R4DS Whole game

SHH

2023-07-03

Whole game

Goal of this part is a rapid overview of the main tools of data science: **importing, tidying, transforming, visualizing**

---

1. Data Visualization

1.1. Introduction

```
# tidyverse packages
# install.packages('tidyverse')
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

It tells you which functions from the tidyverse conflict with functions in base R or other packages.

```
# install.packages('palmerpenguins')
# install.packages('ggthemes')
library(palmerpenguins)
library(ggthemes)
```

Use palmerpenguins package, which include the `penguins` dataset. Also the ggthemes package offers a colorblind sage color palette

2. First Steps

Do penguins with longer filppers weigh more or less than penguins with shorter flippers? What does the relationship between flipper length and body mass look like? Is it positive? negative? linear? nonlinear? Does the relationship vary by the species of the penguins? How about by the island where the penguin lives?

The penguins data frame

- **Variable**: quantity, quality, or property that you can measure
- **Value**: the state of a variable when you measure it
- **Observation**: set of measurements made under similar conditions
- **Tabular data**: set of values, each associated with a variable and an observation. Tabular data is tidy if each value is placed in its own "cell", each variable in its own columnm and each observaation in its own row.

In the tidyverse, it use special dataframes called tibbles

```r
penguins
```

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

```r
glimpse(penguins) # str(penguins)
```

```
## Rows: 344
## Columns: 8
## $ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adel~
## $ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgerse~
## $ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, ~
## $ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, ~
## $ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186~
## $ body_mass_g       <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, ~
## $ sex               <fct> male, female, female, NA, female, male, female, male~
## $ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007~
```

Ultimate goal Our ultimate goal is to create visualization displaying the relationship between flipper lengths and body masses of these penguins, taking into consideration the species of the penguin.

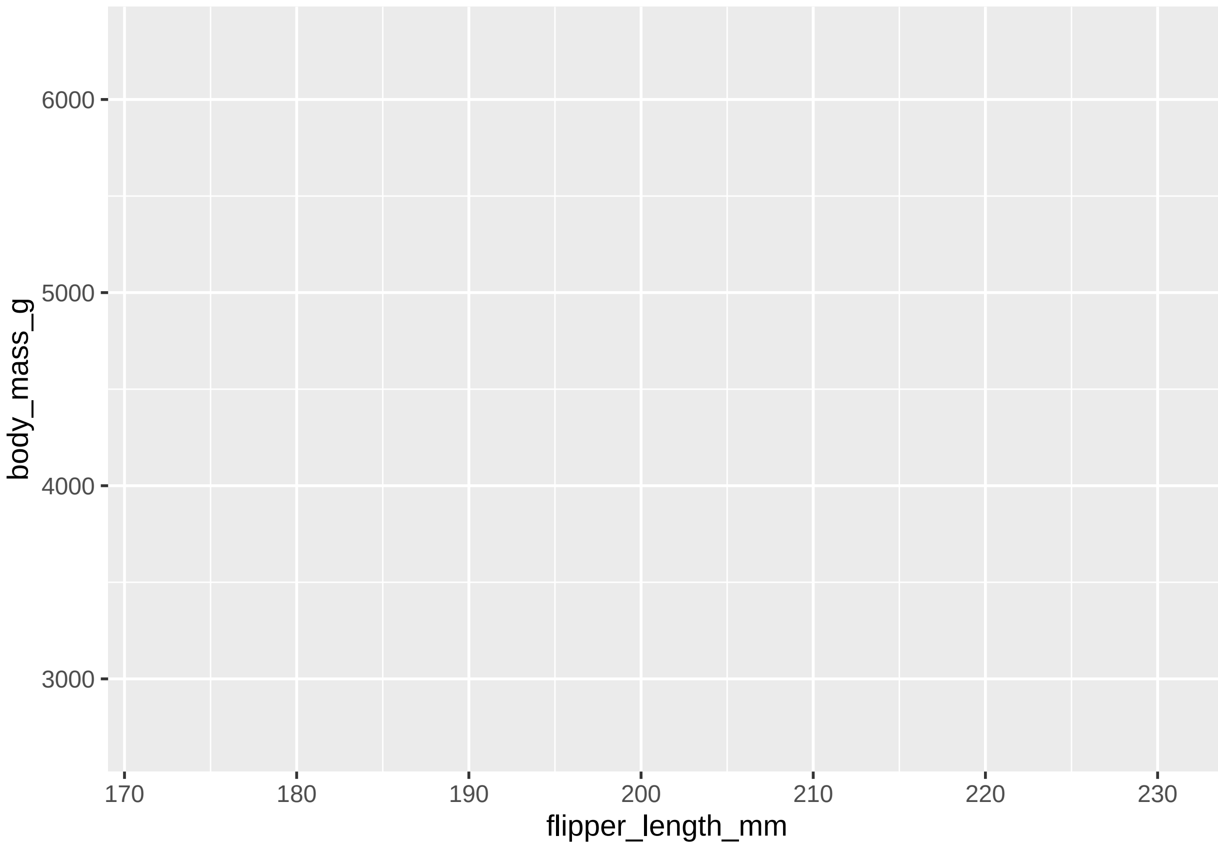
Creating a ggplot In `ggplot2`, we begin a plot with the function `ggplot()`. It defines a plot object that you then add layers to. arguments are

- `data`: dataset to use in the graph
- `mapping`: defines how variables in our dataset are mapped to visual properties(aesthetic) of our plot

```
ggplot(data = penguins)
```


It creates empty graph that is primed to display the data. We can think of it like an empty canvas we'll paint the reaming layers of our plot onto.

```
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g)
)
```



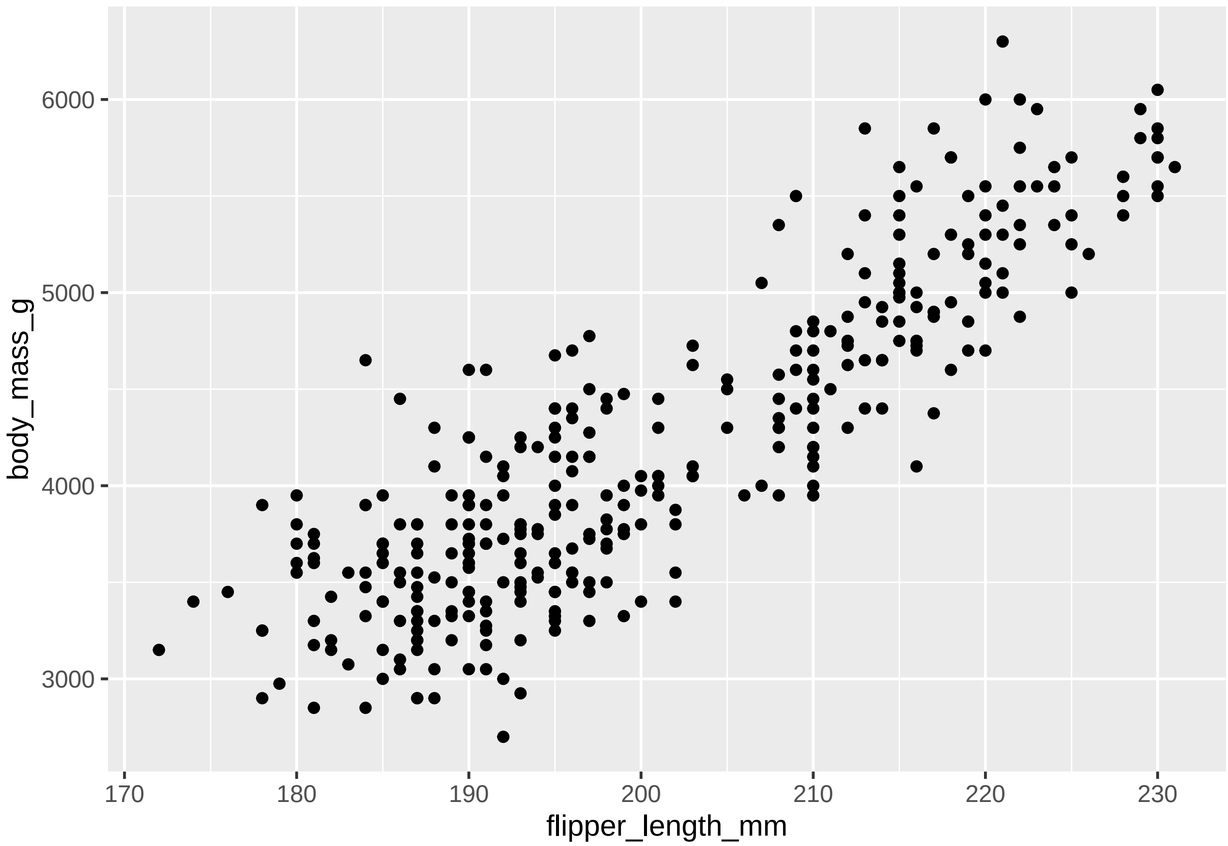
`mapping` argument is always definend in the `aes()` function, and `x`, `y` areguments of `aes()` specify which variables to map to the x and y axes.

We need to define a **geom**: the geometrical object that a plot uses to represent data.

- geom_bar()
- geom_line()
- geom_point()
- geom_boxplot()
- etc…

```
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g)
) +
  geom_point()
```

```
## Warning: Removed 2 rows containing missing values (`geom_point()`).
```



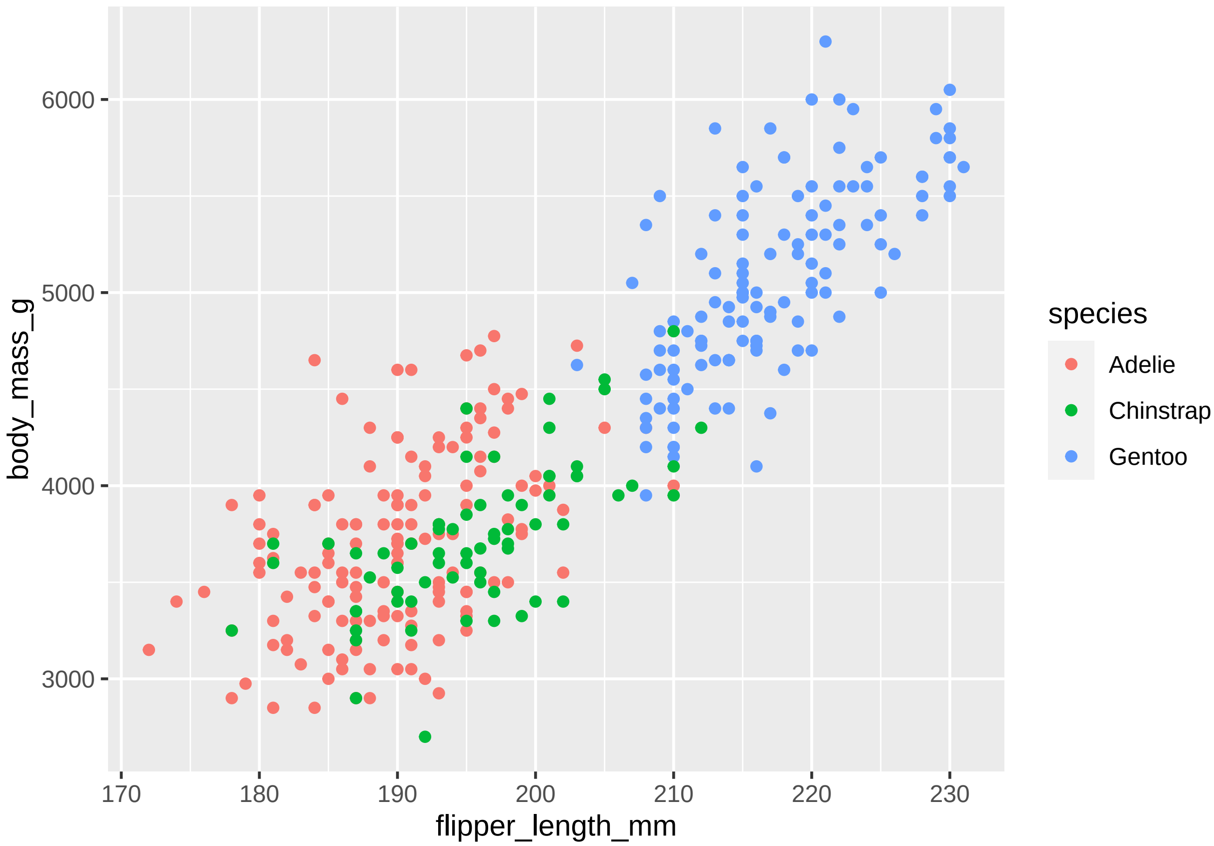
Warning message: ggplot2 subscribes to the philosophy that missing values should never silently go missing.

- Q: What does the relationship between flipper length and body mass look like?
- A: The relationship appears to be positive, fairly linear, and moderately strong. Penguins with longer flippers are generally larger in terms of their body mass.

Adding aesthetics and layers It is always a good idea to be skeptical of any apparent relationship between two variables and ask if there may be other variables that explain or change the nature of this apparent relationship.
For example, does the relationship between flipper length and body mass differ by species?

