# Programacion en R

Alfredo Aburto Alcudia: Curso de analisis de datos con Google

2024-01-02

## Contents

## Introducción

El presente documento tiene como objetivo mostrar las notas realizadas durante la formación de Google como analista de datos junior. Lo anterior, con el fin de fungir como guía de consulta en temas básicos y como última instancia prácticar en la utilización de R markdown para la creación de informes. Ya que anteriormente todas las descripciones teoricas se habian realizado como comentarios sobre archivos de extension ".R".

## Estructuras de datos en R

Una forma de crear una vector es utilizar la función c() llamada función "combinar" Los vectores atómicos solo pueden contener elementos del mismo tipo.

### Vectores

```
vector_01 <- c(2.5,4.9,1.3)
vector_01
```

### Definición de un vector númerico

```
## [1] 2.5 4.9 1.3
```

```
vector_02 <- c(2L, 3L, 9L)
vector_02
```

### Definición de un vector de números enteros

```
## [1] 2 3 9
```

```
vector_03 <- c("Ciencia", "Datos", "Tecnología")
vector_03
```

### Definición de un vector de caracteres y valores logicos

```
## [1] "Ciencia"    "Datos"      "Tecnología"
```

```
vector_04 <- c(TRUE, TRUE, FALSE, TRUE, FALSE, FALSE)
vector_04
```

```
## [1]  TRUE  TRUE FALSE  TRUE FALSE FALSE
```

### DETERMINAR PROPIEDADES DE VECTORES (Tipo y longitud)

```
typeof(vector_03)
```

**Determinar el tipo de vector con el que se trabaja**

```
## [1] "character"
```

```
is.logical(vector_02)
```

**Verificar si un vector es de un tipo específico usando la función 'is'**

```
## [1] FALSE
```

```
is.integer(vector_02)
```

```
## [1] TRUE
```

```
length(vector_04)
```

**Determinar la longitud del vector**

```
## [1] 6
```

**Asignación de nombres a los elementos de un vector** Definimos el vector x como ejemplo

```
x <- c(1, 3, 5)
```

Asignamos nombres a cada elemento del vector x con la función 'names()'

```
names(x) <- c("a", "b", "c")
x # Pasamos por consola el vector x
```

```
## a b c
## 1 3 5
```

La asignación de nombres a los elementos de un vector puede ser útil para hacer que el código sea más legible y comprensible, ya que puedes referirte a los elementos por sus nombres en lugar de sus índices. Por ejemplo, para acceder al elemento "c" del vector x

```
x["c"]
```

```
## c
## 5
```

Con lo anterior enviamos por pantalla al elemento c del vector x

### LISTAS

Las listas difieren de los vectores atómicos porque sus elementos pueden ser de cualquier tipo. Además las listas pueden contener a otras listas.

```
lista_01 <- list("alfredo", 17L, 8.3, FALSE)
lista_01
```

**Definición de una lista utilizando la funcion list()**

```
## [[1]]
## [1] "alfredo"
##
## [[2]]
## [1] 17
##
## [[3]]
## [1] 8.3
##
## [[4]]
## [1] FALSE
```

```
lista_02 <- list(2L, 3.9273, "a", TRUE, list("b", FALSE))
lista_02
```

**Definición de listas anidadas**

```
## [[1]]
## [1] 2
##
## [[2]]
## [1] 3.9273
##
## [[3]]
## [1] "a"
##
## [[4]]
## [1] TRUE
##
## [[5]]
## [[5]][[1]]
## [1] "b"
##
## [[5]][[2]]
## [1] FALSE
```

**Determinar la estructura de una lista** Si se desea saber qué tipos de elementos contiene una lista, puedes utilizar la función str().

```
str(lista_02)
```

```
## List of 5
##  $ : int 2
##  $ : num 3.93
##  $ : chr "a"
##  $ : logi TRUE
##  $ :List of 2
##   ..$ : chr "b"
##   ..$ : logi FALSE
```

Al ejecutar la función str(), R mostrará la estructura de datos de la lista mediante la descripción de sus elementos y tipos

**Nombrando los elementos de una lista** Al igual que con los vectores, podemos nombrar los elementos de una lista, asignandole a cada elemento un nombre cuando definimos la lista.

```r
lista_03 = list('Temperatura (K)' = 20.5, "Presion (atm)" = 1L)
lista_03 # Enviamos por pantalla lista_03
```

```
## $`Temperatura (K)`
## [1] 20.5
##
## $`Presion (atm)`
## [1] 1
```

---

## Fechas y horas

Para trabajar con fechas y horas utilizammos el paqute lubridate. Para ello es necesario instalar y cargar los paquetes tidyverse y lubridate el cual es parte de tidyverse - Instalacion de paquete tydyverse

```r
install.packages("tidyverse")
```

```
## Installing package into '/home/alfredo_aburto/R/x86_64-pc-linux-gnu-library/4.3'
## (as 'lib' is unspecified)
```

- Carga de paquetes tidyverse y lubridate utilizando la función 'library()'

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.2
## -- 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
```

Lubridate es un paquete diseñado para trabajar con fechas y horas de una manera más sencilla y cómoda. Algunas de las funciones y características clave incluyen: - Parseo de fechas: Facilita la conversión de texto a objetos de fecha y hora. - Manipulación de fechas: Proporciona funciones para extraer y manipular componentes de fechas, como días, meses, años, horas y minutos. - Operaciones con fechas: Permite realizar cálculos y operaciones entre fechas de manera intuitiva. - Manejo de zonas horarias: Facilita el trabajo con zonas horarias y la conversión entre ellas.

### Trabajando con fechas y horas

**Tipos** En R hay tres tipos de datos que hacen referencia a un instante en el tiempo - Fecha: ("2002-10-17") - Hora de un dia: ("23:54:46 UTC") - Fecha-hora: ("2010-04-25 16:12:05 UTC")

La hora se expresa en UTC, que quiere decir Hora Universal Coordinada, más comúnmente conocida como tiempo civil. Este es el estándar principal que regula los relojes y la hora mundial.

Obtener la fecha actual (mes, año, día)

```r
today()
```

```
## [1] "2024-01-02"
```

Obtener fecha y hora actual

```
now()
```

```
## [1] "2024-01-02 19:09:49 CST"
```

Al trabajar con R, hay tres modos posibles de crear formatos de fecha-hora: - Desde una cadena - Desde una fecha individual - Desde un objeto de fecha/hora existente

R crea fechas en el formato estándar yyyy-mm-dd (año-mes-día) por defecto.

**Convertir a partir de una cadena** Los datos de fecha/hora a menudo se expresan como cadenas. Puedes convertir cadenas en fechas y fecha-hora utilizando las herramientas provistas por lubridate. Estas herramientas automáticamente trabajan sobre el formato de fecha/hora.

1. Primero, identifica el orden en el año, el mes y el día que aparecen en tus fechas.
2. Ordena las letras y, m y d (año, mes y día) en el mismo orden. Eso te dará el nombre de la función lubridate que analizará tu fecha.

Por ejemplo, para la fecha 2021-01-20, utilizarás el orden ymd:

```
fecha_01 <- ymd("2021-01-20")
fecha_01 # Enviamos por pantalla fecha_01
```

```
## [1] "2021-01-20"
```

mdy Febrero 20th, 2023

```
fecha_02 <- mdy("Febrero 20th, 2023")
fecha_02
```

```
## [1] "2023-02-20"
```

Estas funciones también toman números que no están entre comillas y los convierte al formato yyyy-mm-yy.

```
fecha_03 <- ymd(20210120)
fecha_03
```

```
## [1] "2021-01-20"
```

**Crear componentes de fecha-hora** La función ymd() y sus variantes crean fechas. Para crear una fecha-hora desde una fecha, agrega un guion bajo y una o más de las letras h, m y s (horas, minutos y segundos) al nombre de la función:

```
fecha_04 <- ymd_hms("2021-01-20 20:11:59")
fecha_04
```

```
## [1] "2021-01-20 20:11:59 UTC"
```

Notar el orden en el que se ponen las letras en la función

```
fecha_05 <- ymd_hms("2021-01-20 20:11:59")
fecha_05
```

```
## [1] "2021-01-20 20:11:59 UTC"
```

**Cambiar entre objetos existentes de fecha-hora** Quizas se quiera cambiar entre una fecha-hora y una fecha. Se puede utilizar la función as_date() para convertir una fecha-hora en una fecha.

```
fecha_06 <- as_date(now())
fecha_06
```

```
## [1] "2024-01-02"
```

## Organizar datos

Si nuestra información no esta ordenada no podemos transformar la información en conocimiento. Ordenar nuestros datos nos permite conocer nuevos detalles de nuestros datos.

Para lograrlo utilizaremos las funciones: - arrange() - group_by() - filter()

1. Inserta el paquete tidyverse para poder utilizar su núcleo
2. Obtención de conjunto de datos

```
install.packages("palmerpenguins")
```

```
## Installing package into '/home/alfredo_aburto/R/x86_64-pc-linux-gnu-library/4.3'
## (as 'lib' is unspecified)
```

3. Cargamos nuestros datos

```
library(palmerpenguins)
```

### Ordenar datos por una columna especifica en este caso longitud de su pico

Los siguientes comandos crean un **tibble** y es importante recordar que estos comandos NO modifican mi conjunto de datos, solo estan temporalmente por pantalla

```
penguins %>% arrange(bill_length_mm) # Ordena ascendentemente
```

```
## # 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  Dream               32.1          15.5               188        3050
##  2 Adelie  Dream               33.1          16.1               178        2900
##  3 Adelie  Torgersen           33.5          19                 190        3600
##  4 Adelie  Dream               34            17.1               185        3400
##  5 Adelie  Torgersen           34.1          18.1               193        3475
##  6 Adelie  Torgersen           34.4          18.4               184        3325
##  7 Adelie  Biscoe              34.5          18.1               187        2900
##  8 Adelie  Torgersen           34.6          21.1               198        4400
##  9 Adelie  Torgersen           34.6          17.2               189        3200
## 10 Adelie  Biscoe              35            17.9               190        3450
## # i 334 more rows
## # i 2 more variables: sex <fct>, year <int>
```

```
penguins %>% arrange(-bill_length_mm) # Ordena descendente añandiendo "-"
```

```
## # 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 Gentoo    Biscoe           59.6          17                 230        6050
##  2 Chinstrap Dream            58            17.8               181        3700
##  3 Gentoo    Biscoe           55.9          17                 228        5600
##  4 Chinstrap Dream            55.8          19.8               207        4000
##  5 Gentoo    Biscoe           55.1          16                 230        5850
##  6 Gentoo    Biscoe           54.3          15.7               231        5650
##  7 Chinstrap Dream            54.2          20.8               201        4300
##  8 Chinstrap Dream            53.5          19.9               205        4500
##  9 Gentoo    Biscoe           53.4          15.8               219        5500
## 10 Chinstrap Dream            52.8          20                 205        4550
## # i 334 more rows
## # i 2 more variables: sex <fct>, year <int>
```

Si deseamos crear un nuevo marco de datos que contenga los datos ordentados seguiremos el siguiente ejemplo, con él logramos guardar datos limpios sin perder infromación del conjunto de datos original

