(surveys, weight_kg = weight / 1000)
```

```
mutate(surveys,
       weight_kg = weight / 1000,
       weight_lb = weight_kg * 2.2)
```

group_by() and summarise()

- if you have a column factor (e.g. sex) and want to apply a function to the levels of this factor

```
surveys_g <-group_by(surveys, sex) #does nothing?
summary_sex <- summarize(surveys_g,
                         mean_weight = mean(weight, na.rm = TRUE))
summary_sex
```

```
## # A tibble: 3 x 2
##   sex   mean_weight
##   <chr>       <dbl>
## 1 F            42.2
## 2 M            43.0
## 3 <NA>         64.7
```

another example:

```
surveys_g2 <-group_by(surveys, sex, species_id)
mean_w <- summarize(surveys_g2,
                    mean_weight = mean(weight, na.rm = TRUE))
```

```r
mean_w
```

```
## # A tibble: 92 x 3
## # Groups:   sex [3]
##    sex   species_id mean_weight
##    <chr> <chr>            <dbl>
##  1 F     BA                9.16
##  2 F     DM               41.6
##  3 F     DO               48.5
##  4 F     DS              118.
##  5 F     NL              154.
##  6 F     OL               31.1
##  7 F     OT               24.8
##  8 F     OX               21
##  9 F     PB               30.2
## 10 F     PE               22.8
## # … with 82 more rows
```

arrange sorts by a column

```
arrange(mean_w, mean_weight)


arrange(mean_w, desc(mean_weight))
```

the pipe operator %>%

Classic syntax goes like this

```
object1
object2 <- function1(object1)
object3 <- function2(object2)
```

...or you can nest functions and avoid create intermediary objects

```
object3 <- function2(function1(object1))
```

The pipe operator allows to apply functions sequentially:

```
object3 <- object1 %>% function1() %>% function2()
```

select and filter

```
surveys2 <- filter(surveys, weight < 5)
surveys_sml <- select(surveys2, species_id, sex, weight)

surveys %>%
  filter(weight < 5) %>%
  select(species_id, sex, weight)
```

- you can append `head()` or `View()`

group_by() and summarize()

```r
surveys_g   <- group_by(surveys, sex) #does nothing?
summary_sex <- summarize(surveys_g,
                         mean_weight = mean(weight, na.rm = TRUE))


summary_sex <- surveys %>%
  group_by(sex) %>%
  summarize(mean_weight = mean(weight, na.rm = TRUE))
```

count

```
surveys %>%
    count(sex)

surveys %>%
  count(sex, species)

surveys %>%
  count(sex, species) %>%
  arrange(species, desc(n))
```

challenge

- How many animals were caught in each `plot_type` surveyed?
- Use `group_by()` and `summarize()` to find the mean, min, and max hindfoot length for each species (using `species_id`). Also add the number of observations (hint: see `?n`).

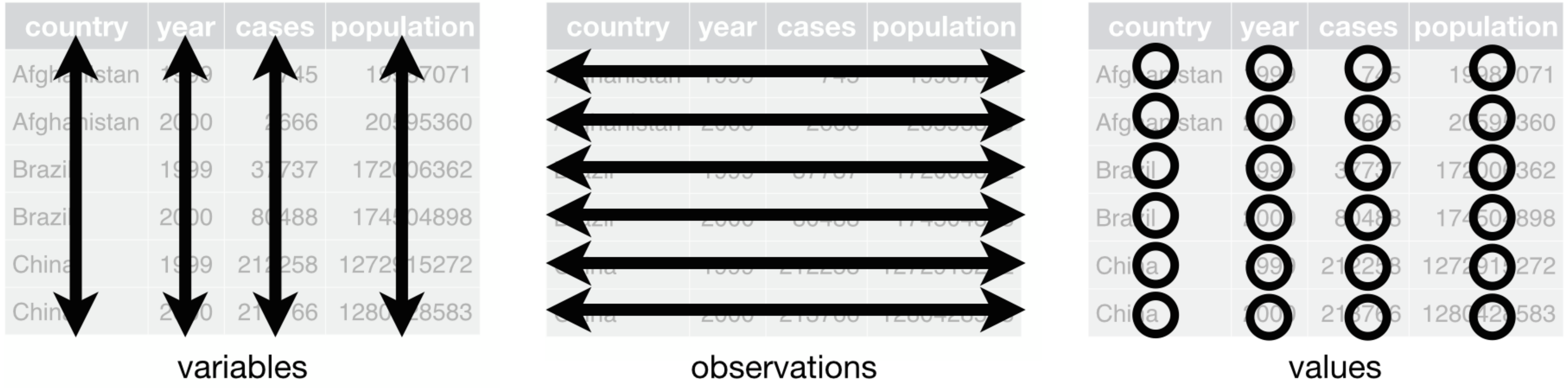
save data!