```
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm,
                y = body_mass_g,
                color = species)
) +
  geom_point()
```

```
## Warning: Removed 2 rows containing missing values (`geom_point()`).
```



Scaling: When a categorical variable is mapped to an aesthetic, ggplot2 will automatically assign a unique value of the aesthetic to each unique level of the variable. ggplot2 will also add a legend that explains which values correspond to which levels

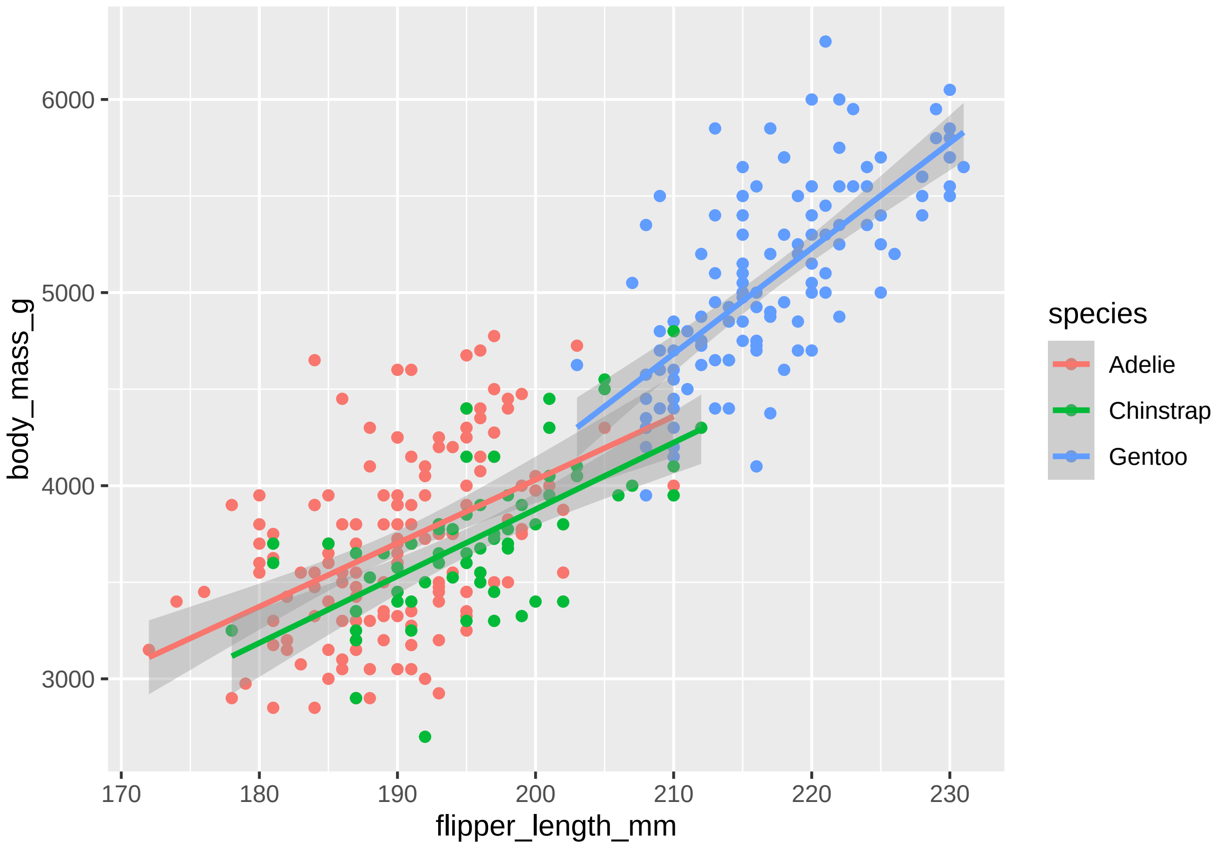
Let's add one more layer: a smooth curve displaying the relationship between body mass and flipper length.

```
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm,
                y = body_mass_g,
                color = species)
) +
  geom_point() +
  geom_smooth(method = 'lm')
```

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

```
## Warning: Removed 2 rows containing non-finite values (`stat_smooth()`).
```

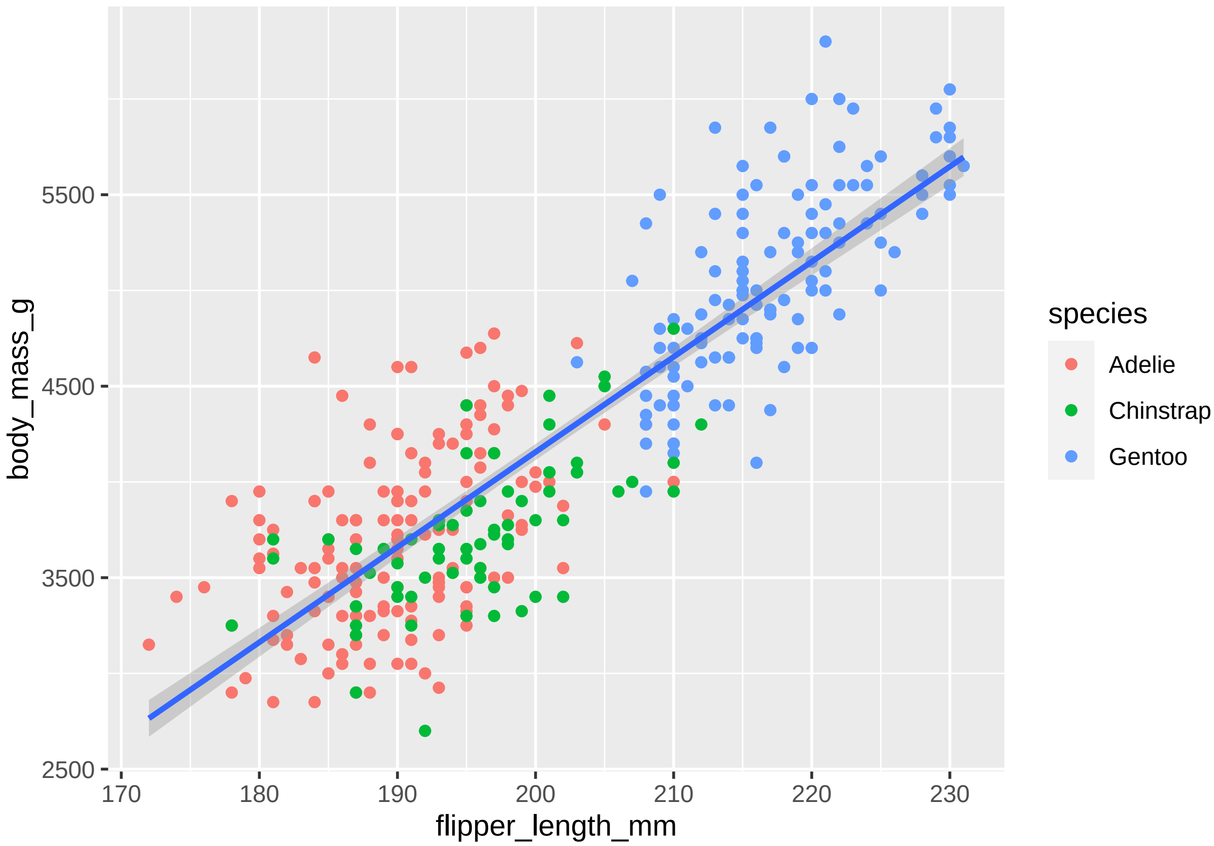
```
## Warning: Removed 2 rows containing missing values (`geom_point()`).
```



When aesthetic mappings are defined in `ggplot()`, at the global level, they are passed down to each of the subsequent geom layers of the plot.
However, each geom function in ggplot2 can also take a `mapping` argument, which allows for aesthetic mappings at the local level.

```
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm,
                y = body_mass_g)
) +
  geom_point(mapping = aes(color = species)) +
  geom_smooth(method = 'lm')
```

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

## Warning: Removed 2 rows containing non-finite values (`stat_smooth()`).

## Warning: Removed 2 rows containing missing values (`geom_point()`).
```



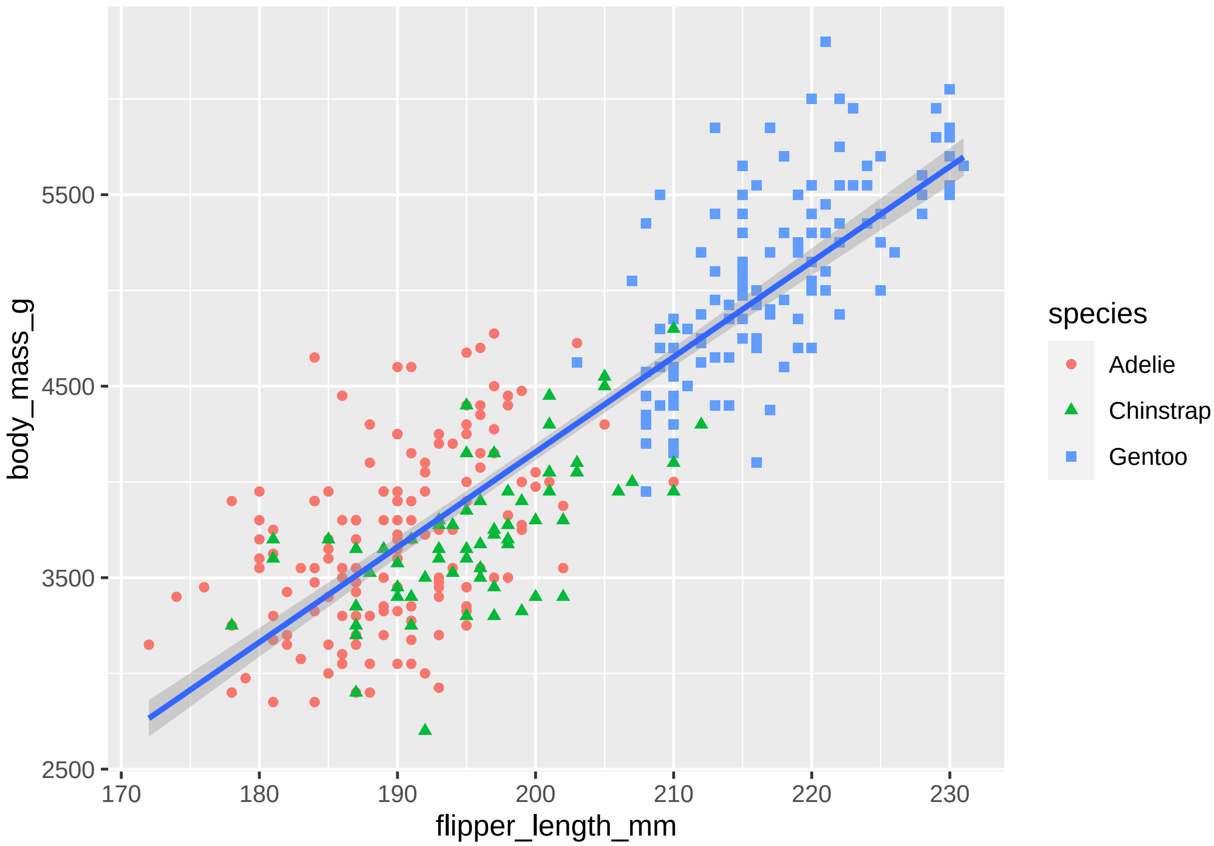
It's generally not a good idea to represent information using only colors on a plot, as people perceive colors differently due to color blindness or other color vision differences.

```r
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm,
                y = body_mass_g)
) +
  geom_point(mapping = aes(color = species, shape = species)) +
  geom_smooth(method = 'lm')
```

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

```
## Warning: Removed 2 rows containing non-finite values (`stat_smooth()`).
```

```
## Warning: Removed 2 rows containing missing values (`geom_point()`).
```



We can improve the labels of out plot using the `labs()` function in a new layer. arguments are