```
penguins_bill_descen <- penguins %>% arrange(-bill_length_mm)
View(penguins_bill_descen)
```

### group_by()

Esta función suele combinarse con otras funciones, nos permite agrupar por alguna columna en particular y luego realizar una operación esos grupos

- Creamos nuestro grupo

```
penguins %>%
  group_by(island) %>% # Agrupamos por isla
  drop_na() %>% # Eliminamos filas con valores nulos en cualquier columna
  summarise(mean_bill_length_mm = mean(bill_length_mm))
```

```
## # A tibble: 3 x 2
##   island    mean_bill_length_mm
##   <fct>                   <dbl>
## 1 Biscoe                   45.2
## 2 Dream                    44.2
## 3 Torgersen                39.0
```

En el fragmento anterior la función summarise se utiliza para resumir los datos dentro de cada grupo, en este caso, calculando la media de una columna especifica.

Otro ejemplo de uso

```
penguins %>%
  group_by(island) %>%
  drop_na() %>%
  summarise(max_bill_length_mm = max(bill_length_mm))
```

```
## # A tibble: 3 x 2
##   island    max_bill_length_mm
##   <fct>                  <dbl>
## 1 Biscoe                  59.6
## 2 Dream                   58
## 3 Torgersen               46
```

Podemos utilizar tanto group_by como summarise para realizar los dos ejemplos anteriores simultasneamente:

```
penguins %>%
  group_by(species, island) %>%
  drop_na() %>%
  summarise(max_bl = max(bill_length_mm), mean_bl = mean(bill_length_mm))
```

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

```
## # A tibble: 5 x 4
## # Groups:   species [3]
##   species   island    max_bl mean_bl
##   <fct>     <fct>      <dbl>   <dbl>
## 1 Adelie    Biscoe      45.6    39.0
## 2 Adelie    Dream       44.1    38.5
## 3 Adelie    Torgersen   46      39.0
```

```
## 4 Chinstrap Dream       58      48.8
## 5 Gentoo    Biscoe      59.6    47.6
```

El fragmento anterior logra realizar varias tareas de limpieza gracias al uso de la canalización o pipes, el resultados es un conjunto de datos que muestra la longitud máxima del pico y el promedio de la longitud del pico por especie que se encuentra en cada isla.

### filter()

Supongamos que deseamos obtener solo los datos sobre los pinguinos Adelie.

```
penguins %>%
  filter(species == "Adelie") # Exactamente igual a la especia Adelie
```

```
## # A tibble: 152 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 142 more rows
## # i 2 more variables: sex <fct>, year <int>
```

## Transformación de Datos

A veces necesitamos dividir una variable entre múltiples columnas o combinar las columnas atuales, o incluso agregar nuevos valores a el marco de datos. Las funciones básicas incluyen:

- separate()
- unite()
- mutate()

### Creamos un marco de datos desde cero

```
id <- c(1:10)
name <- c("John Mendes", "Rob Stewart", "Rachel Abrahamson", "Christy Hickman",
          "Johnson Harper", "Candace Miller", "Carlson Landy", "Pansy Jordan",
          "Darius Berry", "Claudia Garcia")

job_title <- c("Professional", "Programmer", "Management", "Clerical", "Developer",
               "Programmer", "Management", "Clerical", "Developer", "Programmer")

employee <- data.frame(id, name, job_title)

print(employee)
```

```
##    id              name    job_title
## 1   1       John Mendes Professional
## 2   2       Rob Stewart   Programmer
## 3   3 Rachel Abrahamson   Management
```

```
## 4   4   Christy Hickman     Clerical
## 5   5    Johnson Harper    Developer
## 6   6    Candace Miller   Programmer
## 7   7     Carlson Landy   Management
## 8   8      Pansy Jordan     Clerical
## 9   9      Darius Berry    Developer
## 10 10    Claudia Garcia   Programmer
```

### separate()

Permite separar datos de un data frame de una columna en especifico con algun separador que podamos encontrar por ejemplo un espacio

```
employee_separate <- separate(employee, name, into=c("first_name", "last_name"), sep=" ")
employee_separate
```

```
##    id first_name  last_name    job_title
## 1   1       John     Mendes Professional
## 2   2        Rob    Stewart   Programmer
## 3   3     Rachel Abrahamson   Management
## 4   4    Christy    Hickman     Clerical
## 5   5    Johnson     Harper    Developer
## 6   6    Candace     Miller   Programmer
## 7   7    Carlson      Landy   Management
## 8   8      Pansy     Jordan     Clerical
## 9   9     Darius      Berry    Developer
## 10 10    Claudia     Garcia   Programmer
```

Con la función anterior decimos “del data-frame employee en la columna name separa el contenido de cada fila en dos columnas, el primer elemento de la separación lo colocas en la columna”first_name” y el segundo elemento en la columna “last_name”, para encontrar cada elemento debes buscar un espacio en blanco como caracter de separacion.

La función separate tiene un aliado, la función unite.

### unite()

Esta función nos permite fusionar columnas entre sí, es decir, lo opuesto a la funcion separate.

**Trabajando con el data.frame con dos columnas de nombre**

```
unite(employee_separate, 'name', first_name, last_name, sep=" ")
```

```
##    id             name    job_title
## 1   1       John Mendes Professional
## 2   2       Rob Stewart   Programmer
## 3   3 Rachel Abrahamson   Management
## 4   4   Christy Hickman     Clerical
## 5   5    Johnson Harper    Developer
## 6   6    Candace Miller   Programmer
## 7   7     Carlson Landy   Management
## 8   8      Pansy Jordan     Clerical
## 9   9      Darius Berry    Developer
## 10 10    Claudia Garcia   Programmer
```

Con la funcion anterior decimos “Usa el data frame employee_separate, nombra la columna donde se juntaran las columnas como ‘name, une las columnas ’first_name’, y ‘last_name’ y separalos con un espacio”

### mutate()

La función mutate()permite agregar una columna de datos es decir una nueva variable.

```
install.packages("palmerpenguins")
```

```
## Installing package into '/home/alfredo_aburto/R/x86_64-pc-linux-gnu-library/4.3'
## (as 'lib' is unspecified)
```

```
library(palmerpenguins)
data('penguins')
View(penguins)
```

En este dataset, la masa de los pinguinos se encuentra en gramos. Por lo que crearemos una nueva columna con la masa corporal en kg

```
penguins %>%
  mutate(body_mass_kg = body_mass_g/1000, flipper_length_m=flipper_length_mm/1000)
```

```
## # A tibble: 344 x 10
##    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 4 more variables: sex <fct>, year <int>, body_mass_kg <dbl>,
## #   flipper_length_m <dbl>
```

### Mismos datos diferentes resultados

Instalamos el paquete donde se encuentra el data set sobre el cuarteto de Anscombe

```
install.packages('Tmisc')
```

```
## Installing package into '/home/alfredo_aburto/R/x86_64-pc-linux-gnu-library/4.3'
## (as 'lib' is unspecified)
```

Cargamos los datos

```
library(Tmisc)
data(quartet)
View(quartet)
```

Este conjunto de datos contiene 4 conjuntos de ejex x e y. Los datos se pueden resumir a través de diferentes parametros estadisticos. Obtendremos un resumen de cada conjunto de datos con la media, desvest y la correalacion.

Cargamos paquetes necesarios para usar summarize

```
install.packages("dplyr")
```

```
## Installing package into '/home/alfredo_aburto/R/x86_64-pc-linux-gnu-library/4.3'
```

```
## (as 'lib' is unspecified)
```

```
library(dplyr)
```

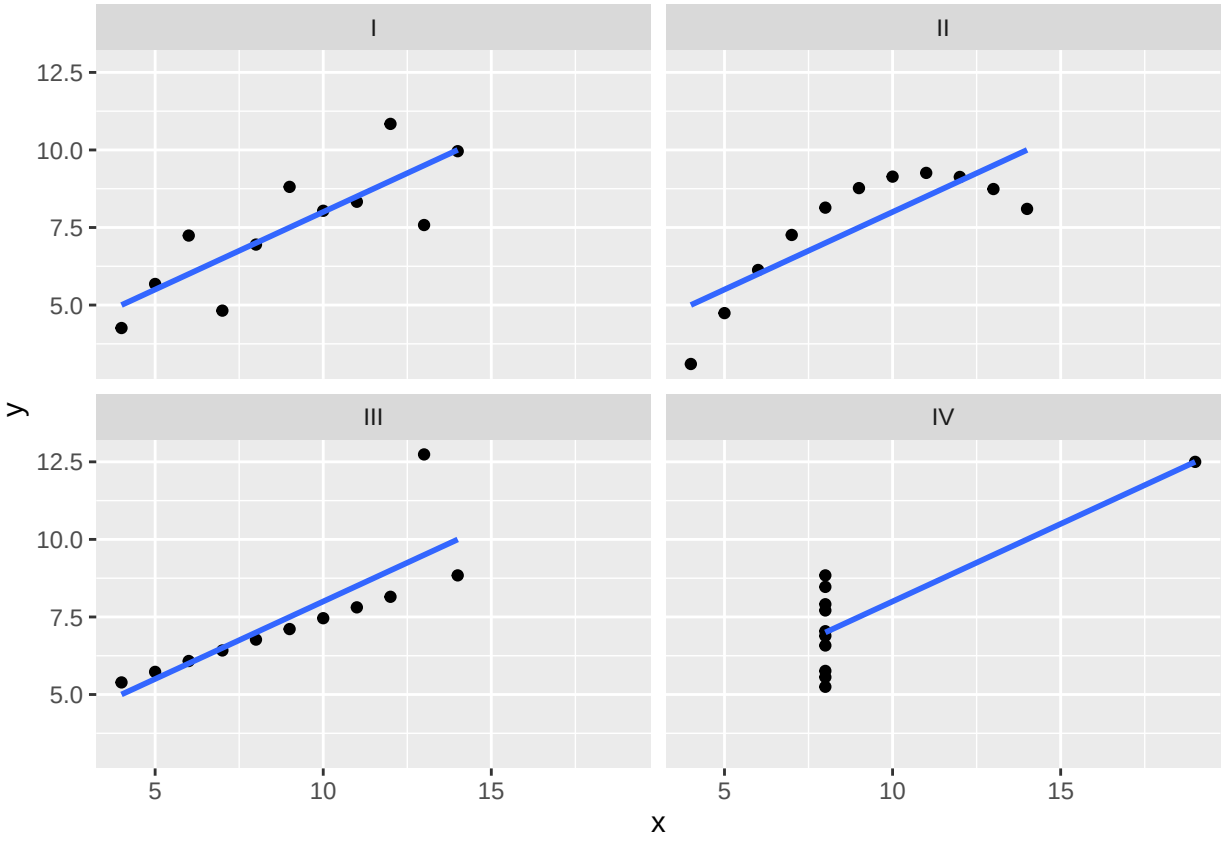
```
quartet %>%
  group_by(set) %>%
  summarize(mean(x), sd(x), mean(y), sd(y), cor(x, y))
```

