R for data science

with tidyverse and ggplot2

Karolina Sienkiewicz
for NGSeminars

September 3, 2020

Contents

I. Intro to tidyverse

1. Why use tidyverse?

- curated *collection* of packages for data science
- packages share data classes and grammar
- low entry threshold
- database-like approach
- **aes**thetically pleasing visualizations
- big community and great resources online

Let's import tidyverse packages:

```
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------------
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

The output information provides as with a useful insight into name conflicts. The more packages you have attached, the more likely some of conflicts appear.

You can also suppress this message by running:

```
suppressPackageStartupMessages(library(tidyverse))
```

2. Pipe operator %>%

It allows as to chain operations together for convenience and better readability.

```
as.character(round(mean(seq(100))))
```

```
## [1] "50"
```

```
seq(100) %>%
  mean() %>%
  round() %>%
  as.character()
```

```
## [1] "50"
```

3. *tibbles*

Basic data object in tidyverse. It's basically an improved/modernized data.frame.

```
tibble(numbers=c(1,2,3),names=c('a','b','c'))
```

```
## # A tibble: 3 x 2
##   numbers names
##     <dbl> <chr>
## 1       1 a
## 2       2 b
## 3       3 c
```

```
tb <- .Last.value #let's save our tibble on variable
```

How do we make conversions between data.frames and tibbles?

```
# convert tibble to data frame
df <- as.data.frame(tb)
# add rownames
row.names(df) <- c("row1", "row2", "row3")
df
```

```
##      numbers names
## row1       1     a
## row2       2     b
## row3       3     c
```

```
# convert data.frame to tibble
as_tibble(df)
```

```
## # A tibble: 3 x 2
##   numbers names
##     <dbl> <chr>
## 1       1 a
## 2       2 b
## 3       3 c
```

```
as_tibble(df, rownames = "row_names")
```

```
## # A tibble: 3 x 3
##   row_names numbers names
##   <chr>       <dbl> <chr>
## 1 row1            1 a
## 2 row2            2 b
## 3 row3            3 c
```

How to add new rows to the tibble?

```
#adding new rows
add_row(tb, numbers=4, names="d")
```

```
## # A tibble: 4 x 2
##   numbers names
##     <dbl> <chr>
## 1       1 a
## 2       2 b
## 3       3 c
## 4       4 d
```

```
tb
```

```
## # A tibble: 3 x 2
##   numbers names
##     <dbl> <chr>
## 1       1 a
## 2       2 b
## 3       3 c
```

```
tb <- add_row(tb, numbers=4, names="d")
tb
```

```
## # A tibble: 4 x 2
##   numbers names
##     <dbl> <chr>
## 1       1 a
## 2       2 b
## 3       3 c
## 4       4 d
```

How to add new column to the tibble?

```
# we can treat the tibble as a data.frame
tb$squres <- tb$numbers^2
tb
```

```
## # A tibble: 4 x 3
##   numbers names squres
##     <dbl> <chr>  <dbl>
## 1       1 a          1
## 2       2 b          4
## 3       3 c          9
## 4       4 d         16
```

```
# or we can use tidyverse 'mutate' operation
tb %>% mutate(cubes = squres*numbers)
```

```
## # A tibble: 4 x 4
##   numbers names squres cubes
##     <dbl> <chr>  <dbl> <dbl>
## 1       1 a          1     1
## 2       2 b          4     8
## 3       3 c          9    27
## 4       4 d         16    64
```

We will talk more about 'mutate' and other tidyverse operation soon.

How to save and read local data file?

```
# saving tibble to file
write_tsv(tb, path = "new_tibble.tsv")

# loading local data
countries <- read_tsv('countries.tsv')
```

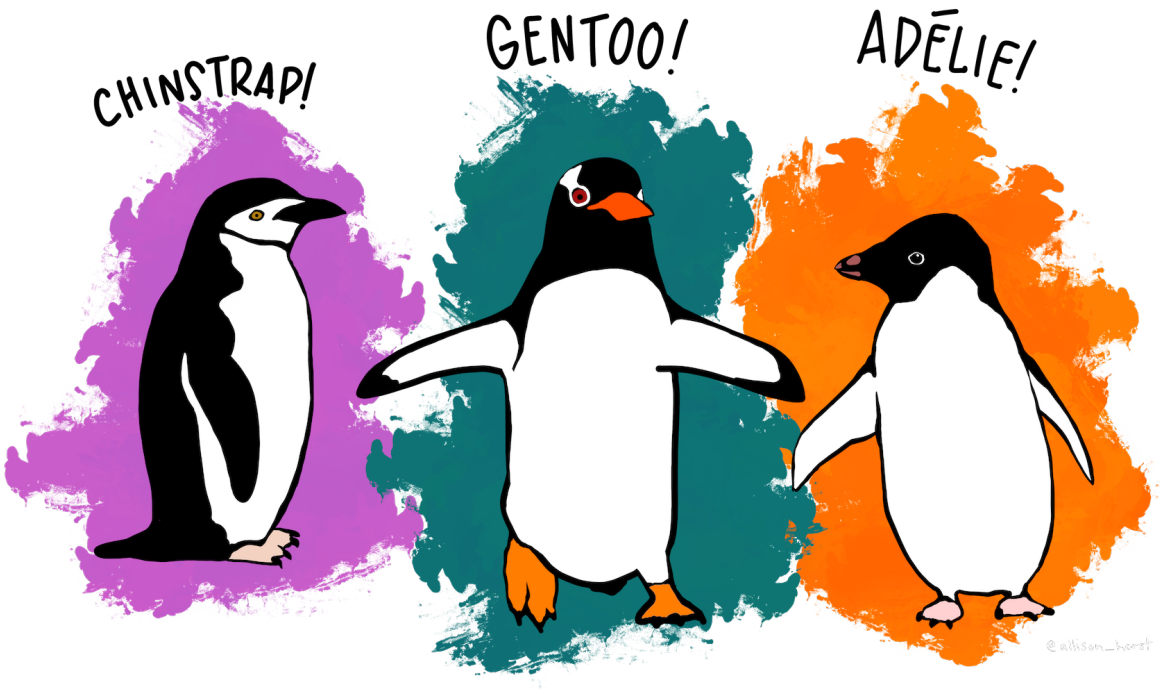
```
## Parsed with column specification:
## cols(
##   year = col_double(),
##   sex = col_character(),
##   Australia = col_double(),
##   Germany = col_double(),
##   Poland = col_double(),
##   USA = col_double()
## )
```

```
# explicitly specyfying column types in loaded data
countries <- read_tsv('countries.tsv', col_types=c(col_double(), col_character()))
# accepting default inferred coulumn types
countries <- read_tsv('countries.tsv', col_types=cols())
countries
```

```
## # A tibble: 6 x 6
##    year sex    Australia  Germany   Poland       USA
##   <dbl> <chr>      <dbl>    <dbl>    <dbl>     <dbl>
## 1  1995 Female   9063508 41930010 19808312 134441472
## 2  1995 Male     8990481 39730955 18779284 128313798
## 3  2000 Female   9619222 42071655 19715504 140752000
## 4  2000 Male     9537815 40115959 18547799 134554000
## 5  2015 Female  11950850 41511847 19596817 163189523
## 6  2015 Male    11826927 41362080 19608451 158229297
```

This example file was created based on estimated population data available at "United Nations data" *website*.

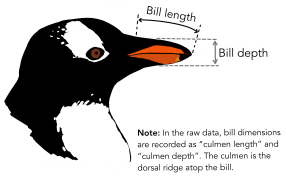
Other data we are going to explore today is penguins data set, which contains information about 3 species of penguins. Data was collected and by *Dr. Kristen Gorman* and the *Palmer Station, Antarctica LTER*, a member of the *Long Term Ecological Research Network*. Artwork by *@allison_horst*.




```
library(palmerpenguins)
penguins
```

```
## # A tibble: 344 x 8
##    species island bill_length_mm bill_depth_mm flipper_length_~ body_mass_g
##    <fct>   <fct>           <dbl>         <dbl>             <int>       <int>
##  1 Adelie  Torge~           39.1          18.7               181        3750
##  2 Adelie  Torge~           39.5          17.4               186        3800
##  3 Adelie  Torge~           40.3          18                 195        3250
##  4 Adelie  Torge~           NA            NA                  NA          NA
##  5 Adelie  Torge~           36.7          19.3               193        3450
##  6 Adelie  Torge~           39.3          20.6               190        3650
##  7 Adelie  Torge~           38.9          17.8               181        3625
##  8 Adelie  Torge~           39.2          19.6               195        4675
##  9 Adelie  Torge~           34.1          18.1               193        3475
## 10 Adelie  Torge~           42            20.2               190        4250
## # ... with 334 more rows, and 2 more variables: sex <fct>, year <int>
```

Tip: remember that you can still use usual 'data.frame-like' operations on this data set. Try checking dimensions with *dim(penguins)* or looking at whole dataset with *View(penguins)*.



Alright, now that we have some data, let's talk about basic operations on data you can do with tidyverse!

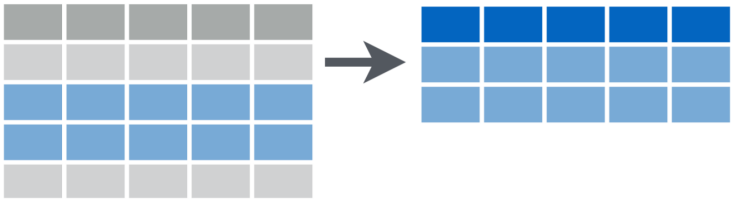
4. Basic tidyverse operations

Here are some useful commands you can use to explore your data. Included figures are taken from the *Data Wrangling with dplyr nd tidyr Cheat Sheet*.

(1) Subsetting data

- use **select** to select columns by names (or helper function)

- use **filter** to select rows that meet logical criteria / select rows based on value in specific column



(2) Ordering data

- use **arrange** to order your rows by a column value (from row to high)
- combine **arrange** with **desc** to reverse the order

(3) Updating your data

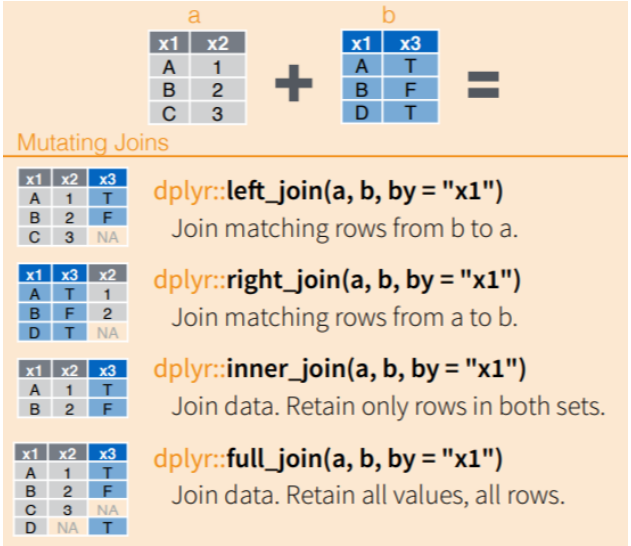
- use **mutate** to add new columns or change values in existing columns
- use **separate** to separate values in column into multiple columns and **unite** to reverse the process

(4) Grouping and summarizing data

- use **group_by** to group the rows based on value in column -> it will add additional layer of organization for other functions
- use **summarise** to summarise your data into single row of values, combine it with **group_by** to calculate statistics for your groups

(5) Combining data sets

- use **left_join** or **right_join** to add columns from one data set to compatible rows of second data set
- use **inner_join** to combine data sets and preserve only common rows
- use **full_join** to combine data sets and preserve all rows



II. Data exploration & Plotting

1. *Palmer penguins* dataset

Let's try using this operations to explore our data sets. How about we take a "glimpse" into our penguin dataset?