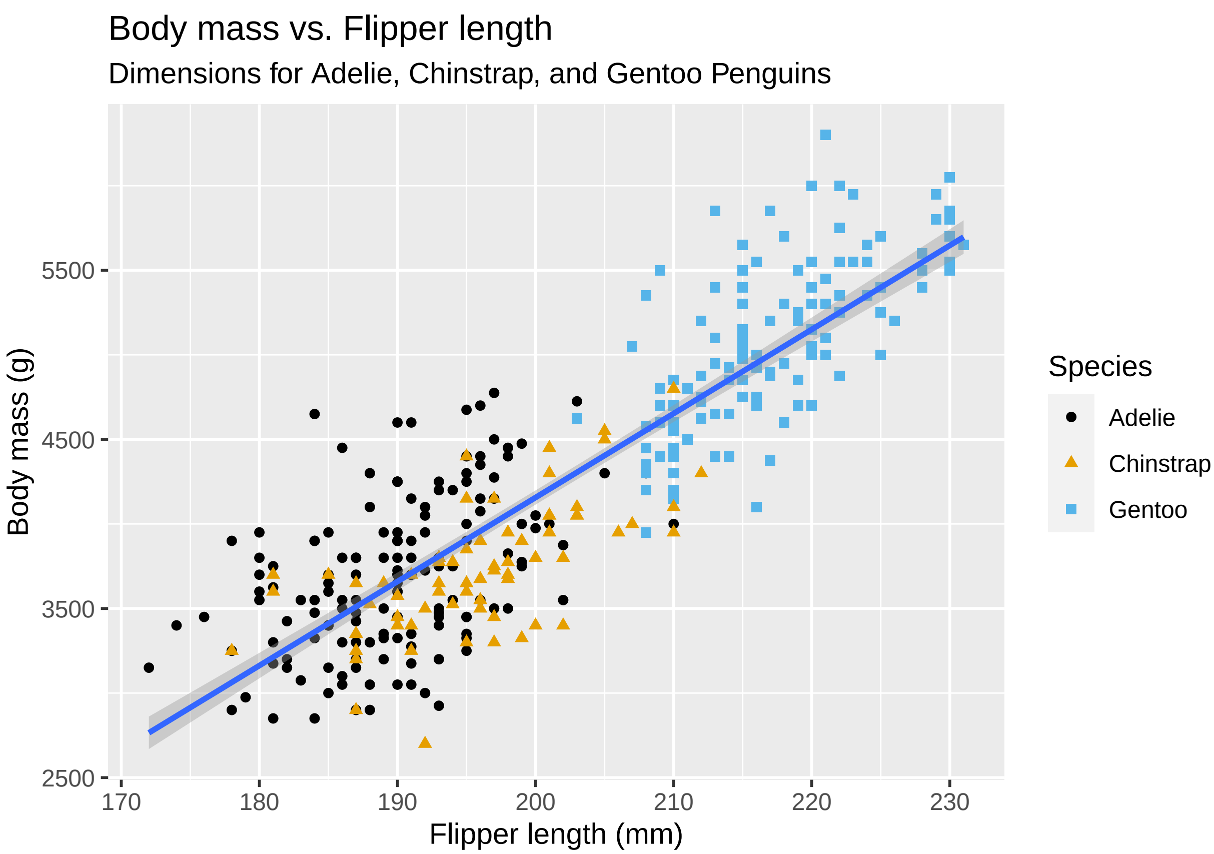
- `title`
- `subtitle`
- `x`
- `y`
- `color` and `shape`: define the label for the legend
- `scale_color_colorblind()`: imporve the color palette to be colorblind safe(from ggthemes package)

```r
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm,
                y = body_mass_g)
) +
  geom_point(mapping = aes(color = species, shape = species)) +
  geom_smooth(method = 'lm') +
  labs(
    title = 'Body mass vs. Flipper length',
    subtitle = 'Dimensions for Adelie, Chinstrap, and Gentoo Penguins',
    x = 'Flipper length (mm)',
    y = 'Body mass (g)',
    color = 'Species',
    shape = 'Species'
  ) +
  scale_color_colorblind()
```

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

```
## Warning: Removed 2 rows containing non-finite values (`stat_smooth()`).
```

```
## Warning: Removed 2 rows containing missing values (`geom_point()`).
```



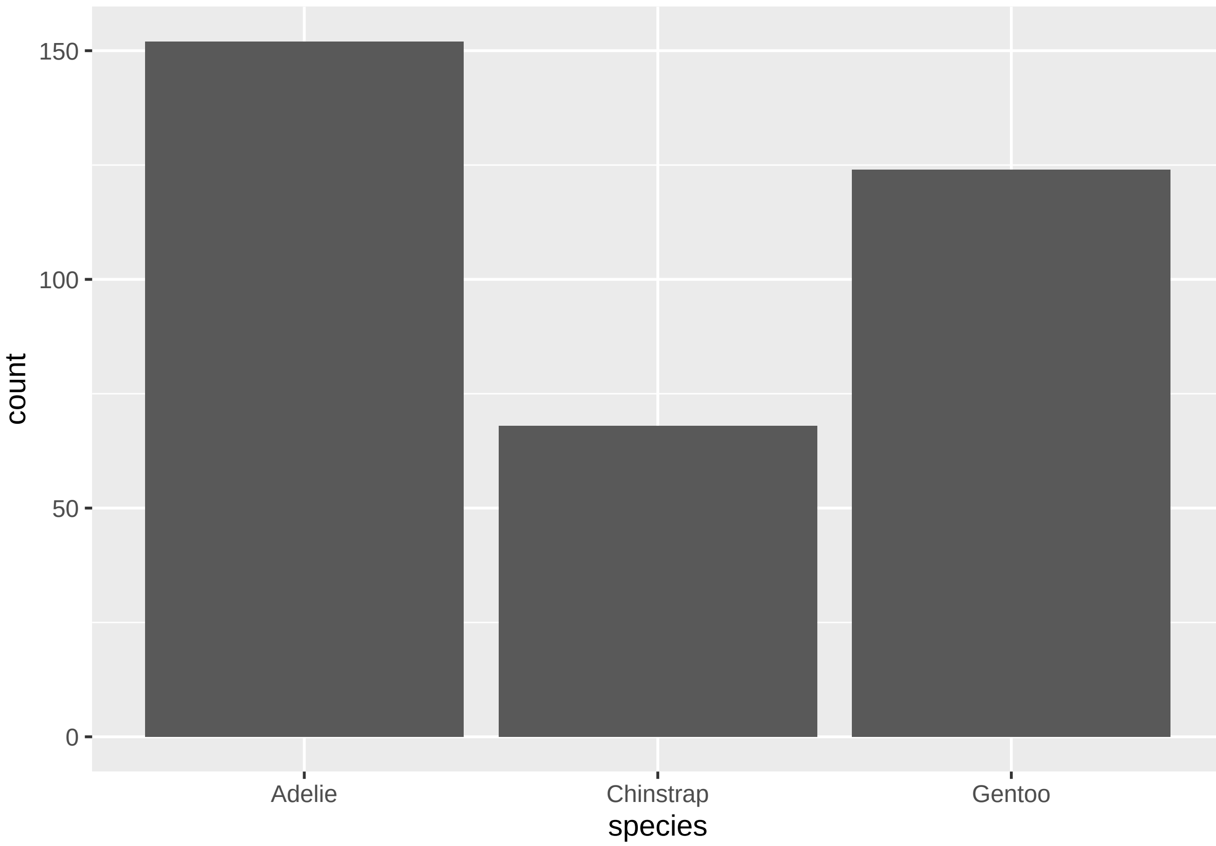
1.2. Visualizing distributions

How to visualize the distribution of a variable depends on the type of variable

- `Categorical`
- `Numerical`

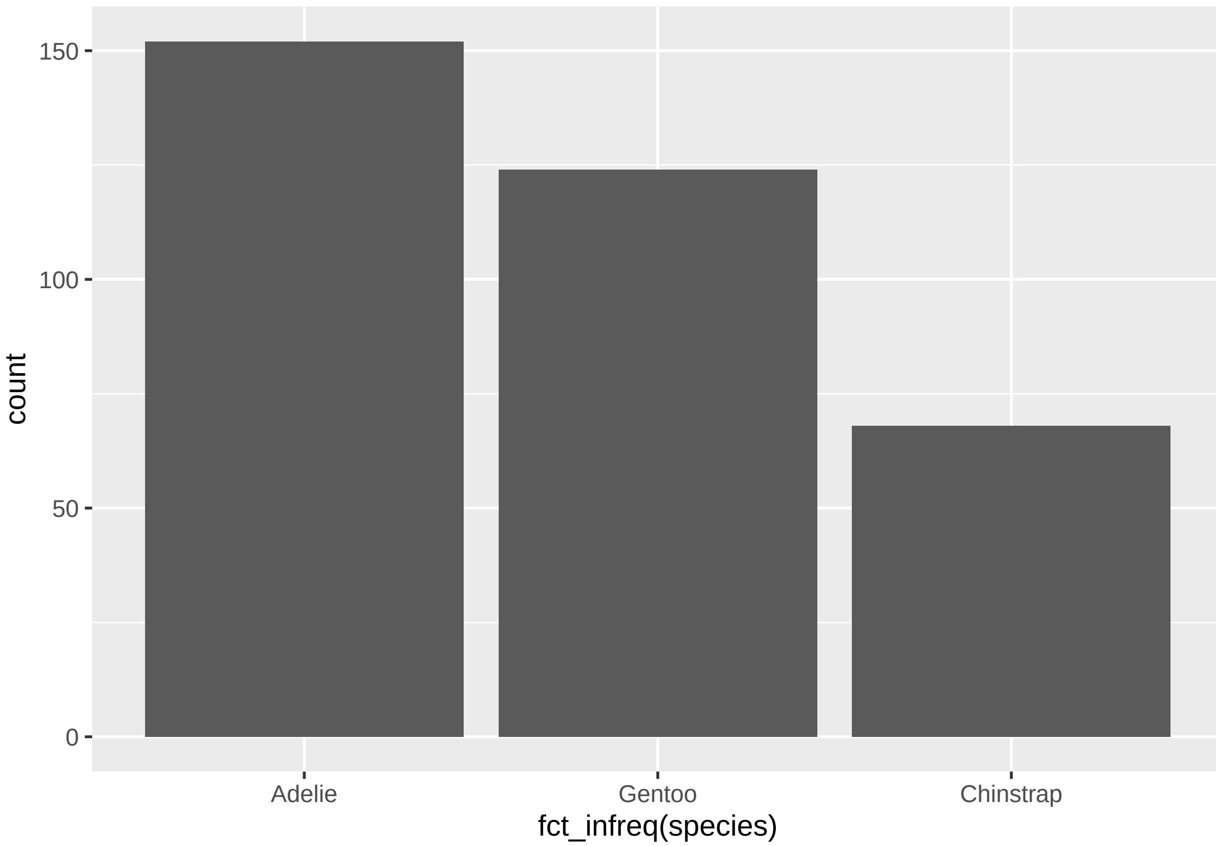
A categorical variable A variable is categorical if it can only take one of a small set of values. To examine the distribution of a categoriccal variable, we can use a bar chart.

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



In bar plots of categorical variables with non-ordered levels, its often preferable to reorder the bars based of their frequencies. It requires transforming the variable to a factor and then reordering the levels of that factor.