```
## # A tibble: 4 x 6
##   set   `mean(x)` `sd(x)` `mean(y)` `sd(y)` `cor(x, y)`
##   <fct>     <dbl>   <dbl>     <dbl>   <dbl>       <dbl>
## 1 I             9    3.32      7.50    2.03       0.816
## 2 II            9    3.32      7.50    2.03       0.816
## 3 III           9    3.32      7.5     2.03       0.816
## 4 IV            9    3.32      7.50    2.03       0.817
```

Al observar la salida del comando anterior podemos ver que tanto la media, como la desviacion estandar y la correlacion entre las variables x e y en cada cuadrante es identica. Pero mirar unicamente el resumen estadistico puede resultar engañoso. Para notar esto, creaeremos algunos gráficos:

```
ggplot(quartet,aes(x,y)) + geom_point() + geom_smooth(method=lm, se=FALSE) +
  facet_wrap(~set)
```

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




Los gráficos generados muestran las grandes diferencias que hay para cada cuadrante

### Función de sesgo

En R podemos cuantificar el sesgo comparando el resultado real de nuestros datos con el resultado previsto. Existe explicacion estadistica detras de lo mencionado anteriormente, sin embargo, con la funcion bias no es necesario hacer el cálculo en forma manual.

**bias()** La función bias calcula el promedio en que el resultado real supera al resultado previsto. Está incuido en el paqueto de diseño Sim.

Si el modelo no tiene sesgo, el resultado debería ser bastante cercano a cero. Un resultado alto significa que tus datos podrían estar sesgados.

- Instalando y cargando el paquete

```
install.packages("SimDesign")
```

```
## Installing package into '/home/alfredo_aburto/R/x86_64-pc-linux-gnu-library/4.3'
## (as 'lib' is unspecified)
```

```
library(SimDesign)
```

Usaremos la función bias para comparar las temperaturas pronosticadas con las temperaturas reales.

- Tomando una muestra del conjunto de datos de temperatura

```
actual_temp <- c(68.3, 70, 72.4, 71, 67, 70)
predicted_temp <- c(67.9, 69, 71.5, 70, 67, 69)
```

- Usando bias

```
bias_01 = bias(actual_temp, predicted_temp)
bias_01
```

```
## [1] 0.7166667
```

- Otro escenario

Trabajaremos para una tienda de juegos, la tienda viene llevando un registro de cuantas copias de juegos nuevos venden en la fecha de lanzamiento. Quieren comparar estas cifras con sus ventas reales para saber si los pedidos de stock oinciden con sus necesidades reales

```
actual_sales <- c(150, 203, 137, 247, 116, 287)
predicted_sales <- c(200, 300, 150, 250, 150, 300)

