A tutorial on basic scientific data analysis and visualization in R

Eliot McKinley

9/24/2021

Introduction

This tutorial gives examples of some common statistical and data visualization techniques that you ay be used to in Excel or Prism using R. The advantages of using R include the ability to work with larger data sets, better control of you analysis methods, the availability of online tutorials and primers for almost anything you'd like to do, and, most importantly, everything will look better than using Excel or Prism.

Topics include: 1. Loading Data 2. Exploring and Summarizing Data 3. Saving Data 4. Basic Data Visualization 5. T-tests, Wilcoxon tests, and ANOVA 6. Correlations 7. PCA and Dimensionality Reduction 8. (Slightly More) Advanced Data Visualization 9. Making publication quality plots and panels

We will be working in the tidyverse, which is an ecosystem for data science. https://www.tidyverse.org/

We will be making all plots using the package {ggplot2} rather than the base R plotting. {ggplot2} provides an intuitive and powerful system for data visualization that is more flexible and beautiful than base R.

First we will install (if necessary) and load the {tidyverse} and {palmerpenguins} packages.

```
# check if packages are loaded
packages = c("tidyverse", "palmerpenguins")
if (length(setdiff(packages, rownames(installed.packages()))) > 0) {
  install.packages(setdiff(packages, rownames(installed.packages())))
}
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.0     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

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

```
library(palmerpenguins)
```

1. Loading data

We are going to use the Palmer Penguins data set which is included in the package {palmerpenguins}. When the package is loaded you have access to the variable "penguins" which contains this data. You can assign it to a different variable name.

```
my_penguins = penguins
```

More often your data is not available from a package so you need to read the data in to R. If your data is saved as a csv file you can pass the file path to your data into the read_csv function in order to import into R.

```
my_penguins_csv = read_csv("./penguins.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   species = col_character(),
##   island = col_character(),
##   bill_length_mm = col_double(),
##   bill_depth_mm = col_double(),
##   flipper_length_mm = col_double(),
##   body_mass_g = col_double(),
##   sex = col_character(),
##   year = col_double()
## )
```

read_csv() prints the names and data types of each column in the csv file.

If your data is saved as an Excel file, you have to install and load another library first, {redxl}. Then call the read_excel funtion

```
packages = c("readxl")
if (length(setdiff(packages, rownames(installed.packages()))) > 0) {
  install.packages(setdiff(packages, rownames(installed.packages())))
}
library(readxl)

my_penguins_excel = read_excel("penguins.xlsx")
```

There are packages available to read all types of data, including fcs files for flow cytometry {flowCore} and google sheets {googlesheets4},

2. Exploring and Summarizing Data

Now that our data is loaded into R, we can do stuff with it. As all the data sets that we imported are identical, we will just stick with "my_penguins". These are in the format of data frames.

There are a few different ways to view your data.

head(data) will show you the first 6 rows of data, or head(data, n) will show you n rows of data.

```
head(my_penguins)
```

```
## # A tibble: 6 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~           39.1          18.7              181        3750 male
## 2 Adelie  Torge~           39.5          17.4              186        3800 fema~
## 3 Adelie  Torge~           40.3          18                195        3250 fema~
## 4 Adelie  Torge~           NA            NA                 NA          NA <NA>
## 5 Adelie  Torge~           36.7          19.3              193        3450 fema~
## 6 Adelie  Torge~           39.3          20.6              190        3650 male
## # ... with 1 more variable: year <int>
```

```
head(my_penguins, 10)
```

```
## # A tibble: 10 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
## # ... with 2 more variables: sex <fct>, year <int>
```

You can view all the data using View(data), or by clicking on the variable name in the "Environment" window in the upper right corner of RStudio. This may take a while if your data set is very large.

```
View(my_penguins)
```

glimpse() gives you a bit more information

```
glimpse(my_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~
```

summary() will give you some general summary info for each of your variables.

```
summary(my_penguins)
```

```
##       species          island    bill_length_mm  bill_depth_mm
##  Adelie   :152   Biscoe   :168   Min.   :32.10   Min.   :13.10
##  Chinstrap: 68   Dream    :124   1st Qu.:39.23   1st Qu.:15.60
##  Gentoo   :124   Torgersen: 52   Median :44.45   Median :17.30
##                                  Mean   :43.92   Mean   :17.15
##                                  3rd Qu.:48.50   3rd Qu.:18.70
##                                  Max.   :59.60   Max.   :21.50
##                                  NA's   :2       NA's   :2
##  flipper_length_mm  body_mass_g       sex           year
##  Min.   :172.0     Min.   :2700   female:165   Min.   :2007
##  1st Qu.:190.0     1st Qu.:3550   male  :168   1st Qu.:2007
##  Median :197.0     Median :4050   NA's  : 11   Median :2008
##  Mean   :200.9     Mean   :4202                Mean   :2008
##  3rd Qu.:213.0     3rd Qu.:4750                3rd Qu.:2009
##  Max.   :231.0     Max.   :6300                Max.   :2009
##  NA's   :2         NA's   :2
```

You can also do your own summaries. This will utilize the dplyr package which provides many powerful tools to manipulate your data frames. In this case let's say we want to group the penguins by species and then calculate the mean and standard deviation for each species:

```
my_penguins %>%  # %>% is a pipe, it applies a function to the data prior, in this case "my_penguins"
  group_by(species) %>% #this specifies species as the group
  summarize( #summarize specifies that you want to summarise each group
    mean_bill = mean(bill_length_mm, na.rm = TRUE), #na.rm will ignore any NAs in the data
    std_bill = sd(bill_length_mm, na.rm=TRUE))
```

```
## # A tibble: 3 x 3
##   species   mean_bill std_bill
##   <fct>         <dbl>    <dbl>
## 1 Adelie         38.8     2.66
## 2 Chinstrap      48.8     3.34
## 3 Gentoo         47.5     3.08
```

dplyr has many operators summarized in this cheat sheet: https://github.com/rstudio/cheatsheets/raw/master/data-transformation.pdf

Let's say you want to summarize based upon species and year, it is as simple as adding year to your group_by() function call.

```
my_penguins %>%  # %>% is a pipe, it applies a function to the data prior, in this case "my_penguins"
  group_by(species, year) %>% #this specifies species and year as the group
  summarize( #summarize specifies that you want to summarise each group
    mean_bill = mean(bill_length_mm, na.rm = TRUE), #na.rm will ignore any NAs in the data
    std_bill = sd(bill_length_mm, na.rm=TRUE))
```

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

```
## # A tibble: 9 x 4
```

```
## # Groups:   species [3]
##   species    year mean_bill std_bill
##   <fct>     <int>     <dbl>    <dbl>
## 1 Adelie     2007      38.8     2.47
## 2 Adelie     2008      38.6     2.98
## 3 Adelie     2009      39.0     2.56
## 4 Chinstrap  2007      48.7     3.47
## 5 Chinstrap  2008      48.7     3.62
## 6 Chinstrap  2009      49.1     3.10
## 7 Gentoo     2007      47.0     3.27
## 8 Gentoo     2008      46.9     2.64
## 9 Gentoo     2009      48.5     3.19
```

There are plenty of other things you can do with dplyr. For example, if you want to get rid of data earlier than 2008, you can filter by year.

```
my_penguins %>%  # %>% is a pipe, it applies a function to the data prior, in this case "my_penguins"
  filter(year >= 2008) %>% #this filters out years prior to 2008
  group_by(species, year) %>% #this specifies species as the group
  summarize(#summarize specifies that you want to summarize each group
    mean_bill = mean(bill_length_mm, na.rm = TRUE),#na.rm will ignore any NAs in the data
    std_bill = sd(bill_length_mm, na.rm = TRUE)
  )
```

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

```
## # A tibble: 6 x 4
## # Groups:   species [3]
##   species    year mean_bill std_bill
##   <fct>     <int>     <dbl>    <dbl>
## 1 Adelie     2008      38.6     2.98
## 2 Adelie     2009      39.0     2.56
## 3 Chinstrap  2008      48.7     3.62
## 4 Chinstrap  2009      49.1     3.10
## 5 Gentoo     2008      46.9     2.64
## 6 Gentoo     2009      48.5     3.19
```

3. Saving Data

When creating summary tables above, we didn't assign them to a variable, so they were printed to the console. If we assign them to "my_penguins_summary" we now have a new variable that you may want to save for later use.

```
my_penguins_summary = my_penguins %>%  # %>% is a pipe, it applies a function to the data prior, in thi
  filter(year >= 2008) %>% #this filters out years prior to 2008
  group_by(species, year) %>% #this specifies species as the group
  summarize(#summarize specifies that you want to summarize each group
    mean_bill = mean(bill_length_mm, na.rm = TRUE),#na.rm will ignore any NAs in the data
    std_bill = sd(bill_length_mm, na.rm = TRUE)
  )
```

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

The most common way to save data is as a csv file. A csv file is versitile and can be read in any number of programs including Excel. In order to save a csv file we will use the write_csv function. You need to specify which variable you want to save and a path and/or filename to save it to.

```
write_csv(my_penguins_summary, "penguins summary.csv")
```

If you have large data sets and will continue working in R with them, it may be advantageous to save as an RDS file. RDS files compress data better than csv for large files so can save space and can also be opened quicker than a csv using readRDS().

```
saveRDS(my_penguins_summary, "penguins summary.rds")
```

4. Basic Data Visualization

As mentioned before, we will be skipping plotting with base R and using {ggplot2} instead. ggplot2 works by layering plots (geoms) and annotations on a common coordinate system. There are many different built in plot types available including scatter splots, bar plots, box plots, dot plots, and violin plots.

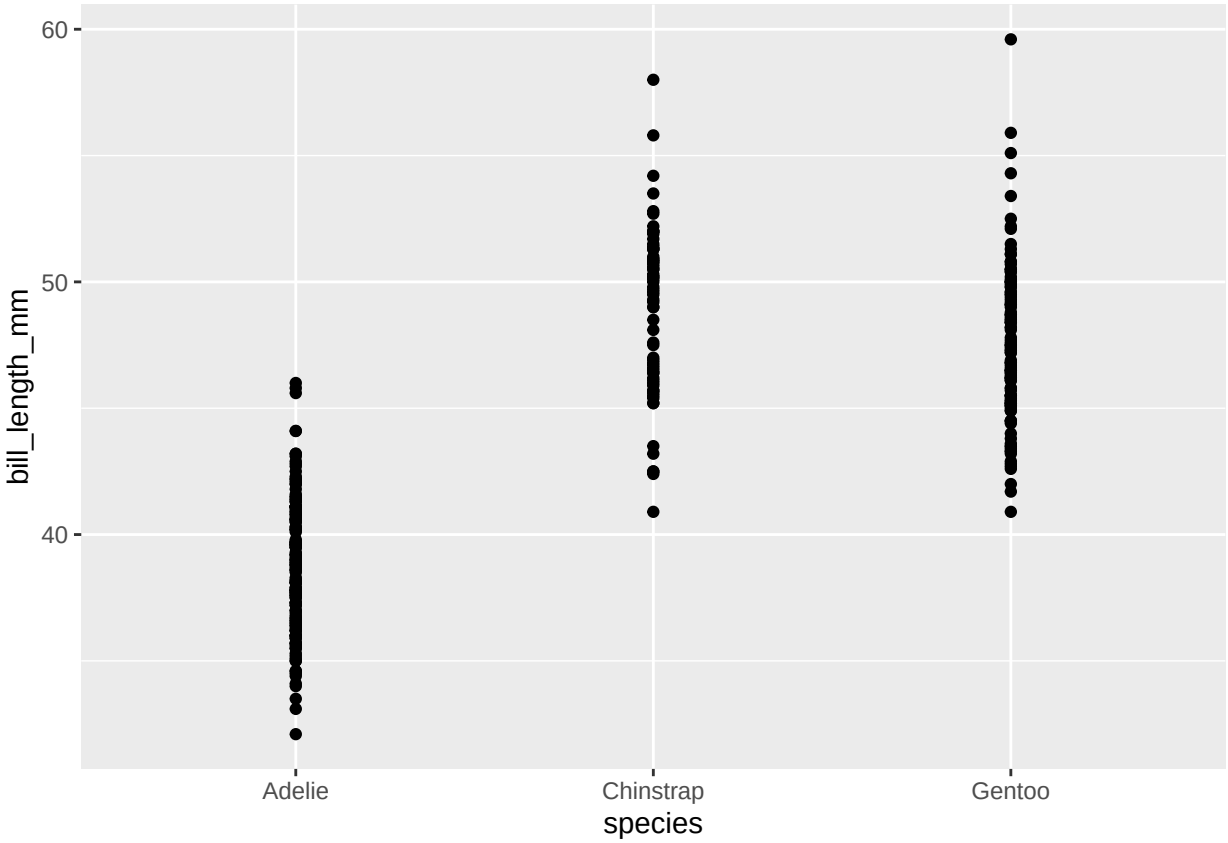
A fantastic ggplot2 tutorial by Cedric Scherer can be found here: https://cedricscherer.netlify.app/2019/08/05/a-ggplot2-tutorial-for-beautiful-plotting-in-r/

A cheat sheet of available geoms and other helpful hints is here: https://github.com/rstudio/cheatsheets/raw/master/data-visualization.pdf

A full reference book for ggplot2 is here: https://ggplot2-book.org/index.html

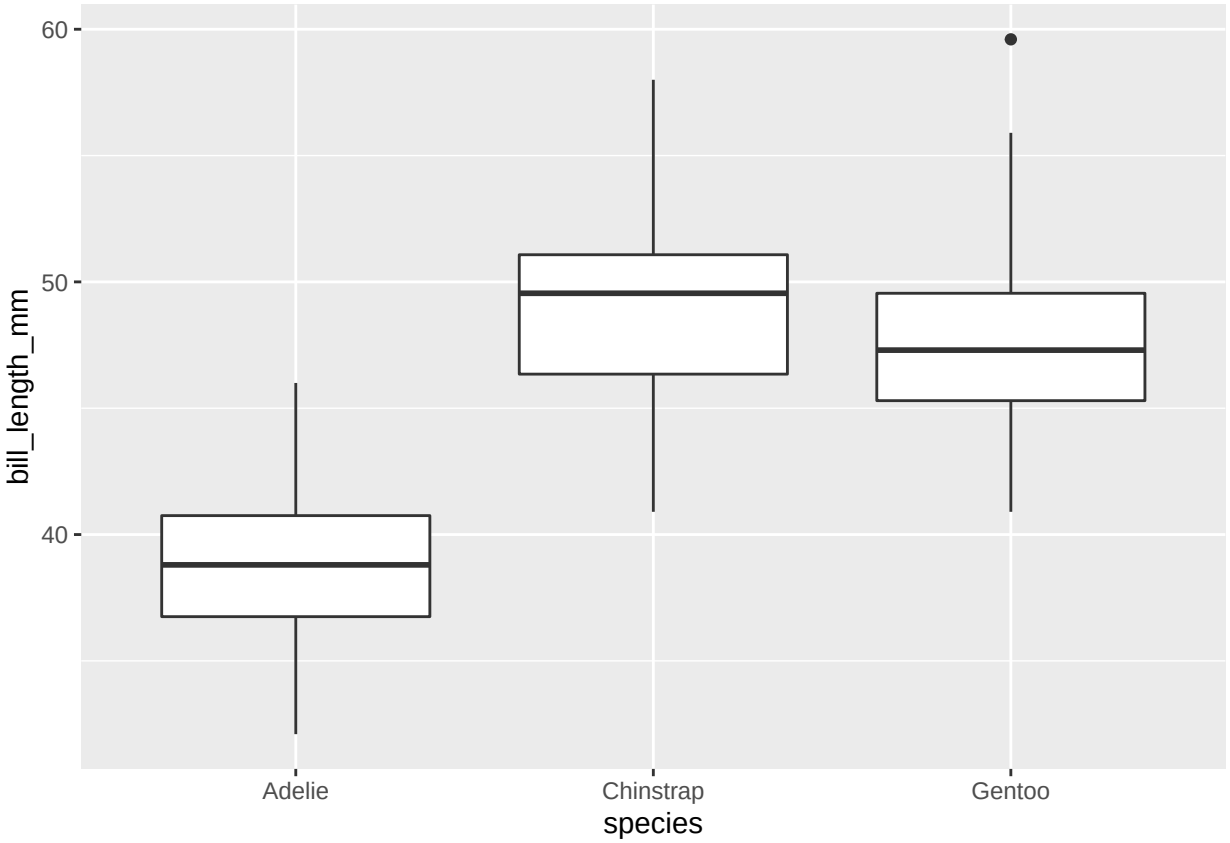
We will start just plotting each replicate of bill length for each penguin species.