```r
surveys <- readr::read_csv("data/raw/portal_data_joined.csv")
```

```
##
## ── 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(),
```

tidy data as a philosohpy

- datasets should be organized as **observations in rows** and **variables in columns** -> "tidy data"



- what is an observation? wht is the sampling unit?

some examples

```r
library(tidyr)
table1
table2
table3
table4a #cases
table4b #population
```

pivot_longer() and pivot_wider()

```
library(tidyr)
table4a
table4a %>%
  pivot_longer(cols = c(`1999`, `2000`), names_to = "year", values_to =

table4b %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "popul
```

pivot_wider()

```
table2 %>%
    pivot_wider(names_from = type, values_from = count)
```

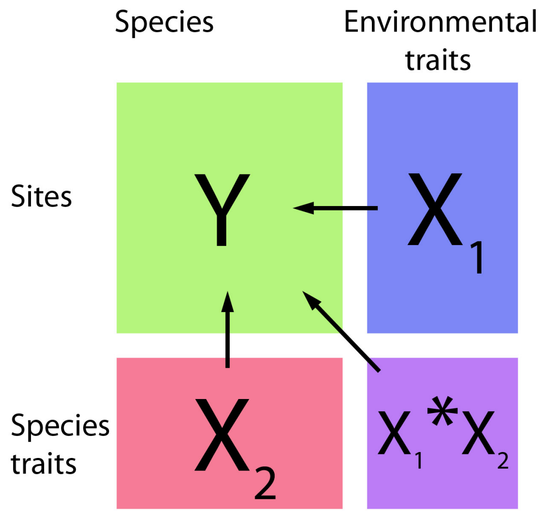
other options for tidying data

```r
table3
table3 %>%
  separate(rate, into = c("cases", "population"))
table3 %>%
  separate(rate, into = c("cases", "population"), convert = TRUE)