```r
glimpse(penguins)
```

```
## Rows: 344
## Columns: 8
## $ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, A...
## $ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torge...
## $ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34....
## $ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18....
## $ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, ...
## $ body_mass_g       <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 347...
## $ sex               <fct> male, female, female, NA, female, male, female, m...
## $ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2...
```

Here is the data column specification:

- *species* a factor denoting penguin species
- *island* a factor denoting island in Palmer Archipelago, Antarctica
- *bill_length_mm* a number denoting bill length (in millimeters)
- *bill_depth_mm* a number denoting bill depth (in millimeters)
- *flipper_length_mm* an integer denoting flipper length (in millimeters)
- *body_mass_g* an integer denoting body mass (in grams)
- *sex* a factor denoting penguin sex
- *year* variable denoting the study year

Tip: Pay special attention to data types inferred by tidyverse. This data was already cleaned up and prepared for analysis. Factor type variables will be very important for grouping and plotting our data.

2. How many penguins of each species is in the dataset?

We are only interested in 'species' column, so we can subset the dataset. (This is an optional step, just for exercise and visual purposes.)

```r
select(penguins, species)
```

```
## # A tibble: 344 x 1
##    species
##    <fct>
##  1 Adelie
##  2 Adelie
##  3 Adelie
##  4 Adelie
##  5 Adelie
##  6 Adelie
##  7 Adelie
##  8 Adelie
##  9 Adelie
## 10 Adelie
## # ... with 334 more rows
```

How to do it when using %>% operator?

```
penguins %>%
  select(species)
```

```
## # A tibble: 344 x 1
##    species
##    <fct>
##  1 Adelie
##  2 Adelie
##  3 Adelie
##  4 Adelie
##  5 Adelie
##  6 Adelie
##  7 Adelie
##  8 Adelie
##  9 Adelie
## 10 Adelie
## # ... with 334 more rows
```

Bonus (1): Other example uses of **select**:

```
# select two columns
penguins %>% select(species, sex)
```

```
## # A tibble: 344 x 2
##    species sex
##    <fct>   <fct>
##  1 Adelie  male
##  2 Adelie  female
##  3 Adelie  female
##  4 Adelie  <NA>
##  5 Adelie  female
##  6 Adelie  male
##  7 Adelie  female
##  8 Adelie  male
##  9 Adelie  <NA>
## 10 Adelie  <NA>
## # ... with 334 more rows
```

```
# select everything but 'species' column
penguins %>% select(- species)
```

```
## # A tibble: 344 x 7
##    island  bill_length_mm bill_depth_mm flipper_length_~ body_mass_g sex    year
##    <fct>            <dbl>         <dbl>            <int>       <int> <fct> <int>
##  1 Torger~           39.1          18.7              181        3750 male   2007
##  2 Torger~           39.5          17.4              186        3800 fema~  2007
##  3 Torger~           40.3          18                195        3250 fema~  2007
##  4 Torger~           NA            NA                 NA          NA <NA>   2007
##  5 Torger~           36.7          19.3              193        3450 fema~  2007
##  6 Torger~           39.3          20.6              190        3650 male   2007
```

```
##  7 Torger~           38.9          17.8             181        3625 fema~  2007
##  8 Torger~           39.2          19.6             195        4675 male   2007
##  9 Torger~           34.1          18.1             193        3475 <NA>   2007
## 10 Torger~           42            20.2             190        4250 <NA>   2007
## # ... with 334 more rows
```

```
# select coumms with regular expression
penguins %>% select(matches("bill*"))
```

```
## # A tibble: 344 x 2
##    bill_length_mm bill_depth_mm
##             <dbl>         <dbl>
##  1           39.1          18.7
##  2           39.5          17.4
##  3           40.3          18
##  4           NA            NA
##  5           36.7          19.3
##  6           39.3          20.6
##  7           38.9          17.8
##  8           39.2          19.6
##  9           34.1          18.1
## 10           42            20.2
## # ... with 334 more rows
```

Bonus (2): How to extract values from the tibble? (Here we use *head* function to show only first values in the vector)

```
# Option 1 -> deframing the vector
penguins %>%
  select(species) %>%
  deframe() %>%
  head()
```

```
## [1] Adelie Adelie Adelie Adelie Adelie Adelie
## Levels: Adelie Chinstrap Gentoo
```

```
# Option 2 -> selecting 'in place'
penguins %>%
  .$species %>%
  head()
```

```
## [1] Adelie Adelie Adelie Adelie Adelie Adelie
## Levels: Adelie Chinstrap Gentoo
```

Moving on to our exercise...

Let's try counting our penguins with **summarise** function:

```
penguins %>%
  summarise(count=n())
```

```
## # A tibble: 1 x 1
##   count
##   <int>
## 1   344
```

Well that's all of our penguins. If we are interested in looking at different species we should group data with **group_by** before.

```
# just grouping our penguins does not visually affect the data
penguins %>%
  group_by(species)
```

```
## # A tibble: 344 x 8
## # Groups:   species [3]
##    species island bill_length_mm bill_depth_mm flipper_length_~ body_mass_g
##    <fct>   <fct>           <dbl>         <dbl>            <int>       <int>
##  1 Adelie  Torge~           39.1          18.7              181        3750
##  2 Adelie  Torge~           39.5          17.4              186        3800
##  3 Adelie  Torge~           40.3          18                195        3250
##  4 Adelie  Torge~           NA            NA                 NA          NA
##  5 Adelie  Torge~           36.7          19.3              193        3450
##  6 Adelie  Torge~           39.3          20.6              190        3650
##  7 Adelie  Torge~           38.9          17.8              181        3625
##  8 Adelie  Torge~           39.2          19.6              195        4675
##  9 Adelie  Torge~           34.1          18.1              193        3475
## 10 Adelie  Torge~           42            20.2              190        4250
## # ... with 334 more rows, and 2 more variables: sex <fct>, year <int>
```

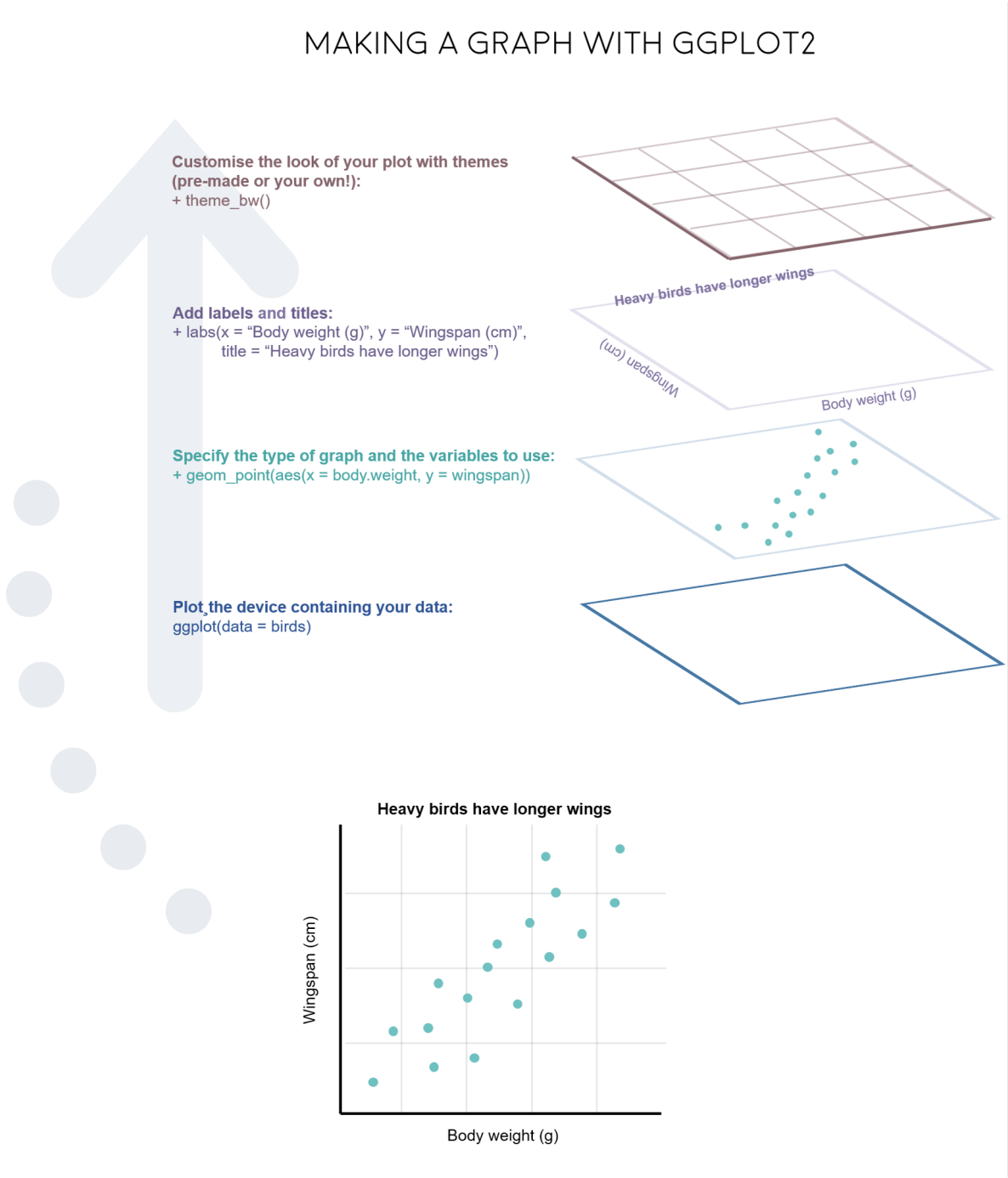
```
penguins %>%
  group_by(species) %>%
  summarise(count=n())
```

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

This looks nice, but it would look even better if we plot it

3. How to plot in ggplot2?

The **ggplot2** has very specific *grammar*, which provides the rules on how to effectively create plots. We construct the plots step, by step by adding additional levels of organization. This *layering* of plot elements provides us with a great control over our plot and high level of possible customization. Take a look at this figure below (created and made available by *Coding Club* on CC BY 4.0 license) to get preview of what we are going to do.



Hmm... I wonder if our penguins with bigger flippers are also heavier? We will check this out in a moment! Now let's try to create a plot showing a distribution of species in our dataset.

(1) Create base ggplot object