bias_02 <- bias(actual_sales, predicted_sales)
bias_02
```

```
## [1] -35
```

Si el resultado del bias es **positivo** significa que estamos por abajo de lo pronosticado. Por el contrario, un resultado **negativo** indica que estamos por arriba del pronostico. En nuestro ultimo escenario, significaria que estamos pidiendo demasiado stock para las fechas de lanzamiento.

El sesgo se refiere a la tendencia sistemática de los errores, es decir, si las predicciones tienden a ser sistemáticamente más altas o más bajas que los valores reales.

## Visualizacion de datos

En R existen multiples paquetes que permiten crear visualizacion de datos. Plotly te permite realizar una gran variedad de funciones de visualización, por otro lado, RGL se enfoca en soluciones específicas como visualizaciones 3D.

Algunos de los paquetes más populares incluyen: - ggplot2 - Plotly - Lattice - RGL - Dygraphs - Leaflet - Highcharter - Patchwork, - gganimate y ggridges.

**ggplot2**

Creado por el estadístico y desarrolador Hadley Wickham en 2005, inspirado por el libro "The Grammar of Graphics", un estudio académico sobre la visualizacion de datos, escrito por el cientifico de la computación Leland Wilkinson.

De igual forma que los humanos usamos la gramatica para establecer normas para crear oraciones, la gramática de los gráficos nos da normas para armar cualquier tipo de visualización.

ggplot2 tiene algunos bloques de constricción básicos que puedes usar para crear diagramas. Cuando tenemos claros estos bloques de consturccion básicos podemos usarlos para crear muchos tipos de diagramas diferentes. Además, puedes agregar o quitar capas de detalles a tu diatrama sin cambiar su estructura básica o los datos subyacentes.

Puedes crear todo tipo de diagramas diferentes, entre ellos, diagramas de dispersión, gráficos de barras, diagramas de línea, y muchos más.

Puedes cambiar los colores, el diseño y las dimensiones de tus diagramas y gregar elementos de texto como títulos, leyendas y etiquetas. Con solo un poco de código puedes crear representaciones visuales de alta calidad.

Además, ggplot2 te deja combinar manipulación y visualización de datos usando el operador de canalización. Ggplot2 también tiene muchísimas funciones que satisfacen todas tus necesidades de visualización de datos.

Nos enfocaremos en algunos conceptos centrales en ggplot2: - estética - figuras geométricas - facetas - etiquetas y anotaciones.

**Estética**

La estética incluye cosas como el tamaño, la forma y el color de tus puntos de datos. Piensa en una estética como una conexión o mapeo entre una característica visual en tu diagrama y una variable en tus datos.

**Figuras geométricas**

Una figura geométrica se refiere al objeto geométrico usado para representar tus datos. Por ejemplo, puedes usar puntos para crear un diagrama de dispersión, barras para crear un gráfico de barras o líneas para crear un diagrama de líneas. Puedes elegir una figura geométrica que se adapte al tipo de datos que tienes. Los puntos muestran la relación entre dos variables cuantitativas. Las barras muestran una variable cuantitativa que varía entre diferentes categorías.

**Facetas**

Las facetas te permiten mostrar grupos más pequeños, o subconjuntos, de datos. Con las facetas, puedes crear diagramas separados para todas las variables en tu conjunto de datos.

**Funciones label y annotate**

Por último, las funciones label y annotate te dejan personalizar tu diagrama. Puedes agregar texto como títulos, subtítulos y leyendas para comunicar el propósito de tu diagrama o destacar datos importantes.

**Proceso de creación de un diagrama**

1. Preparar datps

El paquete ggplot2 te permite usar código R para especificar el conjunto de datos, la figura geométrica y la estética de tu diagrama. Para hacerlo, primero elige un conjunto de datos con el que trabajar.

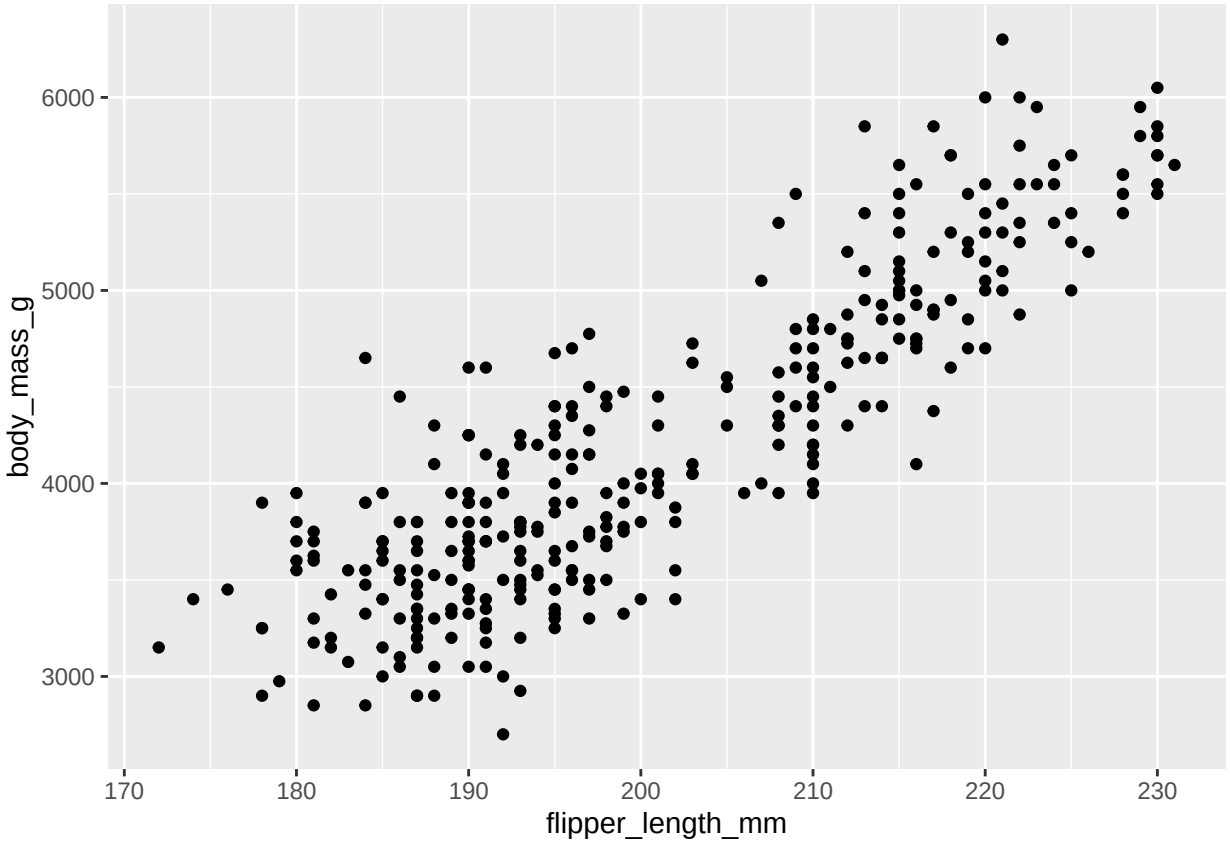
```
library(ggplot2)
library(palmerpenguins)
data("penguins")
View(penguins)
```

2. Creación del diagrama

Supongamos que quieres modelar la relación entre masa corporal y longitud de aletas en las tres especies de pingüino. Puedes elegir una figura geométrica específica que se adapte al tipo de datos que tienes. Los puntos muestran la relación entre dos variables cuantitativas. Un diagrama de dispersión de puntos sería una manera eficaz de mostrar la relación entre las dos variables. Puedes colocar longitud de aleta en el eje X y masa corporal en el eje Y.

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

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




- ggplot(data = penguins): En ggplot 2, comienzas un diagrama con la función ggplot(). La función ggplot() crea un sistema de coordenadas al que puedes agregar capas. El primer argumento de la función ggplot() es el conjunto de datos a usar en el diagrama. En este caso, es "penguins".
- +: Luego agregas un símbolo "+" para agregar una nueva capa a tu diagrama. Completas el diagrama agregando una o más capas a ggplot().
- geom_point(): Luego eliges una figura geométrica agregando una función geométrica. La función geom_point() usa puntos para crear diagramas de dispersión, la función geom_bar usa barras para crear gráficos de barras, etc. En este caso, elige la función geom_point para crear un diagrama de

dispersión de puntos. El paquete ggplot2 viene con muchas funciones geométricas diferentes.

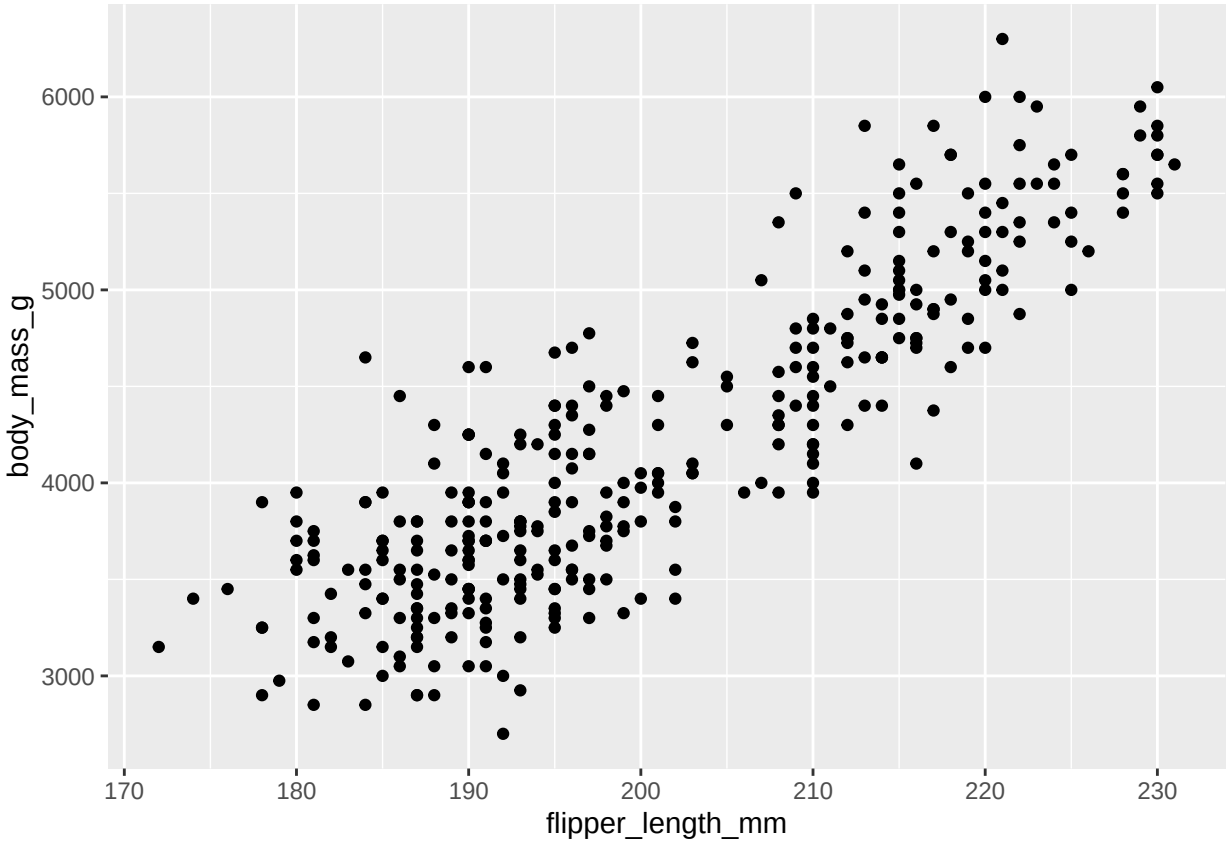
- (mapping = aes(x = flipper_length_mm, y = body_mass_g)): Cada función geométrica en ggplot2 toma un argumento de mapeo. Esto define cómo se aplican variables de tu conjunto de datos a propiedades visuales. El argumento de mapeo siempre se utiliza en conjunto con la función aes(). Los argumentos X e Y de la función aes() especifican qué variables aplicar al eje X y al eje Y del sistema de coordenadas. En este caso, quieres aplicar la variable "flipper_length_mm" al eje X y la variable "body_mass_g" al eje Y.

El diagrama muestra una relación positiva entre las dos variables. Es decir, cuanto más grande es el pingüino, más larga es la aleta.

- **Consejo profesional**: Puedes escribir la misma sección de código que aparece arriba usando una sintaxis diferente con el argumento de mapeo dentro de la llamada de ggplot():

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

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




### Modificar la apariencia de las visualizaciones

Con la estética podemos cambiar la apariencia de nuestros datos en la visualizacion que estamos crando. Existen tres atributos básicos a tener en cuenta al crear visualizaciones de datos.

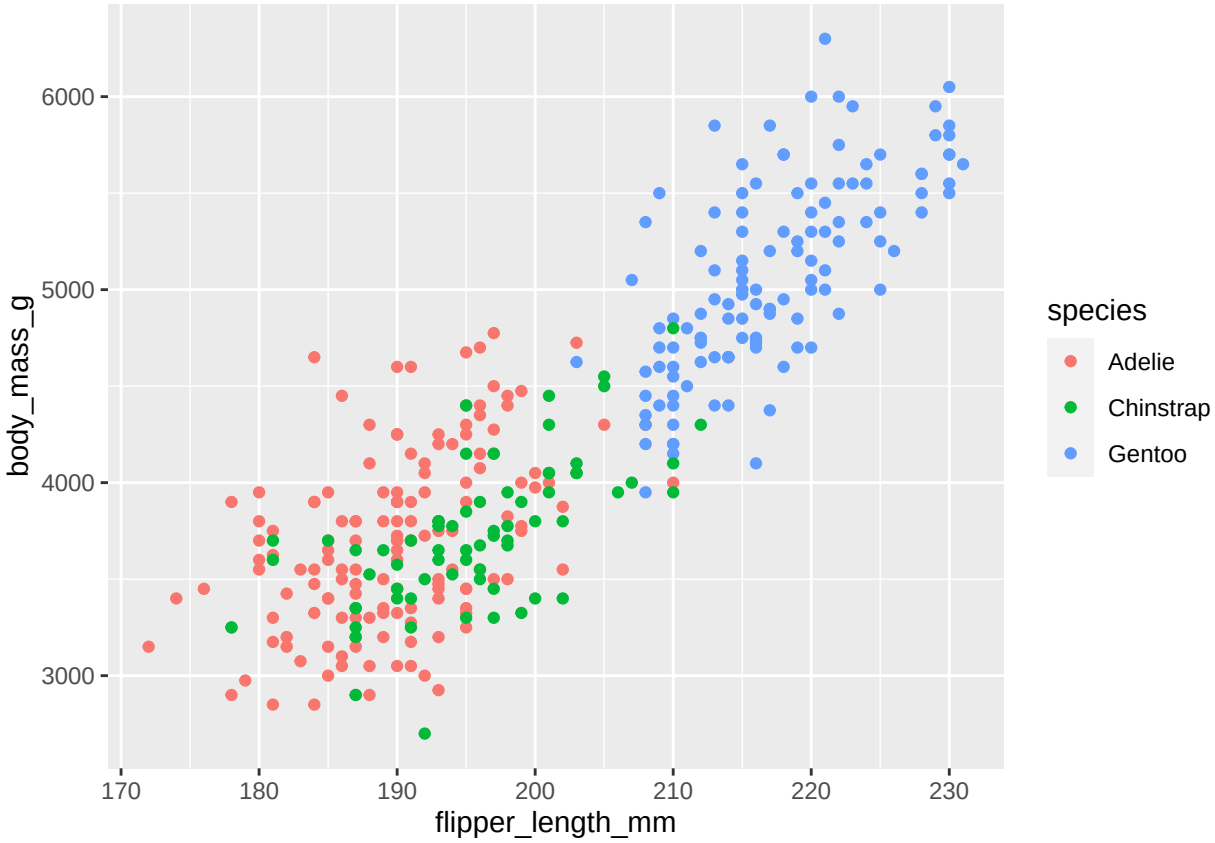
La estética se define como una propiedad visual de un objeto de tu diagrama y los tres atributos eséticos son los siguientes:

- color

- tamaño
- forma
- color: te permite modificar el color de todos los puntos de tu diagrama o el color de cada grupo de datos.

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

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

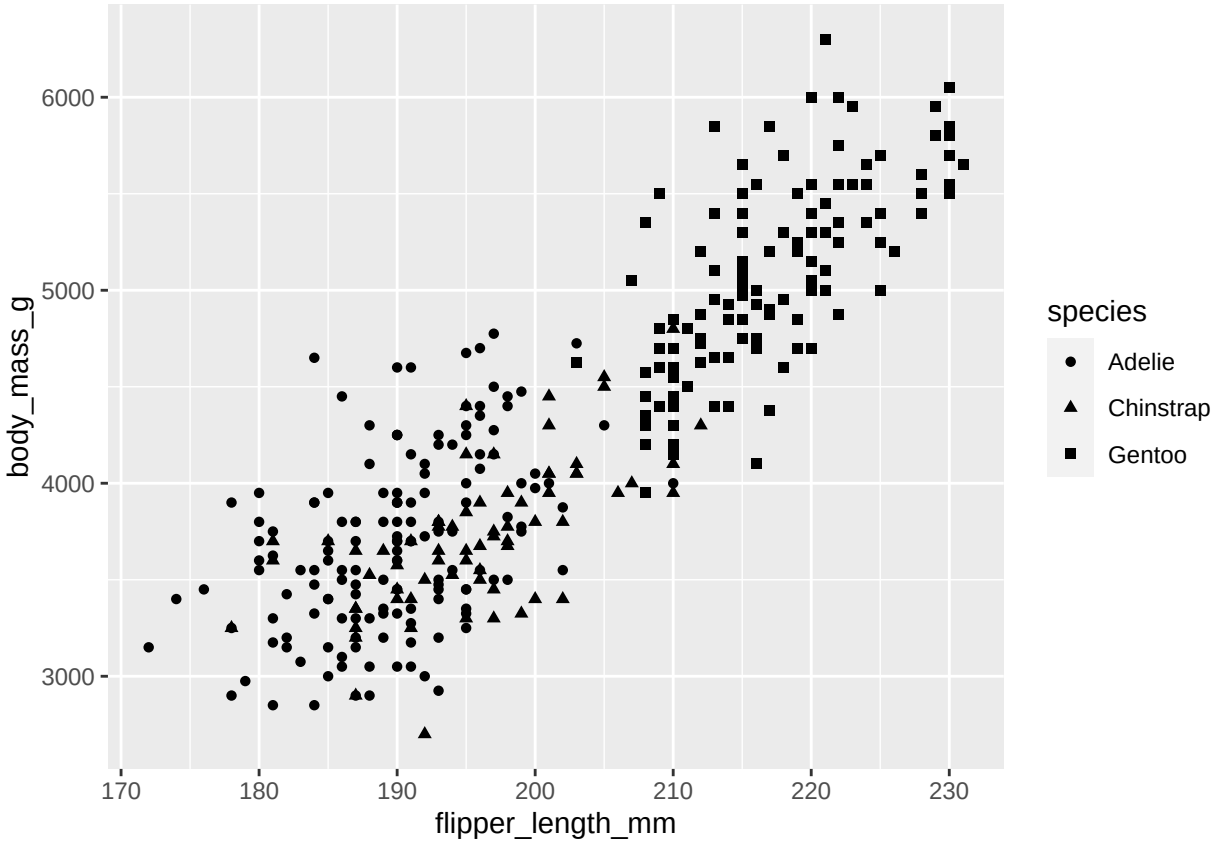



En el bloque anterior decimos que cambie el color por cada especie diferente.

- tamaño: te permite modificar el tamaño de los puntos de tu diagrama por grupo de datos
- forma: te permite modificar la forma de los puntos de tu diagrama por grupo de datos

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

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

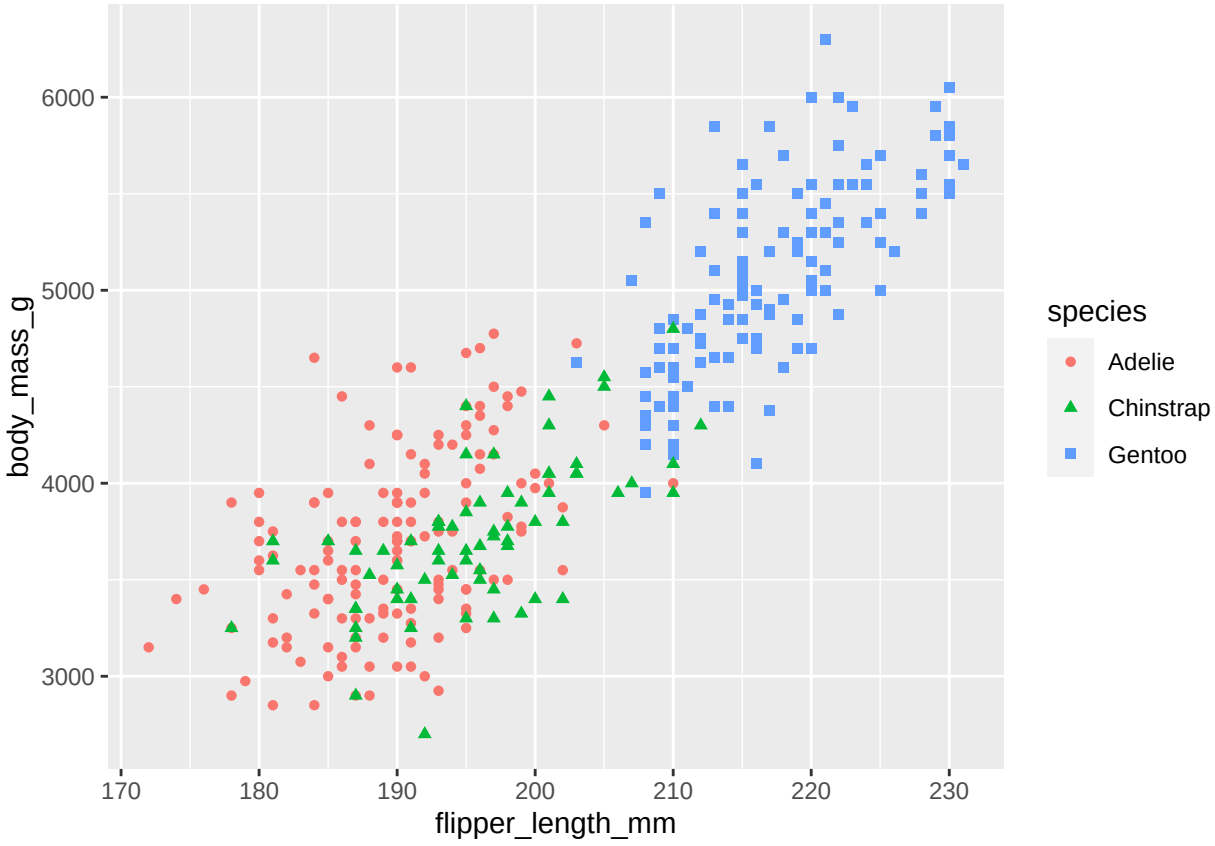



Aqui destacamos las diferentes especies usando diferentes formas geometricas aplicando la variable especies a la estetica shape

Podemos **aplicar más de una estetica a una misma variable**

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

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

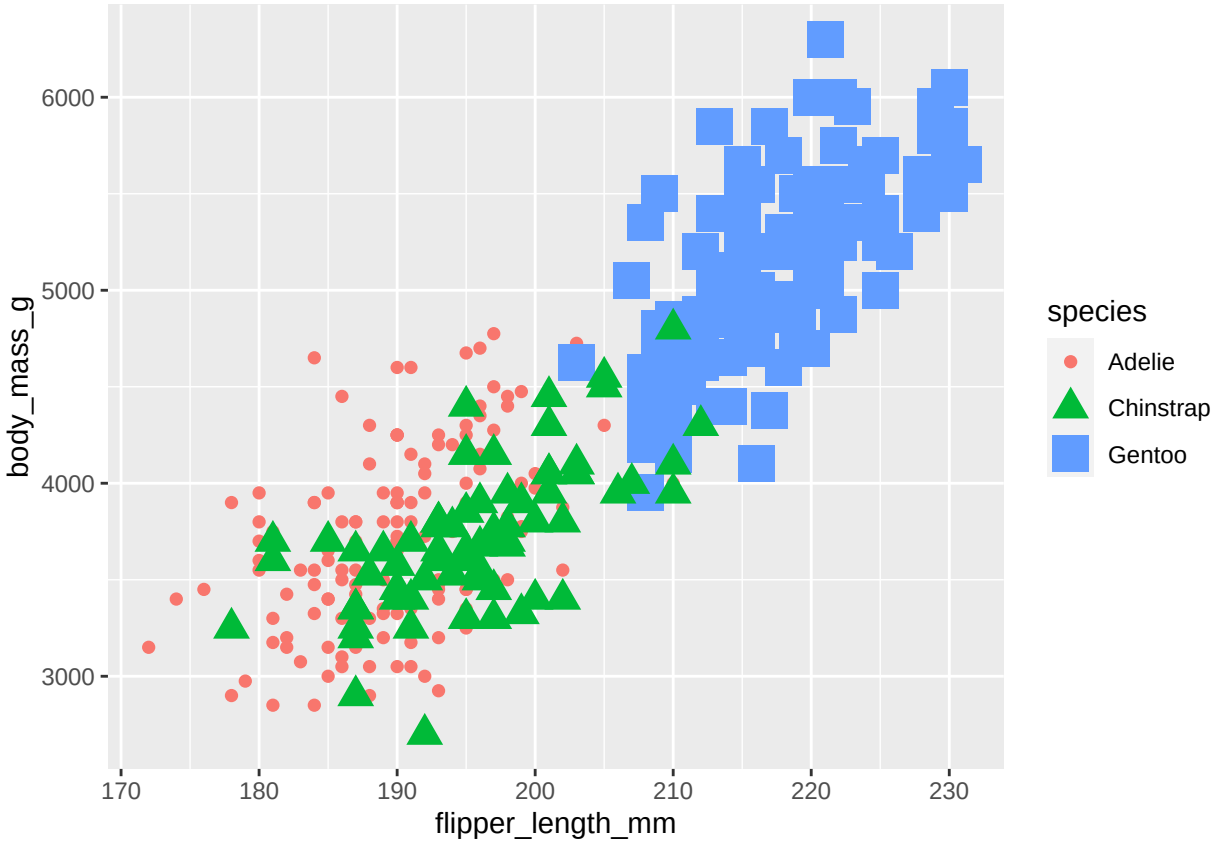



**Aplicando las tres esteticas**

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