table5 %>%
  unite(new, century, year)
```

working with several tables: merges and joins

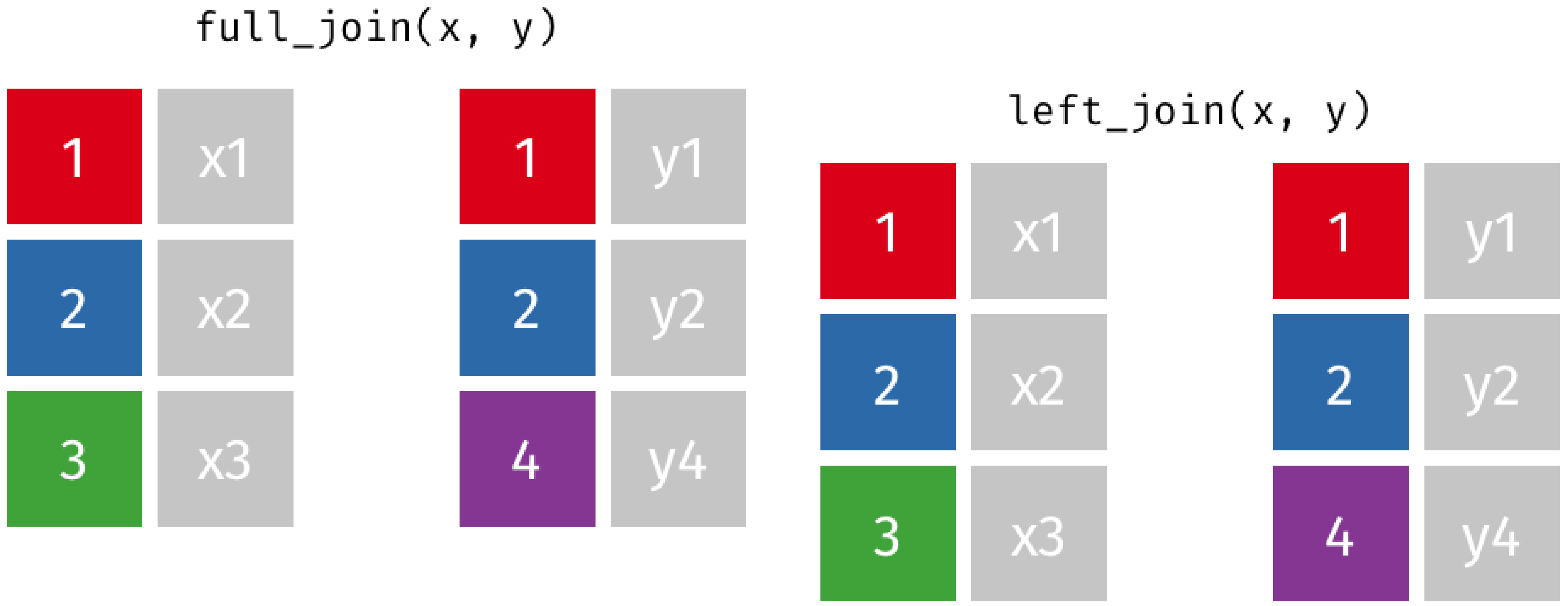
- in real analysis settings you will have many tables that are related
- relational datasets/databases
- in ecology for example:
  - sites x species
  - sites x environmental conditions
  - species x characteristics
  - individuals x individual measurement



working with several tables

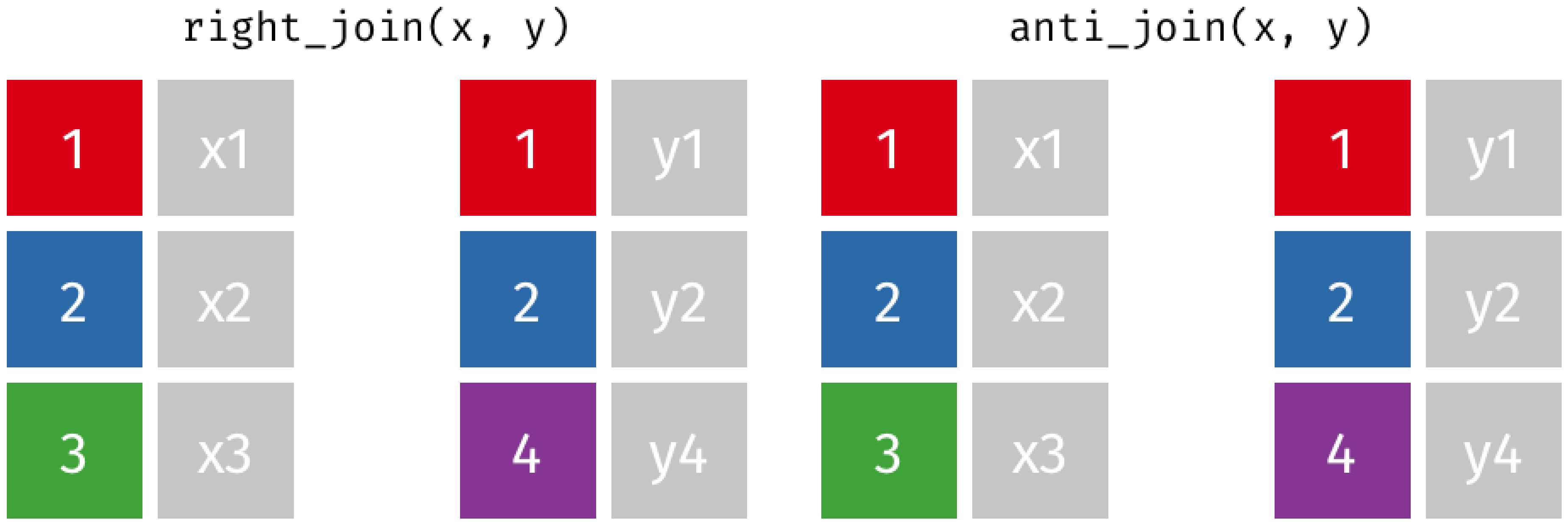
- keep the data as simple as possible, even if that means having different tables
- for each data table have in mind the **sampling unit**. is it the species, is it the plot? is it the individual, the city?
- have a **unique identifier for each observation** so you can merge the data

working with several tables


```
full_join(x, y)
```
```
left_join(x, y)
```

gadenbuie/tidyexplain

working with several tables



gadenbuie/tidyexplain

a short example

```r
tidy4a <- table4a %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "cases
tidy4b <- table4b %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "popul

left_join(tidy4a, tidy4b)
```

```
## Joining, by = c("country", "year")

## # A tibble: 6 x 4
##   country     year  cases population
##   <chr>       <chr> <int>      <int>
## 1 Afghanistan 1999    745   19987071
## 2 Afghanistan 2000   2666   20595360
## 3 Brazil      1999  37737  172006362
## 4 Brazil      2000  80488  174504898
```

our survey data was created like that

```
download.file("https://ndownloader.figshare.com/files/3299483",
              "./data/raw/species.csv")
download.file("https://ndownloader.figshare.com/files/10717177",
              "./data/raw/surveys.csv")
download.file("https://ndownloader.figshare.com/files/3299474",
              "./data/raw/plots.csv")
```

our survey data was created like that