```
my_penguins %>%  # start with the data you want to plot
  ggplot(aes(x = species, y = bill_length_mm)) + # you have to establish aesthetics, in this case the x
  geom_point() # this adds each data point, NAs will be excluded
```



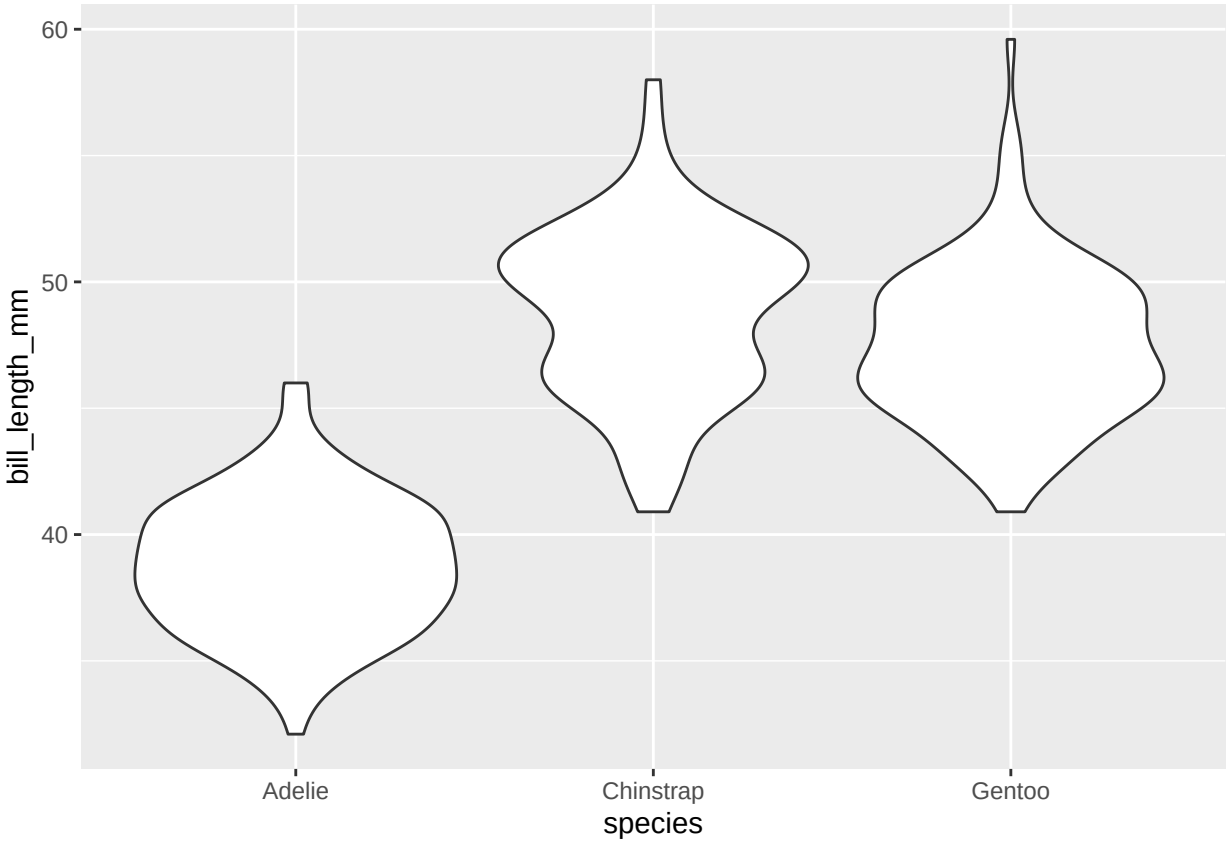
Box plots are created similarly

```
my_penguins %>%  # start with the data you want to plot
  ggplot(aes(x = species, y = bill_length_mm)) + # you have to establish aesthetics, in this case the x
  geom_boxplot() # this creates box plots, NAs will be excluded
```



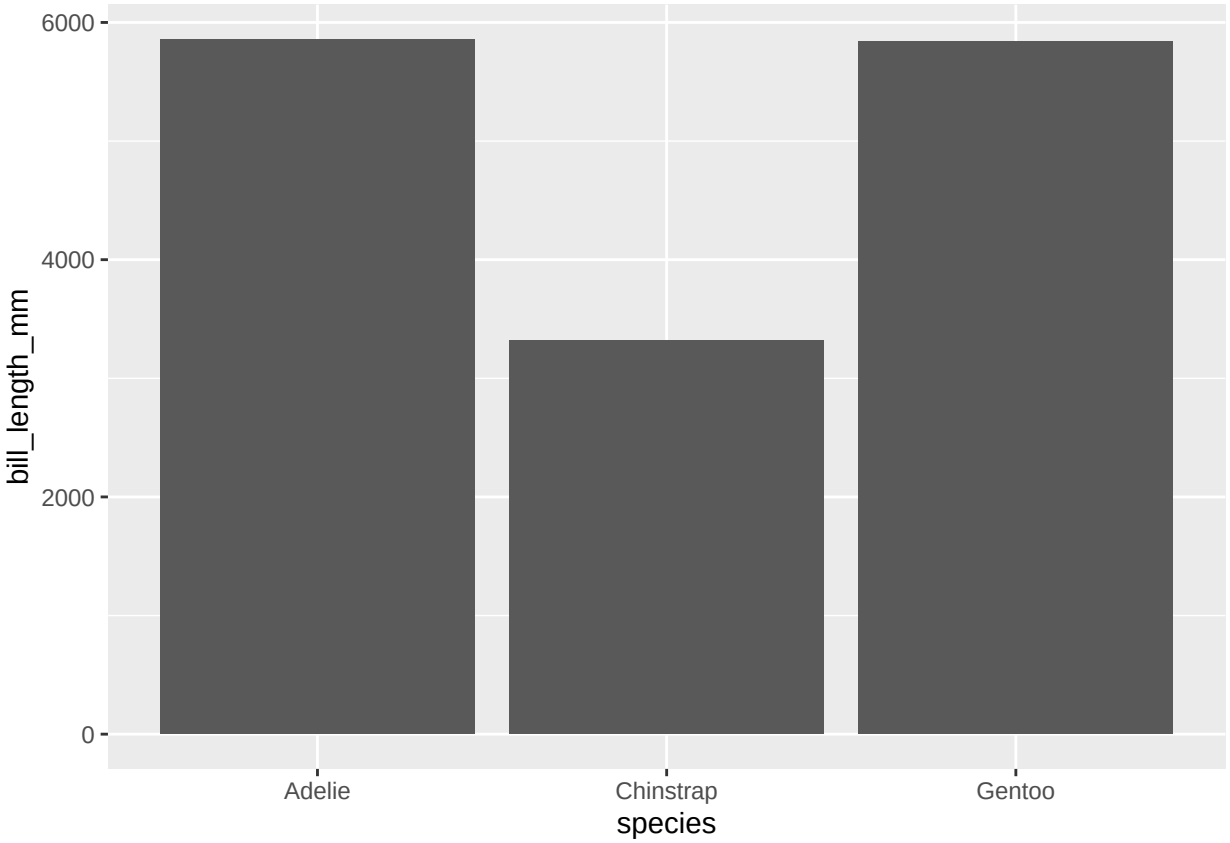
Violin plots

```
my_penguins %>%  # start with the data you want to plot
  ggplot(aes(x = species, y = bill_length_mm)) + # you have to establish aesthetics, in this case the x
  geom_violin() # this creates violin plots, NAs will be excluded
```



If you try to make a bar plot with geom_bar(), you will get an error since you can't have an x and y aesthetics without specifying a transformation or "stat". Stat identity will add up all the bill lenghts, which is likely not what you want.

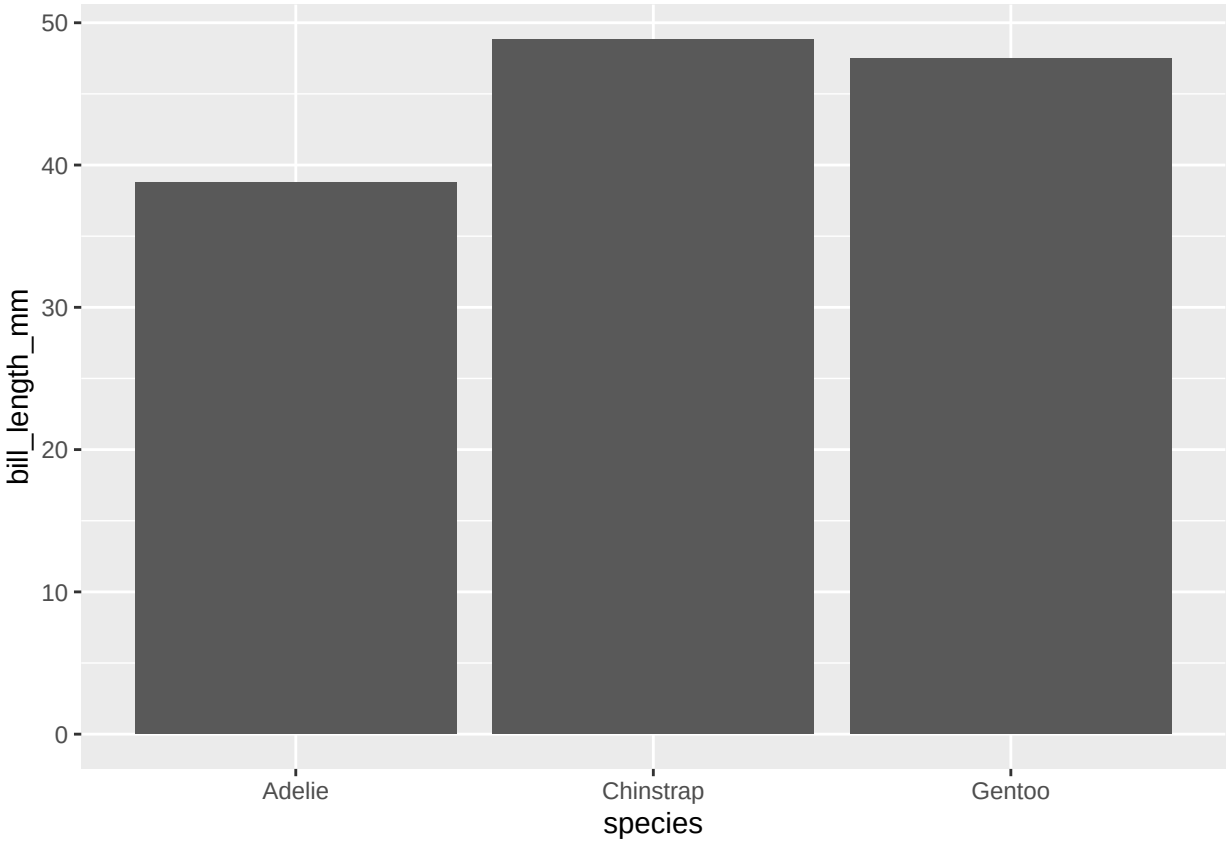
```
my_penguins %>%  # start with the data you want to plot
  ggplot(aes(x = species, y = bill_length_mm)) + # you have to establish aesthetics, in this case the x
  geom_bar(stat = "identity") # this creates bar plots, NAs will be excluded
```



To get a bar plot like you normally expect, with averages on the y-axis instead of sums, you use stat_summary which defaults to the mean.

```
my_penguins %>%  # start with the data you want to plot
  ggplot(aes(x = species, y = bill_length_mm)) + # you have to establish aesthetics, in this case the x
  stat_summary(geom = "bar") # this creates bar plots, NAs will be excluded
```

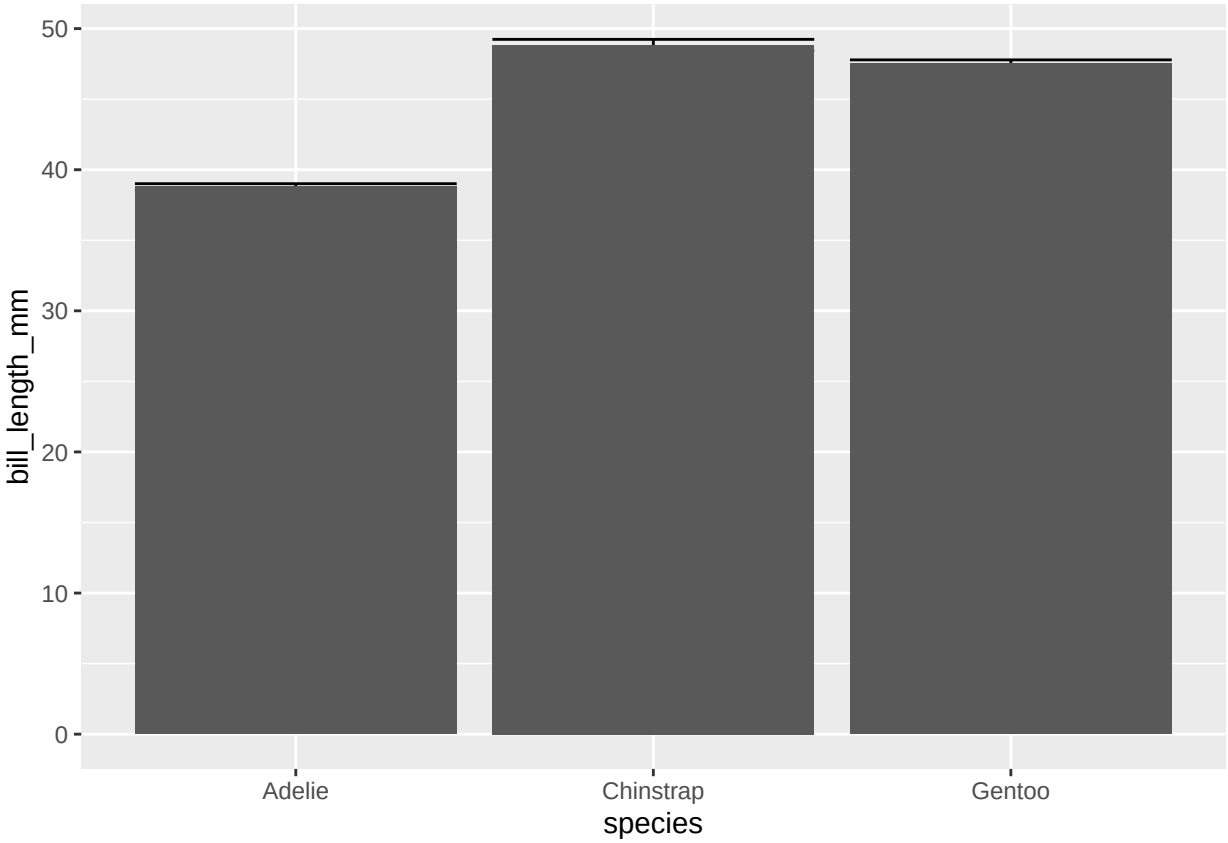
```
## No summary function supplied, defaulting to `mean_se()`
```



You can add an error bar with another call to stat_summary, here it defaults to mean and standard error. Note that I put it above the bar function call so that the bar would be plotted on top of the errorbar. Order matters with {ggplot2}.

```
my_penguins %>%  # start with the data you want to plot
  ggplot(aes(x = species, y = bill_length_mm)) + # you have to establish aesthetics, in this case the x
  stat_summary(geom = "errorbar") + #adds an error bar with se as the range
  stat_summary(geom = "bar")  # this creates bar plots, NAs will be excluded
```

```
## No summary function supplied, defaulting to `mean_se()`
## No summary function supplied, defaulting to `mean_se()`
```

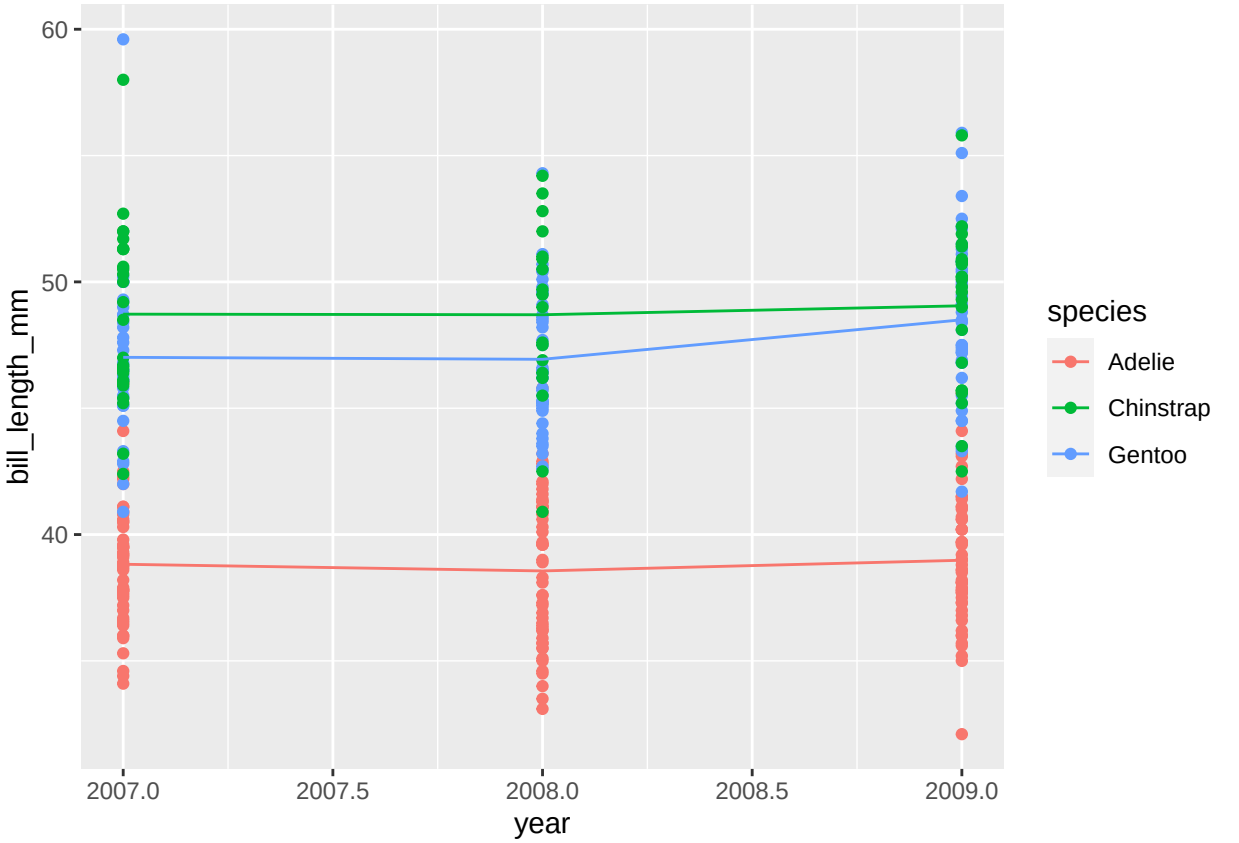


However, **I highly recommend you do not use these types of plots**, especially in publications. Bar plots hide the real distribution and number of replicates of the data and there are better ways to visualize data that will be discussed later.

For time series data, you can change the x-axis to "year" and can add a color aesthetic to differentiate the penguin species and a line connecting the mean values.

```
my_penguins %>%  # start with the data you want to plot
  ggplot(aes(x = year, y = bill_length_mm, color = species)) + # you have to establish aesthetics, in t
  geom_point() + # this creates points plots, NAs will be excluded+
  geom_line(stat = "summary") #this creates lines from each mean for species and year
```

```
## No summary function supplied, defaulting to `mean_se()`
```



To save a plot, use the ggsave function. It will automatically save the last plot generated, or if you save a plot to a variable, you can specify which plot to save. You can specify image size as well.

```
ggsave("last plot.png")
```

```
## Saving 6.5 x 4.5 in image
```

```
## No summary function supplied, defaulting to `mean_se()`
```

```
plot_to_save = my_penguins %>%  # start with the data you want to plot
  ggplot(aes(x = species, y = bill_length_mm)) + # you have to establish aesthetics, in this case the x
  geom_violin(aes(color = species)) # this creates violin plots, NAs will be excluded

ggsave(
  "specific plot.png",
  plot = plot_to_save,
  width = 5,
  height = 5
)
```

5. T-tests and Wilcoxon tests

To run a t-test, you need two vectors to compare. If these are already columns in your data, then this is easy. Just use t.test() and pass the two columns. If the data is paired, you add "paired = TRUE".