```
## Warning: Using size for a discrete variable is not advised.
```

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

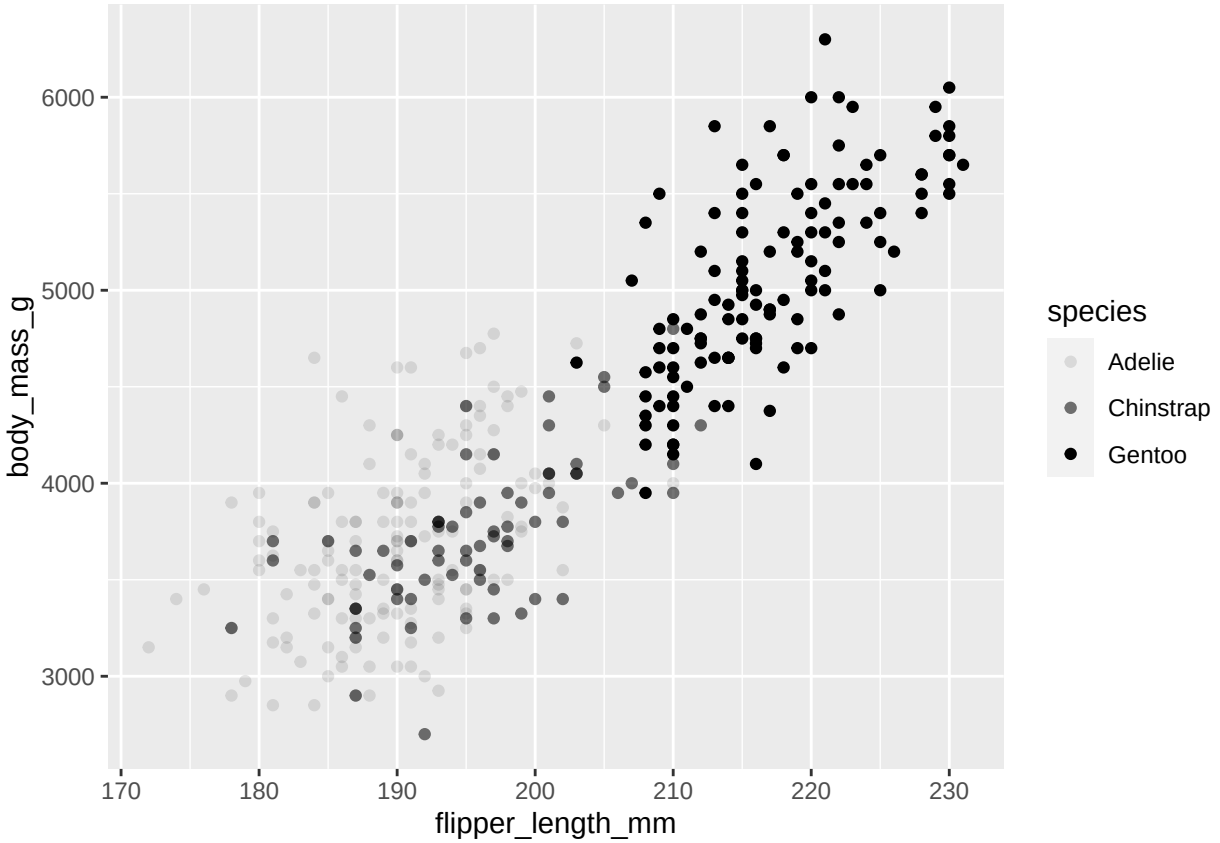



- estetica alpha Esta estetica mapaea nuestros datos con la estetica alfa que controla la transparencia de los puntos.

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