```r
library(readr)
library(dplyr)
species <- read_csv("./data/raw/species.csv")
head(species)
surveys <- read_csv("./data/raw/surveys.csv")
head(surveys)
plots <- read_csv("./data/raw/plots.csv")
head(plots)
left_join(surveys, species) %>% left_join(plots) %>% dim()
```

data visualization with ggplot2

ggplot2

- **ggplot2** separates the data from the aesthetics part and allows layers of information to be added sequentially with `+`

```
ggplot(data = <data>,
       mapping = aes(<mappings>)) +
  geom_xxx()
```

- **data**
- **mappings**: the specific variables (x, y, z, group...)
- **geom_xxx()**: functions for plotting options `geom_point()`, `geom_line()`

cheat sheet link

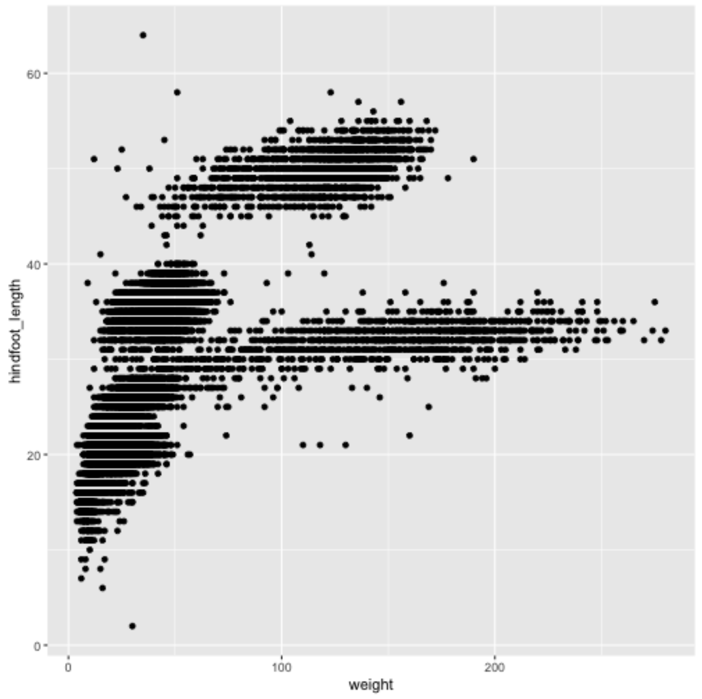
ggplot2 plots are built sequentially in layers

```r
library(ggplot2)
library(readr)

surveys_complete <- read_csv("data/processed/surveys_mod_dplyr.csv")
```

ggplot2 plots are built sequentially in layers

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

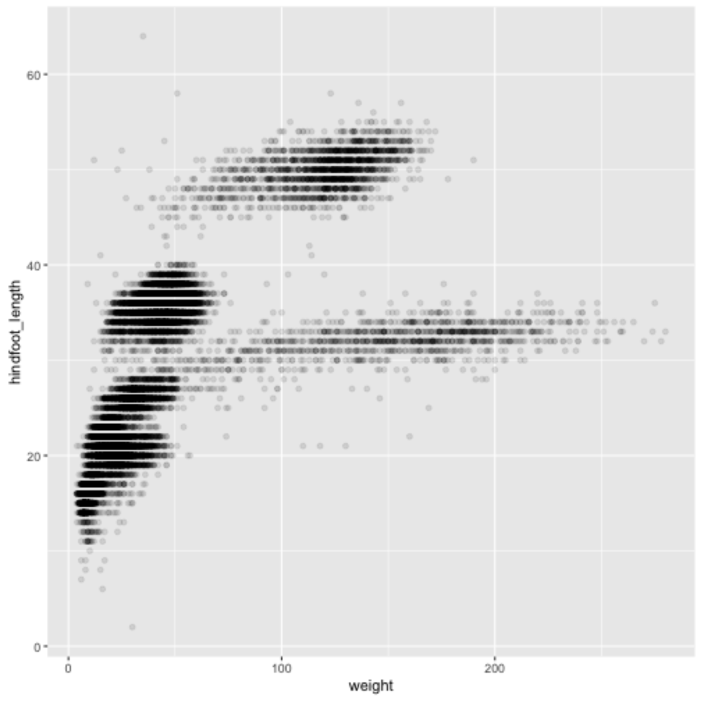


you can assign a plot to an object and build on it

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

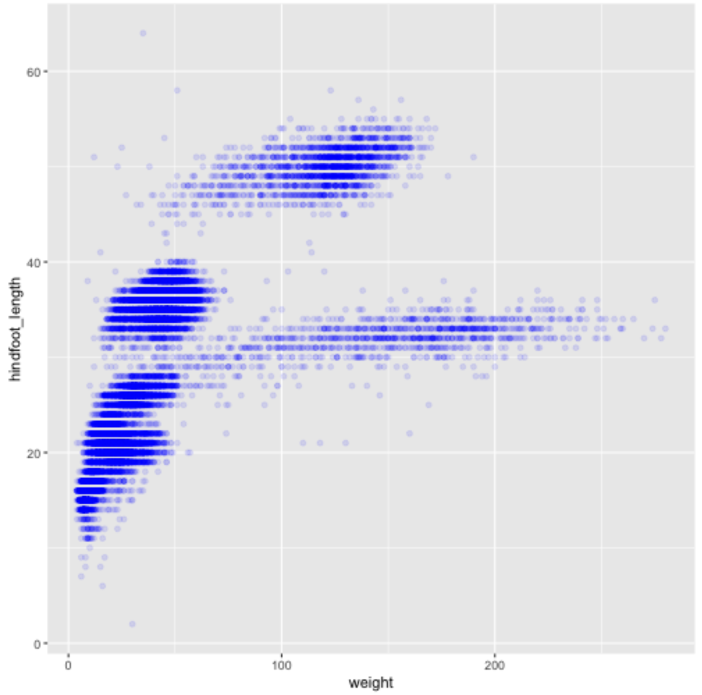
ggplot2 plots are built sequentially in layers