```
t.test(my_penguins$bill_length_mm, my_penguins$body_mass_g) #the $ operator allows you to get a column
```

```
##
##  Welch Two Sample t-test
##
## data:  my_penguins$bill_length_mm and my_penguins$body_mass_g
## t = -95.878, df = 341.03, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -4243.130 -4072.534
## sample estimates:
##  mean of x  mean of y
##   43.92193 4201.75439
```

```
t.test(my_penguins$bill_length_mm, my_penguins$body_mass_g, paired = TRUE)
```

```
##
##  Paired t-test
##
## data:  my_penguins$bill_length_mm and my_penguins$body_mass_g
## t = -96.269, df = 341, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -4242.784 -4072.881
## sample estimates:
## mean of the differences
##               -4157.832
```

Above, you can see that bill length is highly significantly different than body mass. But that is a nonsensical test, since one is a length and one is a weight. If you wanted to compare bill length between two species, you can just make two new vectors and then run t.test().

```
gentoo = penguins %>% filter(species == "Gentoo") %>% pull(body_mass_g) #pull gives you a vector
chinstrap = penguins %>% filter(species == "Chinstrap") %>% pull(body_mass_g)

t.test(gentoo, chinstrap)
```

```
##
##  Welch Two Sample t-test
##
## data:  gentoo and chinstrap
## t = 20.628, df = 170.4, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  1214.416 1471.440
## sample estimates:
## mean of x mean of y
##  5076.016  3733.088
```

Alternatively, you can use parwise_t_test from {rstatix} to get pairwise comparisons between groups.

```
packages = c("rstatix")
if (length(setdiff(packages, rownames(installed.packages()))) > 0) {
  install.packages(setdiff(packages, rownames(installed.packages())))
}
library(rstatix)
```

```
##
## Attaching package: 'rstatix'
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
my_penguins %>%
  pairwise_t_test(body_mass_g ~ species)
```

```
## # A tibble: 3 x 9
##   .y.       group1   group2     n1    n2        p p.signif    p.adj p.adj.signif
## * <chr>     <chr>    <chr>   <int> <int>    <dbl> <chr>       <dbl> <chr>
## 1 body_mas~ Adelie   Chinst~   152    68 6.31e- 1 ns       6.31e- 1 ns
## 2 body_mas~ Adelie   Gentoo    152   124 5.42e-77 ****     1.63e-76 ****
## 3 body_mas~ Chinstr~ Gentoo     68   124 3.21e-56 ****     6.42e-56 ****
```

To use a non-parametric test, you do the same thing, but use wilcox.test() or pairwise_wilcox_test()

```
wilcox.test(gentoo, chinstrap)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  gentoo and chinstrap
## W = 8233, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0
```

```
my_penguins %>%
  pairwise_wilcox_test(body_mass_g ~ species)
```

```
## # A tibble: 3 x 9
##   .y.      group1   group2     n1    n2 statistic        p    p.adj p.adj.signif
## * <chr>    <chr>    <chr>   <int> <int>     <dbl>    <dbl>    <dbl> <chr>
## 1 body_ma~ Adelie   Chinst~   152    68     4831  4.85e- 1 4.85e- 1 ns
## 2 body_ma~ Adelie   Gentoo    152   124      400. 2.98e-42 8.94e-42 ****
## 3 body_ma~ Chinstr~ Gentoo     68   124      131  1.67e-28 3.34e-28 ****
```

ANOVA is similar, however you have to use summary to see the results

```
penguin_anova = aov(body_mass_g ~ species, data = my_penguins)
summary(penguin_anova)
```

```
##              Df    Sum Sq  Mean Sq F value Pr(>F)
## species       2 146864214 73432107   343.6 <2e-16 ***
## Residuals   339  72443483   213698
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 2 observations deleted due to missingness
```

6. Correlations

Correlations, cor.test() work similarly to t-tests, but you have to have the same number of measurements for each vector. You can also specify which correlation method you'd like to calculate, Pearson's is the default.

```
cor.test(my_penguins$bill_length_mm, my_penguins$body_mass_g, use = "complete.obs") #"complete.obs here
```

```
##
##  Pearson's product-moment correlation
##
## data:  my_penguins$bill_length_mm and my_penguins$body_mass_g
## t = 13.654, df = 340, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.5220040 0.6595358
## sample estimates:
##       cor
## 0.5951098
```

```
cor.test(
  my_penguins$bill_length_mm,
  my_penguins$body_mass_g,
  use = "complete.obs",
  method = "spearman"
)
```

```
##
##  Spearman's rank correlation rho
##
## data:  my_penguins$bill_length_mm and my_penguins$body_mass_g
## S = 2774758, p-value < 2.2e-16
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##       rho
## 0.5838003
```

You can quickly get visualizations and correlations from each variable using ggpairs from {GGgally}.

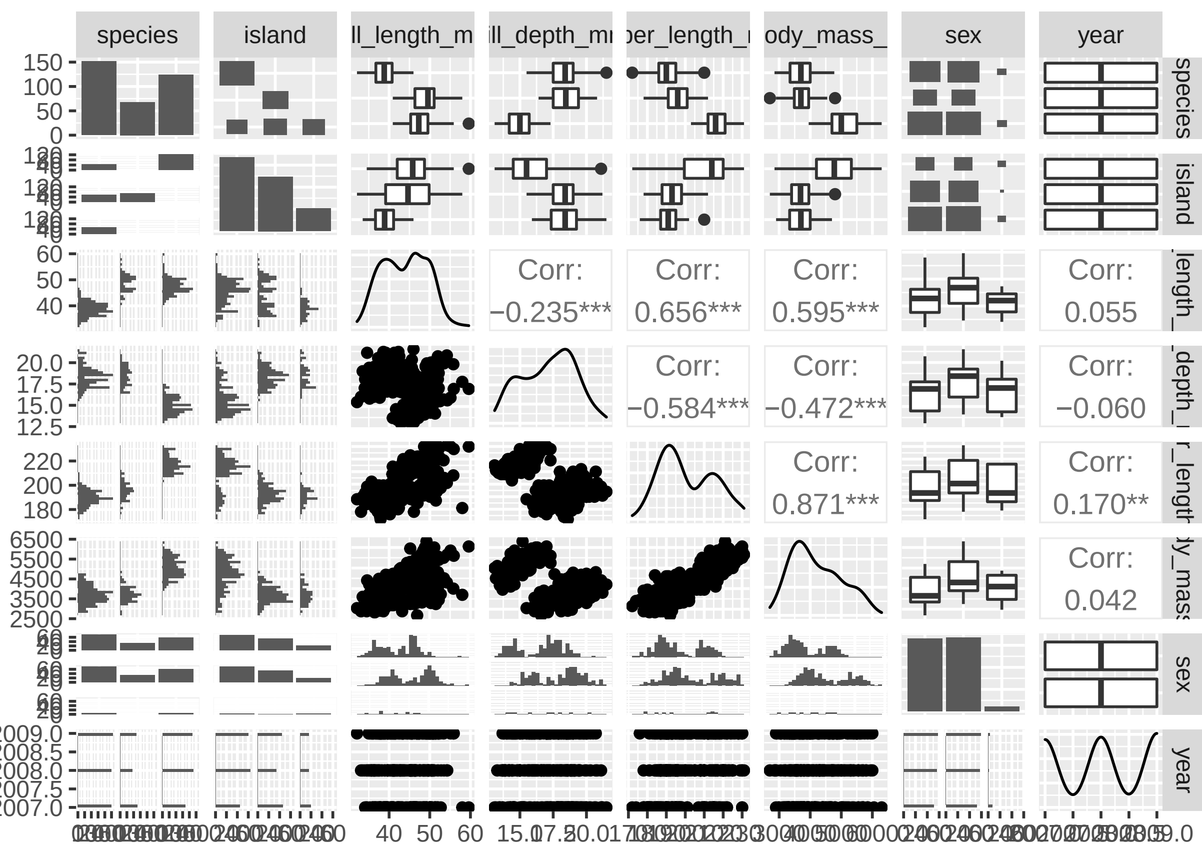
```
packages = c("GGally")
if (length(setdiff(packages, rownames(installed.packages()))) > 0) {
  install.packages(setdiff(packages, rownames(installed.packages())))
}
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
ggpairs(my_penguins)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

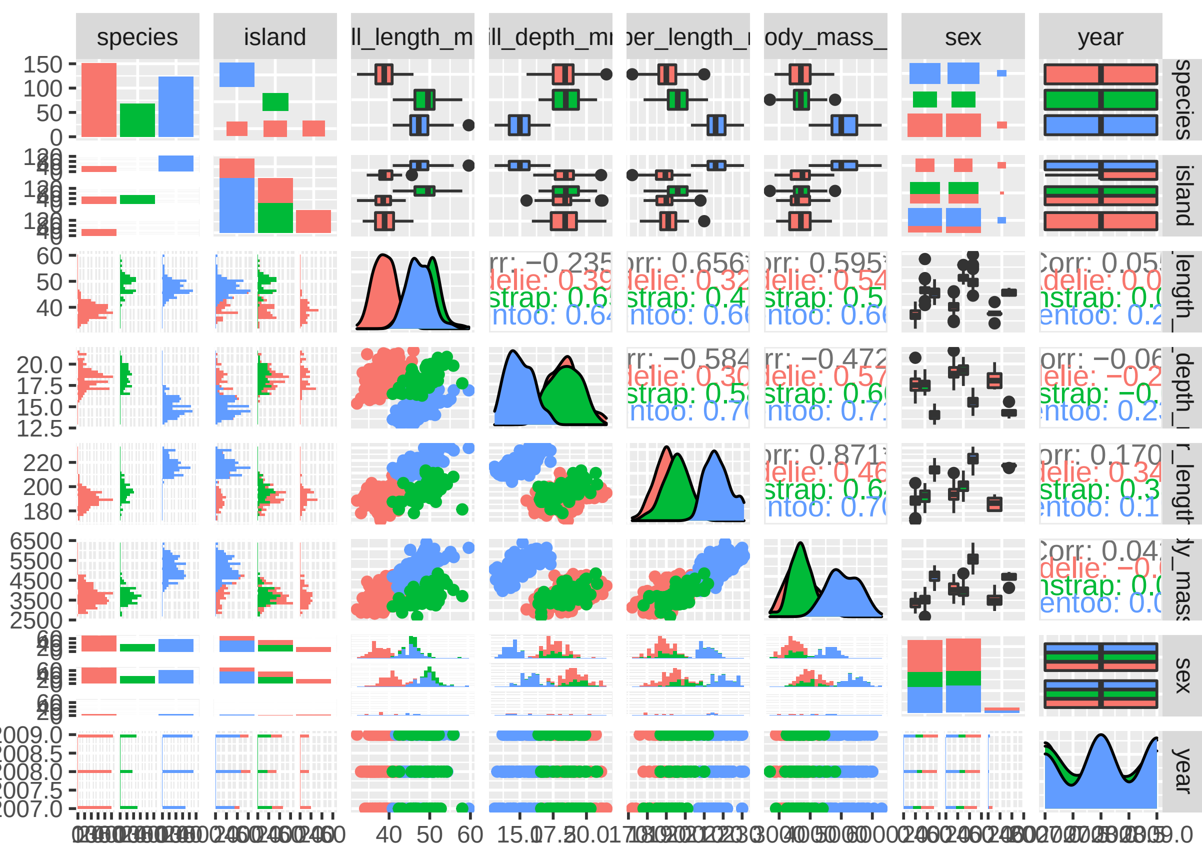
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



You can add colors for groups as well.

```
ggpairs(my_penguins,
mapping = ggplot2::aes(color=species))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



7. PCA and Dimensionality Reduction

Principal Components Analysis is pretty easy in R as well. In the case below we have to filter the variables to only those that are numbers (you can't run PCA on text), remove year since it likely is not useful in this

case, as well as remove rows with NA values. We also specify scale = TRUE in order to center each variable around 0.

For a more comprehensive PCA tutorial, see here: https://clauswilke.com/blog/2020/09/07/pca-tidyverse-style/

```
pca = my_penguins %>%
  select(where(is.numeric)) %>%
  select(-year) %>%
  filter(!is.na(bill_length_mm)) %>%
  #na.omit() %>% #we could just use na.omit here, but I wanted to control exactly which rows were remov
  prcomp(scale = TRUE)
```

You can look at the variance explained by each component in a couple of different ways:

```
summary(pca)
```

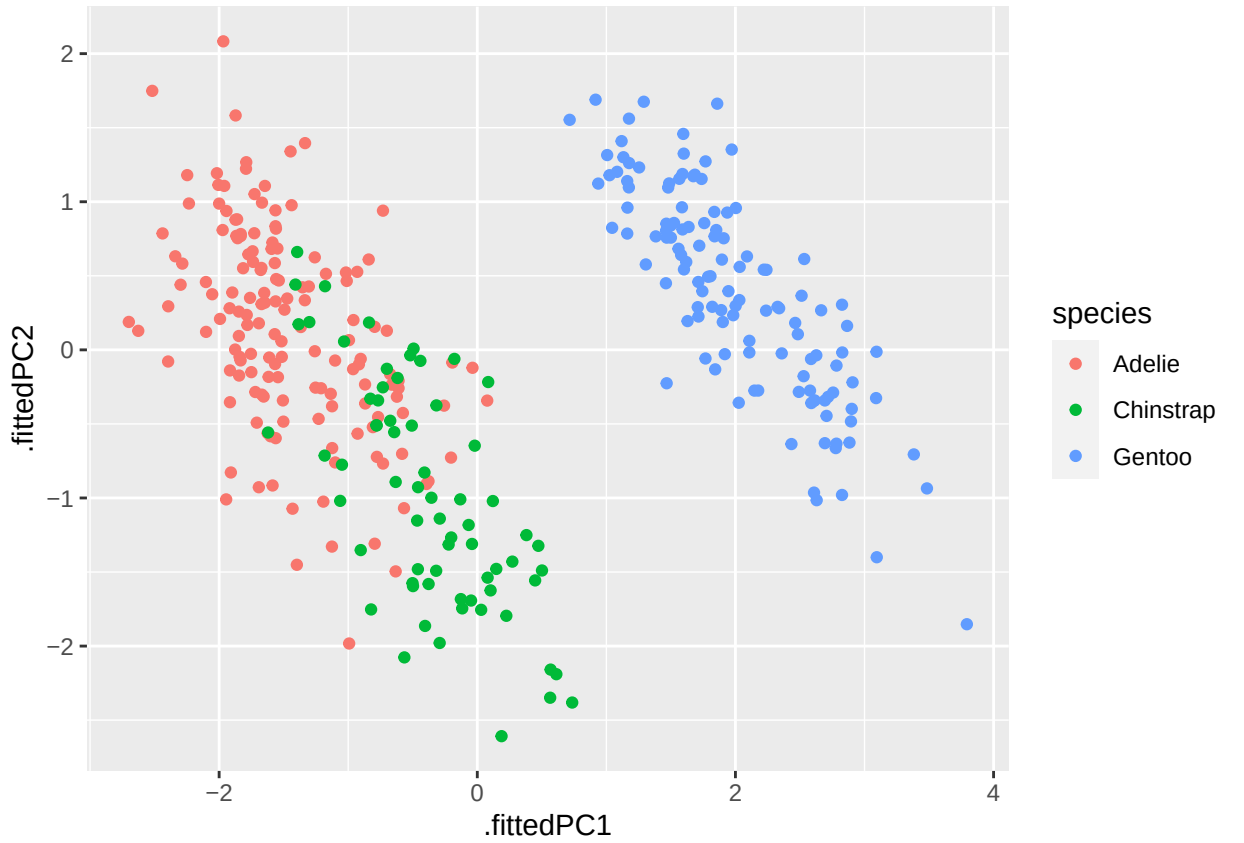
```
## Importance of components:
##                           PC1    PC2     PC3     PC4
## Standard deviation     1.6594 0.8789 0.60435 0.32938
## Proportion of Variance 0.6884 0.1931 0.09131 0.02712
## Cumulative Proportion  0.6884 0.8816 0.97288 1.00000
```

```
pca %>%
  tidy(matrix = "eigenvalues")
```

```
## # A tibble: 4 x 4
##      PC std.dev percent cumulative
##   <dbl>   <dbl>   <dbl>      <dbl>
## 1     1   1.66   0.688       0.688
## 2     2   0.879  0.193       0.882
## 3     3   0.604  0.0913      0.973
## 4     4   0.329  0.0271      1
```

And when you plot the results and color by species, you get some nice separation between them. Gentoo are clearly seperate, which Chinstrap and Adelie have some overlapping points.