```
penguins %>%
  group_by(species) %>%
```

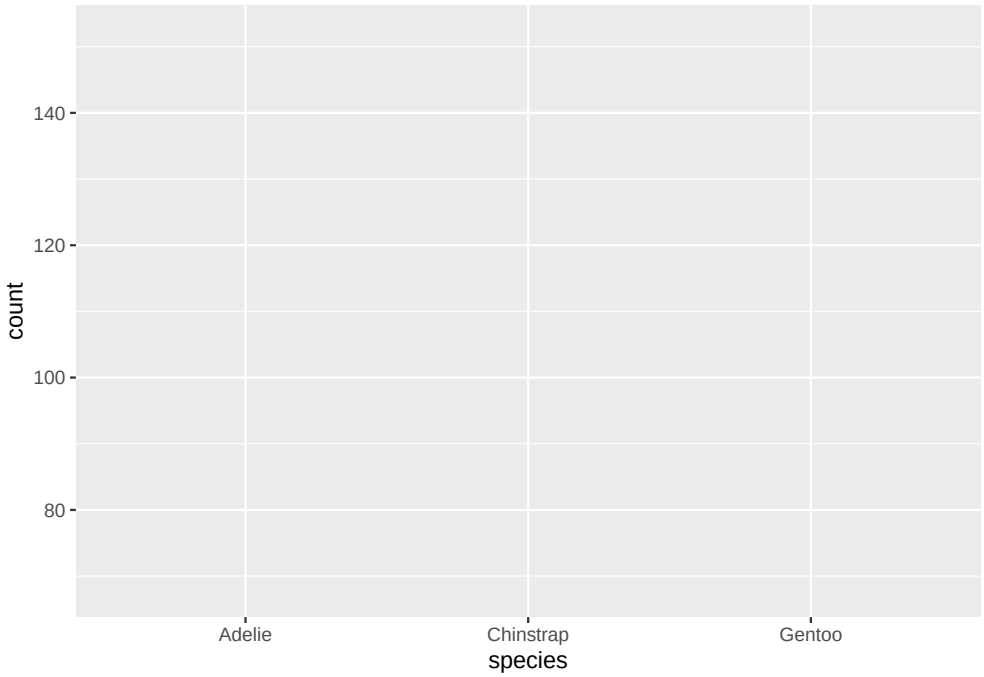
```
  summarise(count=n()) %>%
  ggplot() # or ggplot(penguins)
```


Well... that didn't work. However, we didn't even specify what we want to plot yet so it's understandable.

(2) Add aesthetics to the plot

Aesthetics specify what we want to plot (map variables to x and y *axis* and other parts of plot such as its color). Here we want to show the difference in species distribution (let's add nice colors on top of it).

```
penguins %>%
  group_by(species) %>%
  summarise(count=n()) %>%
  ggplot(aes(x=species, y=count, color=species))
```

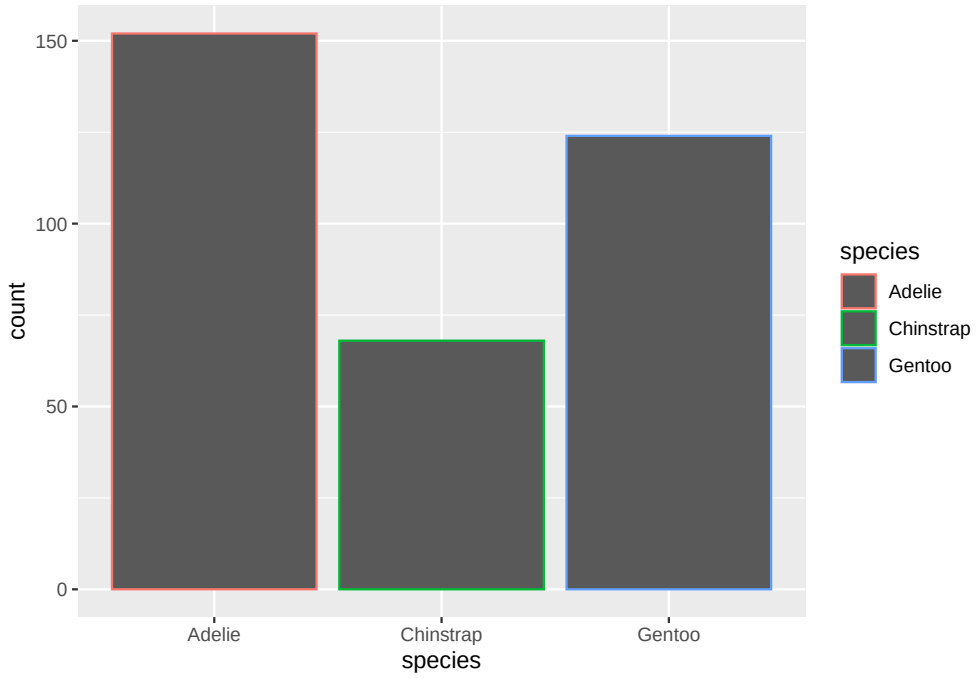


That's a good start. Now the question is how exactly we want to show the data?

(3) Add geometric layers

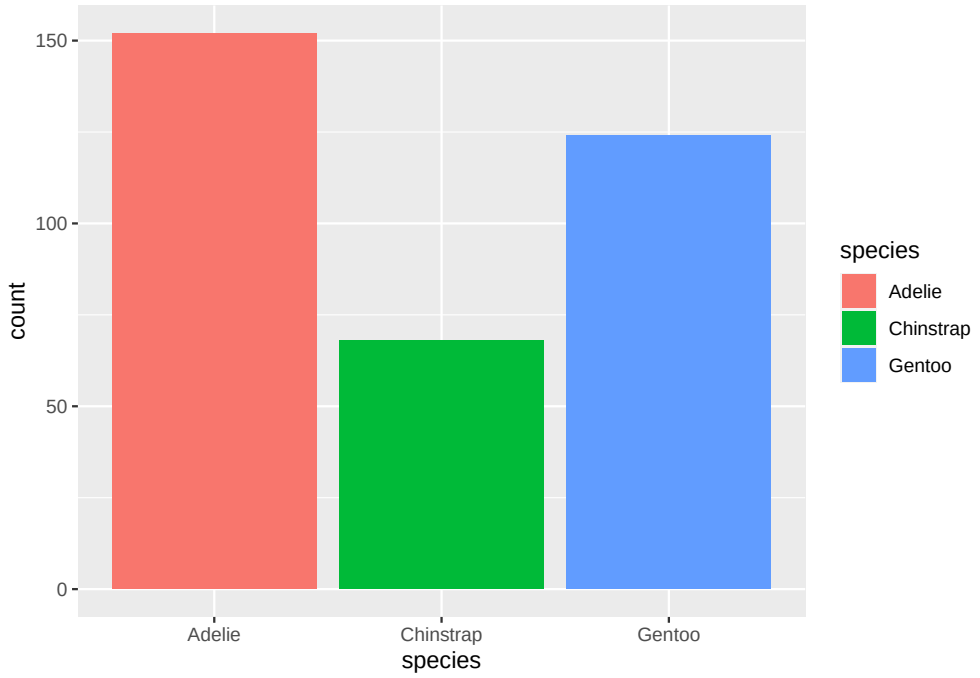
There is many different types of plots. Here because we want to show a difference in one categorical variable across groups, let's aim for simple barplot.

```
# we use 'identity' when we build barplot with x and y values,
#by defult it requires only x values
penguins %>%
  group_by(species) %>%
  summarise(count=n()) %>%
  ggplot(aes(x=species, y=count, color=species)) +
  geom_bar(stat="identity")
```



My friends and foes we have a plot! Only... the colors are not quite alright. What happened? ggplot uses two different parameters for setting colors, in general *color/colour* defines the color of the geom outline and *fill* specifies what color the geom is filled with. Let's correct it!

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



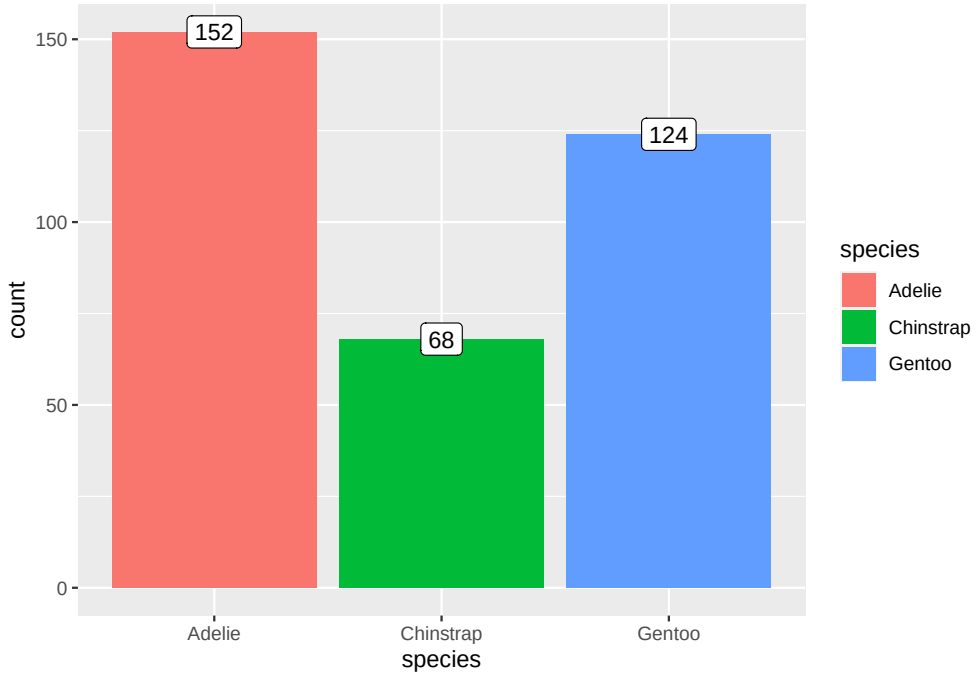
Bonus (3): Note about *aes()* placement. By placing aesthetics in *ggplot()* function, they will be applied to every *geom* function. I recommend binding our aesthetics to specific *geoms* instead. This way we can mix and match *geoms* with different mappings.