```
weight_hind +
    geom_point(alpha = 0.1) #transparency
```



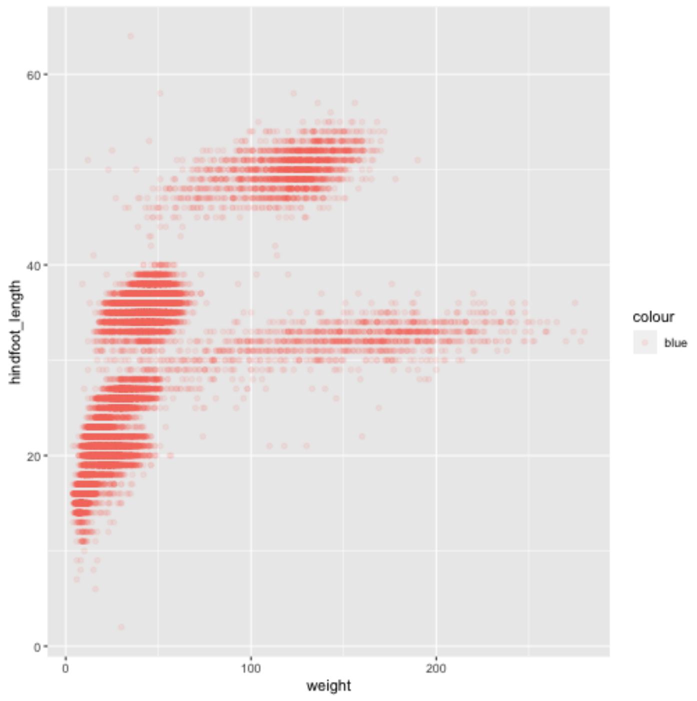
ggplot2 plots are built sequentially in layers

```r
weight_hind +
    geom_point(alpha = 0.1, color = "blue") #color
```



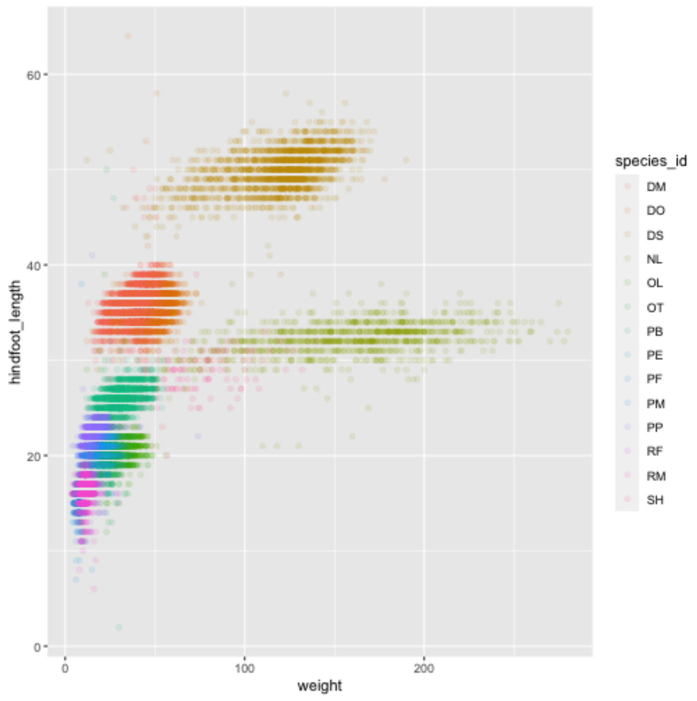
ggplot2 plots are built sequentially in layers