```
pca %>%
  augment(my_penguins %>% filter(!is.na(bill_length_mm))) %>% # add the original penguin data back
  ggplot(aes(x = .fittedPC1, y = .fittedPC2, color = species)) +
  geom_point(size = 1.5)
```

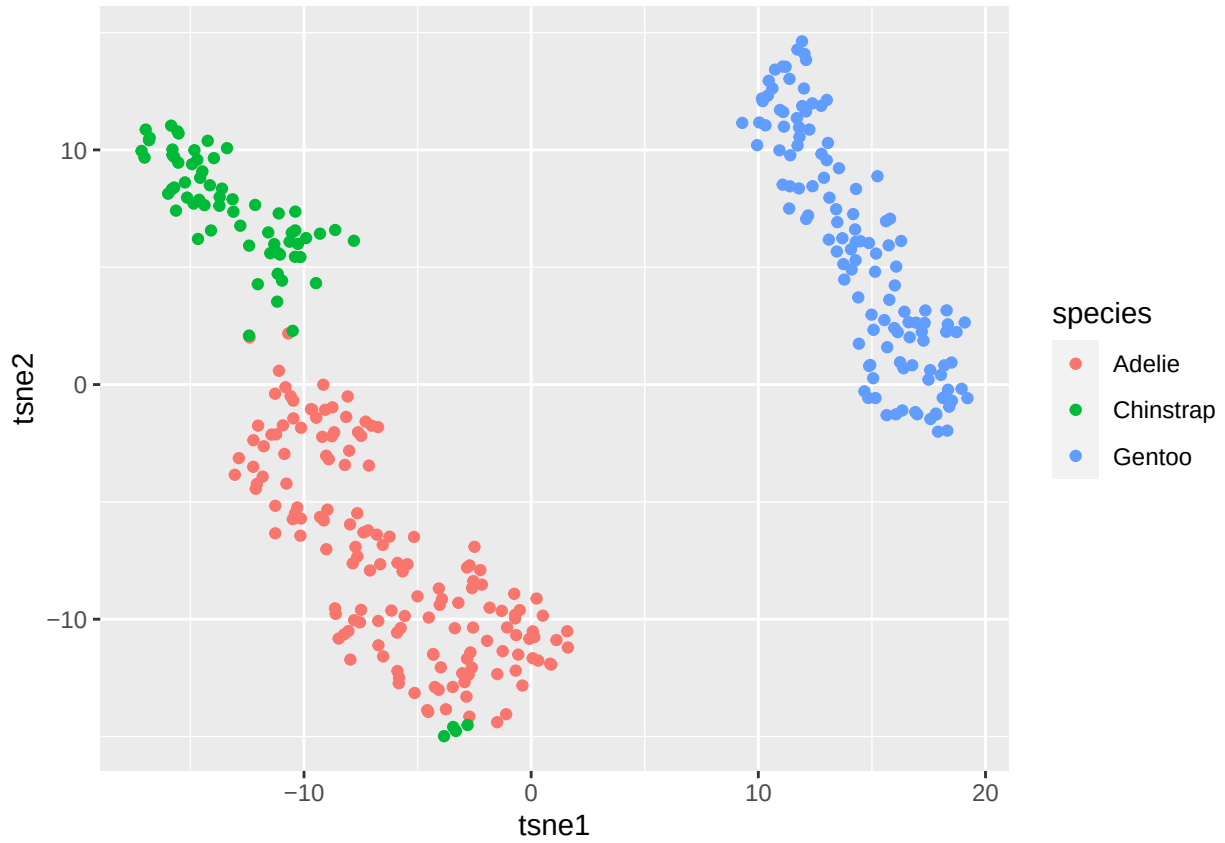


t-SNE works similarly, but fitst you need to install and load {Rtsne}

```
packages = c("Rtsne")
if (length(setdiff(packages, rownames(installed.packages()))) > 0) {
  install.packages(setdiff(packages, rownames(installed.packages())))
}
library(Rtsne)

tsne = my_penguins %>%
  select(where(is.numeric)) %>%
  select(-year) %>%
  filter(!is.na(bill_length_mm)) %>%
  scale() %>% #need to scale the data first
  Rtsne()
```

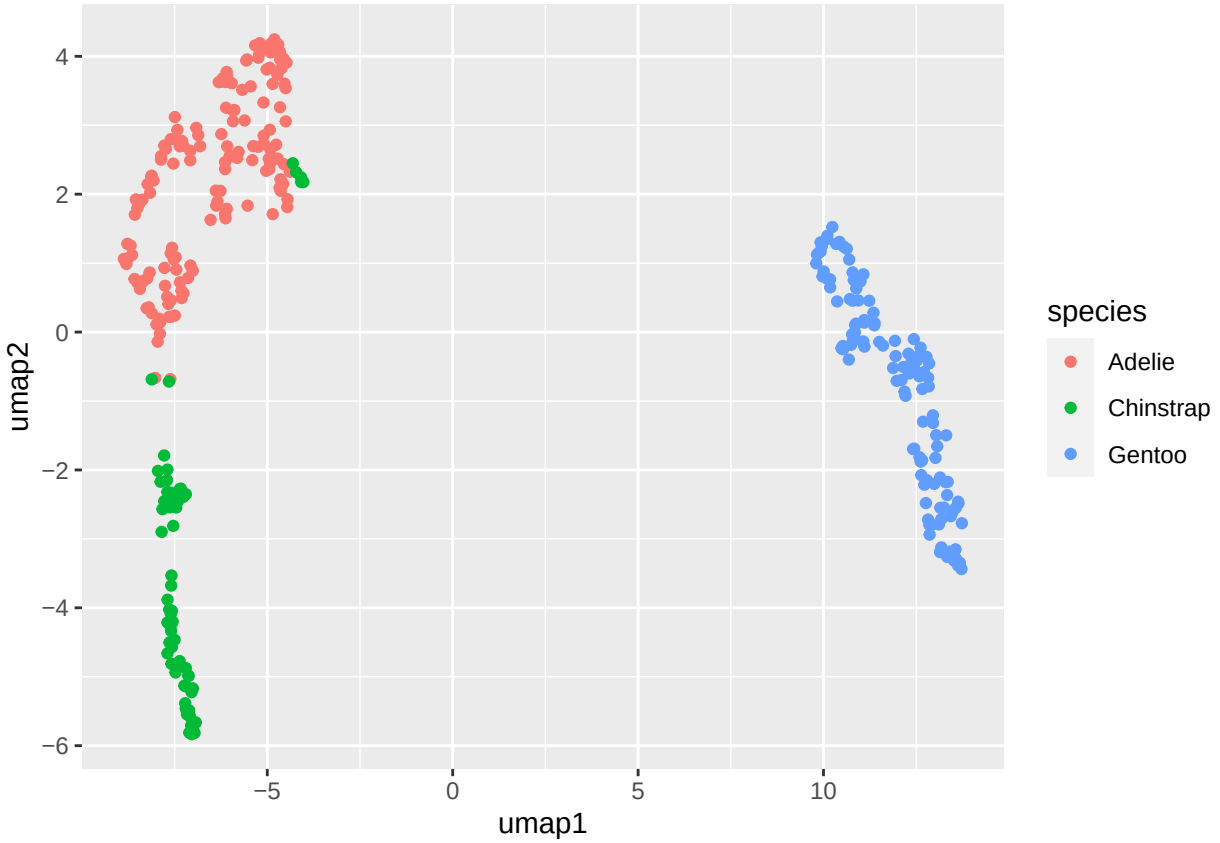
We can't just use the augment function with t-SNE like with did with PCNA, so we'll have to put the t-SNE coordinates back in another way (there are a bunch of different ways you could do this). And then plot. This gives a bit better separation between the Adelie and Chinstrap data than PCA.

```
my_penguins %>%
  filter(!is.na(bill_length_mm)) %>%
  mutate(tsne1 = tsne$Y[, 1], #this adds the 1st tsne component
         tsne2 = tsne$Y[, 2]) %>%  #this adds the 1st tsne component
  ggplot(aes(x = tsne1, y = tsne2, color = species)) +
  geom_point(size = 1.5)
```



If you prefer UMAP it is very similar to t-SNE, but you have to install and load {umap}

```
packages = c("umap")
if (length(setdiff(packages, rownames(installed.packages()))) > 0) {
  install.packages(setdiff(packages, rownames(installed.packages())))
}
library(umap)

umap = my_penguins %>%
  select(where(is.numeric)) %>%
  select(-year) %>%
  filter(!is.na(bill_length_mm)) %>%
  scale() %>% #need to scale the data first
  umap()

my_penguins %>%
  filter(!is.na(bill_length_mm)) %>%
  mutate(umap1 = umap$layout[, 1], #this adds the 1st tsne component
         umap2 = umap$layout[, 2]) %>%  #this adds the 1st tsne component
  ggplot(aes(x = umap1, y = umap2, color = species)) +
  geom_point(size = 1.5)
```



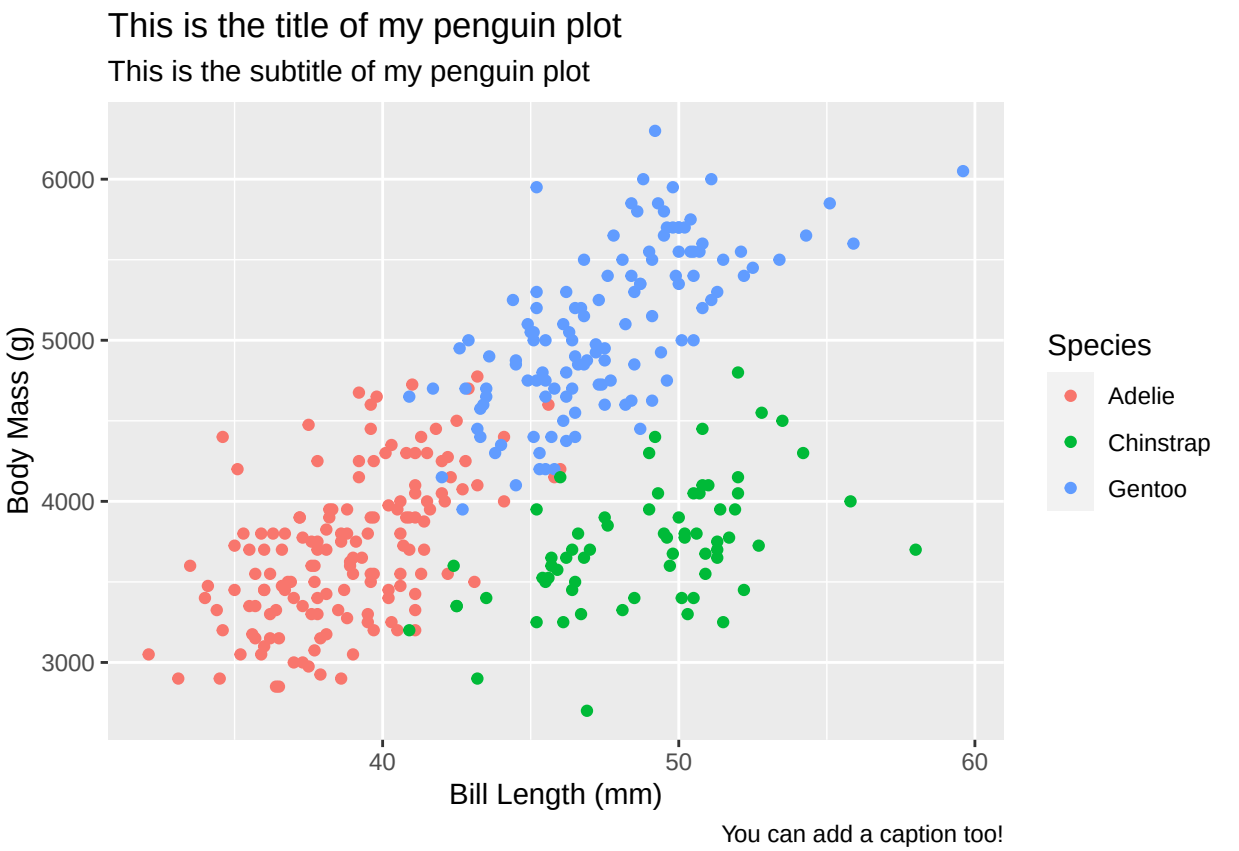
8. (Slightly More) Advanced Data Visualization

While the default plots that {ggplot2} generates are prefectly fine, you'll likely want to change some aspects.

First, changing the labels on axes, legend, and adding a title: just add labs() to your plot and specify which you want to change and provide some text.

We will work from the a scatter plot of bill length vs body weight and color of species. We wll save it first as my_plot and then build off of it going forward.

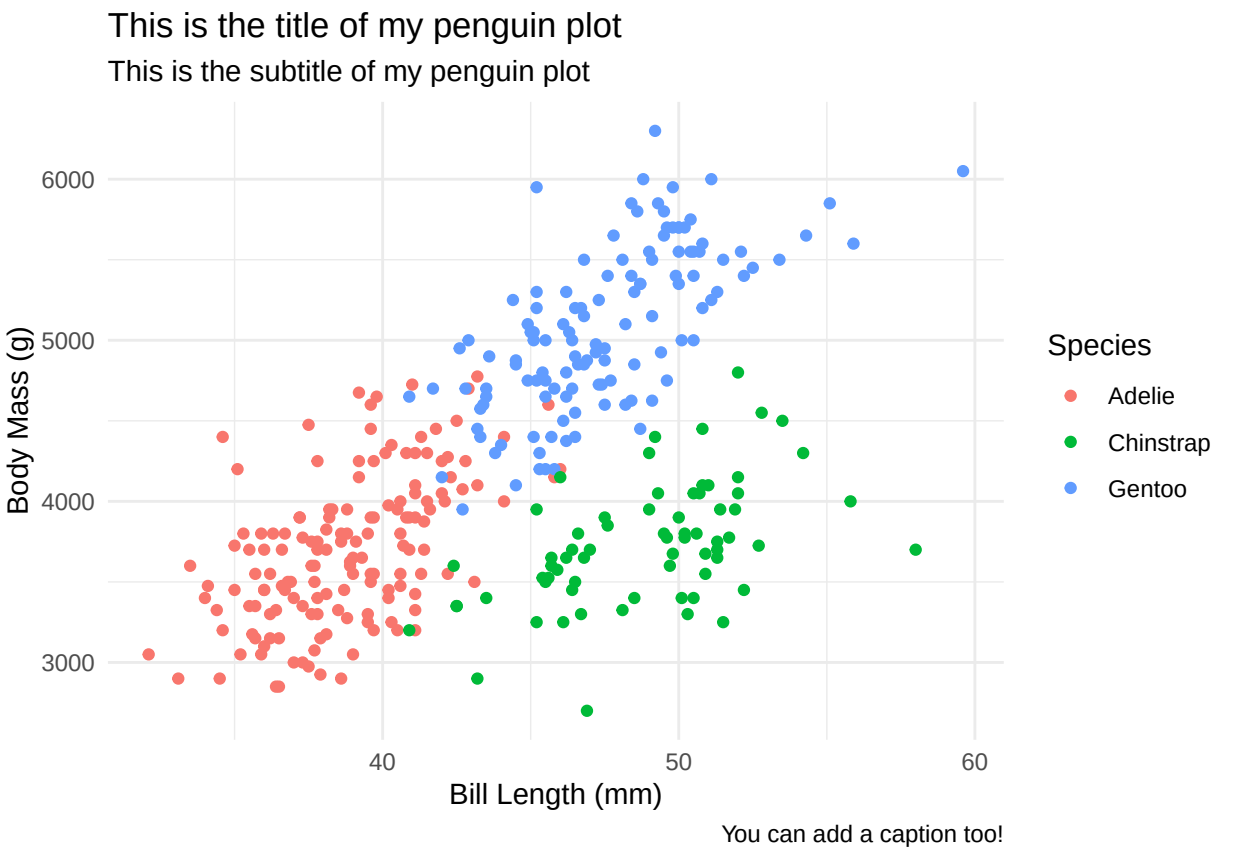
```
#base plot
my_plot = my_penguins %>%
  ggplot(aes(x = bill_length_mm, y = body_mass_g, color = species)) +
  geom_point()

my_plot = my_plot +
  labs(
    x = "Bill Length (mm)",
    y = "Body Mass (g)",
    color = "Species",
    title = "This is the title of my penguin plot",
    subtitle = "This is the subtitle of my penguin plot",
    caption = "You can add a caption too!"
  )

my_plot
```



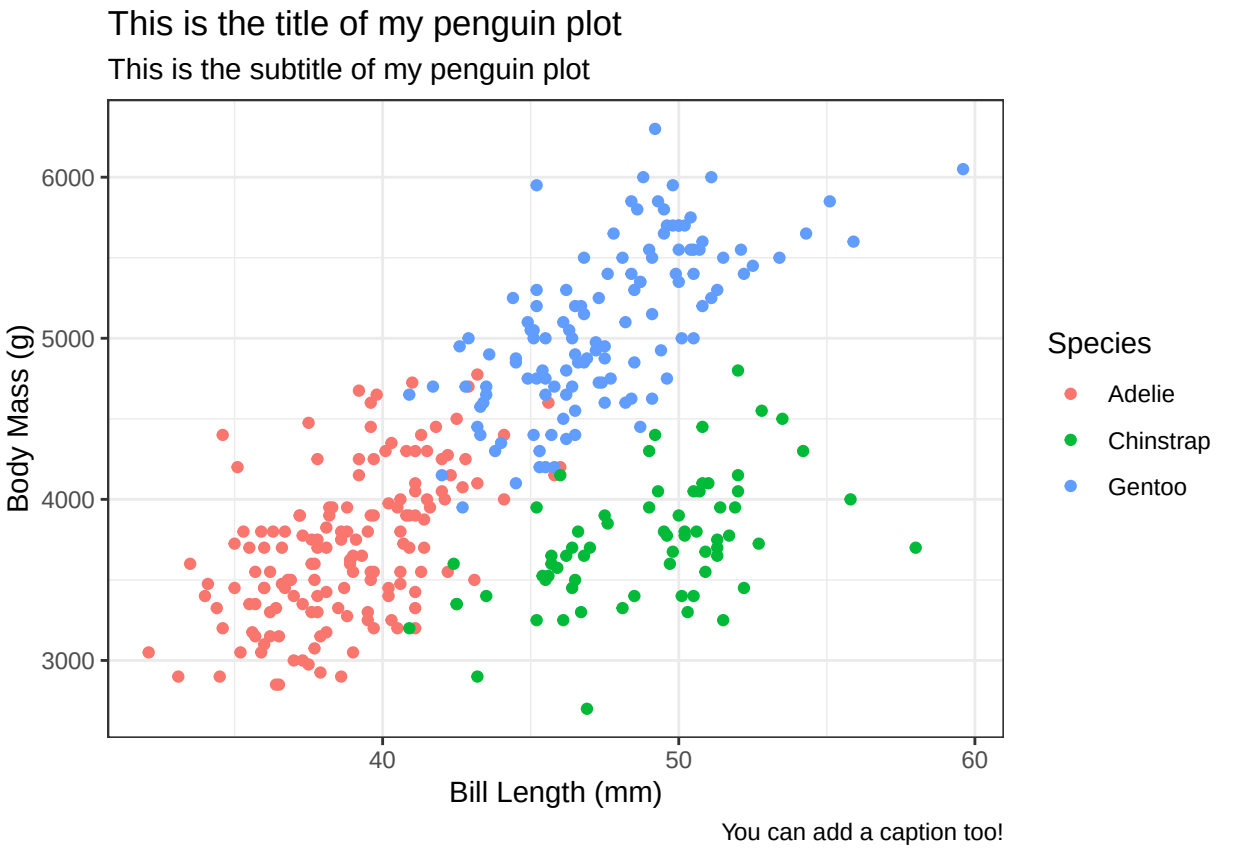
Essentially any component of the plot can be changed using the theme() function: https://ggplot2.tidyverse.org/reference/theme.html

Luckily, there are some already built in functions that will change much of this for you automatically. Some common ones are theme_minimal() and theme_bw(). And check out {ggtheme} if you want some more.