```
# this is an equivalent of last commend
penguins %>%
  group_by(species) %>%
  summarise(count=n()) %>%
  ggplot() +
  geom_bar(aes(x=species, y=count, color=species), stat="identity")
```

(3) Mixing different geoms

Let's try adding labels with specific count to our plot with *geom_label* (We can also use *geom_text*).

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



This looks very good already but there is much more things we could do.

(3) Customizing your plots.

We definitely cannot cover all the different way you can improve your plot, but here are some examples of possible customizations:

- updating axis labels and title (all can be changed with *labs* function, or by adding *xlab*, *ylab* and *ggtitle*)
- add a bit of transparency to our bars (with *alpha* parameter) so we can see our grid a bit better
- changing color palette to color-blind friendly palette
- adding new *theme** to our plot (quick change of plot appearance, see more *here* and *here*)
- editing our theme, ex. to change legend placements, removing legend altogether if it's not needed or changing fonts

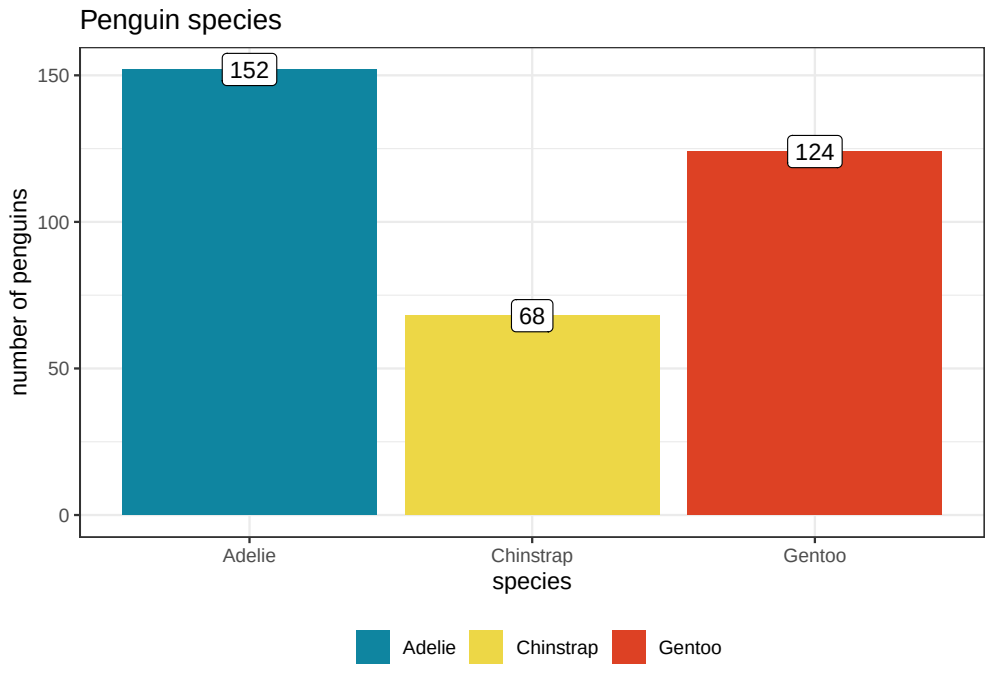
Also not applicable here (some we will see later on):

- changing axis limits (with *xlim* and *ylim*)
- changing shape of data points
- adding error bars / error bar ribbons
- fitting linear model to scatter plots

Let's add some of that to our plot!

```
library(PNWColors) # great library, with color-blind friendly palettes

penguins %>%
  group_by(species) %>%
  summarise(count=n()) %>%
  ggplot() +
  geom_bar(aes(x=species, y=count, fill=species), stat="identity") +
  geom_label(aes(x=species, y=count, label=count)) +
  labs(title="Penguin species", y="number of penguins", fill="") +
  theme_bw() +
  theme(legend.position = "bottom") +
  scale_fill_manual(values = pnw_palette("Bay",3))
```



Let's save our plot for later:

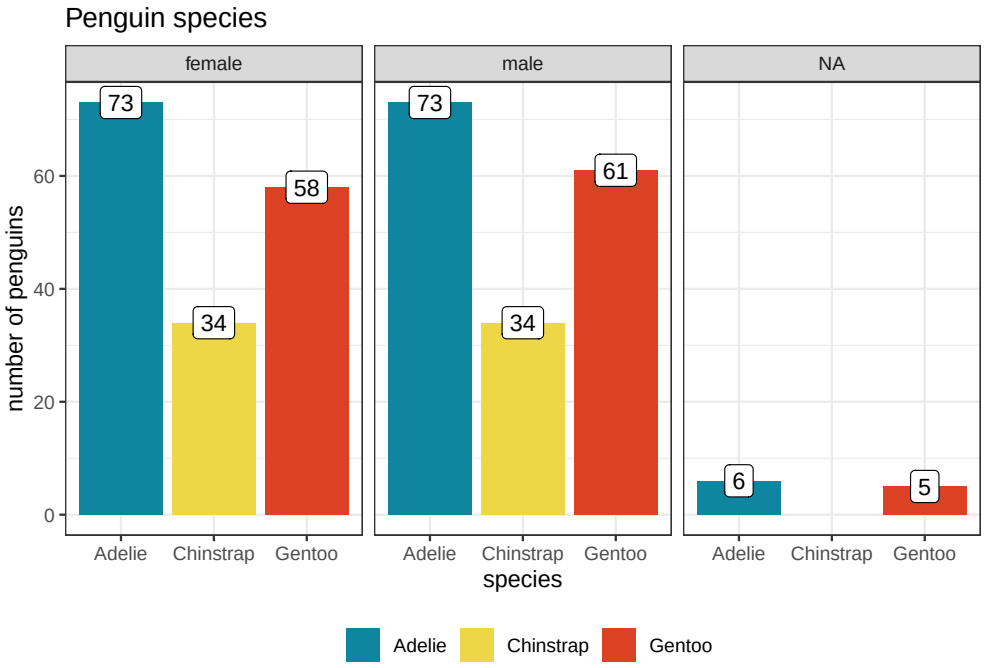
```
species_plot <- last_plot()
```

(4) Facets

Let's look at penguins of both sexes separately. *facet_wrap* and *facet_grid* functions can split data into a additional panels based on one or combination of multiple factor values.

All "small" plots represents the same graphical idiom, but with data from a different level of the faceting variable. This is different then plotting different geoms side by side, which we will see later on.

```
penguins %>%
  group_by(species, sex) %>% #we are adding new grouping variable
  summarise(count=n()) %>%
  ggplot() +
  geom_bar(aes(x=species, y=count, fill=species), stat="identity") +
  geom_label(aes(x=species, y=count, label=count)) +
  labs(title="Penguin species", y="number of penguins", fill="") +
  theme_bw() +
  theme(legend.position = "bottom") +
  scale_fill_manual(values = pnw_palette("Bay",3)) +
  facet_wrap(~sex) # split based on sex
```



The plot looks nice, but unexpectedly we have discovered that some penguins' sex was not identified. This is a good opportunity to practice some data filtering.

```
# how many NA's is in the colum
penguins %>% filter(is.na(sex))
```

```
## # A tibble: 11 x 8
##    species island bill_length_mm bill_depth_mm flipper_length_~ body_mass_g
##    <fct>   <fct>           <dbl>         <dbl>            <int>       <int>
##  1 Adelie  Torge~           NA            NA                NA          NA
##  2 Adelie  Torge~           34.1          18.1             193        3475
##  3 Adelie  Torge~           42            20.2             190        4250
##  4 Adelie  Torge~           37.8          17.1             186        3300
##  5 Adelie  Torge~           37.8          17.3             180        3700
##  6 Adelie  Dream            37.5          18.9             179        2975
##  7 Gentoo  Biscoe           44.5          14.3             216        4100
##  8 Gentoo  Biscoe           46.2          14.4             214        4650
##  9 Gentoo  Biscoe           47.3          13.8             216        4725
## 10 Gentoo  Biscoe           44.5          15.7             217        4875
## 11 Gentoo  Biscoe           NA            NA                NA          NA
## # ... with 2 more variables: sex <fct>, year <int>
```

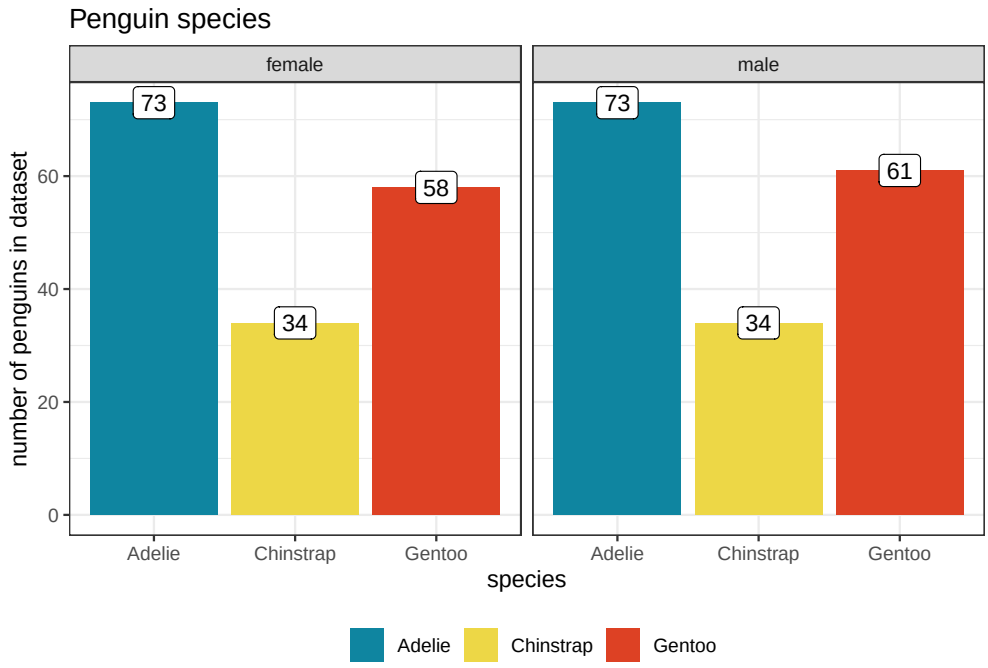
There is 11 rows that need to be filtered out.