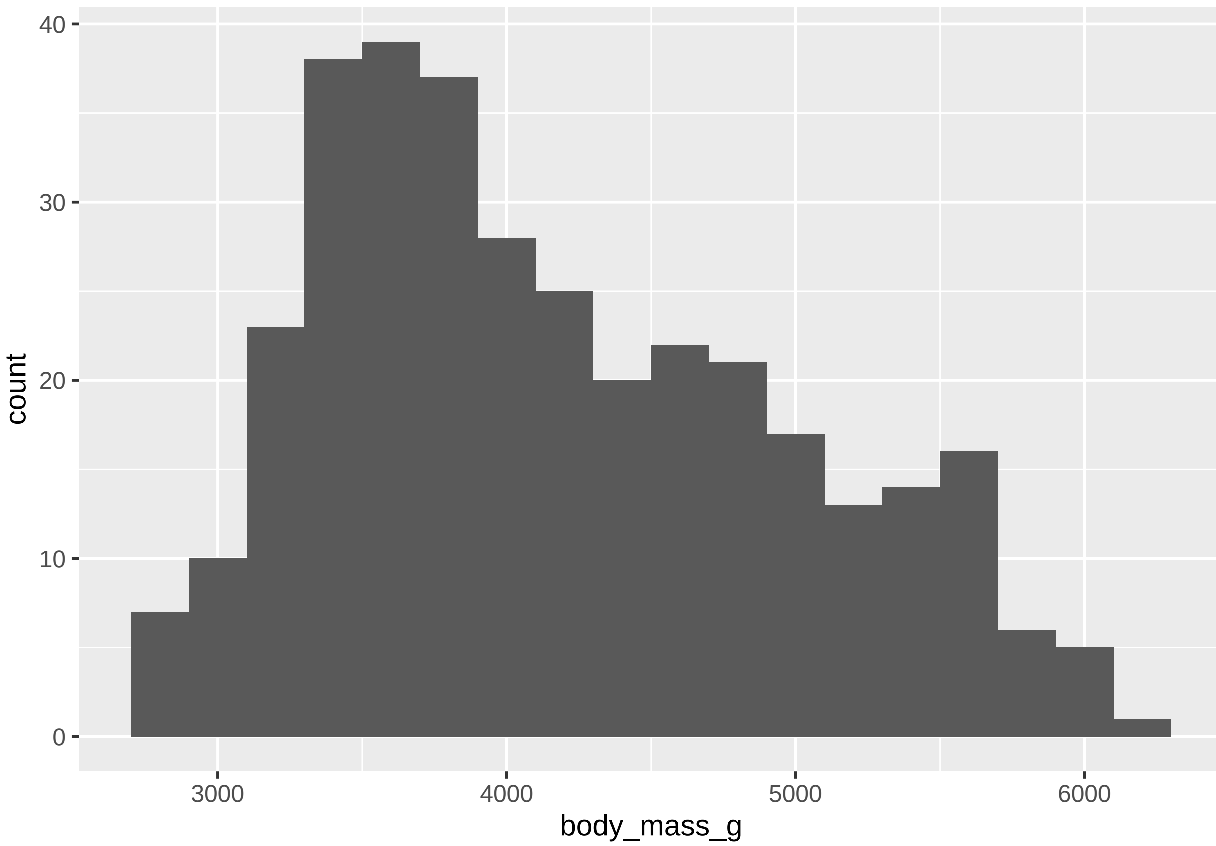
```
ggplot(penguins, aes(x = fct_infreq(species))) +
  geom_bar()
```



A numerical variable A variable is numerical or quantitative if it can take on a wide range of numerical values. Numerical variables can be continuous or discrete.
One commonly used visualization for distributions of continuous variable is a `histogram`

```
ggplot(penguins, aes(x = body_mass_g)) +
  geom_histogram(binwidth = 200)
```

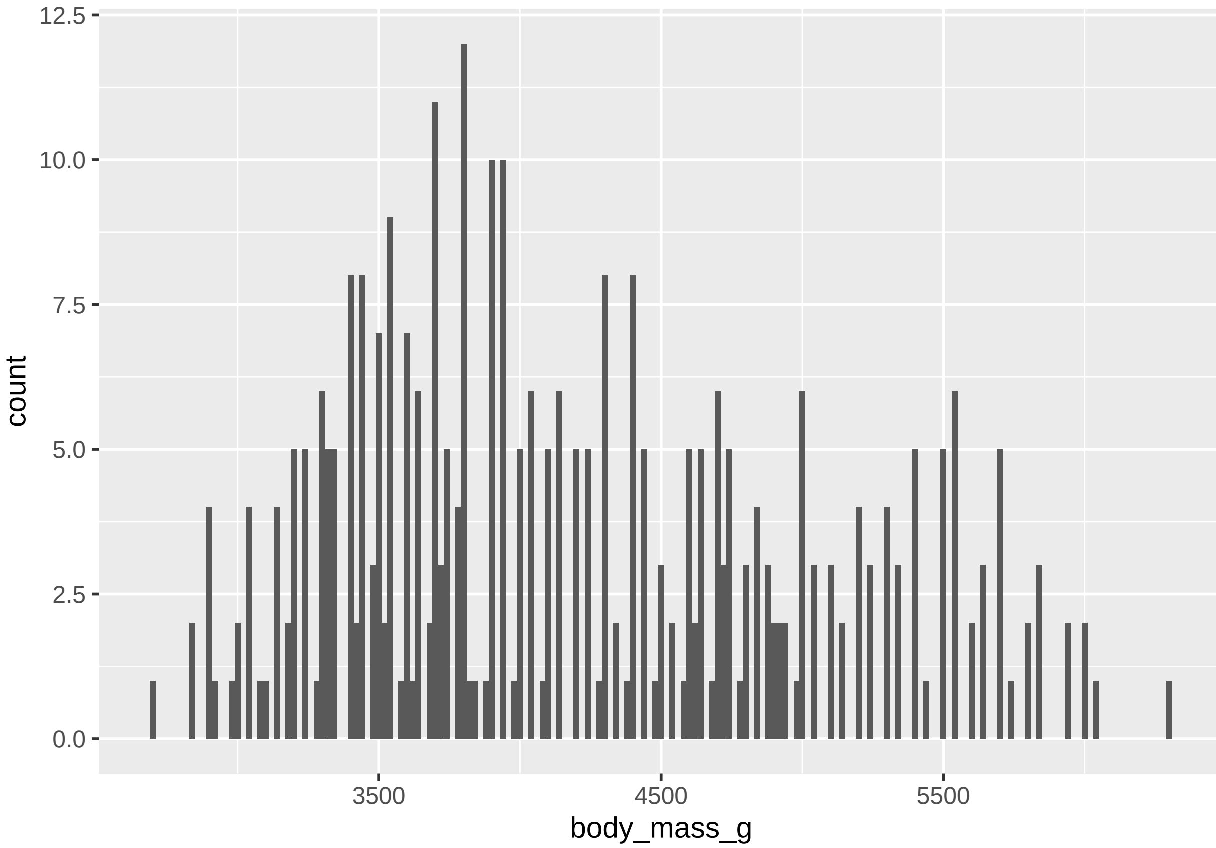
```
## Warning: Removed 2 rows containing non-finite values (`stat_bin()`).
```



A histogram divides the x-axis into equally spaced bins and then uses the height of a bar to display the number of observations that fall in each bin.
Since different binwidths can reveal different patterns, we have to explore a variety of binwidths when working with histogram.

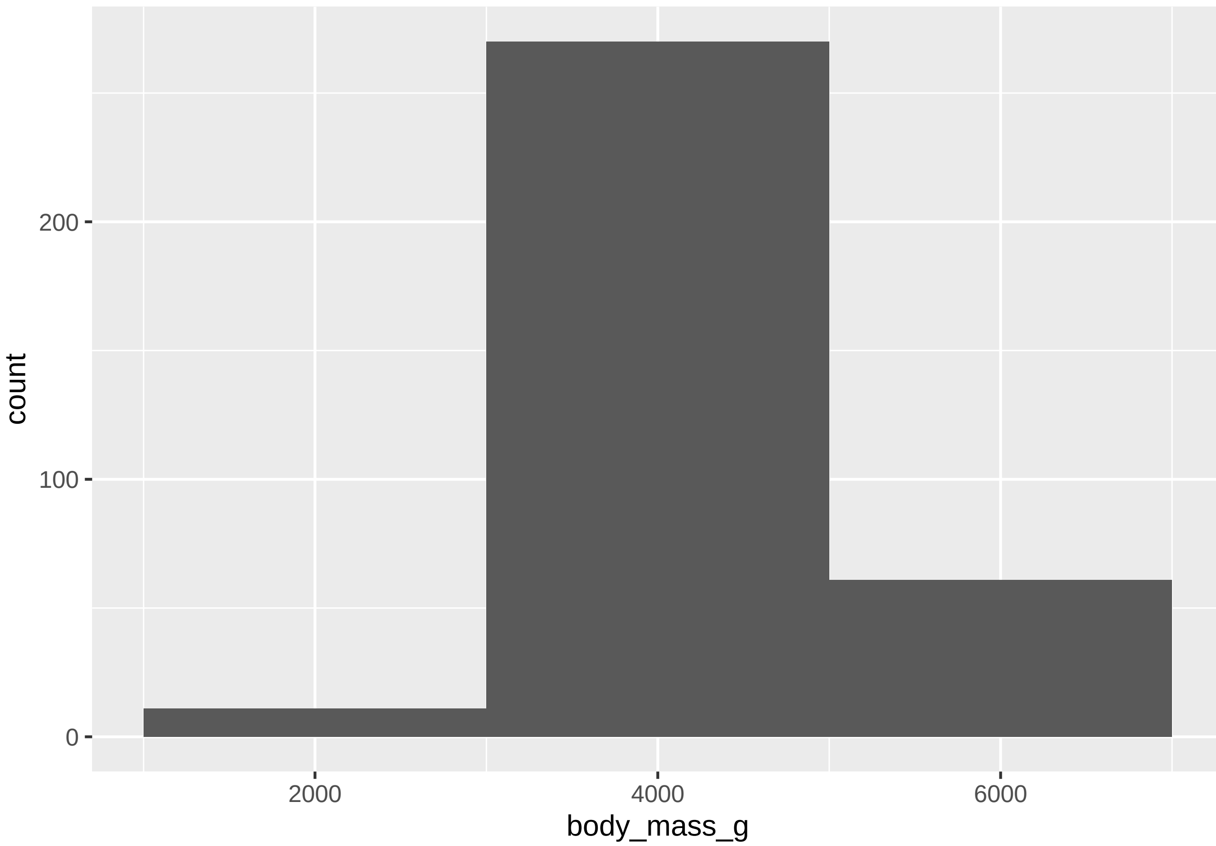
```
ggplot(penguins, aes(x = body_mass_g)) +
  geom_histogram(binwidth = 20)
```

```
## Warning: Removed 2 rows containing non-finite values (`stat_bin()`).
```




```
ggplot(penguins, aes(x = body_mass_g)) +
  geom_histogram(binwidth = 2000)
```

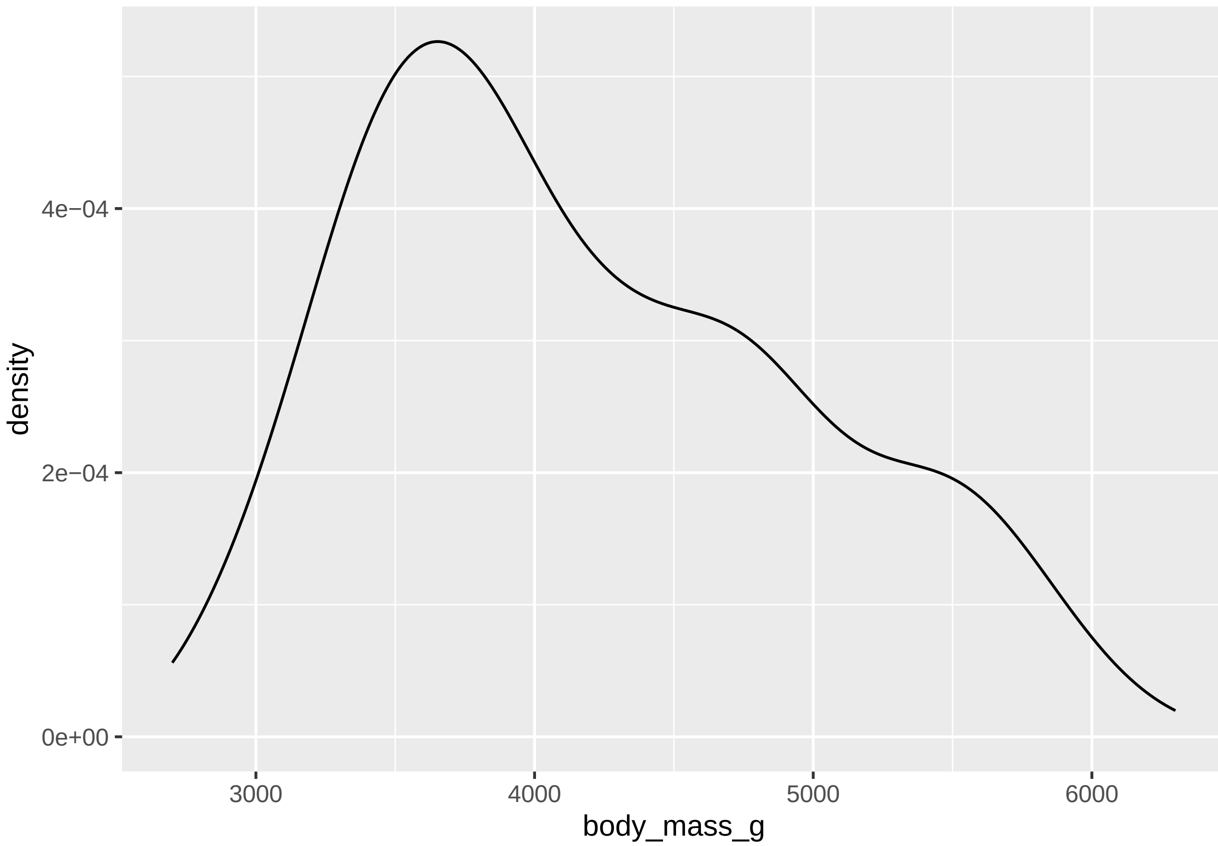
```
## Warning: Removed 2 rows containing non-finite values (`stat_bin()`).
```