```
## Warning: Using alpha for a discrete variable is not advised.
```

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



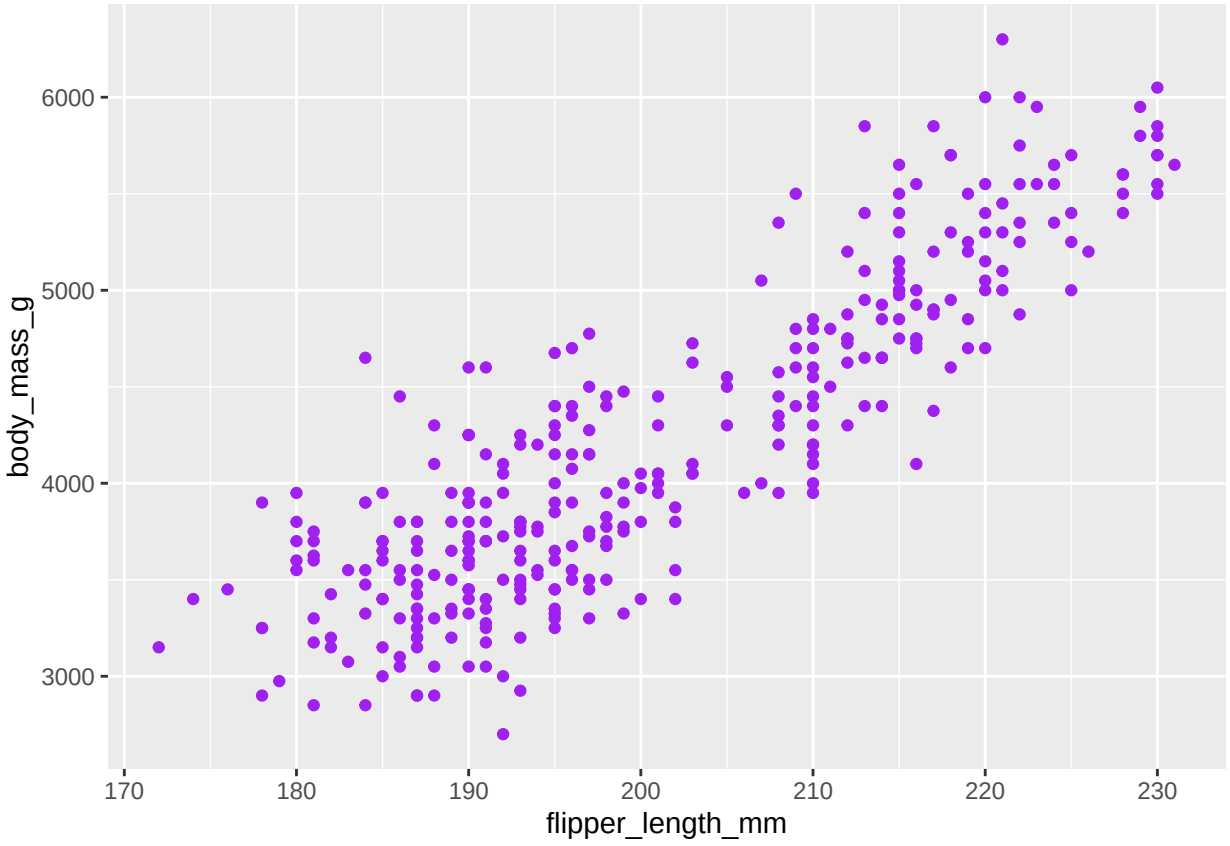

Todo lo relacionado con la estetica de nuestro conjunto de datos ira dentro de la funcion 'aes()', definir los ejes x e y son parte de la estetica. Como se mencionó, modificar el color, tamaño y forma es parte de la estetica, por lo que si deseamos modificarlos iran dentro de la función mecionada.

Con la funcion **aes**, tambien mapeamos nuestros datos lo que significa emparejar una variable especifica en tu conjunto de datos con una estética especifica.

Podemos fijar una estetica por separado de una variable especifica. Digamos que queremos cambiar el color de tosos los puntos a violeta. Pero no queremos aplicar color a una variable especifica como especia. Solo queremos que cada punto de nuestro diagrama de dispersion sea violeta. Para lograrlo debemos colocar nuestro nuevo trozo de codigo fuera de la funcion aes y usar comillas para nuestro valor de color.

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

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




- Usando diferentes funciones geometricas

Anteriormente nos hemos centrado en los gráficos de dispersión de puntos, pero existen diferentes gráficos que se pueden usar en ggplot al modificar las figuras geometricas que representaran a nuestros datos.

En ggplot2 una figura geométrica es un objeto geométrico usado para representar a nuestros datos. Las figuras geometricas incluyen puntos, barras, lineas, etc.

**geom functions**

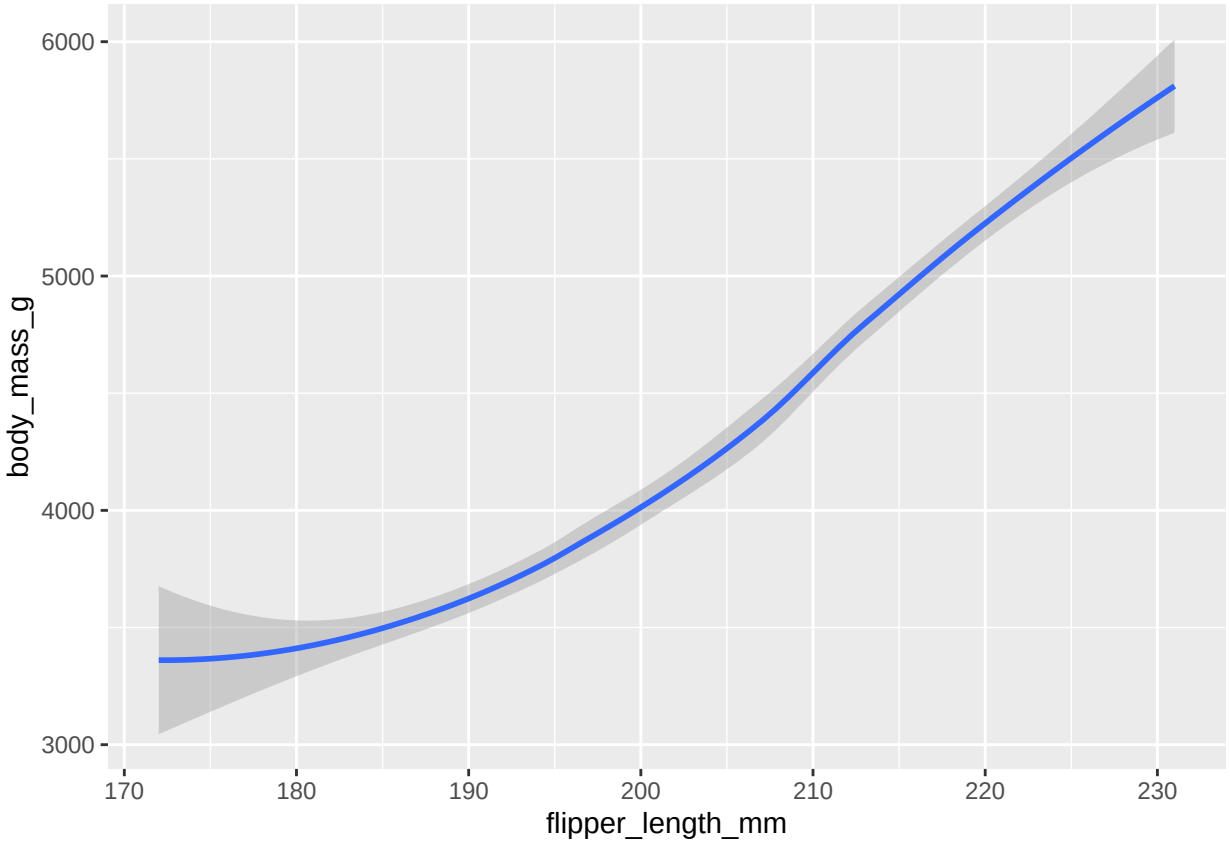
- geom_point(): Para graficos de dispersion de puntos
- geom_bar(): Para gráficos de barras
- geom_line(): Para graficos de linea
- geom_smoot(): Grafica con linea suavizada
- geom_jttler(): Crea diagramas de dispersión y luego agrega una pequeña cantidad de ruido aleatorio, ayuda a lidiar con el trazado excesivo, que sucede cuando los puntos de datos de un diagrama se super ponen unos con otros.

Para modificar la figura geometrica a utilizar es necesario modificar la funcion geom en el codigo. Si queremos modificar el grafico de dispersion que mostraba la relacion entre las variables flliper_length_mm y body_mass_g por una grafica con una linea suave podemos hacerlo como sigue:

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

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

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




La funcion geom_smooth es util para mostrar tendencias generales en nuestros datos. La linea muestra con laridad la relación positiva entre nuestras variables.

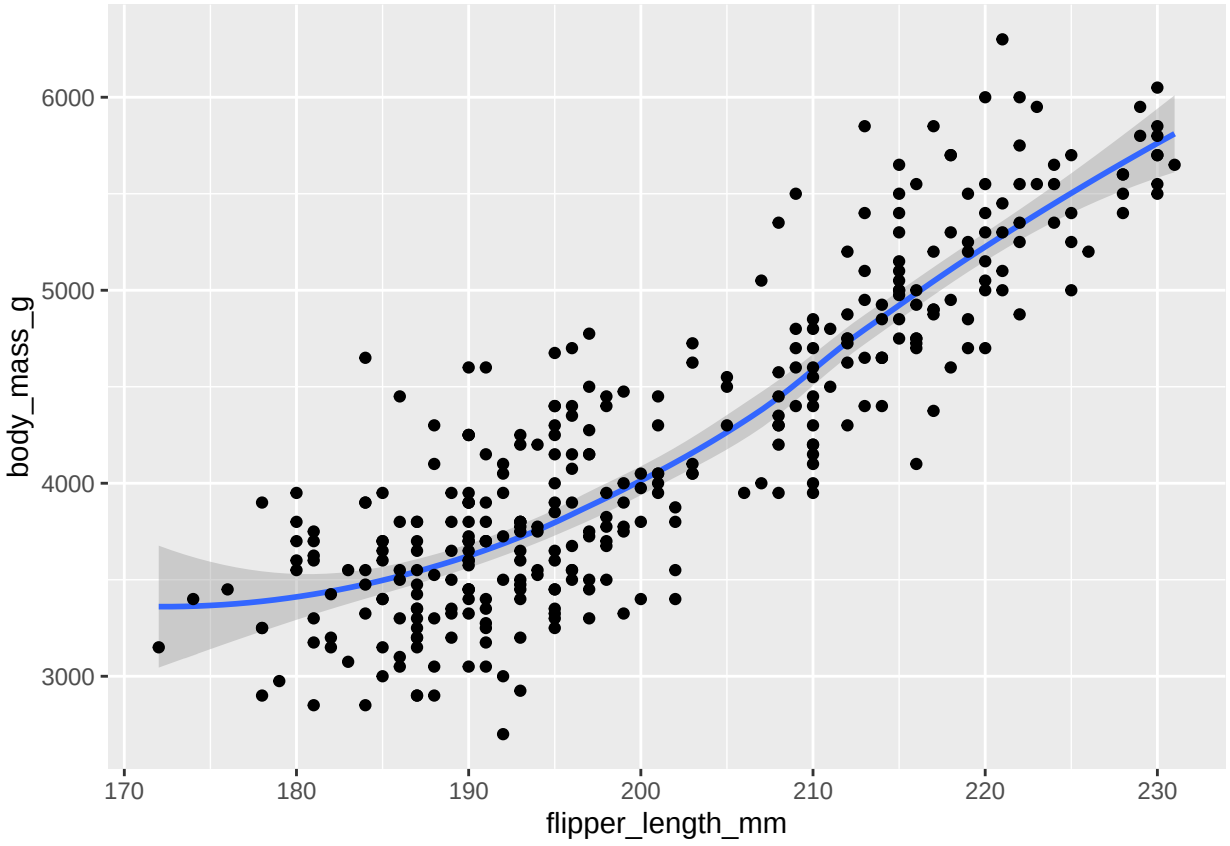
Podemos combinar diferentes geometrias, supongamos que deseamos mostrar tanto la tendencia general y los puntos individuales de nuestros datos.

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