```
# filter out this rows
penguins %>%
  filter(!is.na(sex)) %>% # filter out this rows
  group_by(species, sex) %>%
  summarise(count=n())
```

```
## # A tibble: 6 x 3
## # Groups:   species [3]
##   species   sex    count
##   <fct>     <fct>  <int>
## 1 Adelie    female    73
## 2 Adelie    male      73
## 3 Chinstrap female    34
## 4 Chinstrap male      34
## 5 Gentoo    female    58
## 6 Gentoo    male      61
```

Great! Now let's plot it again:

```
penguins %>%
  filter(!is.na(sex))%>%
  group_by(species, sex) %>% #we are adding new grouping variable
  summarise(count=n()) %>%
  ggplot() +
  geom_bar(aes(x=species, y=count, fill=species), stat="identity") +
  geom_label(aes(x=species, y=count, label=count)) +
  labs(title="Penguin species", y="number of penguins in dataset", fill="") +
  theme_bw() +
  theme(legend.position = "bottom") +
  scale_fill_manual(values = pnw_palette("Bay",3)) +
  facet_wrap(~sex) # split based on sex
```



Bonus (4): More **filter** examples

```
# filter based on catgorical variable
penguins %>% filter(species == 'Adelie')
```

```
## # A tibble: 152 x 8
##    species island bill_length_mm bill_depth_mm flipper_length_~ body_mass_g
##    <fct>   <fct>           <dbl>         <dbl>            <int>       <int>
##  1 Adelie  Torge~           39.1          18.7              181        3750
##  2 Adelie  Torge~           39.5          17.4              186        3800
##  3 Adelie  Torge~           40.3          18                195        3250
##  4 Adelie  Torge~           NA            NA                 NA          NA
##  5 Adelie  Torge~           36.7          19.3              193        3450
##  6 Adelie  Torge~           39.3          20.6              190        3650
##  7 Adelie  Torge~           38.9          17.8              181        3625
##  8 Adelie  Torge~           39.2          19.6              195        4675
##  9 Adelie  Torge~           34.1          18.1              193        3475
## 10 Adelie  Torge~           42            20.2              190        4250
## # ... with 142 more rows, and 2 more variables: sex <fct>, year <int>
```

```
# filter based on threshold for continuous variable
penguins %>% filter(bill_depth_mm>(bill_length_mm/2))
```

```
## # A tibble: 35 x 8
##    species island bill_length_mm bill_depth_mm flipper_length_~ body_mass_g
##    <fct>   <fct>           <dbl>         <dbl>            <int>       <int>
##  1 Adelie  Torge~           36.7          19.3              193        3450
##  2 Adelie  Torge~           39.3          20.6              190        3650
##  3 Adelie  Torge~           34.1          18.1              193        3475
##  4 Adelie  Torge~           38.6          21.2              191        3800
```

```
##  5 Adelie  Torge~           34.6          21.1             198        4400
##  6 Adelie  Torge~           34.4          18.4             184        3325
##  7 Adelie  Biscoe           35.9          19.2             189        3800
##  8 Adelie  Biscoe           35.3          18.9             187        3800
##  9 Adelie  Dream            39.2          21.1             196        4150
## 10 Adelie  Dream            38.8          20               190        3950
## # ... with 25 more rows, and 2 more variables: sex <fct>, year <int>
```

```
# filter based on multiple values of one variable
penguins %>% filter(species %in% c('Adelie', 'Gentoo'))
```

```
## # A tibble: 276 x 8
##    species island bill_length_mm bill_depth_mm flipper_length_~ body_mass_g
##    <fct>   <fct>           <dbl>         <dbl>            <int>       <int>
##  1 Adelie  Torge~           39.1          18.7              181        3750
##  2 Adelie  Torge~           39.5          17.4              186        3800
##  3 Adelie  Torge~           40.3          18                195        3250
##  4 Adelie  Torge~           NA            NA                 NA          NA
##  5 Adelie  Torge~           36.7          19.3              193        3450
##  6 Adelie  Torge~           39.3          20.6              190        3650
##  7 Adelie  Torge~           38.9          17.8              181        3625
##  8 Adelie  Torge~           39.2          19.6              195        4675
##  9 Adelie  Torge~           34.1          18.1              193        3475
## 10 Adelie  Torge~           42            20.2              190        4250
## # ... with 266 more rows, and 2 more variables: sex <fct>, year <int>
```

```
# filter with multiple conditions (use '&' operator as AND, '|' operator as OR)
penguins %>% filter(species == 'Adelie' & island == 'Torgersen')
```

```
## # A tibble: 52 x 8
##    species island bill_length_mm bill_depth_mm flipper_length_~ body_mass_g
##    <fct>   <fct>           <dbl>         <dbl>            <int>       <int>
##  1 Adelie  Torge~           39.1          18.7              181        3750
##  2 Adelie  Torge~           39.5          17.4              186        3800
##  3 Adelie  Torge~           40.3          18                195        3250
##  4 Adelie  Torge~           NA            NA                 NA          NA
##  5 Adelie  Torge~           36.7          19.3              193        3450
##  6 Adelie  Torge~           39.3          20.6              190        3650
##  7 Adelie  Torge~           38.9          17.8              181        3625
##  8 Adelie  Torge~           39.2          19.6              195        4675
##  9 Adelie  Torge~           34.1          18.1              193        3475
## 10 Adelie  Torge~           42            20.2              190        4250
## # ... with 42 more rows, and 2 more variables: sex <fct>, year <int>
```

4. Does heavier penguins have longer flippers?

(1). Let's explore the data

We are interested in penguin body mass and flipper length for different species.

```
penguins %>%
  select(species, body_mass_g, flipper_length_mm)
```

```
## # A tibble: 344 x 3
##    species body_mass_g flipper_length_mm
##    <fct>         <int>             <int>
##  1 Adelie         3750               181
##  2 Adelie         3800               186
##  3 Adelie         3250               195
##  4 Adelie           NA                NA
##  5 Adelie         3450               193
##  6 Adelie         3650               190
##  7 Adelie         3625               181
##  8 Adelie         4675               195
##  9 Adelie         3475               193
## 10 Adelie         4250               190
## # ... with 334 more rows
```

Let's check what species have highest and lowest body mass:

```
penguins %>%
  select(species, body_mass_g, flipper_length_mm) %>%
  arrange(desc(body_mass_g)) #descending order
```

```
## # A tibble: 344 x 3
##    species body_mass_g flipper_length_mm
##    <fct>         <int>             <int>
##  1 Gentoo         6300               221
##  2 Gentoo         6050               230
##  3 Gentoo         6000               220
##  4 Gentoo         6000               222
##  5 Gentoo         5950               223
##  6 Gentoo         5950               229
##  7 Gentoo         5850               213
##  8 Gentoo         5850               217
##  9 Gentoo         5850               230
## 10 Gentoo         5800               229
## # ... with 334 more rows
```

We can also calculate average body mass and flipper length across species.

```
# check mean body mass and flipper length across species
penguins %>%
  group_by(species) %>%
  summarise(mean_mass = mean(body_mass_g, na.rm=TRUE),
            mean_flipper_length = mean(flipper_length_mm, na.rm=TRUE))
```

```
## # A tibble: 3 x 3
##   species   mean_mass mean_flipper_length
##   <fct>         <dbl>               <dbl>
## 1 Adelie        3701.                190.
## 2 Chinstrap     3733.                196.
## 3 Gentoo        5076.                217.
```

Bonus (5): **summarize** function can do much more then just counting penguins and means!

```
# check mean, maximum and minimum body mass of penguins across species
penguins %>%
  group_by(species) %>%
  summarise(count=n(),
            mean_mass = mean(body_mass_g, na.rm=TRUE),
            min_mass = min(body_mass_g, na.rm = TRUE),
            max_mass = max(body_mass_g, na.rm = TRUE))
```

```
## # A tibble: 3 x 5
##   species   count mean_mass min_mass max_mass
##   <fct>     <int>     <dbl>    <int>    <int>
## 1 Adelie      152     3701.     2850     4775
## 2 Chinstrap    68     3733.     2700     4800
## 3 Gentoo      124     5076.     3950     6300
```

```
# calculate mean accross all numeric columns
penguins %>%
  group_by(species) %>%
  summarize(across(where(is.numeric), mean, na.rm = TRUE)) %>%
  select(-year) # remove year column from output
```

```
## # A tibble: 3 x 5
##   species   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##   <fct>              <dbl>         <dbl>             <dbl>       <dbl>
## 1 Adelie              38.8          18.3              190.       3701.
## 2 Chinstrap           48.8          18.4              196.       3733.
## 3 Gentoo              47.5          15.0              217.       5076.
```

Are there differences in average body mass to flipper length ratio across species? Let's use **mutate** to create new column in our dataset.

```
penguins %>%
  group_by(species) %>%
  mutate(ratio=body_mass_g/flipper_length_mm)
```

```
## # A tibble: 344 x 9
## # Groups:   species [3]
##    species island bill_length_mm bill_depth_mm flipper_length_~ body_mass_g
##    <fct>   <fct>           <dbl>         <dbl>            <int>       <int>
##  1 Adelie  Torge~           39.1          18.7              181        3750
##  2 Adelie  Torge~           39.5          17.4              186        3800
##  3 Adelie  Torge~           40.3          18                195        3250
##  4 Adelie  Torge~           NA            NA                 NA          NA
##  5 Adelie  Torge~           36.7          19.3              193        3450
##  6 Adelie  Torge~           39.3          20.6              190        3650
##  7 Adelie  Torge~           38.9          17.8              181        3625
##  8 Adelie  Torge~           39.2          19.6              195        4675
##  9 Adelie  Torge~           34.1          18.1              193        3475
## 10 Adelie  Torge~           42            20.2              190        4250
## # ... with 334 more rows, and 3 more variables: sex <fct>, year <int>,
## #   ratio <dbl>
```

```
penguins %>%
  group_by(species) %>%
  mutate(ratio=body_mass_g/flipper_length_mm) %>%
  summarise(avg_ratio = mean(ratio, na.rm = TRUE))
```

```
## # A tibble: 3 x 2
##   species   avg_ratio
##   <fct>         <dbl>
## 1 Adelie         19.5
## 2 Chinstrap      19.0
## 3 Gentoo         23.3
```

Our results sure look promising, but we cannot really tell only based only on means. Let's plot the data! What type of plot do we want to use? How about we show every penguin as a data point?

(2) Plotting the relationship between two continuous variables

Do we have any missing values?