An alternative visualization for distributions of numerical variables is a **`density plot`**. A density plot is a smoothed-out version of a histogram. It shows fewer details than a histogram but can make it easier to quickly glean the shape of the distribution, particularly with respect to modes and skewness.

```
ggplot(penguins, aes(x = body_mass_g)) +
  geom_density()
```

```
## Warning: Removed 2 rows containing non-finite values (`stat_density()`).
```



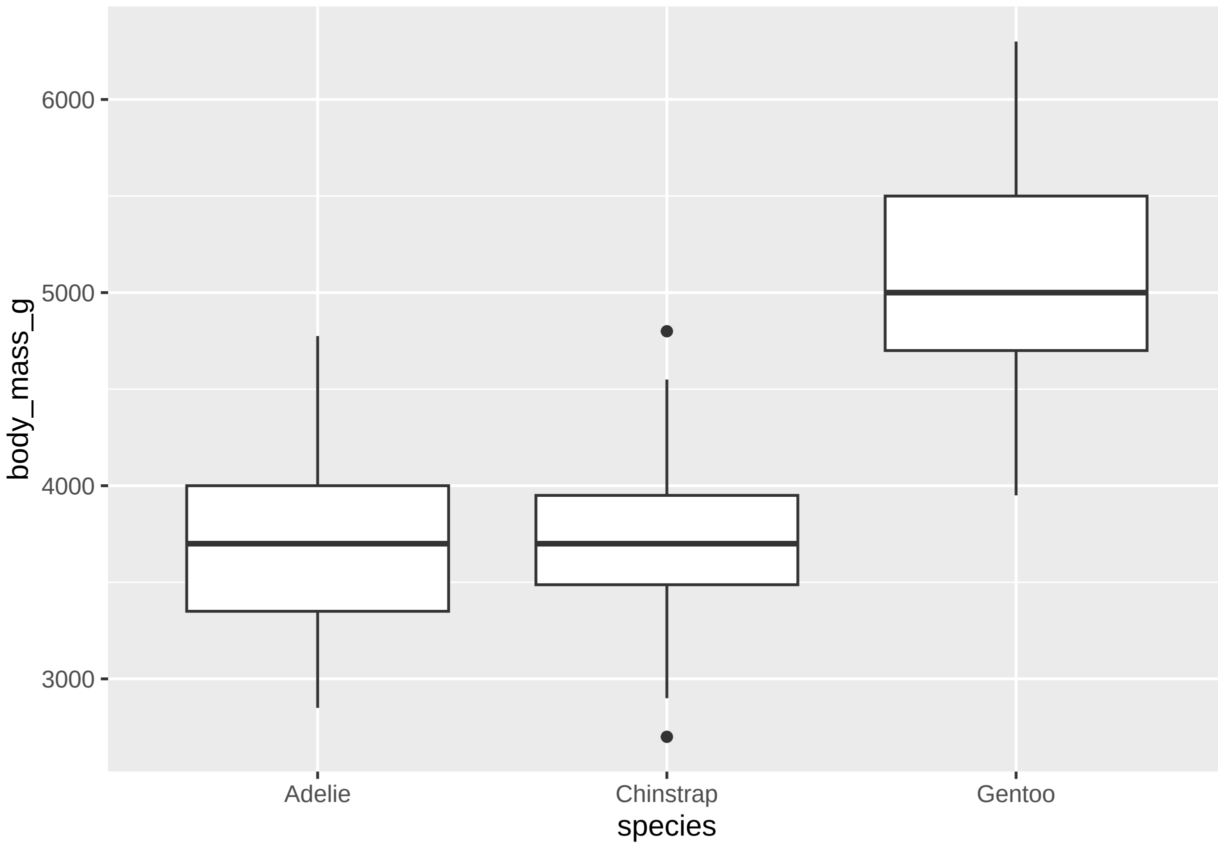
1.3 Visualizing relationhsips

To visualize a relationship we need to have at least two variables.

A numerical and a categorical variable To visualize the relationship between a numerical and a categorical variable we can use `side-by-side box plots`.

```
ggplot(penguins, aes(x = species, y = body_mass_g)) +
  geom_boxplot()
```

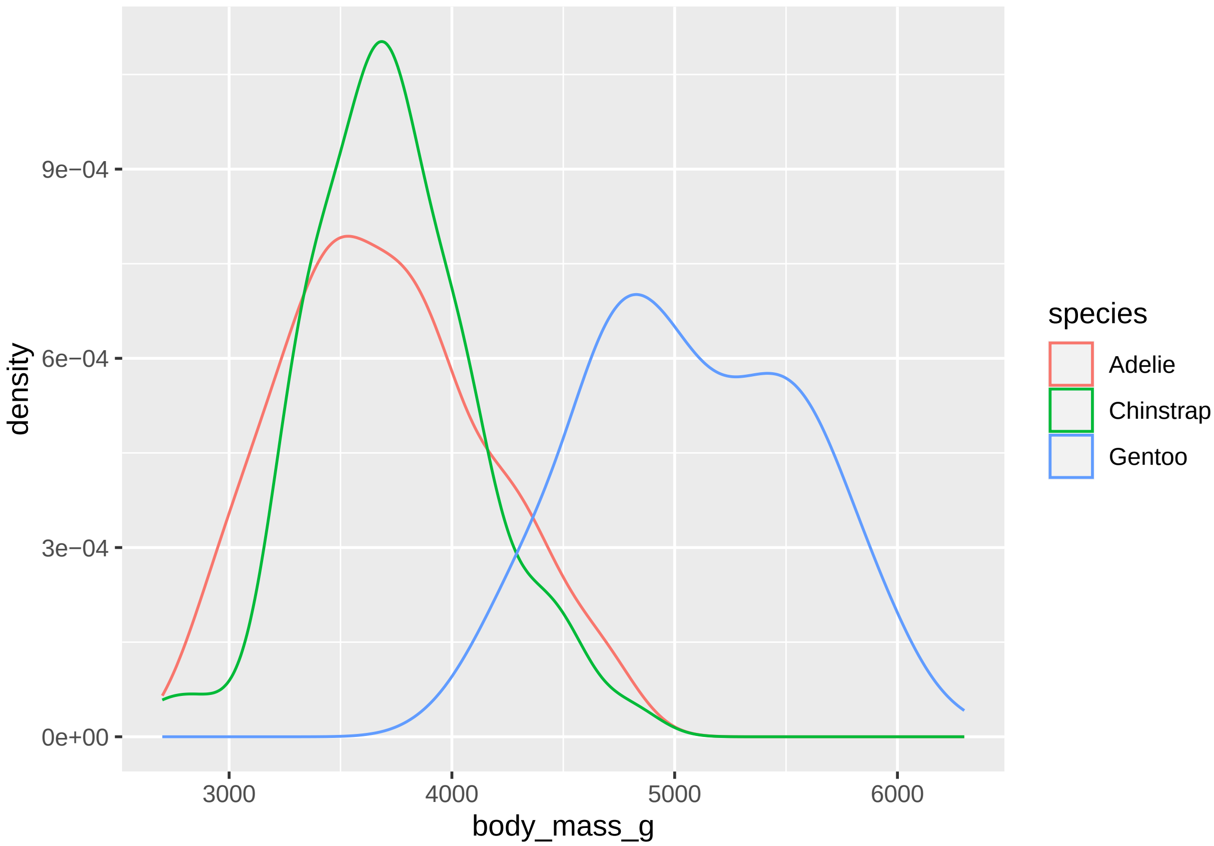
```
## Warning: Removed 2 rows containing non-finite values (`stat_boxplot()`).
```



Alternatively, we can make density plots with `geom_density()`.

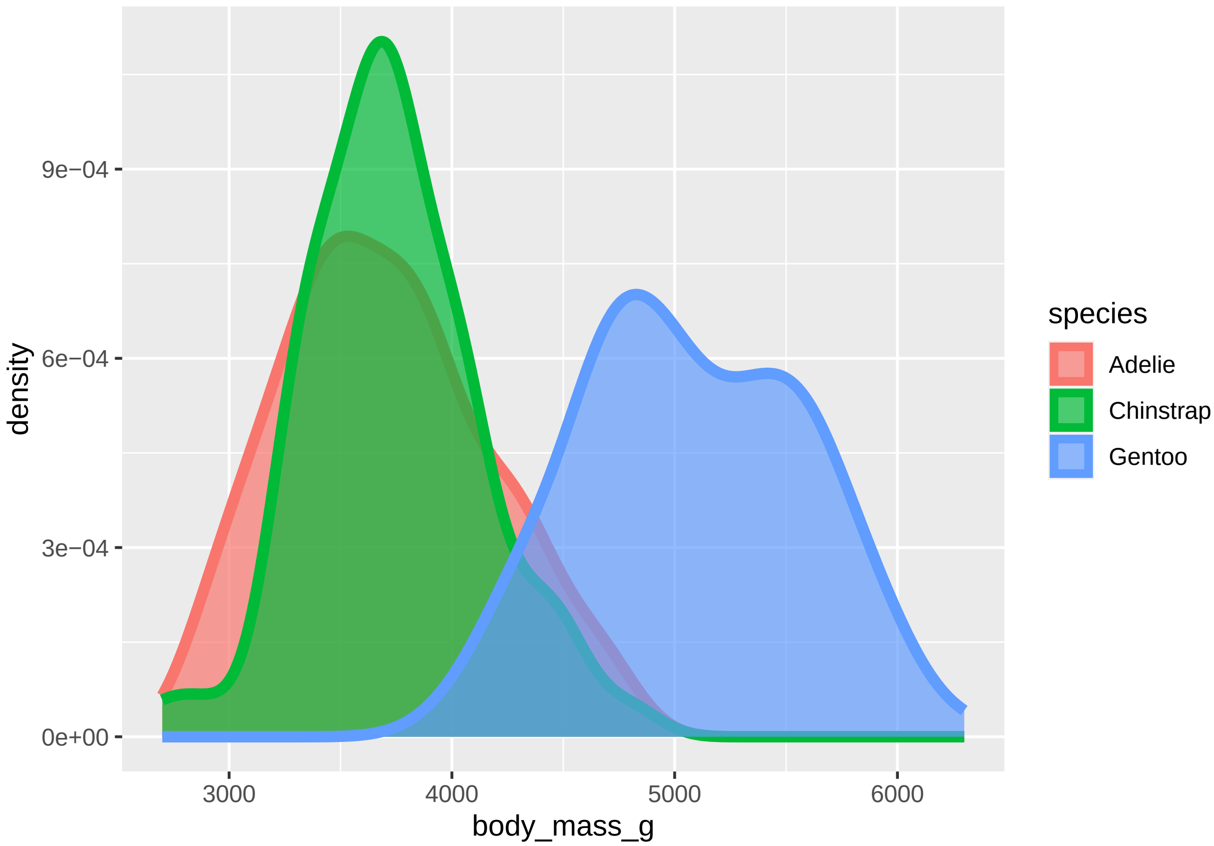
```
ggplot(penguins, aes(x = body_mass_g, color = species)) +
  geom_density()
```

```
## Warning: Removed 2 rows containing non-finite values (`stat_density()`).
```