```
my_plot+
  theme_minimal()
```



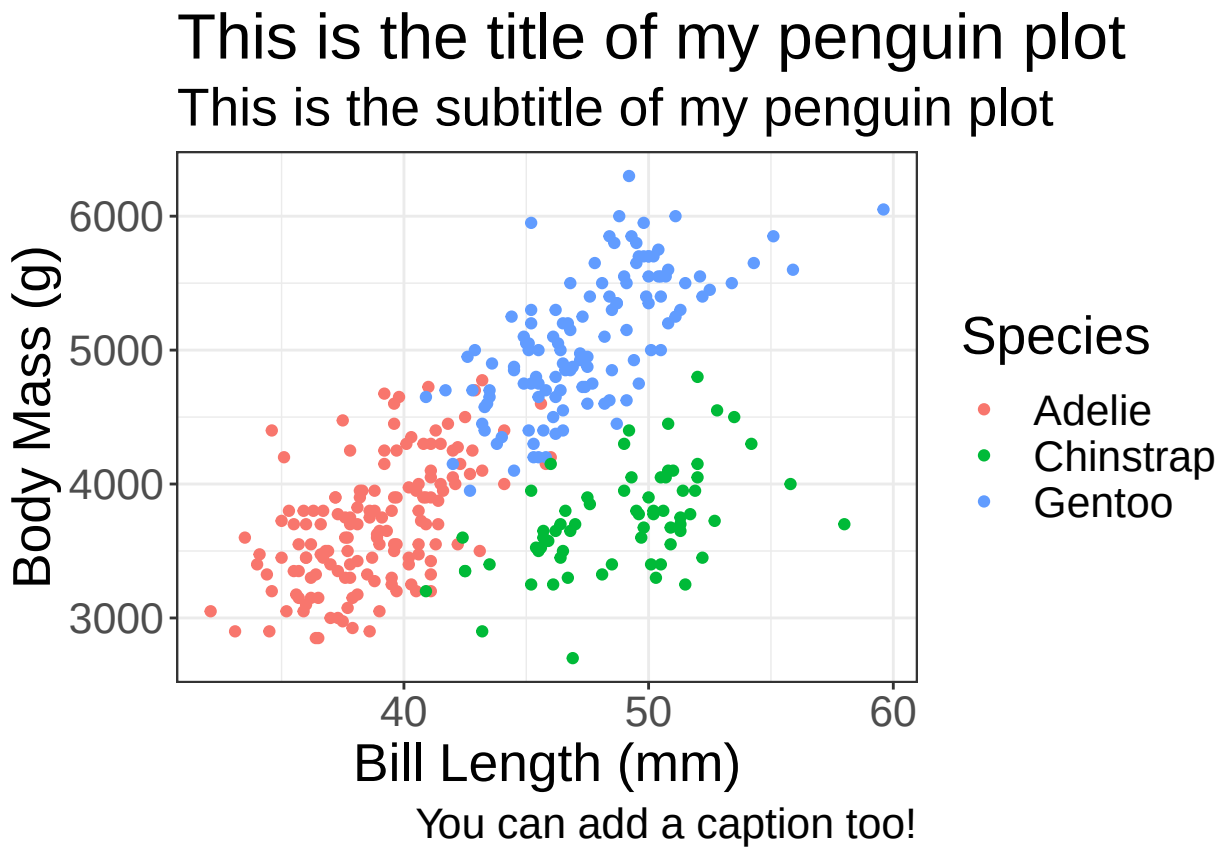

```
my_plot+
  theme_bw()
```



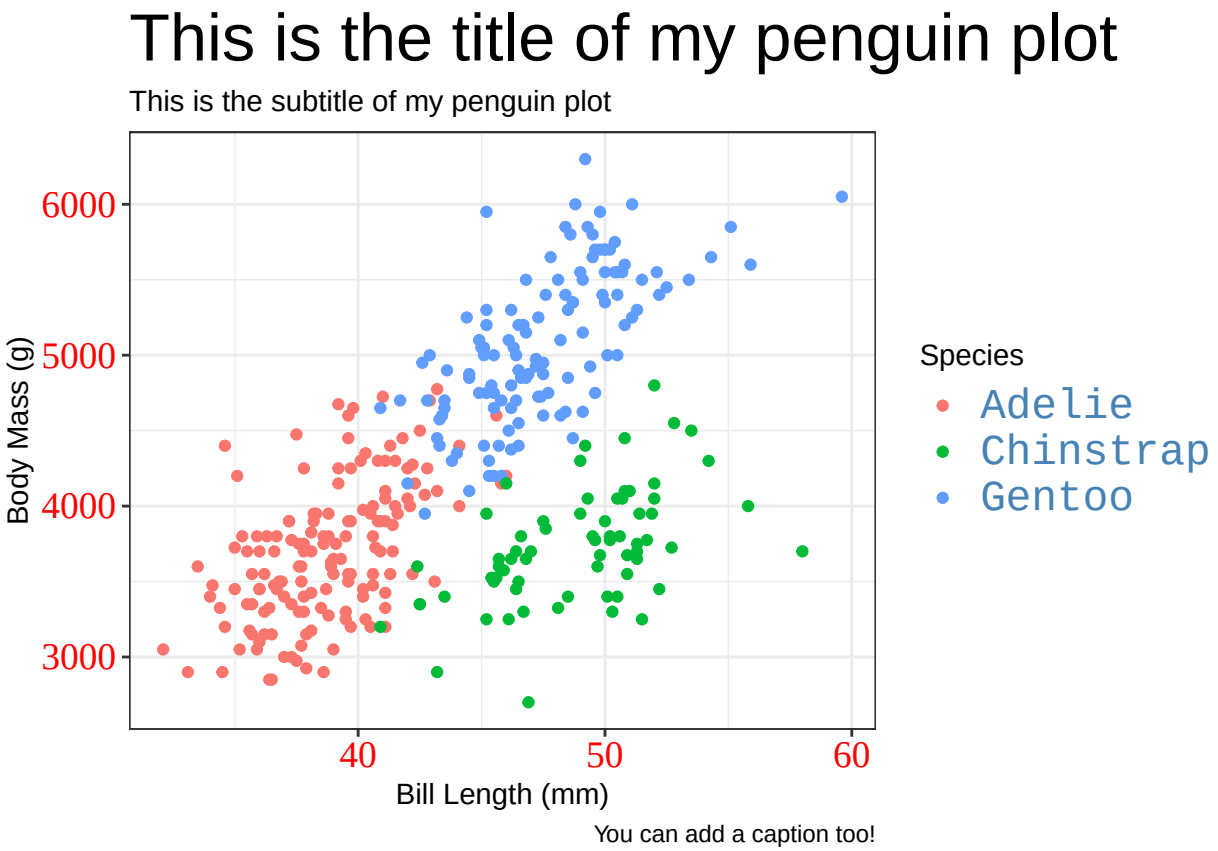
One of the most common things you will do with a plot is to just make the text bigger. Again, we can jut add to the plot we made before.

You can also change size, color or font for specific parts of the plot.

```
my_plot +
  theme_bw() +
  theme(text = element_text(size = 20))
```

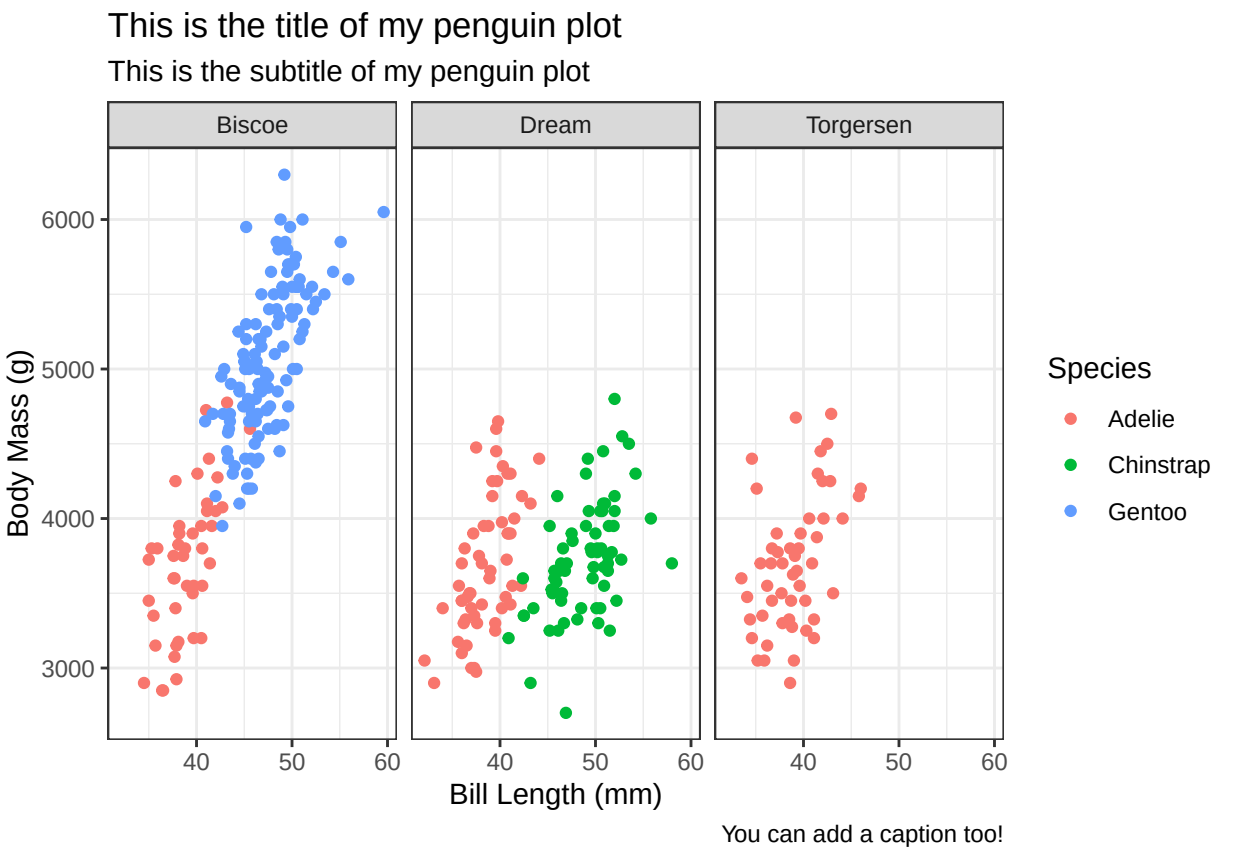



```
my_plot +
  theme_bw() +
  theme(
    plot.title = element_text(size = 25),
    axis.text = element_text(
      size = 14,
      color = "red",
      family = "serif"
    ),
    legend.text = element_text(
      size = 16,
      color = "steelblue",
      family = "mono"
    )
  )
```



You can also separate data out by variables and plot them separately, this is known as faceting. For example, if you want to look at the plots for each island where the penguins live:

```
my_plot+
  theme_bw()+
  facet_wrap(~island)
```

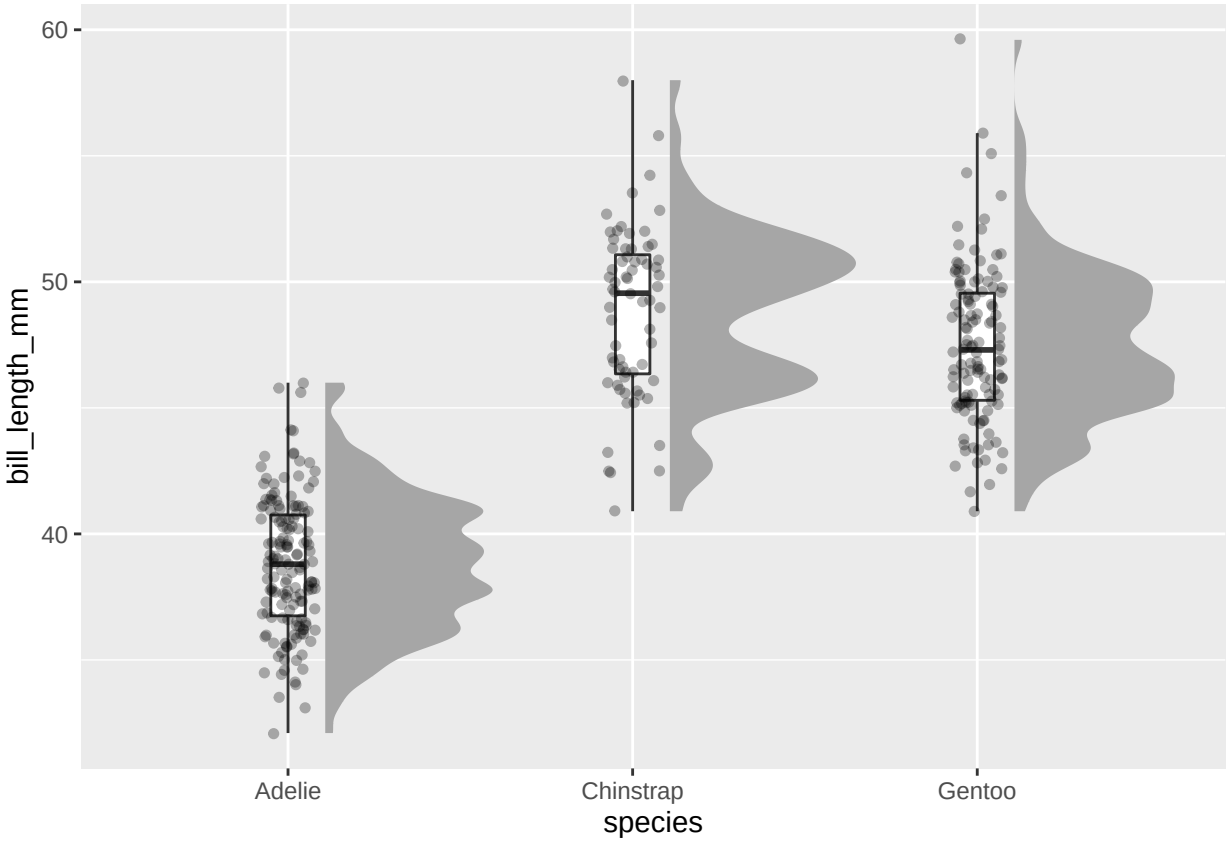


One of my favorite types of plots is a raincloud plot. This type of plot would replace a bar plot, violin plot, or simple boxplot. Raincloud plots do not hide the data and allow you to see the full distribution of the data clearly and quickly. A great tutorial is here: https://www.cedricscherer.com/2021/06/06/visualizing-distributions-with-raincloud-plots-and-how-to-create-them-with-ggplot2/

First we will install a couple of libraries, {ggdist} and {gghalves}.

```
packages = c("ggdist", "gghalves")
if (length(setdiff(packages, rownames(installed.packages()))) > 0) {
  install.packages(setdiff(packages, rownames(installed.packages())))
}
library(ggdist)
library(gghalves)

my_penguins %>%
  ggplot(aes(x = species, y = bill_length_mm)) +
  ggdist::stat_halfeye(
    adjust = .5,
    width = .6,
    .width = 0,
    #this turns off the interval bar
    justification = -.2,
    point_colour = NA #this turns off the median point
  ) +
  geom_boxplot(width = 0.1,
               outlier.shape = NA) + #this makes outliers go away
  geom_point(
```

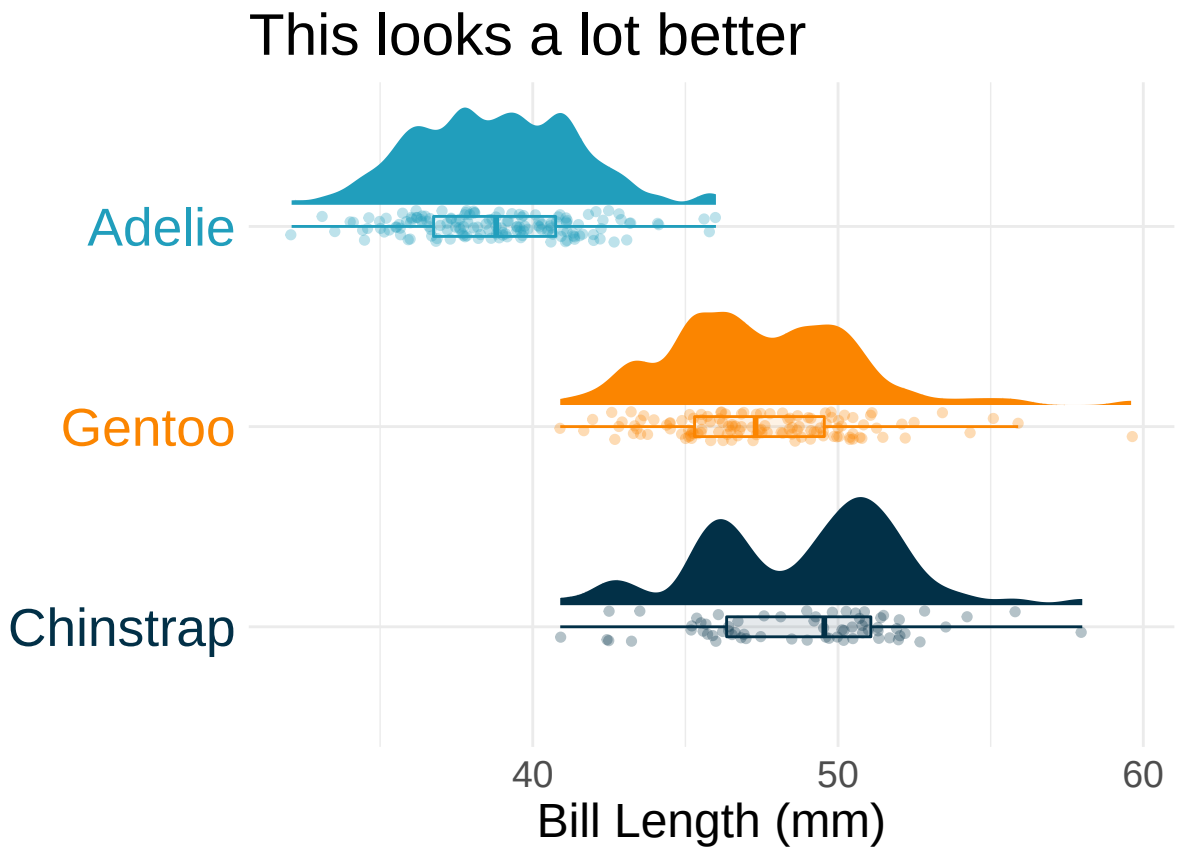
```
    size = 1.3,
    alpha = .3,
    position = position_jitter(seed = 1, width = .08)
  )
```



This is a good base, but we can really make it pop by adding some color and flipping the axes, among other things.