```
## `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()`).
```

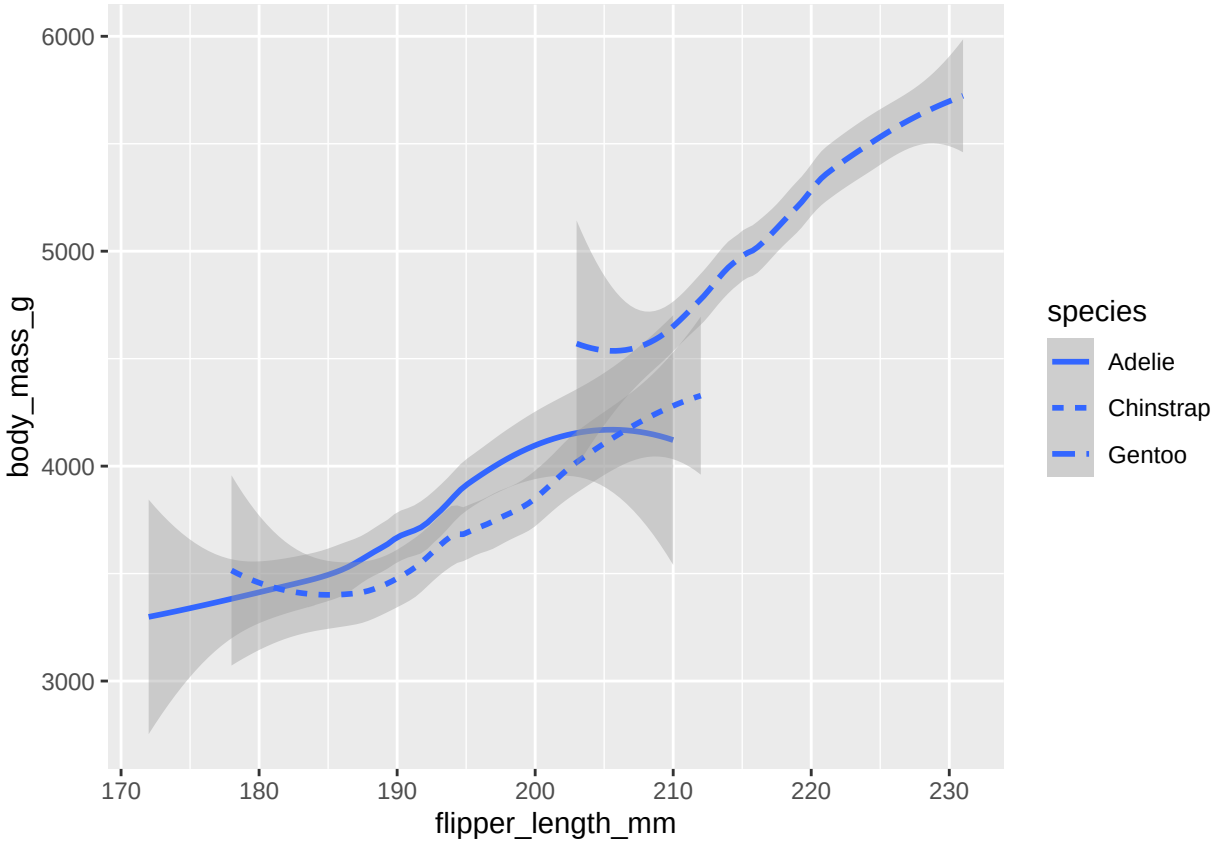



Digamos que ahora queremos modelar una linea separada para cada especie

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

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

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

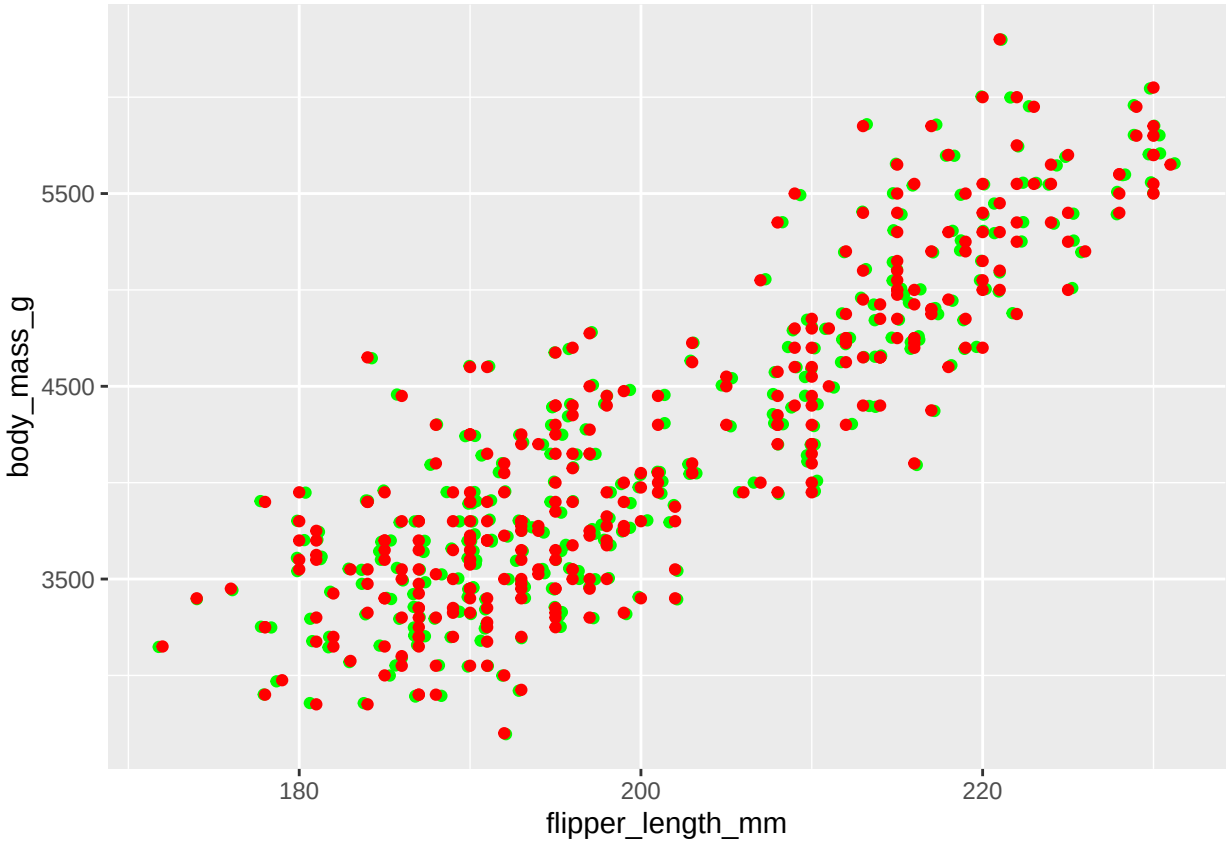



El bloque anterior agrega la estetica linetype para que a cada especie de pinguino le asigne una tipo de linea y una linea suavizada diferente.

Si quisieramos que las lineas fueran iguales pero una diferente para cada especie podemos utilizar la estetica line

```
ggplot(data=penguins) +
  geom_jitter(mapping = aes(x=flipper_length_mm, y=body_mass_g), color='green') +
  geom_point(mapping = aes(x=flipper_length_mm, y=body_mass_g), color='red')
```

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



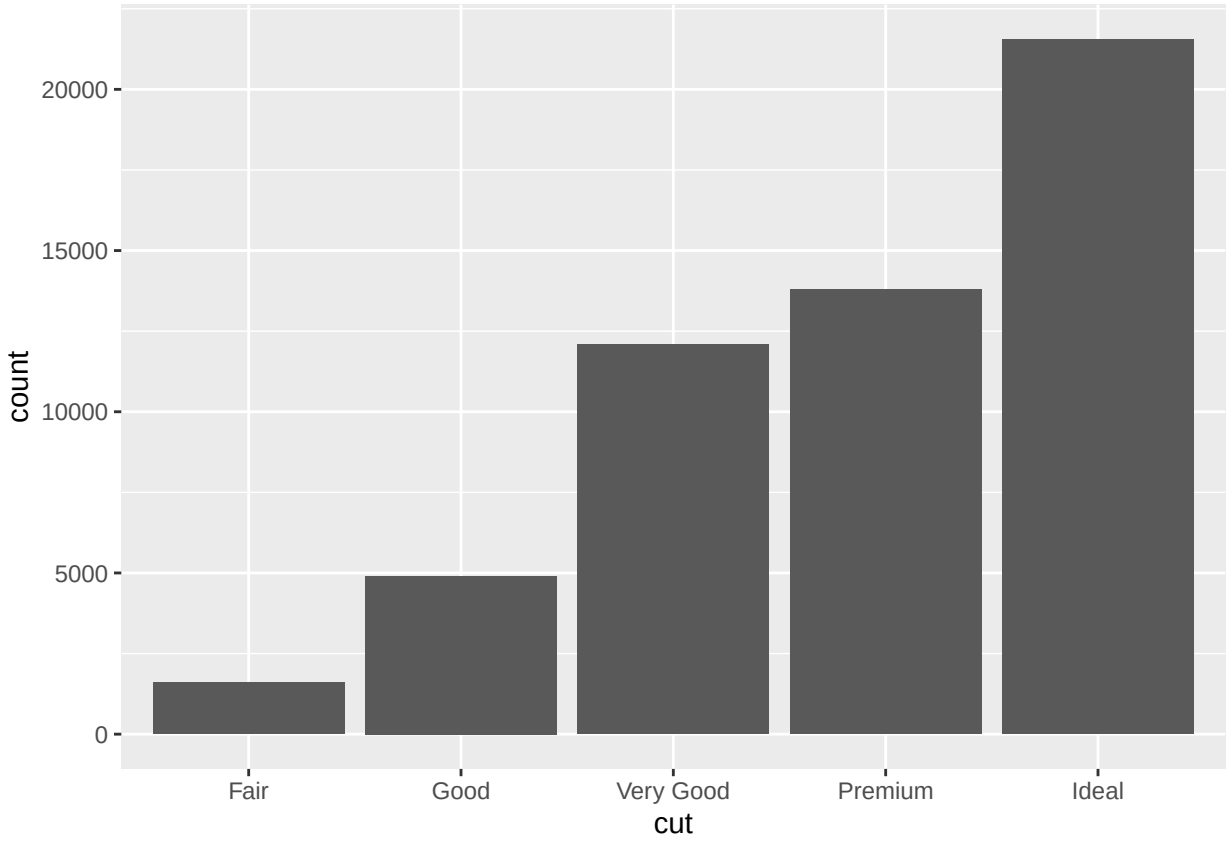

Con el codigo anterior comparamos las geometrias jitter y point.

**Graficas de barras** Para ejemplificar su uso, utilizaremos el conjunto de datos de diamantes, incluidos en ggplot2