```
penguins %>%
  filter(is.na(body_mass_g) | is.na(flipper_length_mm))
```

```
## # A tibble: 2 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_~ body_mass_g sex
##   <fct>   <fct>           <dbl>         <dbl>            <int>       <int> <fct>
## 1 Adelie  Torge~             NA            NA               NA          NA <NA>
## 2 Gentoo  Biscoe             NA            NA               NA          NA <NA>
## # ... with 1 more variable: year <int>
```

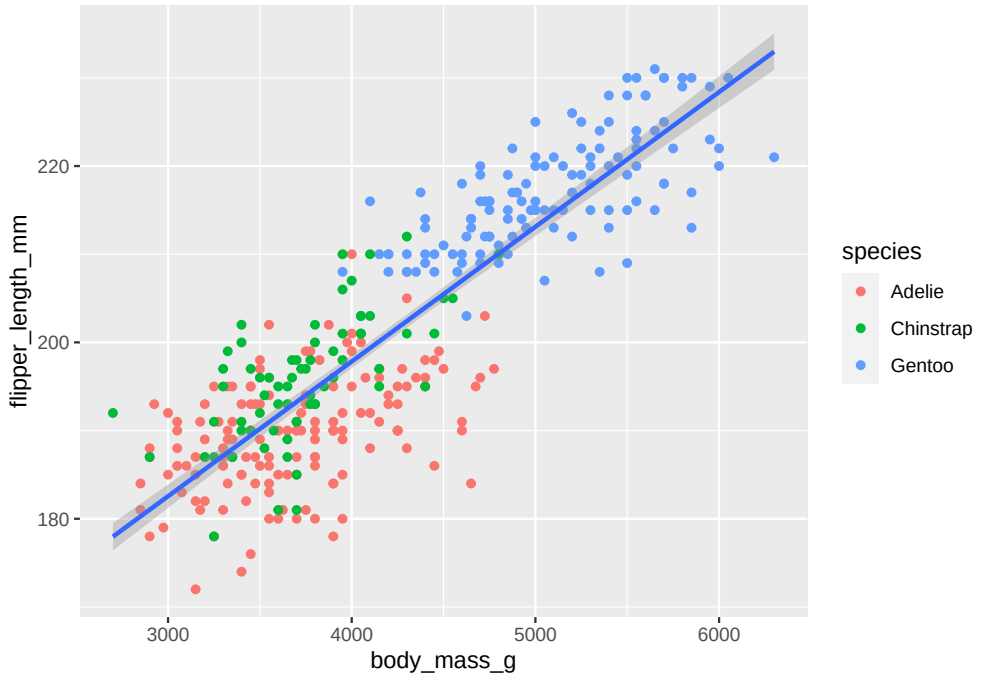
We will have to filter this two penguins out, before plotting:

```
penguins %>%
  filter(!is.na(body_mass_g) & !is.na(flipper_length_mm)) %>%
  ggplot() +
  geom_point(aes(x=body_mass_g, y=flipper_length_mm, color=species))
```

This is great! It looks like there is some correlation here. Also *Gentoo* penguins are clearly the biggest birds. Let's fit linear model to our data to better show trend - gray band is 95% confidence level interval for predictions from model.

```
penguins %>%
  filter(!is.na(body_mass_g) & !is.na(flipper_length_mm)) %>%
  ggplot() +
  geom_point(aes(x=body_mass_g, y=flipper_length_mm, color=species)) +
  geom_smooth(aes(x=body_mass_g, y=flipper_length_mm), method = "lm")
```

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

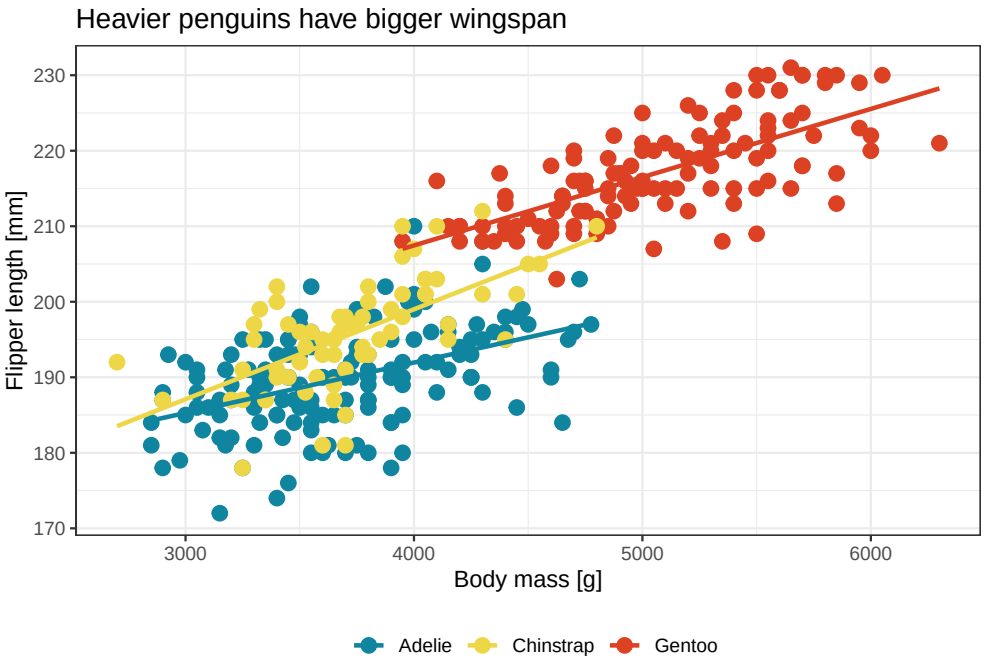


(3) Customizing the plot

This already looks very good! Let's customize this plot a bit and fit model separately to show trend in different species.

```
penguins %>%
  filter(!is.na(body_mass_g) & !is.na(flipper_length_mm)) %>%
  ggplot() +
  geom_point(aes(x=body_mass_g, y=flipper_length_mm, color=species),
             size=3) + # increase size of points
  geom_smooth(aes(x=body_mass_g, y=flipper_length_mm, group=species, color=species),
              method = "lm", se=FALSE) +
  labs(x= "Body mass [g]", y= "Flipper length [mm]",
       title = "Heavier penguins have bigger wingspan", color="") +
  theme_bw() +
  theme(legend.position = "bottom") +
  scale_color_manual(values = pnw_palette("Bay",3)) # use 'color' (not 'fill') palette
```

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



Let's save our plot for later:

```
mass_wingspan_plot <- last_plot()
```

5. What is the distribution of bill length across species?

(1). Let's explore the data

Check mean values of bill length across species.

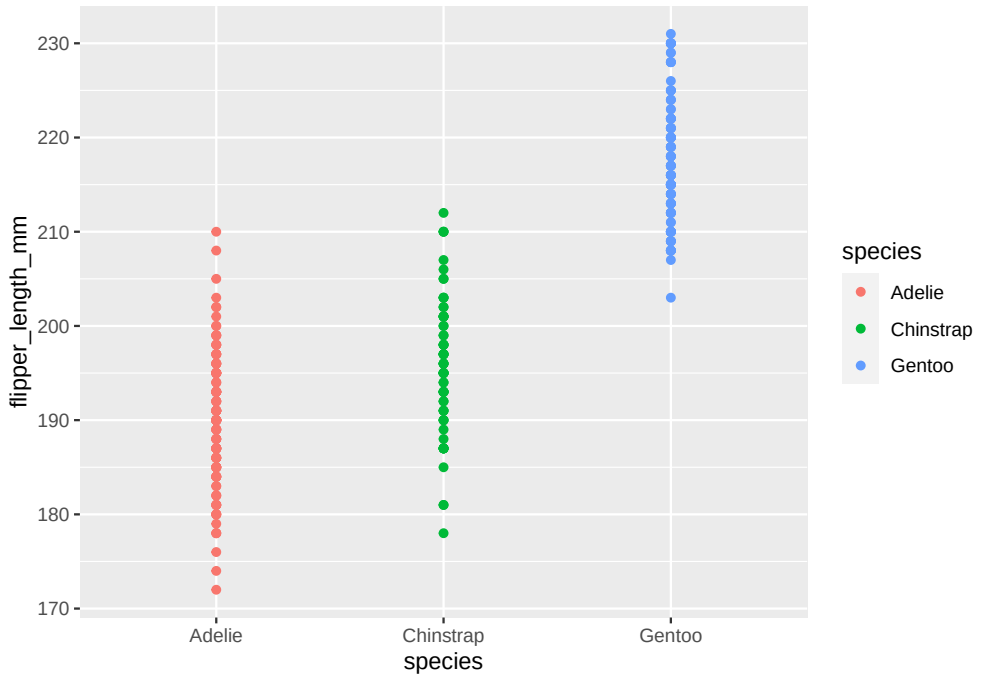
```
penguins %>%
  filter(! is.na(bill_length_mm)) %>% # filter out missing values
  group_by(species) %>%
  summarise(mean_bill_length=mean(bill_length_mm, na.rm = TRUE))
```