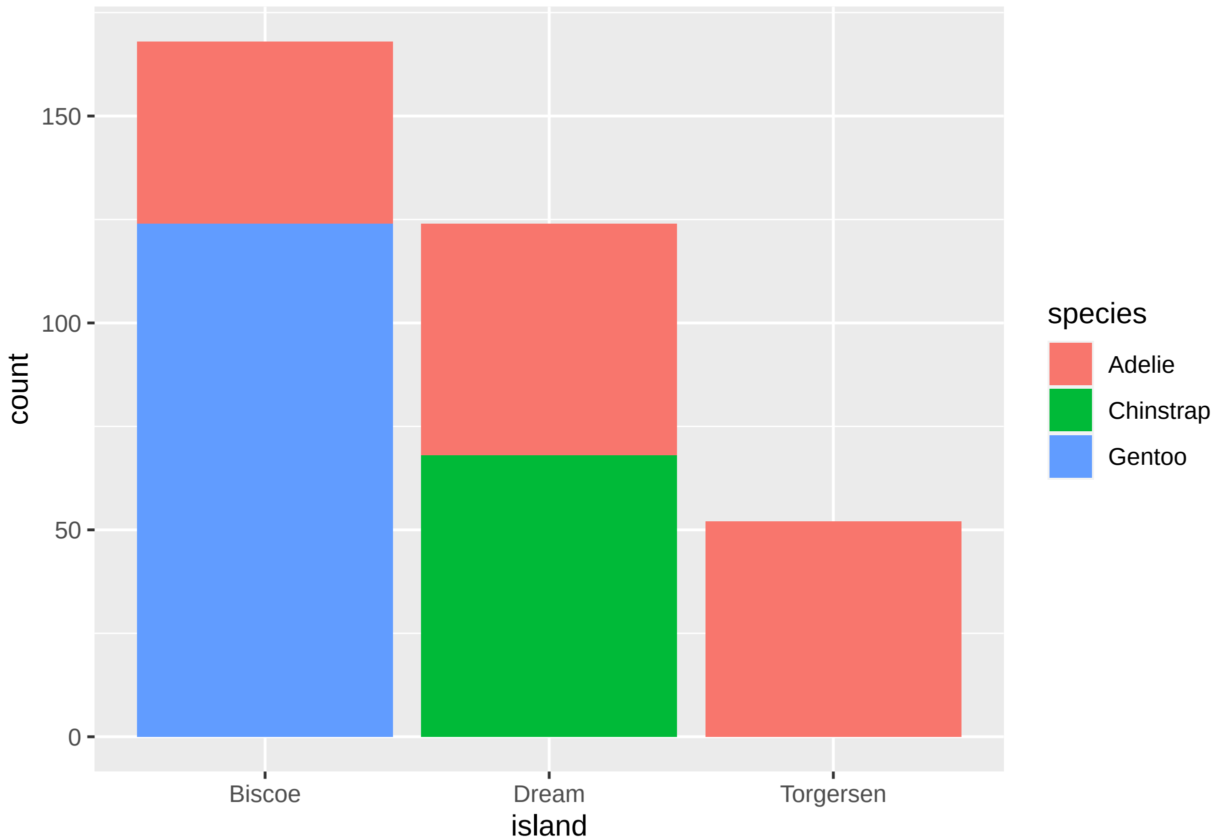

```
ggplot(penguins, aes(x = body_mass_g, color = species, fill = species)) +
  geom_density(linewidth = 2, alpha = 0.7)
```

```
## Warning: Removed 2 rows containing non-finite values (`stat_density()`).
```



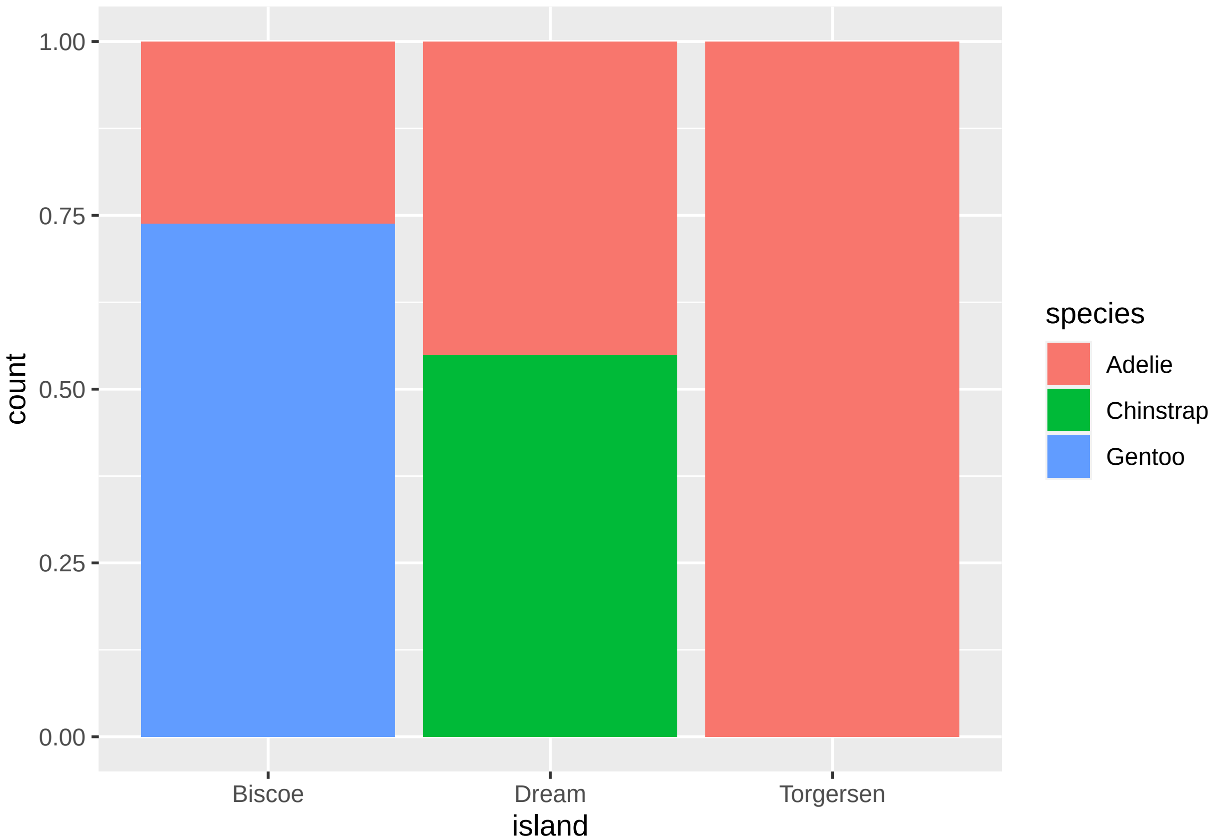
Two categorical variables We can use `stacked bar plot` to visualize the relationship between two categorical variables.

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



The second plot is a relative frequency plot. It is more useful for comparing species distributions across the islands since it's not affected by the unequal numbers of penguins across the islands.

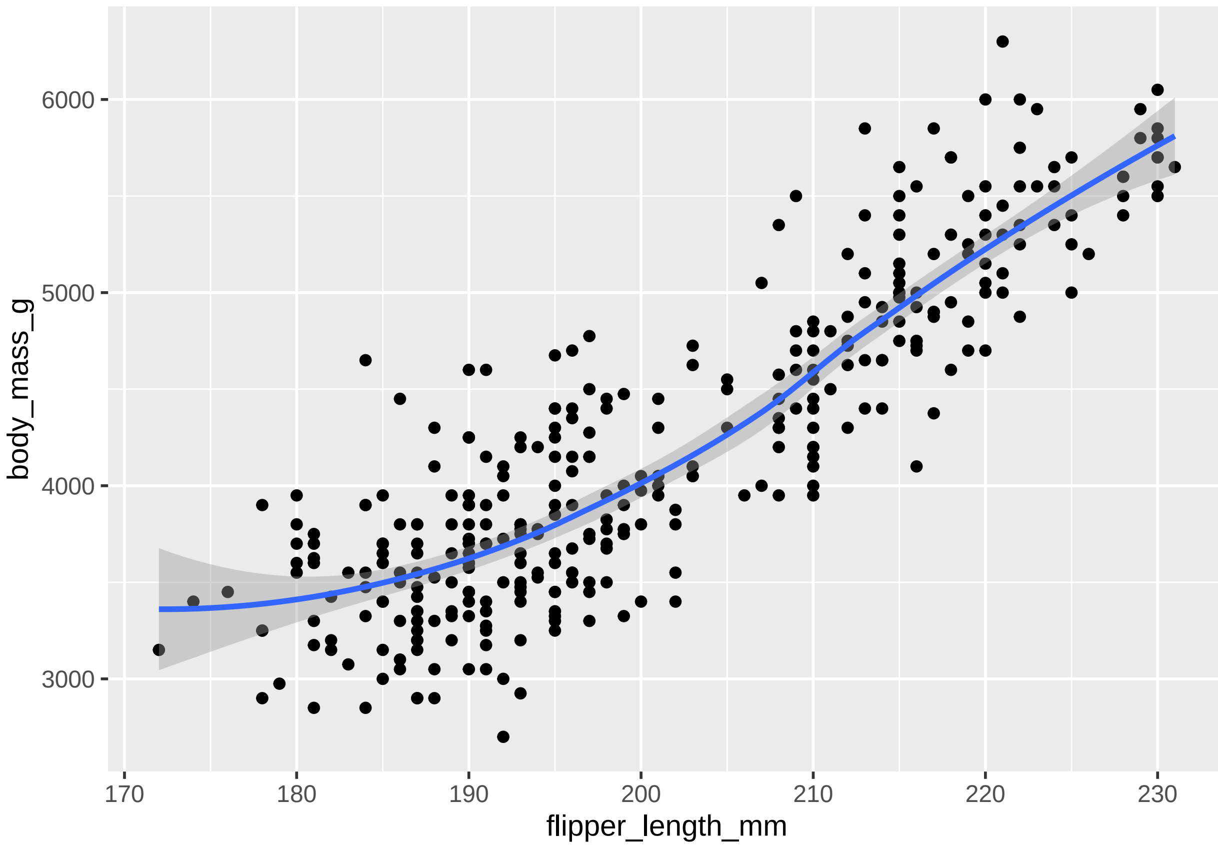
```
ggplot(penguins, aes(x = island, fill = species)) +
  geom_bar(position = 'fill')
```



Two numerical variables For visualizing the relationship between two numerical variables, we can use `scatter plot` and `smooth curves`.

```
ggplot(penguins, aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point() +
  geom_smooth()
```

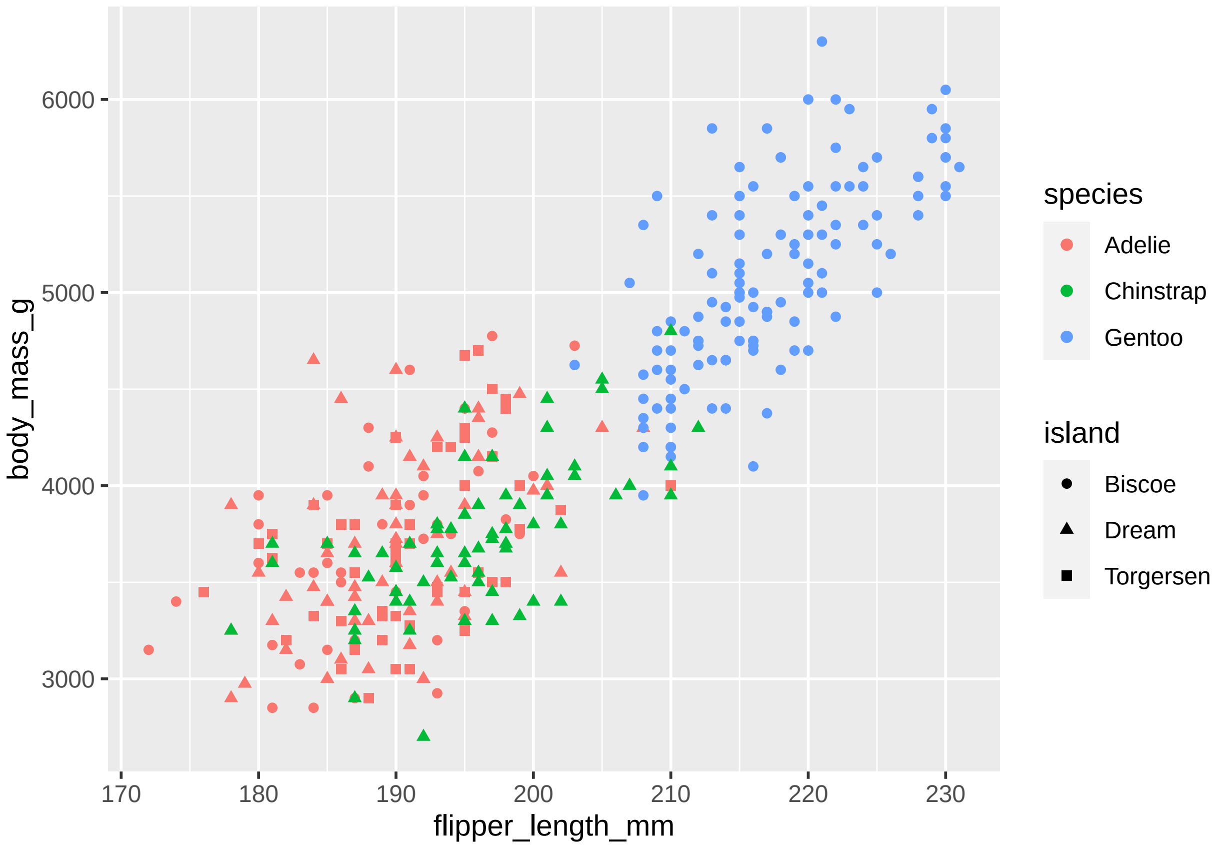
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'

## Warning: Removed 2 rows containing non-finite values (`stat_smooth()`).

## Warning: Removed 2 rows containing missing values (`geom_point()`).
```



Three or more variables We can incorporate more variables into a plot by mapping them to additional aesthetics.