```
my_penguins %>%
  #filter(!is.na(bill_length_mm)) %>%
  ggplot(aes(
    x = reorder(species,-bill_length_mm, median, na.rm = TRUE),
    y = bill_length_mm ,
    color = species,
    fill = species
  )) +
  ggdist::stat_halfeye( #adds the density plot
    adjust = .5,
    width = .6,
    .width = 0, #this turns off the interval bar
    justification = -.2,
    point_colour = NA #this turns off the median point
  ) +
  geom_boxplot(width = 0.1,
               outlier.shape = NA, #this makes outliers go away
               alpha = 0.1) +
```

```
geom_point(
  size = 1.3,
  alpha = .3,
  position = position_jitter(seed = 1, width = .08) #jitter the points
) +
coord_flip() + #flip the x and y axes
labs(x = "",
     y = "Bill Length (mm)",
     title = "This looks a lot better") +
theme_minimal() +
theme(
  legend.position = "none", #turn off the legend
  text = element_text(size = 18),
  axis.text.y = element_text(
    color = c("#023047", "#fb8500", "#219ebc"),
    size = 20
  )
) +
scale_color_manual(values = c("#219ebc", "#023047", "#fb8500")) + #set the colors
scale_fill_manual(values = c("#219ebc", "#023047", "#fb8500"))
```



9. Making publication quality plots and panels

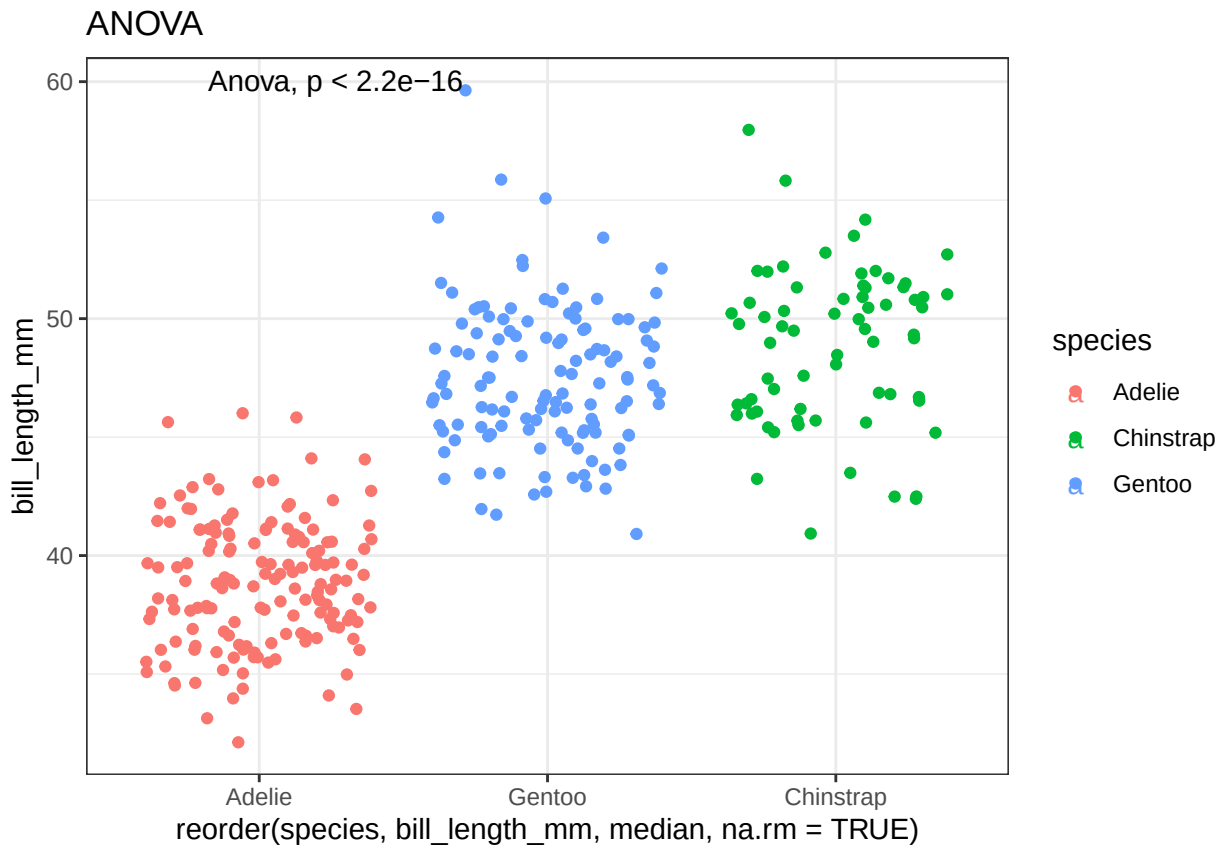
With the help of {ggplot2} and a few other packages you can make your figures all in the comfort of R.

The first is {ggpubr}. It aims to make publication quality plots right off the bat.

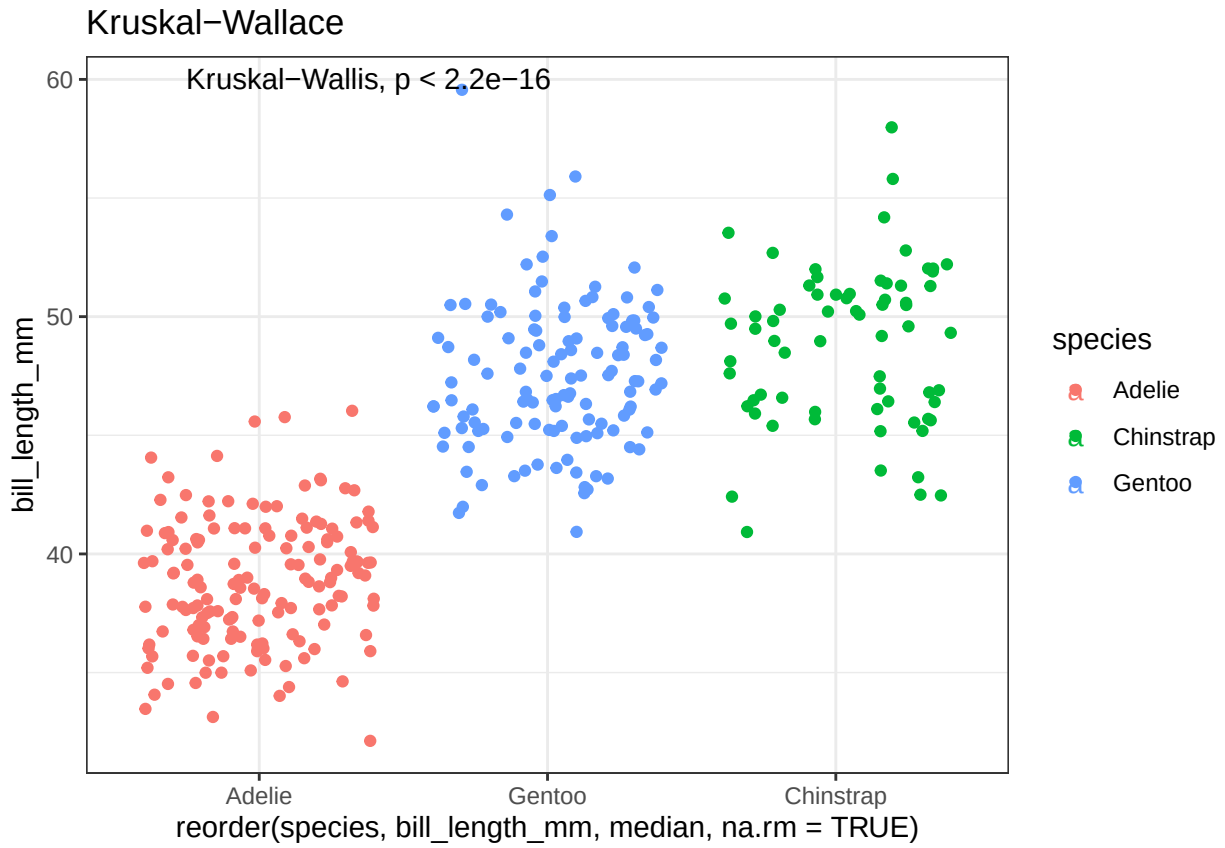
It has numerous functions for adding things like statistical significance automatically. To run ANOVA-like tests, just simply call stat_compare_means.

A good overview of this is here: http://www.sthda.com/english/articles/24-ggpubr-publication-ready-plots/76-add-p-values-and-significance-levels-to-ggplots/

```
packages = c("ggpubr")
if (length(setdiff(packages, rownames(installed.packages()))) > 0) {
  install.packages(setdiff(packages, rownames(installed.packages())))
}
library(ggpubr)

#anova
my_penguins %>%
  #filter(!is.na(bill_length_mm)) %>%
  ggplot(aes(
    x = reorder(species,bill_length_mm, median, na.rm = TRUE),
    y = bill_length_mm ,
    color = species,
    fill = species
  ))+
  geom_jitter()+
  labs(title = "ANOVA")+
  stat_compare_means(method = "anova")+
  theme_bw()
```



```r
#kruskal
my_penguins %>%
  #filter(!is.na(bill_length_mm)) %>%
  ggplot(aes(
    x = reorder(species,bill_length_mm, median, na.rm = TRUE),
    y = bill_length_mm ,
    color = species,
    fill = species
  ))+
  geom_jitter()+
  labs(title = "Kruskal-Wallace")+
  stat_compare_means()+
  theme_bw()
```

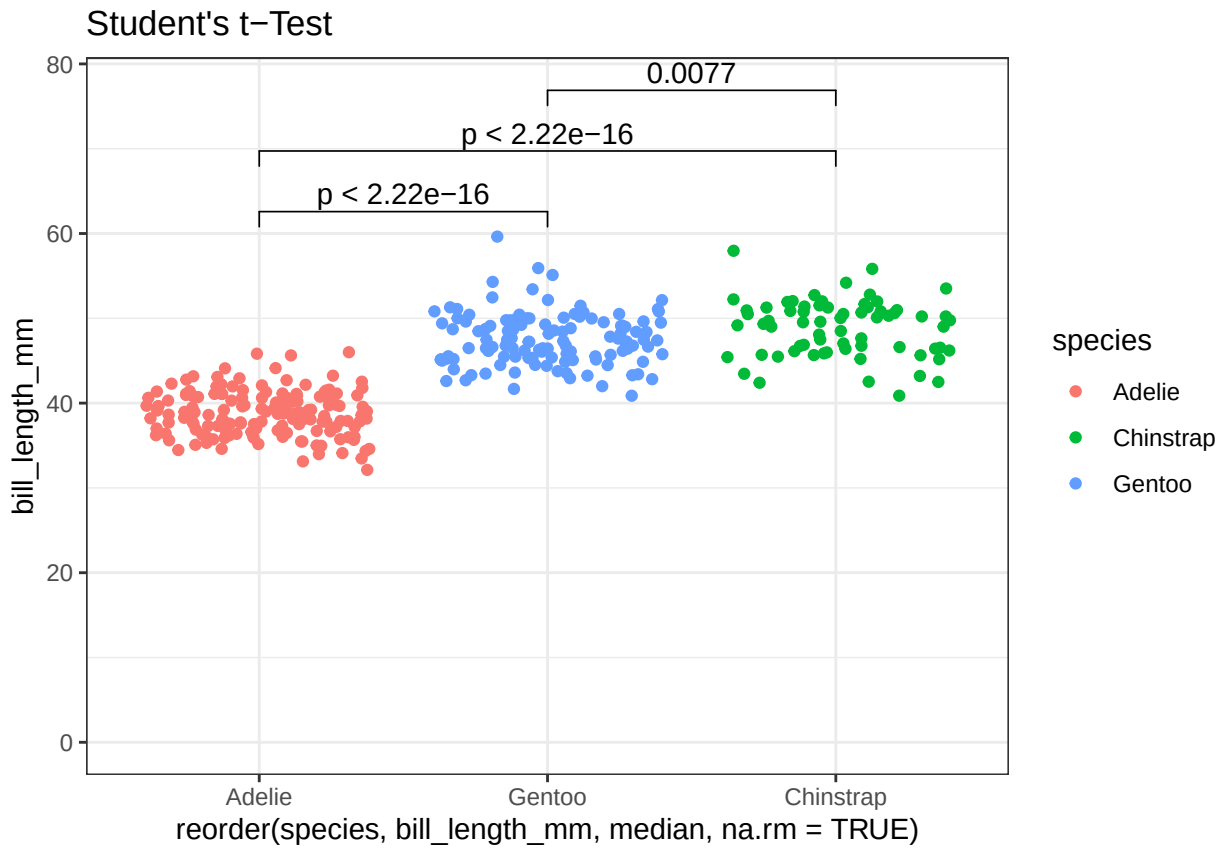


If you want to do pairwise comparison, you have to specify which comparisons you want to make.

```r
penguin_comparisons <- list( c("Adelie", "Gentoo"), c("Adelie", "Chinstrap"), c("Gentoo", "Chinstrap"))

#non-parametric
my_penguins %>%
  #filter(!is.na(bill_length_mm)) %>%
  ggplot(aes(
    x = reorder(species,bill_length_mm, median, na.rm = TRUE),
    y = bill_length_mm ,
    color = species,
    fill = species
```

```
  ))+
  geom_jitter()+
  labs(title = "Wilcoxon")+
  expand_limits(y=0)+ # this makes sure the y axis starts at zero
  stat_compare_means(comparisons = penguin_comparisons)+
  theme_bw()
```

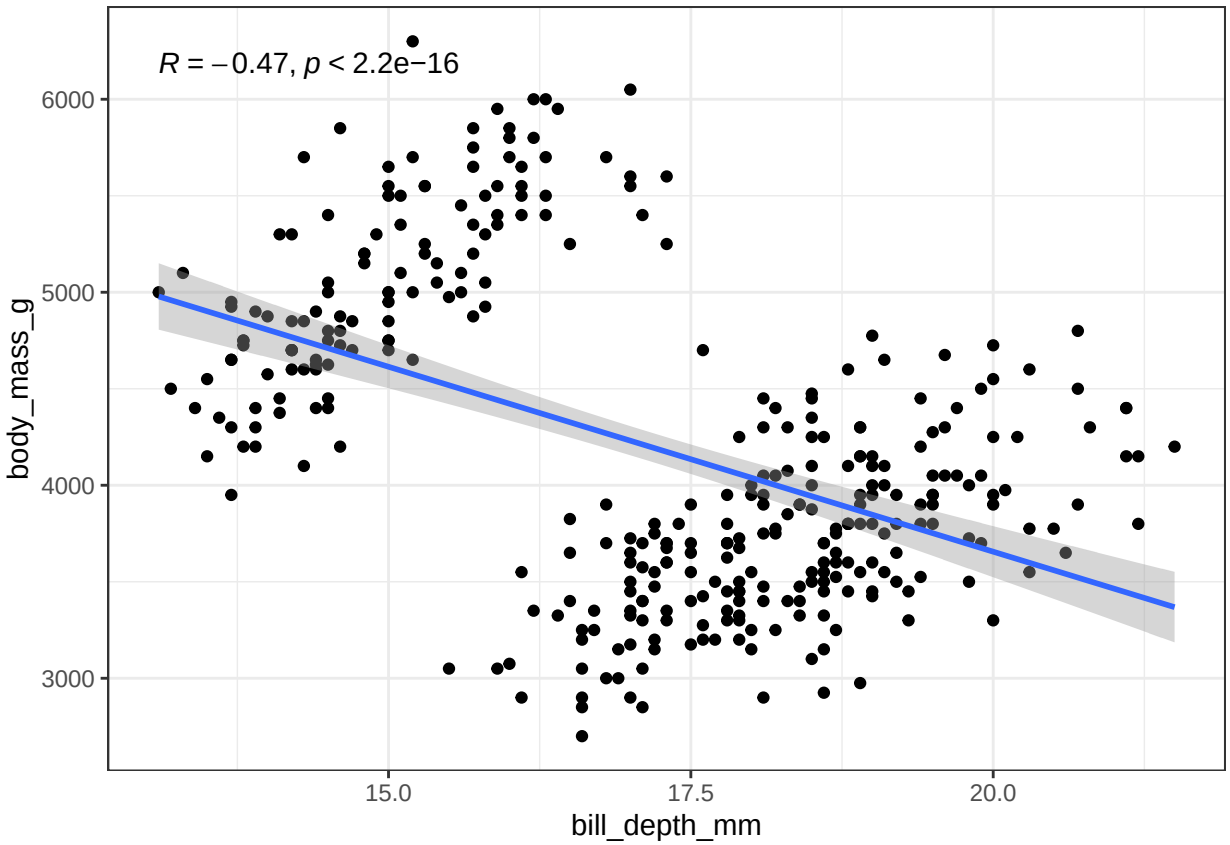

```
#t-test
my_penguins %>%
  #filter(!is.na(bill_length_mm)) %>%
  ggplot(aes(
    x = reorder(species,bill_length_mm, median, na.rm = TRUE),
    y = bill_length_mm ,
    color = species,
    fill = species
  ))+
  geom_jitter()+
  labs(title = "Student's t-Test")+
  expand_limits(y=0)+ # this makes sure the y axis starts at zero
  stat_compare_means(comparisons = penguin_comparisons, method = "t.test")+
  theme_bw()
```



Add the correlation coefficient to a plot by using stat_cor():

```
my_penguins %>%
  ggplot(aes(x=bill_depth_mm, y=body_mass_g))+
  geom_point()+
  geom_smooth(method = "lm")+
  stat_cor(method = "pearson")+
  theme_bw()
```

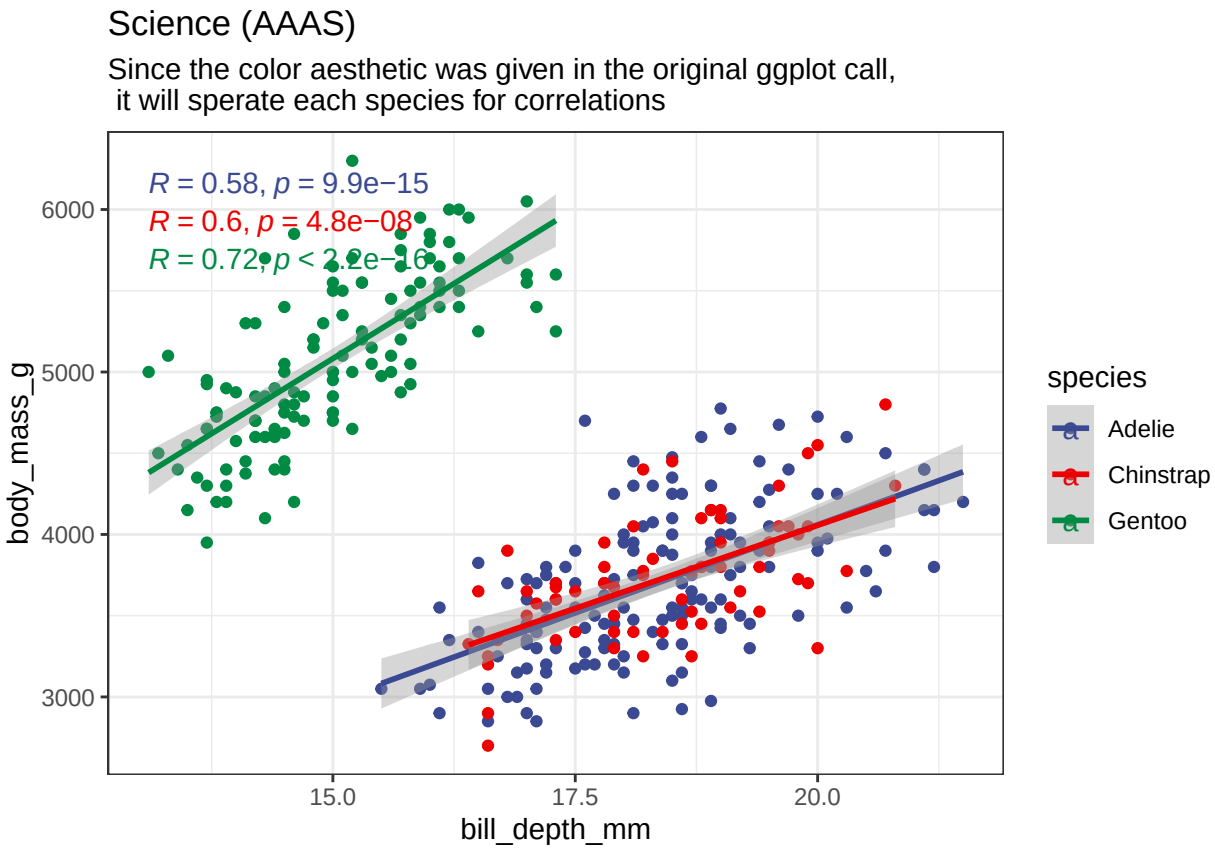
```
## `geom_smooth()` using formula 'y ~ x'
```



The default colors in R leave a lot to be desired. {ggsci} has color palettes for many different journals and publishing groups built in, as well as some fun ones like Futurama and Star Trek. Also depending on where you set your color aesthetic, you can either get the correlations between all penguins, or those in a species.