```
weight_hind +
    geom_point(alpha = 0.1, aes(color = "blue")) #this is a mistake!
```



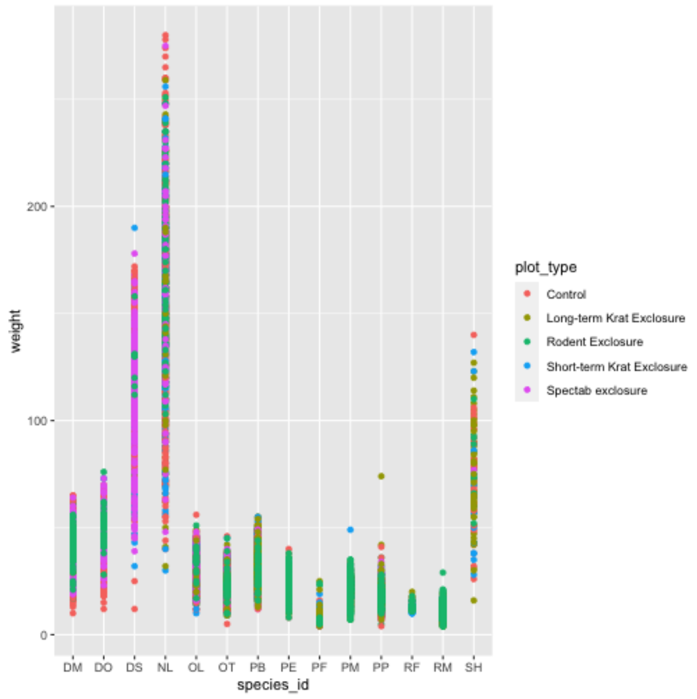
ggplot2 plots are built sequentially in layers

```r
weight_hind +
    geom_point(alpha = 0.1, aes(color = species_id))
```



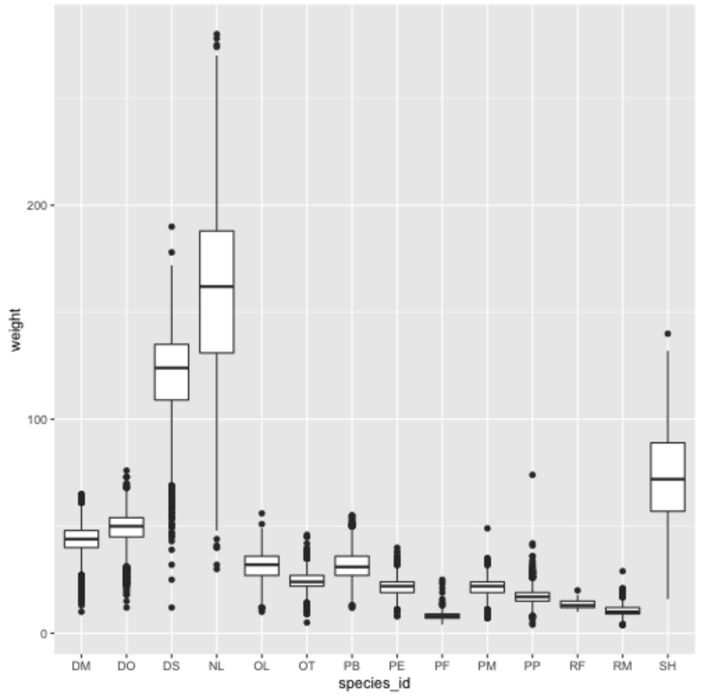
challenge: change x to categorial variable