```
ggplot(penguins, aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point(aes(color = species, shape = island))
```

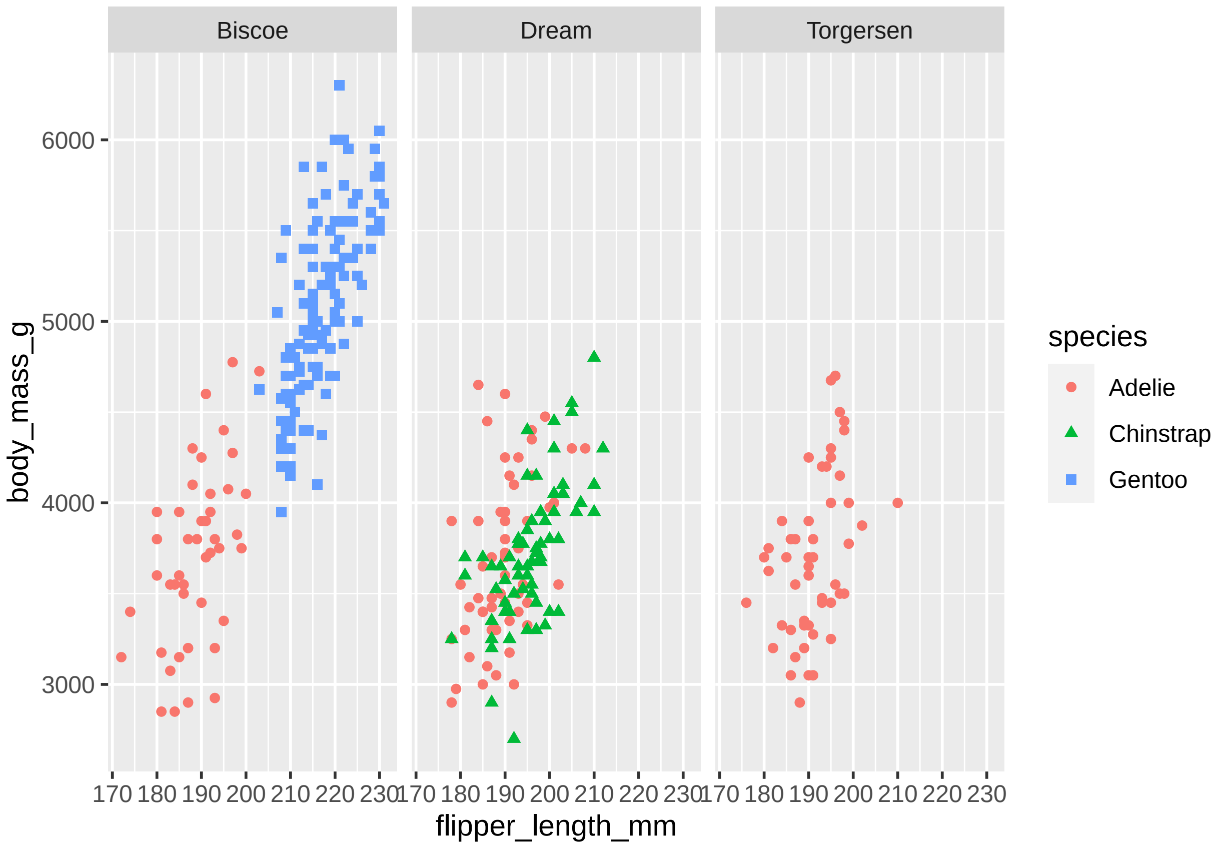
```
## Warning: Removed 2 rows containing missing values (`geom_point()`).
```



However adding too many aesthetic mappings to a plot makes it cluttered and difficult to make sense of. Another way is to split our plot into `facets`. To facet out plot by a single variable, use `facet_wrap()`.

```
ggplot(penguins, aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point(aes(color = species, shape = species)) +
  facet_wrap(~island)
```

```
## Warning: Removed 2 rows containing missing values (`geom_point()`).
```

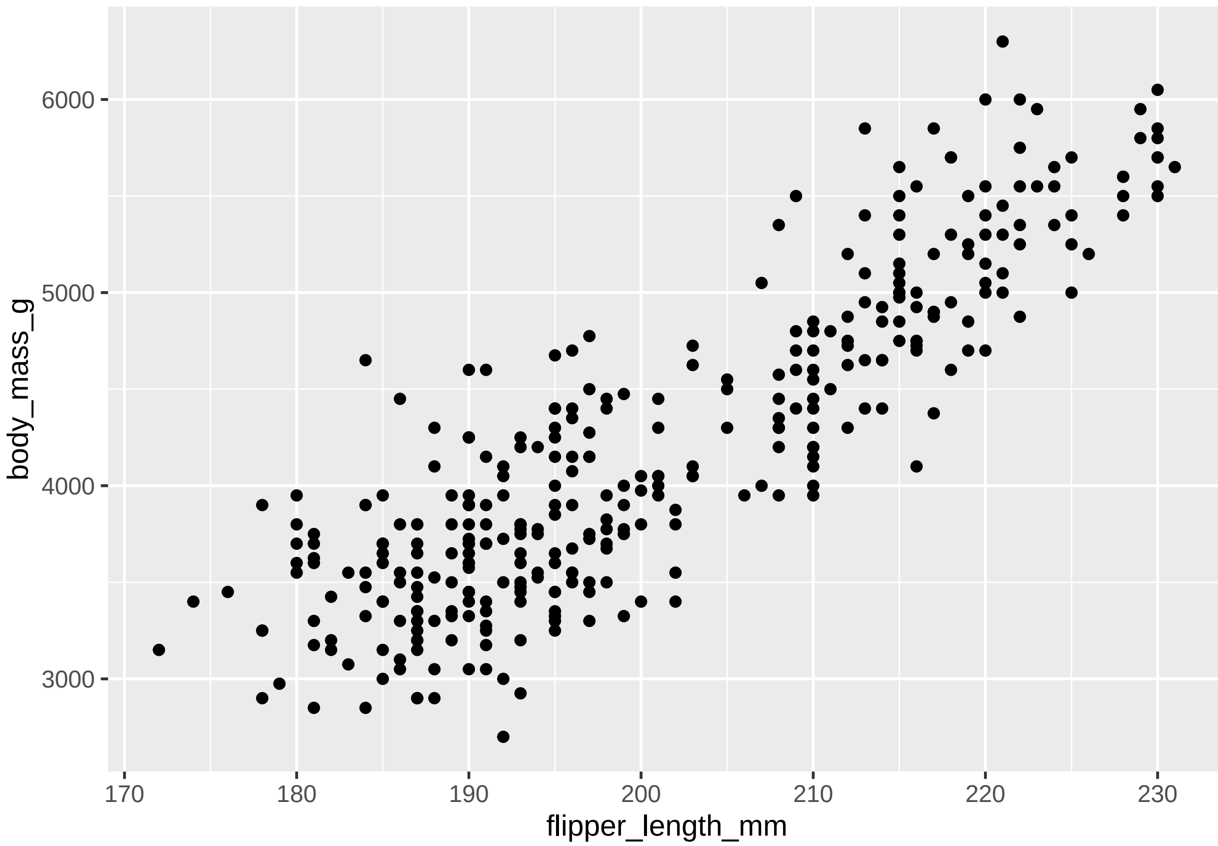


1.4 Saving plots

`ggsave()` will save the plot most recently created to disk. If we don't specify the `width` and `height` they will be taken from the dimensions of the current plotting device.

```
ggplot(penguins, aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point()
```

```
## Warning: Removed 2 rows containing missing values (`geom_point()`).
```




```r
# ggsave(filename = 'penguin-plot.png')
# ggsave(filename = 'penguin-plot.pdf')
```

Data transformation

1. Introduction

It's rare that we get the data in exactly the right form we need to make the graph we want. Often we'll need to create some new variables or summaries. Also we may want to rename the variable or reorder the observations.

Goals - `dplyr` package - overview of all the key tools for tranforming a data frame - understand pipe, which is important tool when combining verbs

```r
library(nycflights13)
library(tidyverse)
```

nycflights13 To explore the basic `dplyr` verbs, we're going to use `nycflights13::flights`.

```r
flights
```

```
## # A tibble: 336,776 x 19
```

```
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

`flights` is a tibble, a special type of data frame used by the tidyverse. The most important difference between tibbles and data frames is the way tibbles print. They are designed for large datasets, so they only show the first few rows and only the columns that fit on one screen.

- View(tibble): open an interactive scrollable and filterable view
- print(tibble, width = Inf): show all columns
- glimpse(tibble)

```
glimpse(flights)
```

```
## Rows: 336,776
## Columns: 19
## $ year           <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2~
## $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
## $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
## $ dep_time       <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 558, ~
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 600, ~
## $ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2, -1~
## $ arr_time       <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 849,~
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 851,~
## $ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7, -1~
## $ carrier        <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6", "~
## $ flight         <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301, 4~
## $ tailnum        <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N394~
## $ origin         <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LGA",~
## $ dest           <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IAD",~
## $ air_time       <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149, 1~
## $ distance       <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 733, ~
## $ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6~
## $ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59, 0~
## $ time_hour      <dttm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013-01-01 0~
```

dplyr basics   Common rules of dplyr

- The first argument is always a data frame

- The subsequent arguments typically describe which columns to operate on, using the vairate names
- The output is always a new data frame

pipe operator `|>` - `x |> f(y)`: f(x, y) - `x |> f(y) |> g(z)`: g(f(x, y), z)

```
flights |>
  filter(dest == 'IAH') |>
  group_by(year, month, day) |>
  summarize(
    arr_delay = mean(arr_delay, na.rm = T)
  )
```

```
## `summarise()` has grouped output by 'year', 'month'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 365 x 4
## # Groups:   year, month [12]
##     year month   day arr_delay
##    <int> <int> <int>     <dbl>
##  1  2013     1     1     17.8
##  2  2013     1     2      7
##  3  2013     1     3     18.3
##  4  2013     1     4     -3.2
##  5  2013     1     5     20.2
##  6  2013     1     6      9.28
##  7  2013     1     7     -7.74
##  8  2013     1     8      7.79
##  9  2013     1     9     18.1
## 10  2013     1    10      6.68
## # i 355 more rows
```

dplyr's verbs are organized into four groups based on what they operate on:

- rows
- columns
- groups
- tables

2. Rows

The most important verbs that operate on rows of a dataset are

- `filter()`
- `arrange()`
- `distinct()`

filter() `filter()` allows us to keep rows based on the values of the columns. When we run `filter()`, dplyr executes the filtering operation, creating a new data frame. It doesn't modify the existing dataset. So if we want to save the result, we must use the assignment operator `<-`.

arguments are:

- data frame
- conditions

```r
# departed more than 120 minutes late
flights |>
  filter(dep_delay > 120)
```

```
## # A tibble: 9,723 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      848           1835       853     1001           1950
##  2  2013     1     1      957            733       144     1056            853
##  3  2013     1     1     1114            900       134     1447           1222
##  4  2013     1     1     1540           1338       122     2020           1825
##  5  2013     1     1     1815           1325       290     2120           1542
##  6  2013     1     1     1842           1422       260     1958           1535
##  7  2013     1     1     1856           1645       131     2212           2005
##  8  2013     1     1     1934           1725       129     2126           1855
##  9  2013     1     1     1938           1703       155     2109           1823
## 10  2013     1     1     1942           1705       157     2124           1830
## # i 9,713 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

we can also use `<` `<=` `>` `>=` `==` `!=` and combine conditions with `&` , `|`. There is a useful shortcut when we are combining | and ==: `%in%`.