```
packages = c("ggsci")
if (length(setdiff(packages, rownames(installed.packages()))) > 0) {
  install.packages(setdiff(packages, rownames(installed.packages())))
}
library(ggsci)

#Science
my_penguins %>%
  ggplot(aes(x=bill_depth_mm, y=body_mass_g, color=species))+ #color here
  geom_point()+
  geom_smooth(method = "lm")+
  stat_cor(method = "pearson")+
  theme_bw()+
  scale_color_aaas()+
  labs(title="Science (AAAS)",
       subtitle = "Since the color aesthetic was given in the original ggplot call,\n it will sperate ea
```

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

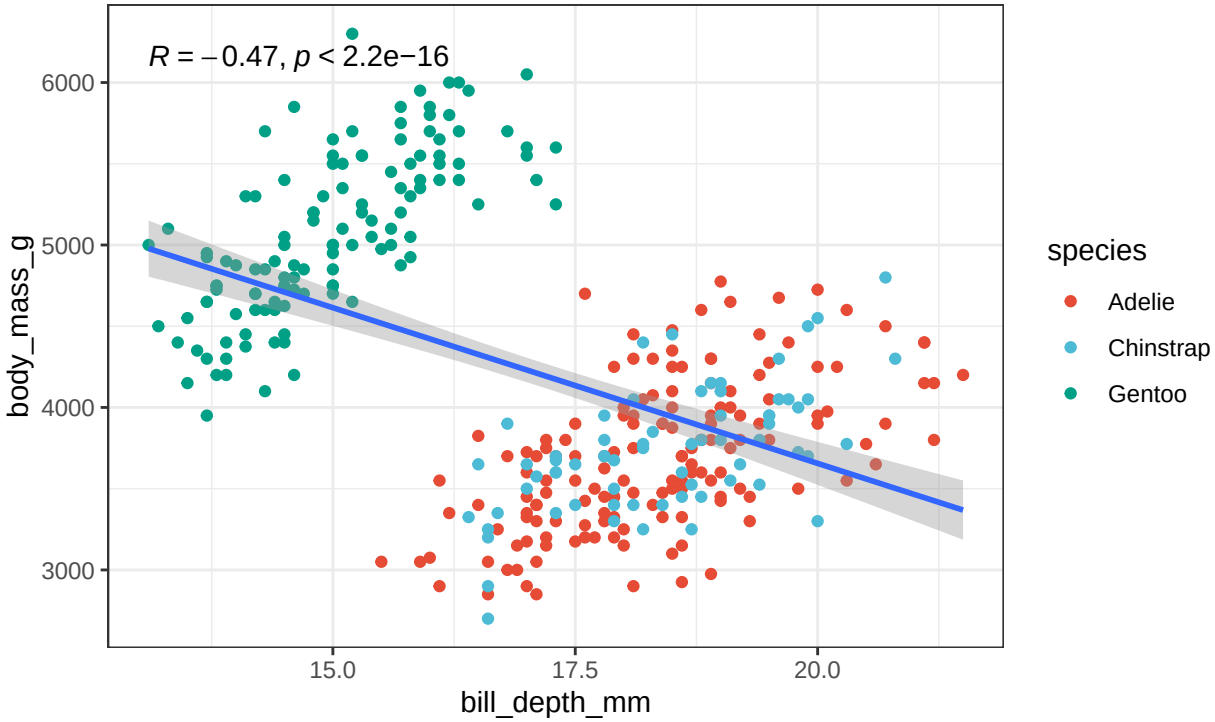



```
#Nature
my_penguins %>%
  ggplot(aes(x=bill_depth_mm, y=body_mass_g))+
  geom_point(aes(color=species))+ #color here
  geom_smooth(method = "lm")+
  stat_cor(method = "pearson")+
  theme_bw()+
  scale_color_npg()+
  labs(title="Nature Publishing Group",
       subtitle = "Since the color aesthetic was given in the geom_point ggplot call,\n it provide a co
```

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

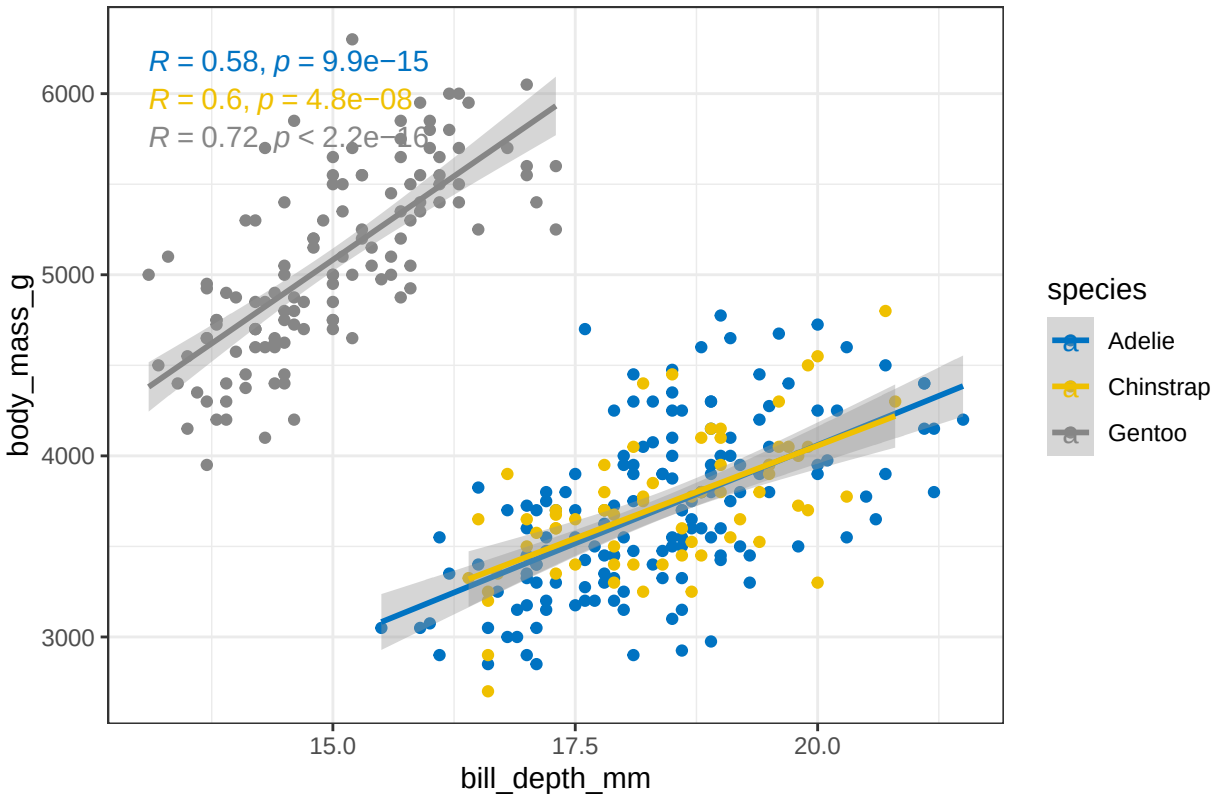
Nature Publishing Group

Since the color aesthetic was given in the geom_point ggplot call,
it provide a correlation for all data points



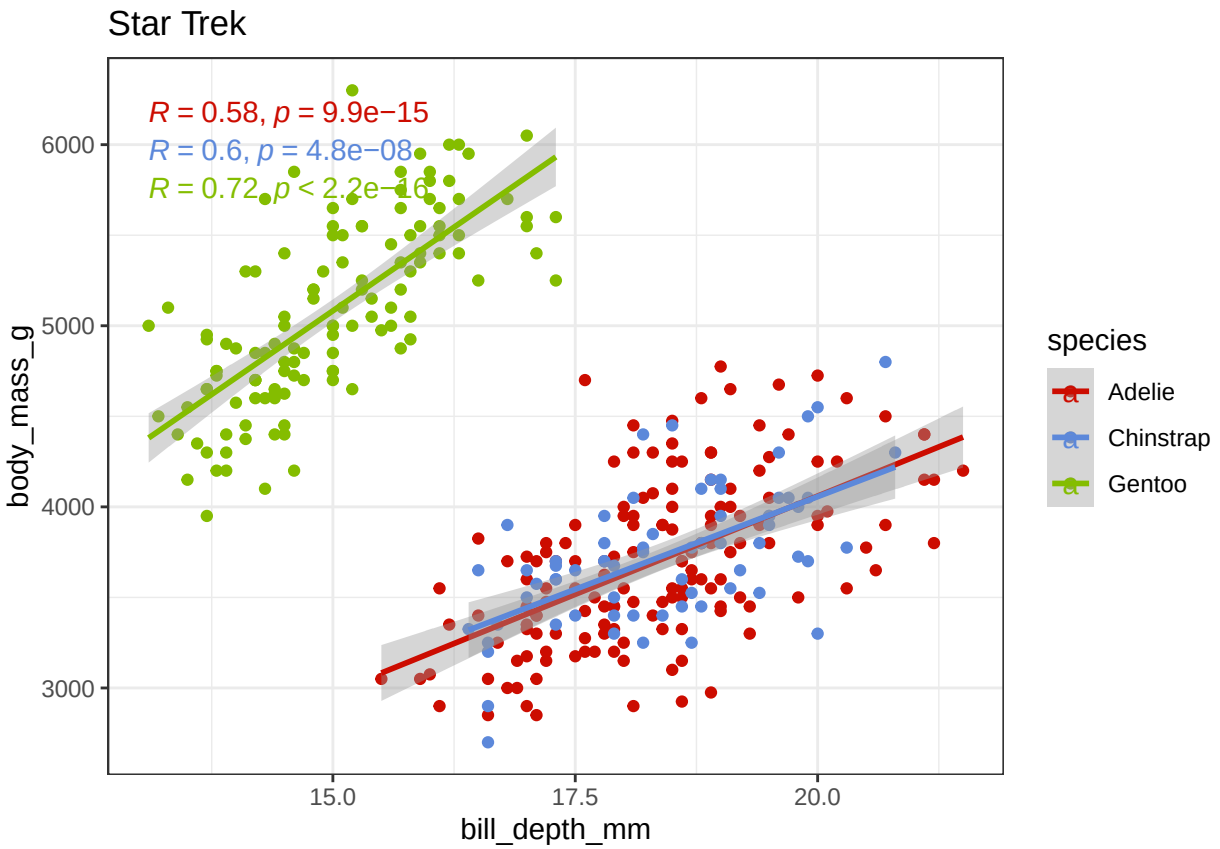

```
my_penguins %>%
  ggplot(aes(x=bill_depth_mm, y=body_mass_g, color=species))+
  geom_point()+
  geom_smooth(method = "lm")+
  stat_cor(method = "pearson")+
  theme_bw()+
  scale_color_jco()+
  labs(title="Journal of Clinical Oncology")
```

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




```
my_penguins %>%
  ggplot(aes(x=bill_depth_mm, y=body_mass_g, color=species))+
  geom_point()+
  geom_smooth(method = "lm")+
  stat_cor(method = "pearson")+
  theme_bw()+
  scale_color_startrek()+
  labs(title="Star Trek")
```

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



In order to make panels, use {patchwork}. {patchwork} allows you to assemble plots into any type of layout you'd like as well as providing annotation. First we will load the package and save a few plots as variables.

For more information see here: https://patchwork.data-imaginist.com/

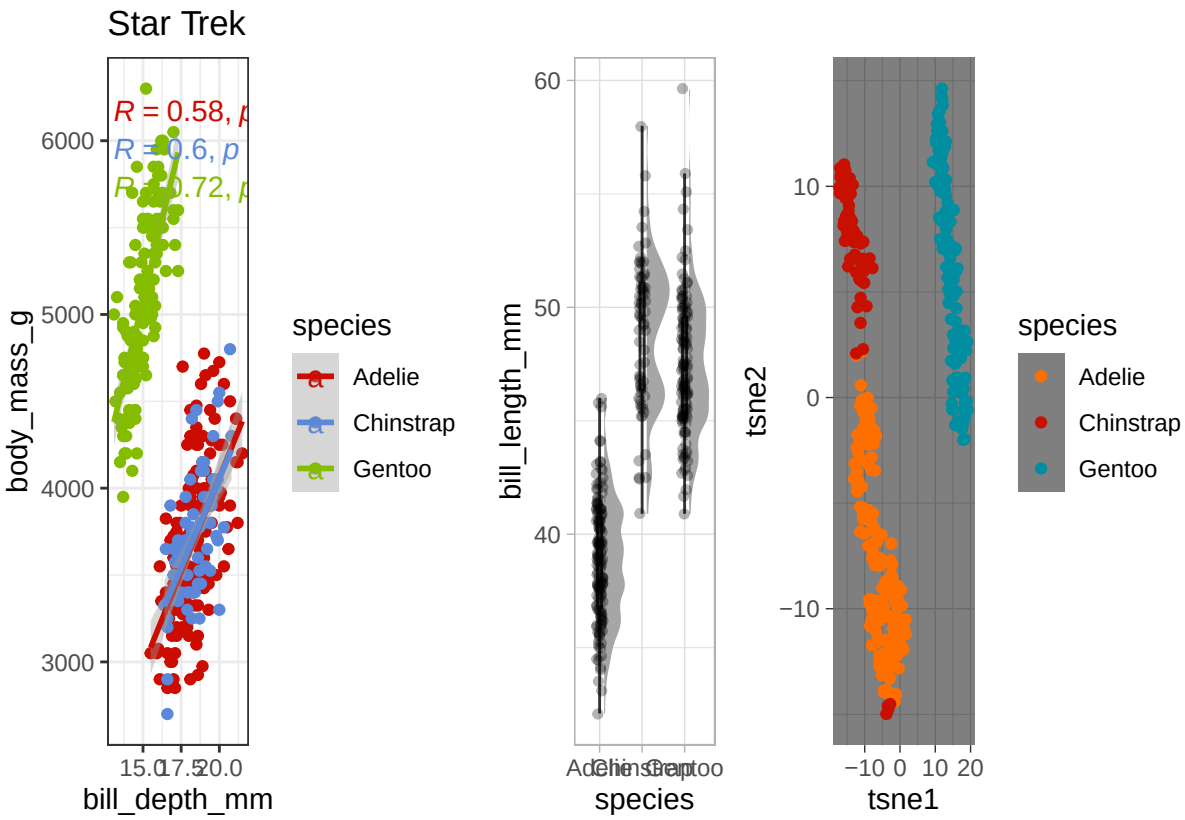
```
packages = c("patchwork")
if (length(setdiff(packages, rownames(installed.packages()))) > 0) {
  install.packages(setdiff(packages, rownames(installed.packages())))
}
library(patchwork)

plot1= my_penguins %>%
  ggplot(aes(x=bill_depth_mm, y=body_mass_g, color=species))+
  geom_point()+
  geom_smooth(method = "lm")+
  stat_cor(method = "pearson")+
  theme_bw()+
  scale_color_startrek()+
  labs(title="Star Trek")

plot2= my_penguins %>%
  ggplot(aes(x = species, y = bill_length_mm)) +
  ggdist::stat_halfeye(
    adjust = .5,
    width = .6,
    .width = 0,
    #this turns off the interval bar
```

```
    justification = -.2,
    point_colour = NA #this turns off the median point
  ) +
  geom_boxplot(width = 0.1,
               outlier.shape = NA) + #this makes outliers go away
  geom_point(
    size = 1.3,
    alpha = .3,
    position = position_jitter(seed = 1, width = .08)
  )+
  theme_light()

plot3=my_penguins %>%
  filter(!is.na(bill_length_mm)) %>%
  mutate(tsne1 = tsne$Y[, 1], #this adds the 1st tsne component
         tsne2 = tsne$Y[, 2]) %>%  #this adds the 1st tsne component
  ggplot(aes(x = tsne1, y = tsne2, color = species)) +
  geom_point(size = 1.5) +
  scale_color_futurama()+
  theme_dark()
```

Once we have the plots, we can start putting them together. You can pretty much treat each plot now as a variable in a math equation, so if you want all three to be next to each other horizontally:

```
plot1 + plot2 + plot3
```

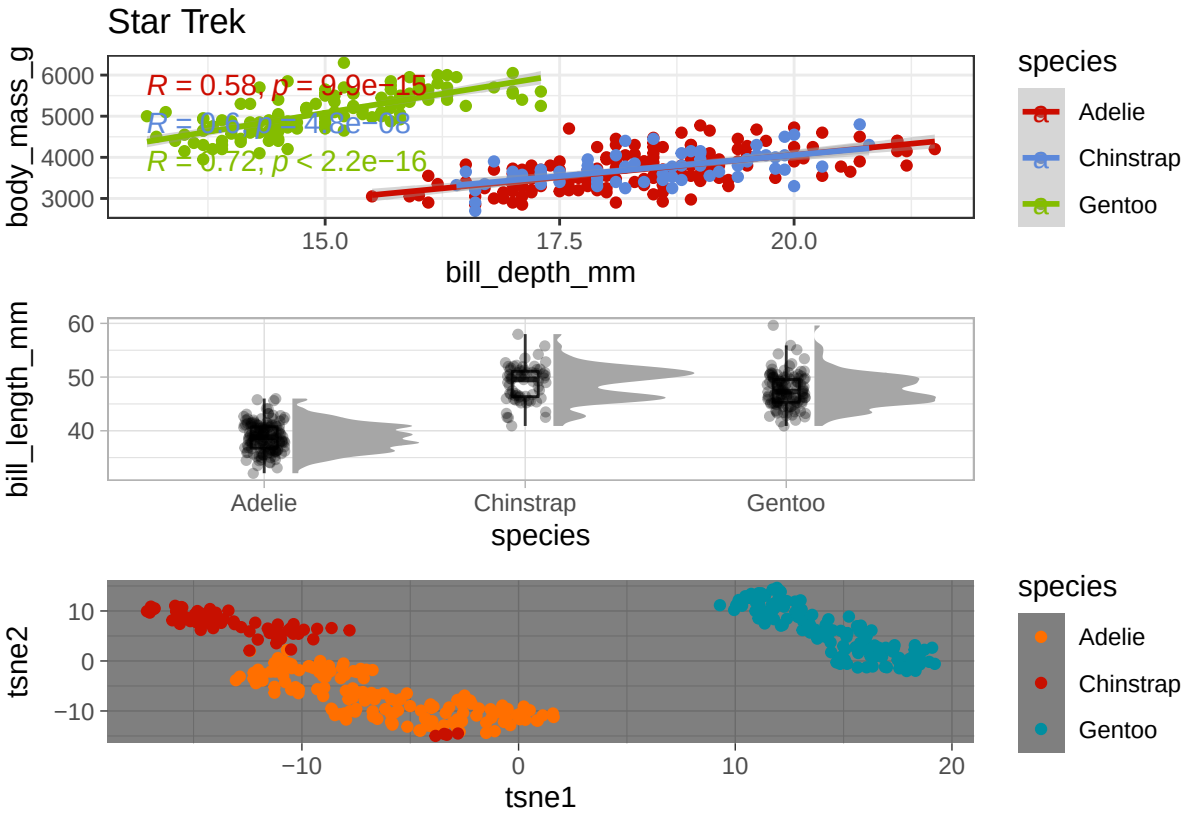
```
## `geom_smooth()` using formula 'y ~ x'
```



Or vertically:

```
plot1 / plot2 / plot3
```

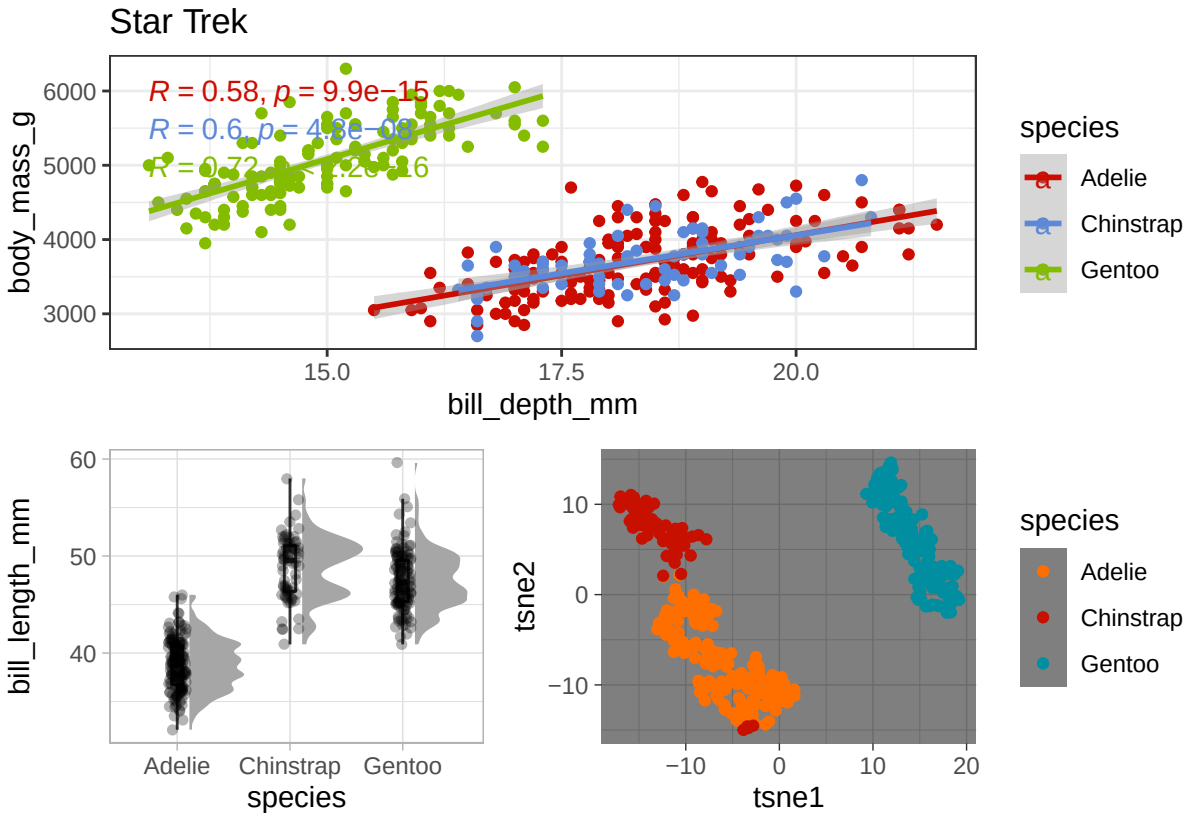
```
## `geom_smooth()` using formula 'y ~ x'
```



You can put one on top of two, or compress two into the space of one, whatever you'd like:

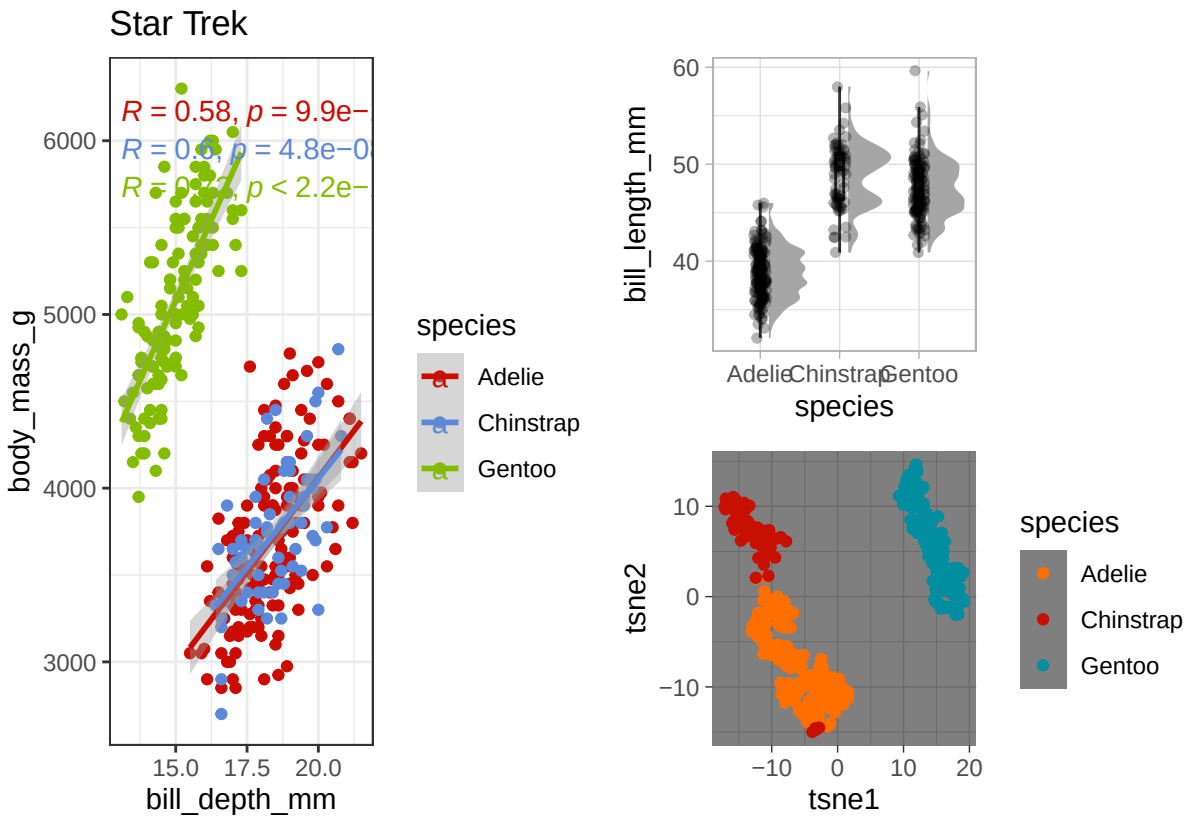
```
plot1 / (plot2 + plot3)
```

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




```
right_side = plot2/plot3
plot1 + right_side
```

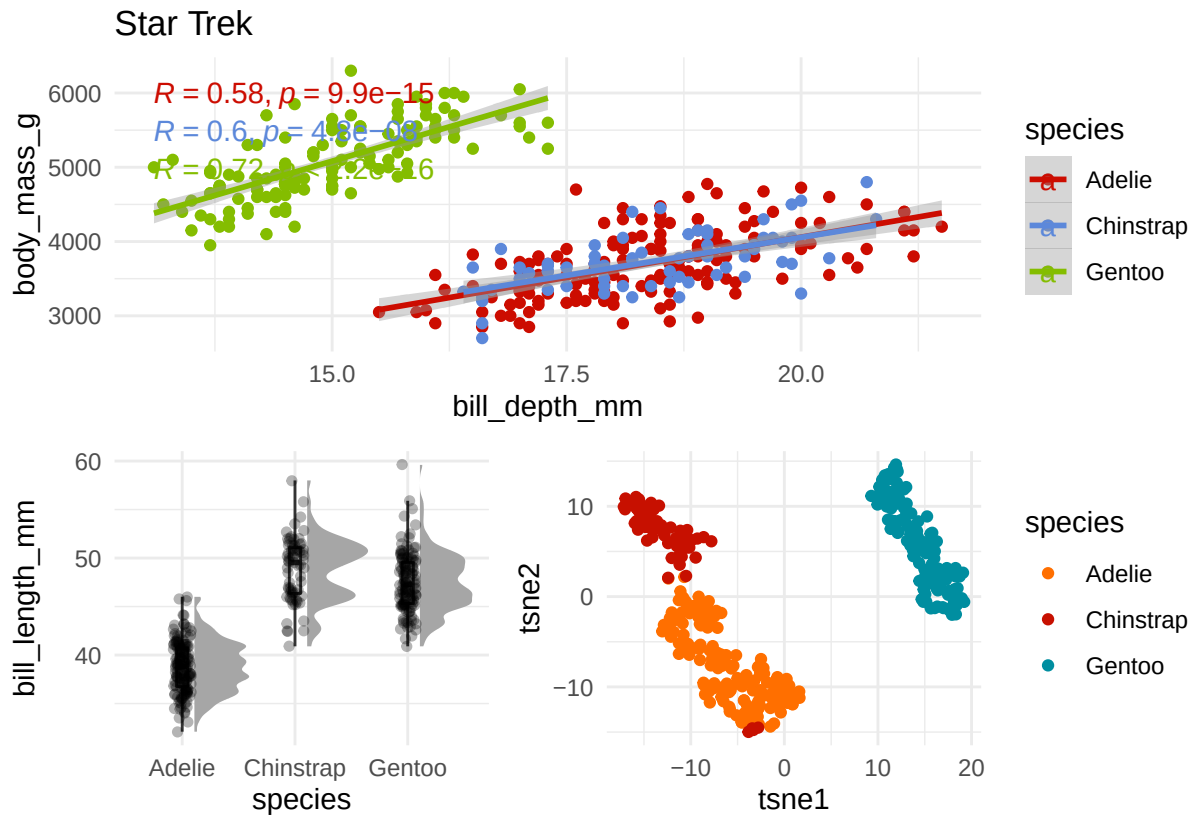
```
## `geom_smooth()` using formula 'y ~ x'
```



If the themes are different, you can make them all the same with &:

```
plot1 / (plot2 + plot3) & theme_minimal()
```

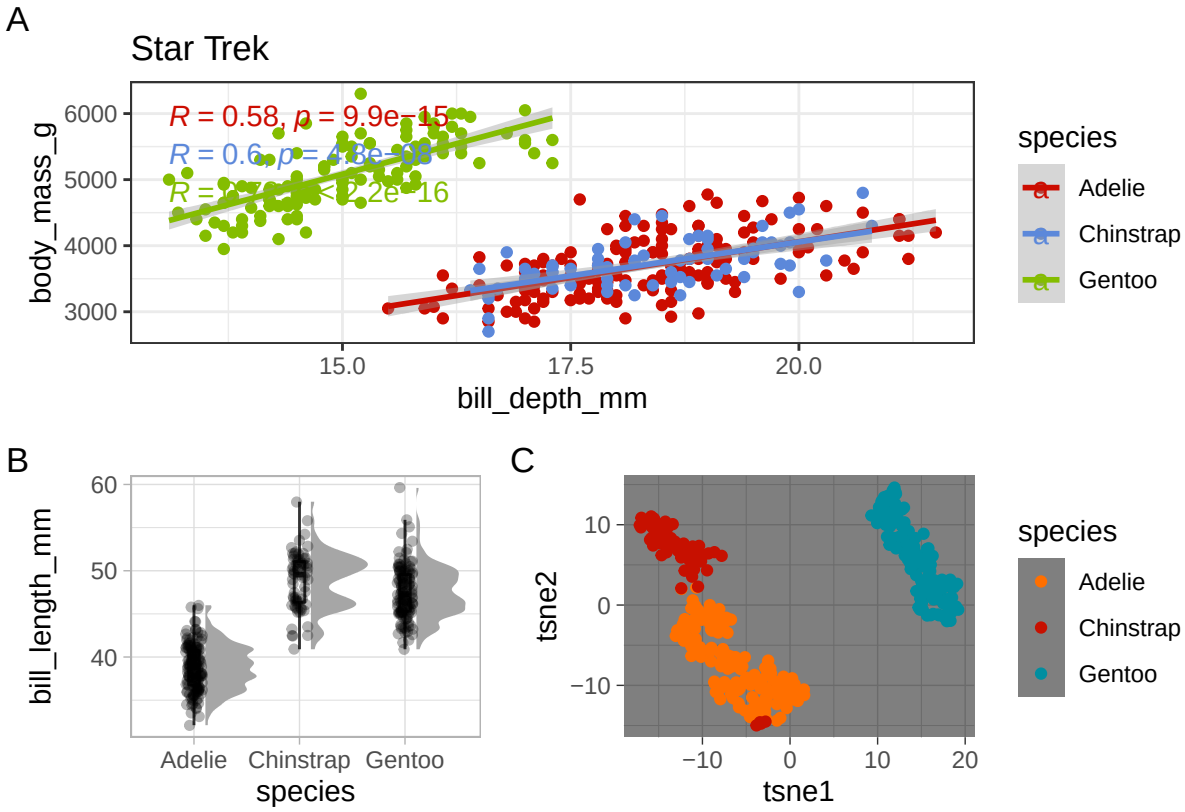
```
## `geom_smooth()` using formula 'y ~ x'
```



To add annotations you use plot_annotation():

```
plot1 / (plot2 + plot3) +
  plot_annotation(tag_levels = "A")
```

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

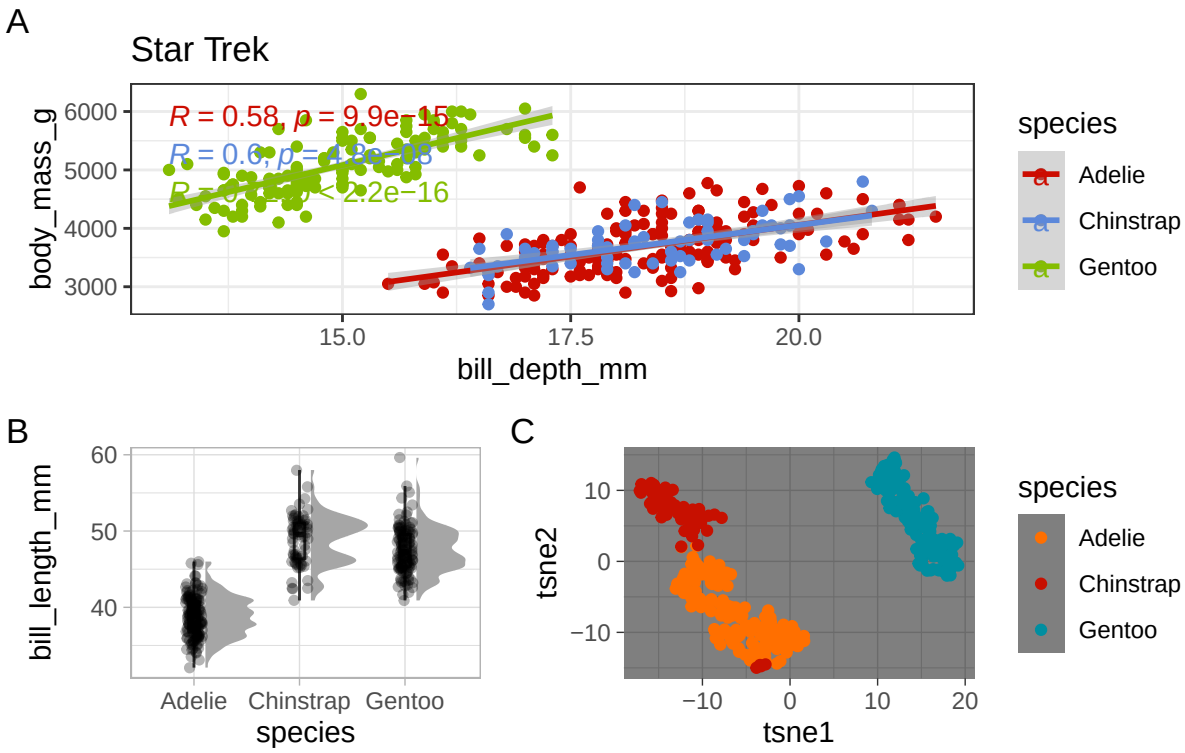


And can add an overall title and change the font:

```
plot1 / (plot2 + plot3) +
  plot_annotation(
    tag_levels = "A",
    title = "You can add a title",
    theme = theme(plot.title = element_text(size = 24, family = "mono", color="steelblue"))
  )
```

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

You can add a title



Conclusion

This tutorial is only just scratching the surface of what you can do in R. If you ever have questions or need help feel free to ask. If you just want some pretty plot and don't want to do it yourself, just ask. The online R community is very welcoming and if you have questions, almost everything has been answered online with a tutorial that is better than this one. Have fun.