```
ggplot(data = surveys_complete,
       mapping = aes(x = species_id, y = weight)) +
   geom_point(aes(color = plot_type))
```



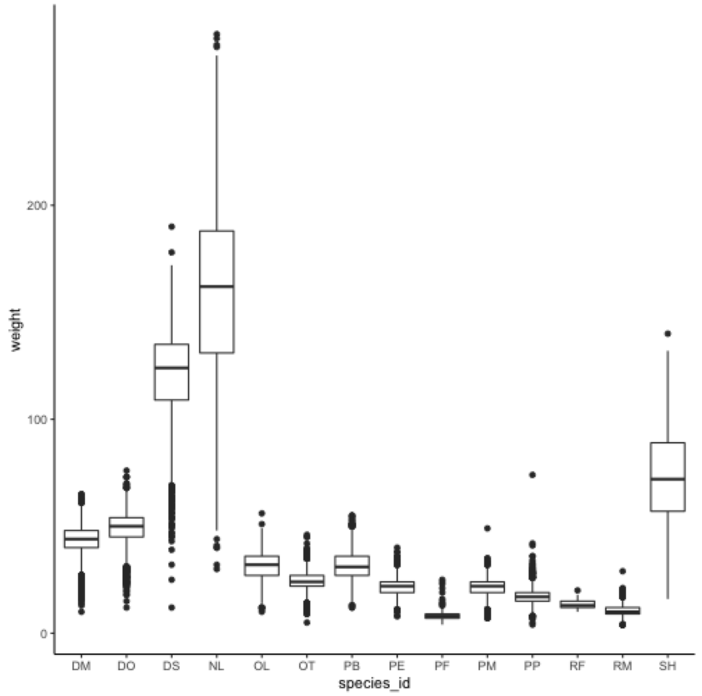
boxplots!

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



theme options theme_

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



add jitter layer

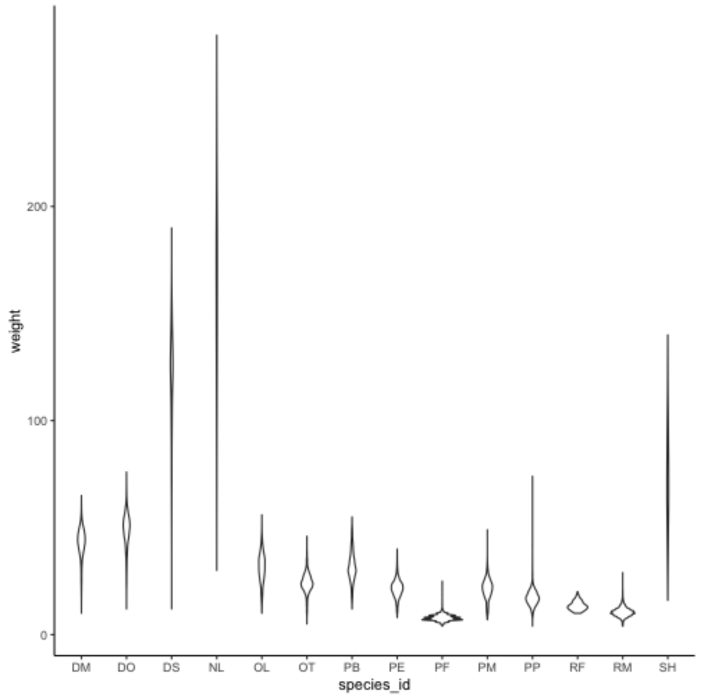
```
ggplot(data = surveys_complete,
       mapping = aes(x = species_id, y = weight)) +
   geom_boxplot() +
  geom_jitter(alpha = 0.3, color = "dodgerblue", width = 0.2) +
  theme_classic()
```

change plot order