```r
# flights that departed on January 1
flights |>
  filter(month == 1 & day == 1)
```

```
## # A tibble: 842 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # i 832 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```r
# flights that departed in January or Februray
flights |>
  filter(month == 1 | month == 2)
```

```
## # A tibble: 51,955 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # i 51,945 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
flights |>
  filter(month %in% c(1, 2))
```

```
## # A tibble: 51,955 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # i 51,945 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
jan1 <- flights |>
  filter(month == 1 & day == 1)
jan1
```

```
## # A tibble: 842 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
```

```
## 8 2013 1 1 557 600 -3 709 723
## 9 2013 1 1 557 600 -3 838 846
## 10 2013 1 1 558 600 -2 753 745
## # i 832 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

arrange()

`arrange()` changes the order of the rows based on the value of the columns. If we provide more than one columns name, each additional column will be used to break ties in the values of preceding columns. Ascending is defualt and when we want to order by descending, use desc(column name).

arguments are:

- data frame
- set of columns

```r
#
flights |>
  arrange(year, month, day, dep_time)
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```r
#
flights |>
  arrange(desc(dep_delay))
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     9      641            900      1301     1242           1530
##  2  2013     6    15     1432           1935      1137     1607           2120
##  3  2013     1    10     1121           1635      1126     1239           1810
##  4  2013     9    20     1139           1845      1014     1457           2210
```

```
## 5 2013 7 22 845 1600 1005 1044 1815
## 6 2013 4 10 1100 1900 960 1342 2211
## 7 2013 3 17 2321 810 911 135 1020
## 8 2013 6 27 959 1900 899 1236 2226
## 9 2013 7 22 2257 759 898 121 1026
## 10 2013 12 5 756 1700 896 1058 2020
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

distinct()

`distinct()` finds all the unique rows in a dataset. However, most of the time, we'll want the distinct combination of some variables, so we can also optionally supply column names. If we want to keep other columns when filtering for unique rows, we can use the `.keep_all = T`

```
# remove duplicate rows
flights |>
  distinct()
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
# find all unique origin and destination pairs
flights |>
  distinct(origin, dest)
```

```
## # A tibble: 224 x 2
##    origin dest
##    <chr>  <chr>
##  1 EWR    IAH
##  2 LGA    IAH
##  3 JFK    MIA
##  4 JFK    BQN
##  5 LGA    ATL
##  6 EWR    ORD
```

```
## 7 EWR    FLL
## 8 LGA    IAD
## 9 JFK    MCO
## 10 LGA   ORD
## # i 214 more rows
```

```
flights |>
  distinct(origin, dest, .keep_all = T)
```

```
## # A tibble: 224 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # i 214 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
# count(): find the number of occurrences
# sort = T: arrange then in descending order of number of occurrences
flights |>
  count(origin, dest, sort = T)
```

```
## # A tibble: 224 x 3
##    origin dest      n
##    <chr>  <chr> <int>
##  1 JFK    LAX   11262
##  2 LGA    ATL   10263
##  3 LGA    ORD    8857
##  4 JFK    SFO    8204
##  5 LGA    CLT    6168
##  6 EWR    ORD    6100
##  7 JFK    BOS    5898
##  8 LGA    MIA    5781
##  9 JFK    MCO    5464
## 10 EWR    BOS    5327
## # i 214 more rows
```

3. columns

There are four important verbs that affect the columns.

- mutate()

- `select()`
- `rename()`
- ‘relocate()“

mutate() The job of `mutate()` is to add new columns that are calculated from the existing columns.

By default, `mutate()` adds new columns on the right hand side of our dataset. `.before` argument add the variables to the left hand side. Also we can use `.after` argument and both in `.before` and `.after` we can use variable name instead of a position.

Alternatively, we can control which variables are kept with the `.keep` argument.

```
flights |>
  mutate(
    gain = dep_delay - arr_delay,
    speed = distance / air_time * 60
  )
```

```
## # A tibble: 336,776 x 21
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # i 336,766 more rows
## # i 13 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>, gain <dbl>, speed <dbl>
```

```
flights |> mutate(
  gain = dep_delay - arr_delay,
  speed = distance / air_time * 60,
  .before = 1
)
```

```
## # A tibble: 336,776 x 21
##     gain speed  year month   day dep_time sched_dep_time dep_delay arr_time
##    <dbl> <dbl> <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1    -9  370.  2013     1     1      517            515         2      830
##  2   -16  374.  2013     1     1      533            529         4      850
##  3   -31  408.  2013     1     1      542            540         2      923
##  4    17  517.  2013     1     1      544            545        -1     1004
##  5    19  394.  2013     1     1      554            600        -6      812
##  6   -16  288.  2013     1     1      554            558        -4      740
##  7   -24  404.  2013     1     1      555            600        -5      913
##  8    11  259.  2013     1     1      557            600        -3      709
```

```
##  9     5  405.  2013     1     1      557            600        -3      838
## 10   -10  319.  2013     1     1      558            600        -2      753
## # i 336,766 more rows
## # i 12 more variables: sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
## #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
flights |>
  mutate(
    gain = dep_delay - arr_delay,
    speed = distance / air_time * 60,
    .after = day
  )
```

```
## # A tibble: 336,776 x 21
##     year month   day  gain speed dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int> <dbl> <dbl>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1    -9  370.      517            515         2      830
##  2  2013     1     1   -16  374.      533            529         4      850
##  3  2013     1     1   -31  408.      542            540         2      923
##  4  2013     1     1    17  517.      544            545        -1     1004
##  5  2013     1     1    19  394.      554            600        -6      812
##  6  2013     1     1   -16  288.      554            558        -4      740
##  7  2013     1     1   -24  404.      555            600        -5      913
##  8  2013     1     1    11  259.      557            600        -3      709
##  9  2013     1     1     5  405.      557            600        -3      838
## 10  2013     1     1   -10  319.      558            600        -2      753
## # i 336,766 more rows
## # i 12 more variables: sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
## #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
flights |>
  mutate(
    gain = dep_delay - arr_delay,
    hours = air_time / 60,
    gain_per_hour = gain / hours,
    .keep = 'used'
  )
```

```
## # A tibble: 336,776 x 6
##    dep_delay arr_delay air_time  gain hours gain_per_hour
##        <dbl>     <dbl>    <dbl> <dbl> <dbl>         <dbl>
##  1         2        11      227    -9 3.78          -2.38
##  2         4        20      227   -16 3.78          -4.23
##  3         2        33      160   -31 2.67         -11.6
##  4        -1       -18      183    17 3.05           5.57
##  5        -6       -25      116    19 1.93           9.83
##  6        -4        12      150   -16 2.5           -6.4
##  7        -5        19      158   -24 2.63          -9.11
##  8        -3       -14       53    11 0.883         12.5
##  9        -3        -8      140     5 2.33           2.14
## 10        -2         8      138   -10 2.3           -4.35
## # i 336,766 more rows
```

select() `select()` allows us to rapidly zoom in on a useful subset using operations based on the names of the variables.

- select columns by name

```
flights |>
  select(year, month, day)
```

```
## # A tibble: 336,776 x 3
##      year month   day
##     <int> <int> <int>
##   1  2013     1     1
##   2  2013     1     1
##   3  2013     1     1
##   4  2013     1     1
##   5  2013     1     1
##   6  2013     1     1
##   7  2013     1     1
##   8  2013     1     1
##   9  2013     1     1
##  10  2013     1     1
## # i 336,766 more rows
```

- select all columns between year and day

```
flights |>
  select(year:day)
```

```
## # A tibble: 336,776 x 3
##      year month   day
##     <int> <int> <int>
##   1  2013     1     1
##   2  2013     1     1
##   3  2013     1     1
##   4  2013     1     1
##   5  2013     1     1
##   6  2013     1     1
##   7  2013     1     1
##   8  2013     1     1
##   9  2013     1     1
##  10  2013     1     1
## # i 336,766 more rows
```

- select all columns except those from year to day

```
# can also use - instead of !
flights |>
  select(!year:day)
```

```
## # A tibble: 336,776 x 16
##    dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier
```

```
##       <int>          <int>     <dbl>    <int>          <int>     <dbl> <chr>
##   1      517            515         2      830            819        11 UA
##   2      533            529         4      850            830        20 UA
##   3      542            540         2      923            850        33 AA
##   4      544            545        -1     1004           1022       -18 B6
##   5      554            600        -6      812            837       -25 DL
##   6      554            558        -4      740            728        12 UA
##   7      555            600        -5      913            854        19 B6
##   8      557            600        -3      709            723       -14 EV
##   9      557            600        -3      838            846        -8 B6
## 10      558            600        -2      753            745         8 AA
## # i 336,766 more rows
## # i 9 more variables: flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

- select all columns that are characters

```
flights |>
  select(where(is.character))
```

```
## # A tibble: 336,776 x 4
##    carrier tailnum origin dest
##    <chr>   <chr>   <chr>  <chr>
##  1 UA      N14228  EWR    IAH
##  2 UA      N24211  LGA    IAH
##  3 AA      N619AA  JFK    MIA
##  4 B6      N804JB  JFK    BQN
##  5 DL      N668DN  LGA    ATL
##  6 UA      N39463  EWR    ORD
##  7 B6      N516JB  EWR    FLL
##  8 EV      N829AS  LGA    IAD
##  9 B6      N593JB  JFK    MCO
## 10 AA      N3ALAA  LGA    ORD
## # i 336,766 more rows
```

There are a number of helper functions we can use within `select()`

- `starts_with()`
- `ends_with()`
- `contains()`
- `num_range('x', 1:3)`

We can rename variables using `=`

```
flights |>
  select(tail_num = tailnum)
```

```
## # A tibble: 336,776 x 1
##    tail_num
##    <chr>
##  1 N14228
##  2 N24211
```

```
##   3 N619AA
##   4 N804JB
##   5 N668DN
##   6 N39463
##   7 N516JB
##   8 N829AS
##   9 N593JB
##  10 N3ALAA
## # i 336,766 more rows
```

```
flights |>
  rename(tail_num = tailnum)
```

rename()

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tail_num <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

relocate() Use `relocate()` to move variables around. By default `relocate()` moves variables to the front. We can also specify where to put them using `.before` and `.after` arguments just like in `mutate()`.

```
flights |>
  relocate(time_hour, air_time)
```

```
## # A tibble: 336,776 x 19
##    time_hour           air_time  year month   day dep_time sched_dep_time
##    <dttm>                 <dbl> <int> <int> <int>    <int>          <int>
##  1 2013-01-01 05:00:00      227  2013     1     1      517            515
##  2 2013-01-01 05:00:00      227  2013     1     1      533            529
##  3 2013-01-01 05:00:00      160  2013     1     1      542            540
##  4 2013-01-01 05:00:00      183  2013     1     1      544            545
##  5 2013-01-01 06:00:00      116  2013     1     1      554            600
##  6 2013-01-01 05:00:00      150  2013     1     1      554            558
##  7 2013-01-01 06:00:00      158  2013     1     1      555            600
##  8 2013-01-01 06:00:00       53  2013     1     1      557            600
```

```
##   9 2013-01-01 06:00:00      140  2013     1     1      557            600
##  10 2013-01-01 06:00:00      138  2013     1     1      558            600
## # i 336,766 more rows
## # i 12 more variables: dep_delay <dbl>, arr_time <int>, sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, distance <dbl>, hour <dbl>, minute <dbl>
```

```r
flights |>
  relocate(year:dep_time, .after = time_hour)
```

```
## # A tibble: 336,776 x 19
##    sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight
##             <int>     <dbl>    <int>          <int>     <dbl> <chr>    <int>
##  1            515         2      830            819        11 UA        1545
##  2            529         4      850            830        20 UA        1714
##  3            540         2      923            850        33 AA        1141
##  4            545        -1     1004           1022       -18 B6         725
##  5            600        -6      812            837       -25 DL         461
##  6            558        -4      740            728        12 UA        1696
##  7            600        -5      913            854        19 B6         507
##  8            600        -3      709            723       -14 EV        5708
##  9            600        -3      838            846        -8 B6          79
## 10            600        -2      753            745         8 AA         301
## # i 336,766 more rows
## # i 12 more variables: tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
## #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>, year <int>,
## #   month <int>, day <int>, dep_time <int>
```

```r
flights |>
  relocate(starts_with('arr'), .before = dep_time)
```

```
## # A tibble: 336,776 x 19
##     year month   day arr_time arr_delay dep_time sched_dep_time dep_delay
##    <int> <int> <int>    <int>     <dbl>    <int>          <int>     <dbl>
##  1  2013     1     1      830        11      517            515         2
##  2  2013     1     1      850        20      533            529         4
##  3  2013     1     1      923        33      542            540         2
##  4  2013     1     1     1004       -18      544            545        -1
##  5  2013     1     1      812       -25      554            600        -6
##  6  2013     1     1      740        12      554            558        -4
##  7  2013     1     1      913        19      555            600        -5
##  8  2013     1     1      709       -14      557            600        -3
##  9  2013     1     1      838        -8      557            600        -3
## 10  2013     1     1      753         8      558            600        -2
## # i 336,766 more rows
## # i 11 more variables: sched_arr_time <int>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```