```
## # A tibble: 3 x 2
##   species   mean_bill_length
##   <fct>                <dbl>
## 1 Adelie                38.8
## 2 Chinstrap             48.8
## 3 Gentoo                47.5
```

Even though we observed that *Gentoo* are the biggest birds, the difference between average bill length is not very big.

(2) Plotting continuous variable across different groups

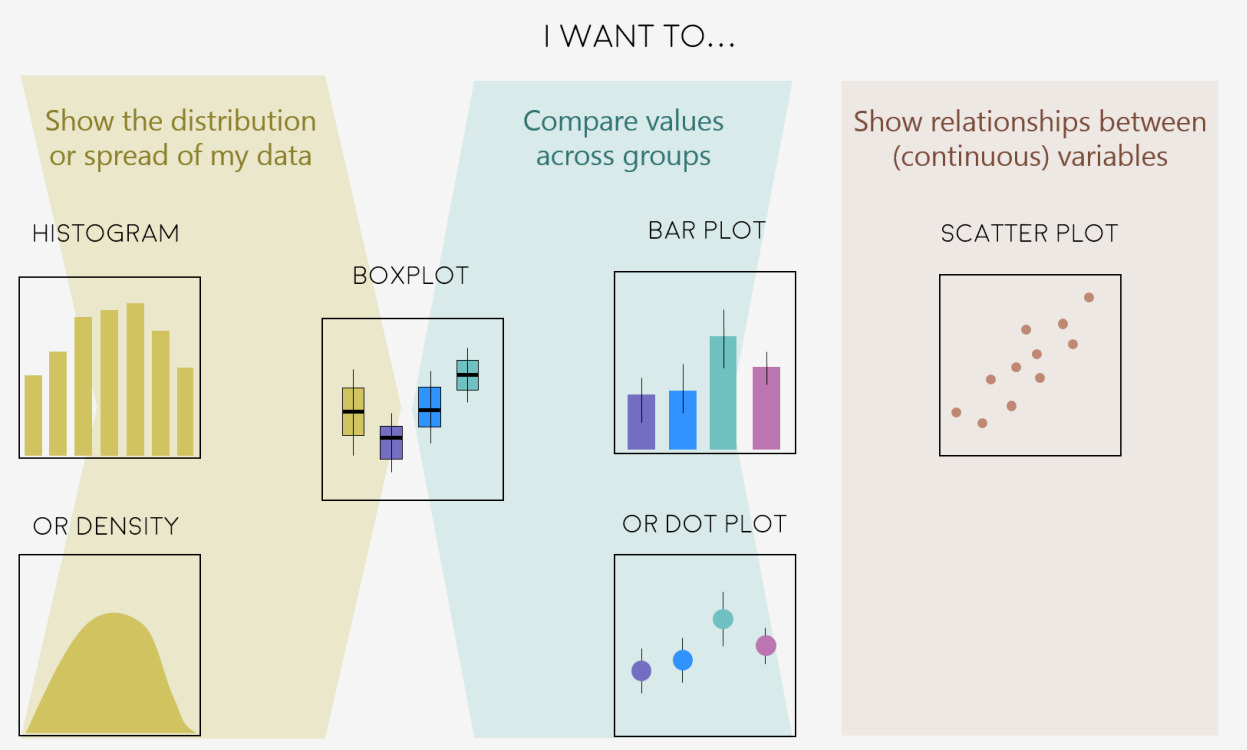
```
penguins %>%
  filter(!is.na(bill_length_mm)) %>%
  ggplot() +
  geom_point(aes(x=species, y=flipper_length_mm, color=species))
```



Hmm... that doesn't look right, we cannot use the same *geom* type. What should we use?

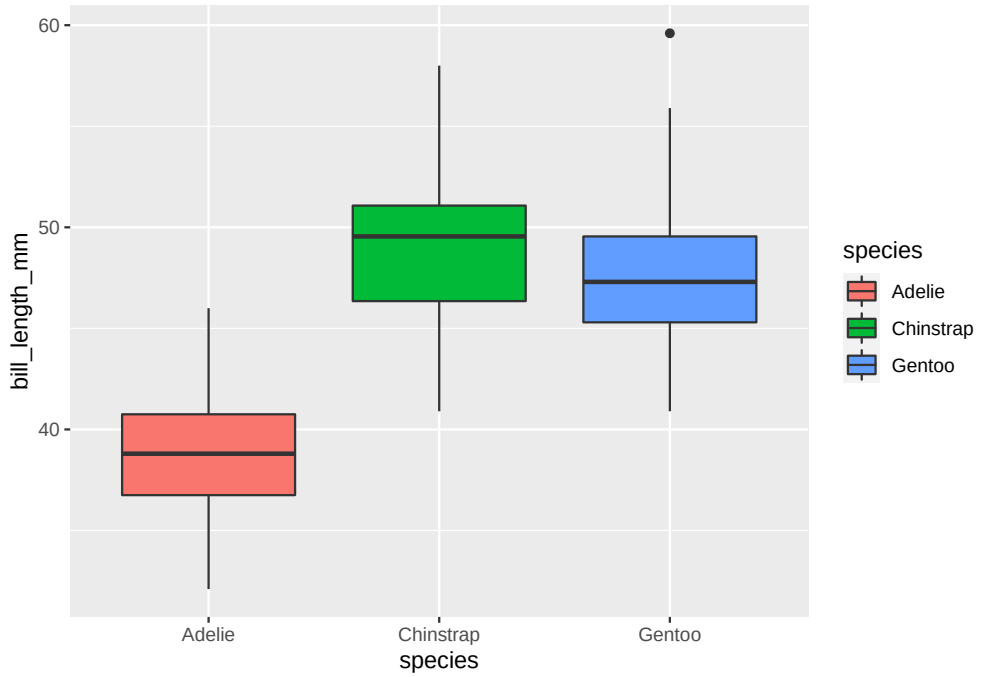
Which plot type is appropriate for our data?

The answer to this questions is not always immediately obvious. The most important thing to consider is what are the types of variables we want to display. Take a look at this figure (also created by *Coding Club*).



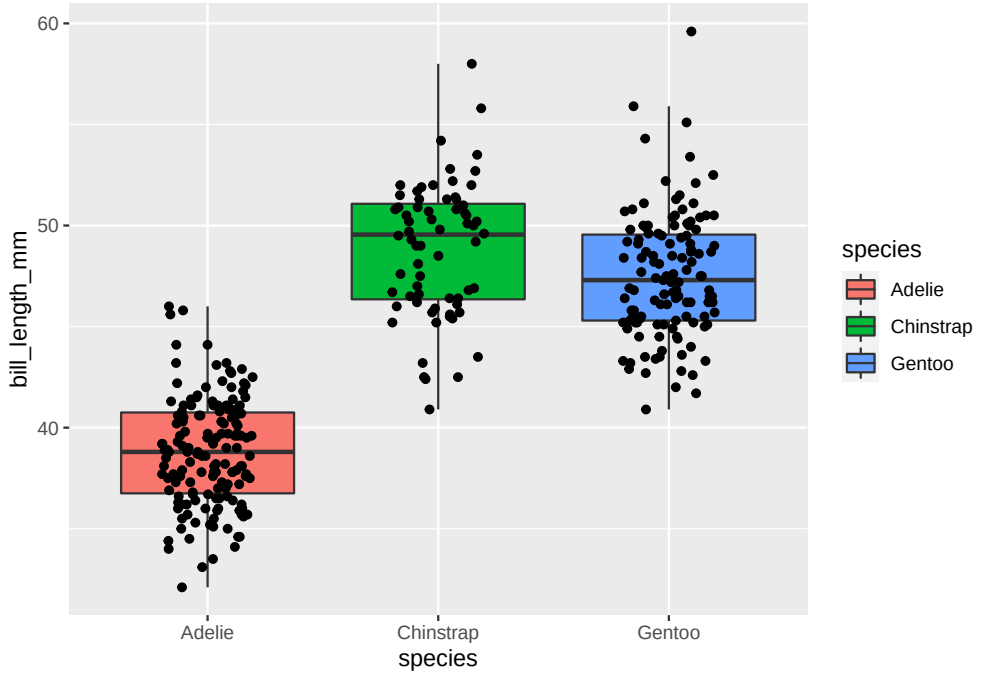
We want to show a distribution of continuous variable across different groups. Let's use boxplot!

```
penguins %>%
  filter(!is.na(bill_length_mm)) %>%
  ggplot() +
  geom_boxplot(aes(x=species,y=bill_length_mm, fill=species))
```



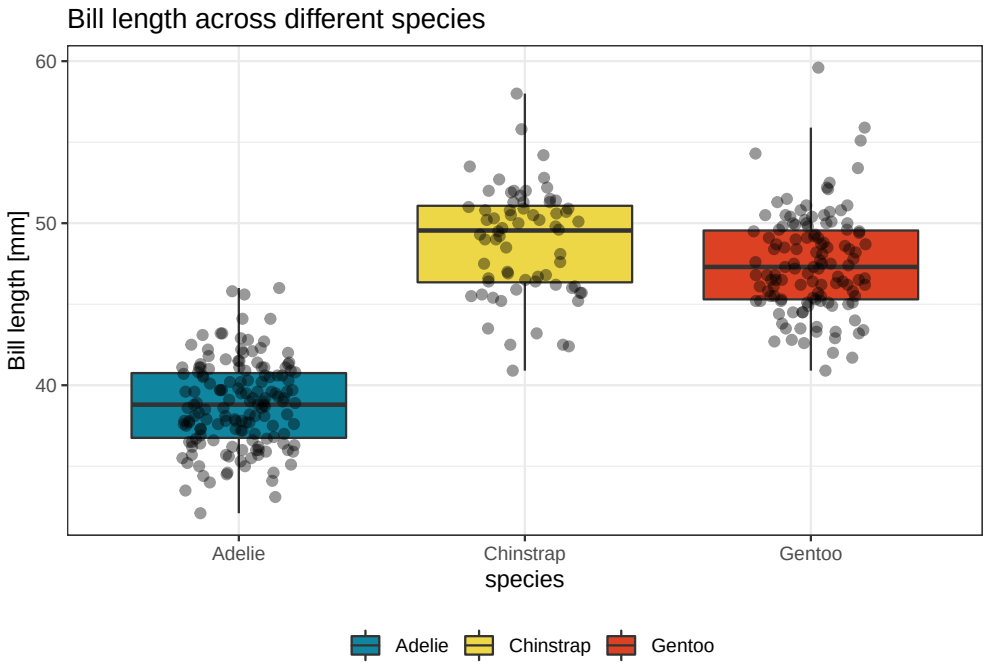
This is an improvement, but if we want to also see our specific data point we can do it with a variation of *geom_point* called *geom_jitter* (it adds a bit of noise to points position, so we will be able to see our observations).

```
penguins %>%
  filter(!is.na(bill_length_mm)) %>%
  ggplot() +
  # hide outliers
  geom_boxplot(aes(x=species,y=bill_length_mm, fill=species), outlier.alpha = 0) +
  # we restrict variation of point heights
  geom_jitter(aes(x=species, y=bill_length_mm), width = 0.2, height = 0)
```



Even though we saw that *Adelie* and *Chinstrap* penguins have similar sizes (or at least wingspan and body mass) it looks like former have much shorter bills. Let's customize our plot and finish!