```
ggplot(data = surveys_complete,
       mapping = aes(x = species_id, y = weight)) +
  geom_jitter(alpha = 0.3, color = "dodgerblue", width = 0.2) +
  geom_boxplot() +
  theme_classic()
```

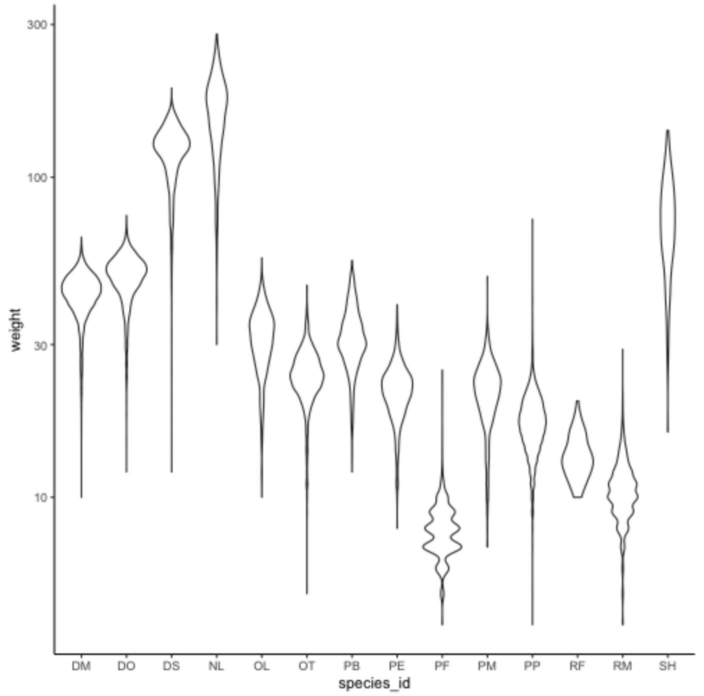
violin plots

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



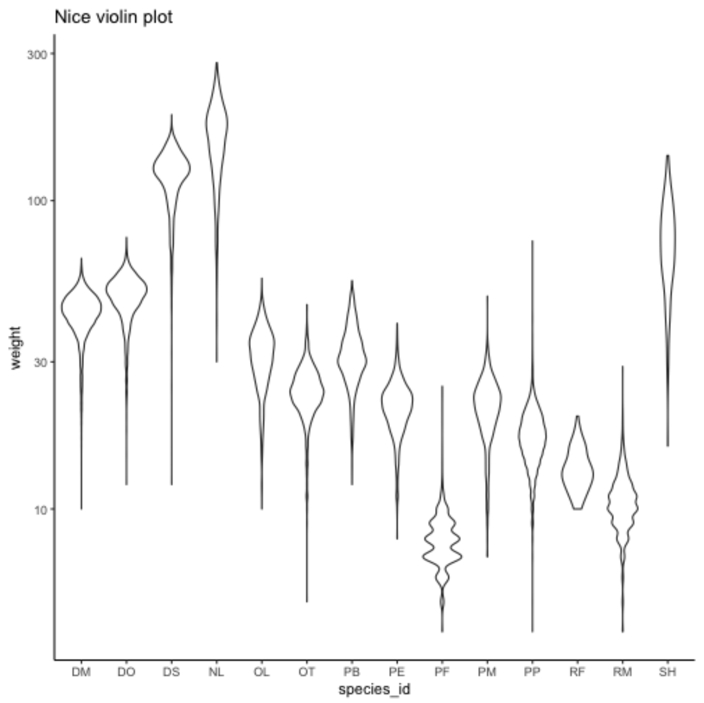
change scale (scale_xx options)

```
p <- ggplot(data = surveys_complete,
       mapping = aes(x = species_id, y = weight)) +
   geom_violin() + scale_y_log10() + theme_classic() #nice!
p
```



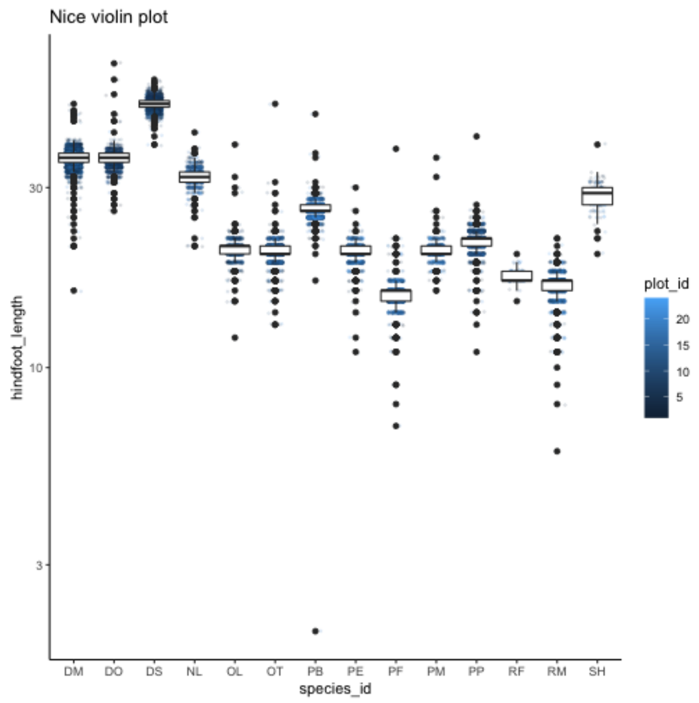
add title `ggtitle()`

```
p +  #remember the plot can be an object
  ggtitle("Nice violin plot")
```



ggplot2 plots are built sequentially in layers

```
ggplot(data = surveys_complete,
       mapping = aes(x = species_id, y = hindfoot_length)) +
   geom_jitter(size = 0.5, alpha = 0.1, width = 0.2, aes(col = plot_id)
   geom_boxplot() + scale_y_log10() + theme_classic() + ggtitle("Nice v
```



what would be next?

- learn to write **functions** for your own workflow and other programming tools to **iterate** these functions accross many inputs (loops and the **purrr** package)

- study R-specific literature such as R4DS and Advanced R

- study specific packages in your area, read their vignettes and documentation, get acquainted with the workflows

- learn tools for **communicating** your results (text, presentations, dashboards): markdown & R markdown, **xaringan** (presentations)

- learn about version control (**git**) to backup and control changes for your projects

references

- R for data science
- Reproducible workflows
- https://github.com/gadenbuie/tidyexplain