```
penguins %>%
  filter(!is.na(bill_length_mm)) %>%
  ggplot() +
  geom_boxplot(aes(x=species,y=bill_length_mm, fill=species), outlier.alpha = 0) +
  geom_jitter(aes(x=species, y=bill_length_mm),
              width = 0.2, height = 0, size=2, alpha=0.4) + # add transparency to points
  labs(y= "Bill length [mm]", title = "Bill length across different species", fill="") +
  theme_bw() +
  theme(legend.position = "bottom") +
  scale_fill_manual(values = pnw_palette("Bay",3))
```



Let's save our plot:

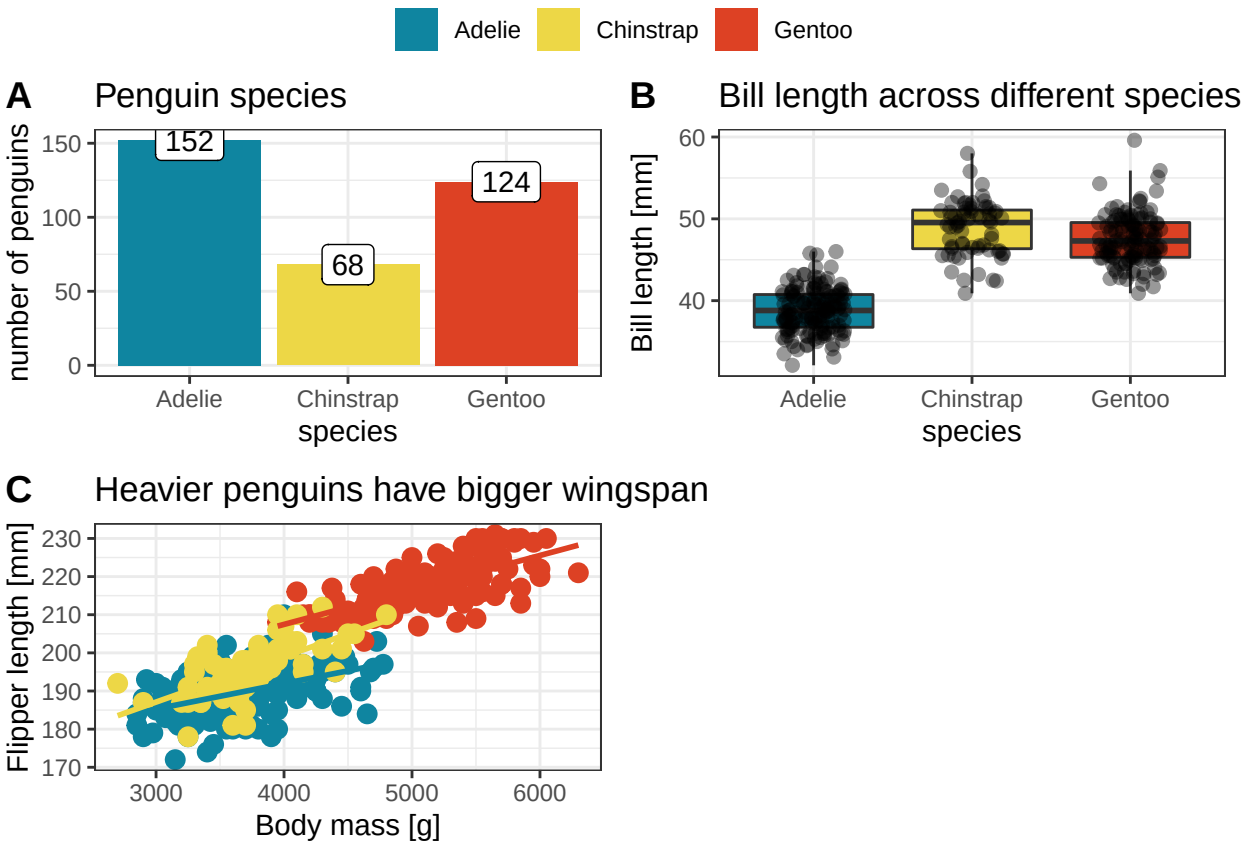
```
bill_plot <- last_plot()
```

(3) Combining different plots

We created three different plots for *palmerpenguins* dataset. How can we see them together? This is a different then creating *facets*, because every plot represents different data. One of the easiest way to display different plots together is to use *ggarrange* function from *ggpubr* package.

```
library(ggpubr)
# the input are variables with our saved plots
ggarrange(species_plot, bill_plot,mass_wingspan_plot,
          nrow=2, ncol=2,# you can specify number of rows and columns
          common.legend = TRUE,
          align = "hv",
          labels = c('A', 'B', 'C'))
```

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

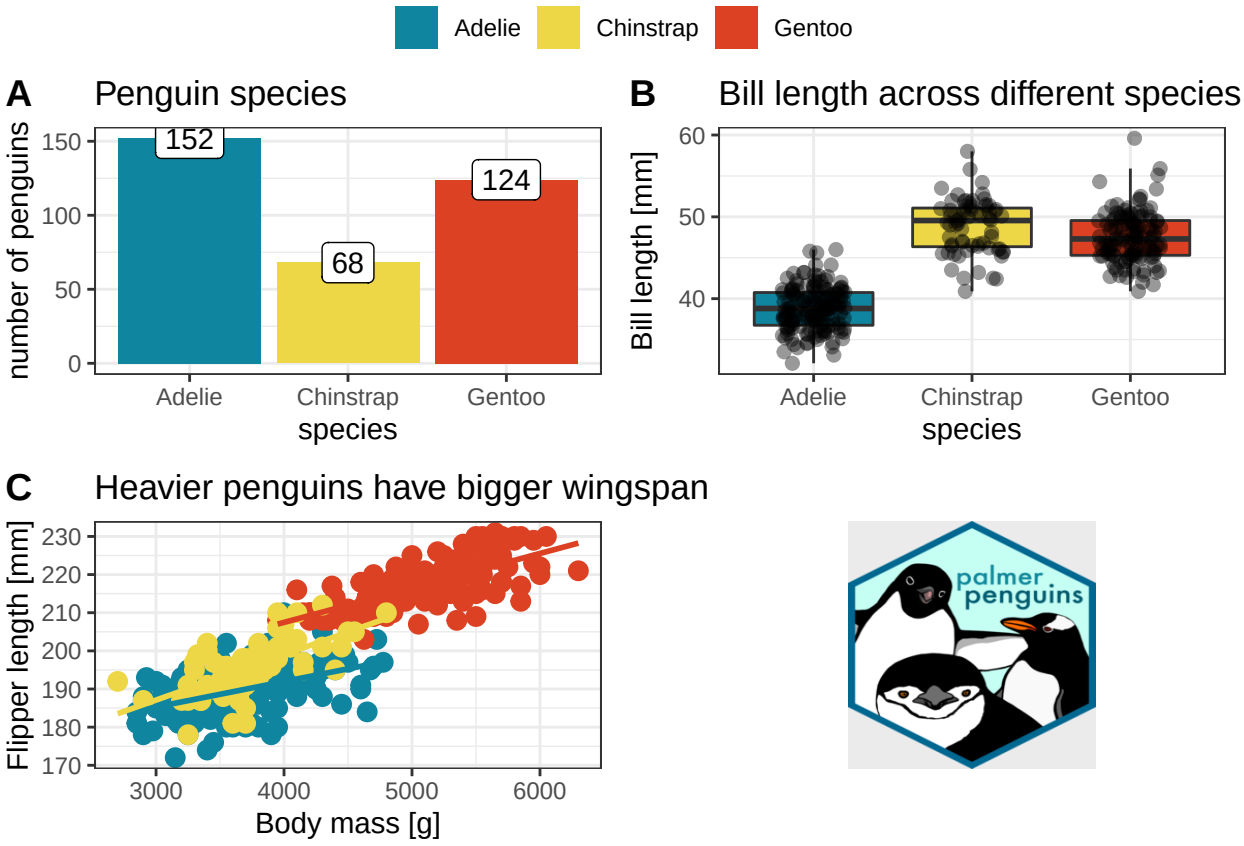


Almost done, let's add something extra!

```
library(png)
logo_png <- readPNG("images/palmer_logo.png")
logo_img <- ggplot() + background_image(logo_png) + coord_fixed() #fix image ratio

ggarrange(species_plot, bill_plot, mass_wingspan_plot, logo_img,
          nrow=2, ncol=2,
          common.legend = TRUE,
          align = "hv",
          labels = c('A', 'B', 'C', ""),
          widths = c(1,1,1,0.5))
```

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



6. Importance of reshaping your data

The *palmerpenguins* dataset is already preprocessed and prepared for analysis. This is not always the case. When using your own data frequently check if your observations are in correct orientation (rows or columns). This is extremely important when plotting data, because most of the operations is based on assumption of variables being in columns of the tibble.

Q: What to do when this is not the case?

A: We have to reshape the data with the help of **gather** and **spread** operations.

Q: How do we decide if our data need reshaping?

A: Think about effect you need to achieve and work your way down!

Let's see an example! Remember *countries* dataset we loaded from the file?

```
countries
```

```
## # A tibble: 6 x 6
##    year sex    Australia  Germany   Poland       USA
##   <dbl> <chr>      <dbl>    <dbl>    <dbl>     <dbl>
## 1  1995 Female   9063508 41930010 19808312 134441472
## 2  1995 Male     8990481 39730955 18779284 128313798
## 3  2000 Female   9619222 42071655 19715504 140752000
## 4  2000 Male     9537815 40115959 18547799 134554000
## 5  2015 Female  11950850 41511847 19596817 163189523
## 6  2015 Male    11826927 41362080 19608451 158229297
```

Let's say we are interested in seeing a (line) plot of population change over time in separate countries. To make things extra complicated we want to look at both sexes together.

What variables we would need?

- country for grouping purposes
- year for *x axis*
- (summed) population for *y axis*

This is what we would call a "*wide*" dataset, each different variable is in the different column. We want to reshape the data into a *long* (or *narrow*) dataset. This is where **gather** operation comes to our aid:

```
countries %>%
  gather(key="country", value="population", -year, -sex)
```

```
## # A tibble: 24 x 4
##     year sex    country   population
##    <dbl> <chr>  <chr>          <dbl>
##  1  1995 Female Australia    9063508
##  2  1995 Male   Australia    8990481
##  3  2000 Female Australia    9619222
##  4  2000 Male   Australia    9537815
##  5  2015 Female Australia   11950850
##  6  2015 Male   Australia   11826927
##  7  1995 Female Germany     41930010
##  8  1995 Male   Germany     39730955
##  9  2000 Female Germany     42071655
## 10  2000 Male   Germany     40115959
## # ... with 14 more rows
```

Let's save this data on variable for our plot.

```
long_countries <- .Last.value
```

We can revert this situation by using a **spread** command. It's very helpful in case if we want to focus only on specific observations.

```
long_countries %>%
  spread(key = country, value=population)
```

```
## # A tibble: 6 x 6
##    year sex    Australia  Germany   Poland       USA
##   <dbl> <chr>      <dbl>    <dbl>    <dbl>     <dbl>
## 1  1995 Female   9063508 41930010 19808312 134441472
## 2  1995 Male     8990481 39730955 18779284 128313798
## 3  2000 Female   9619222 42071655 19715504 140752000
## 4  2000 Male     9537815 40115959 18547799 134554000
## 5  2015 Female  11950850 41511847 19596817 163189523
## 6  2015 Male    11826927 41362080 19608451 158229297
```

Here is the plot we wanted to make:

```r
long_countries %>%
  group_by(year, country) %>%
  summarise(total_population=sum(population)) %>% # we are interested in total population
  ggplot(aes(x=year, y=total_population, group=country, color=country)) +
  geom_line(size=1) +
  geom_point(size=3)  +
  labs(y="estimated population", title = "Population over time", color="") +
  theme_bw() +
  theme(legend.position = "top") +
  scale_color_manual(values = pnw_palette("Sailboat",4))
```