```
data("diamonds")
View(diamonds)
```

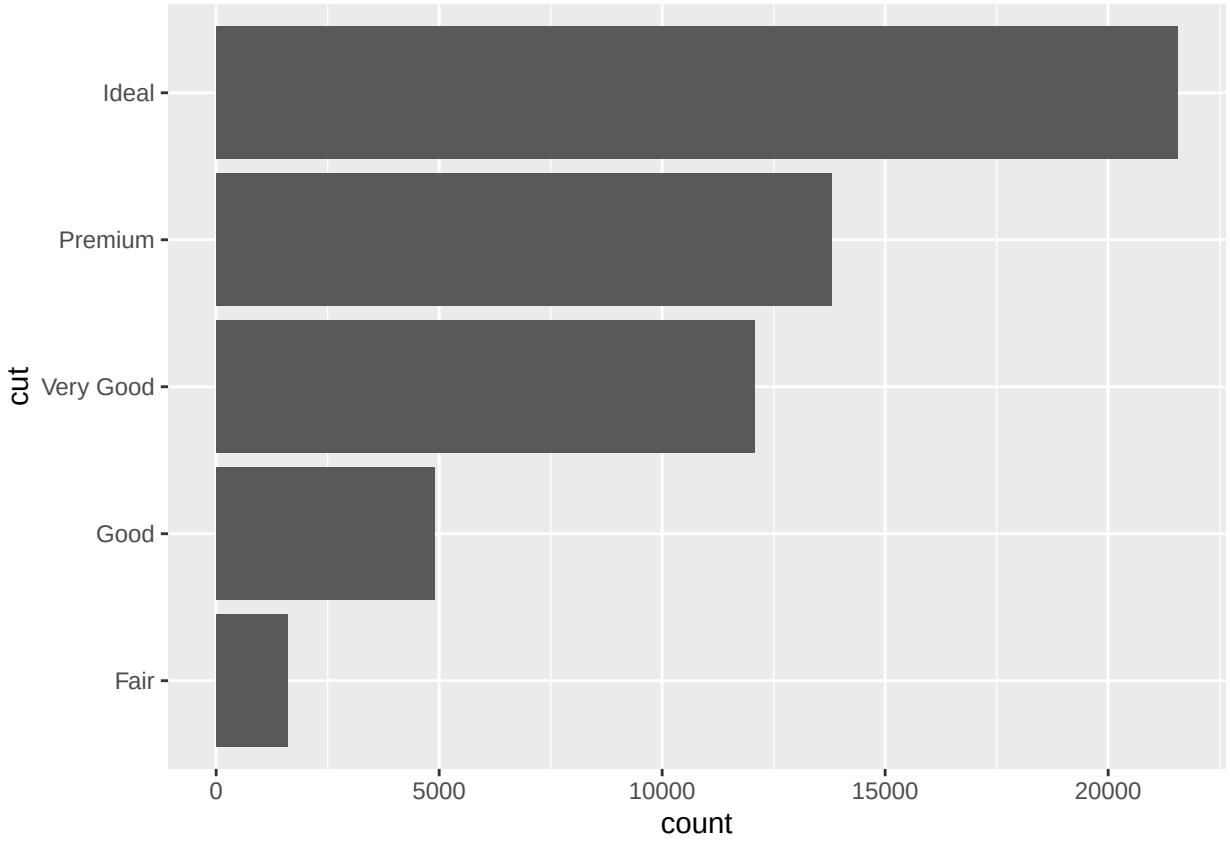
- Creacion de un grafico de barras (vertical)

```
ggplot(data=diamonds) +
  geom_bar(mapping = aes(x=cut))
```




- Creacion de un grafico de barras (horizontal)

```
ggplot(data=diamonds) +
  geom_bar(mapping = aes(y=cut))
```

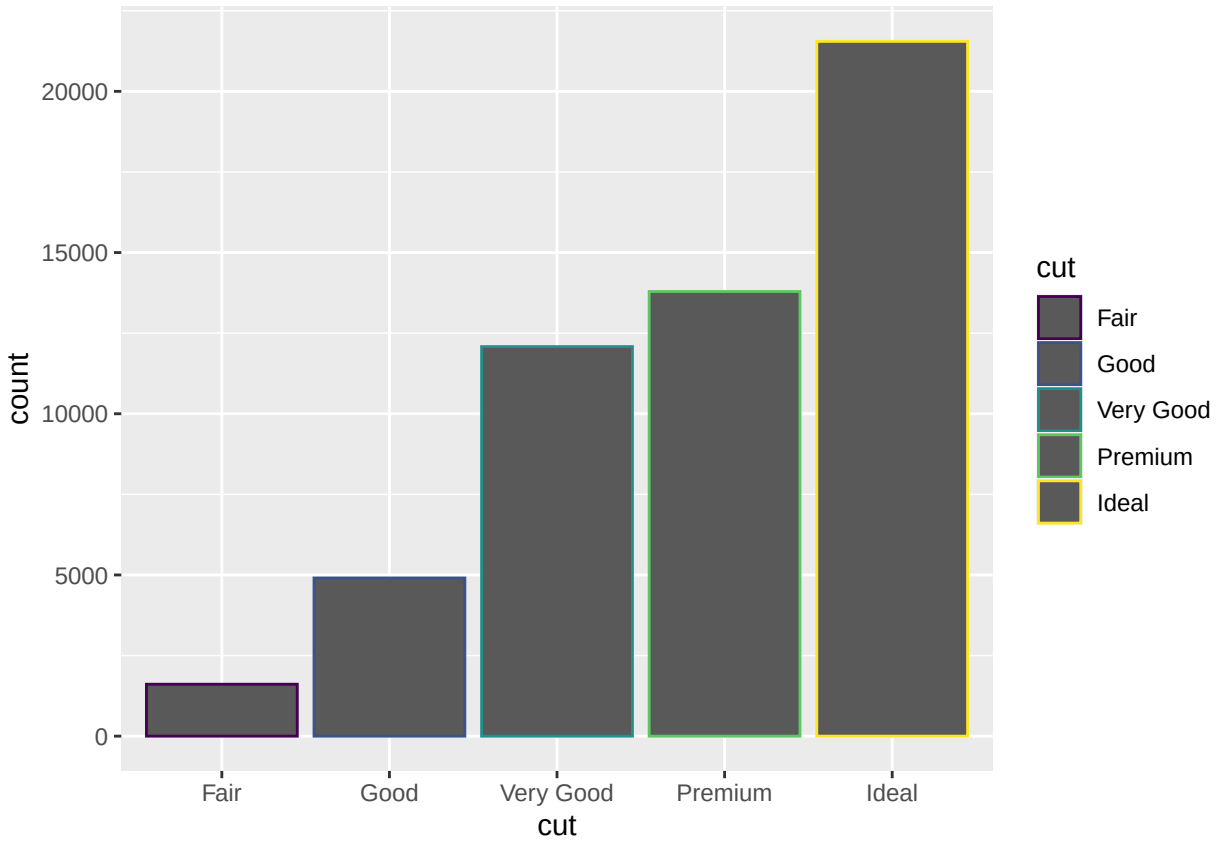



Cuando solo especificamos solo un eje para un grafico de barras, R de manera automatica completa el otro eje con el conteo de observaciones para las categorias existentes en el eje que si se ha especificado.

Esta geometria utiliza las diversas esteticas que hemos mencionado anteriormente, teniendo algunas particularidades como es la estetica fill

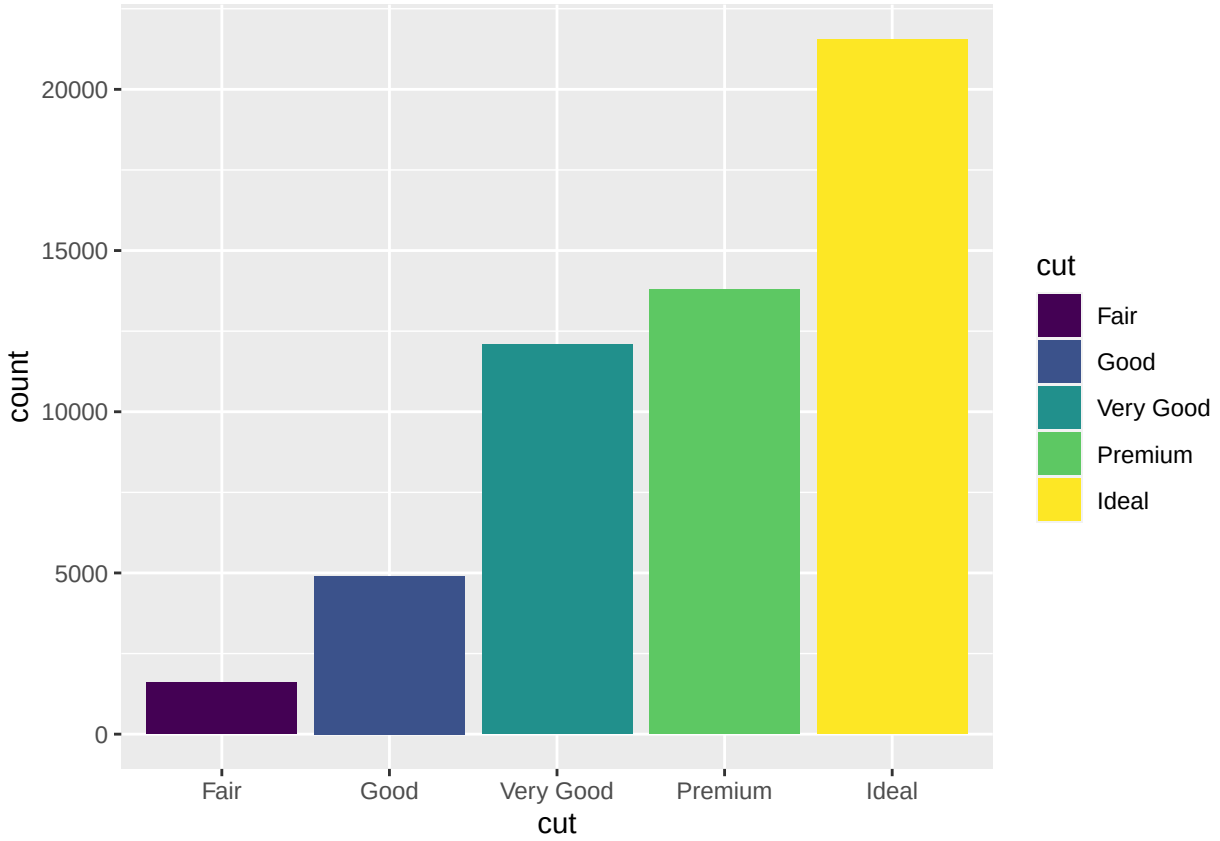
Apliquemos la estetica color a la variable cut:

```
ggplot(data=diamonds) +
  geom_bar(mapping = aes(x=cut, color=cut))
```




El codigo anterior aplica color pero solo a los contronos de nuestras barras, si queremos que el color se aplique tambien en el relleno (que es lo mas comun), podemos usar la estetica fill

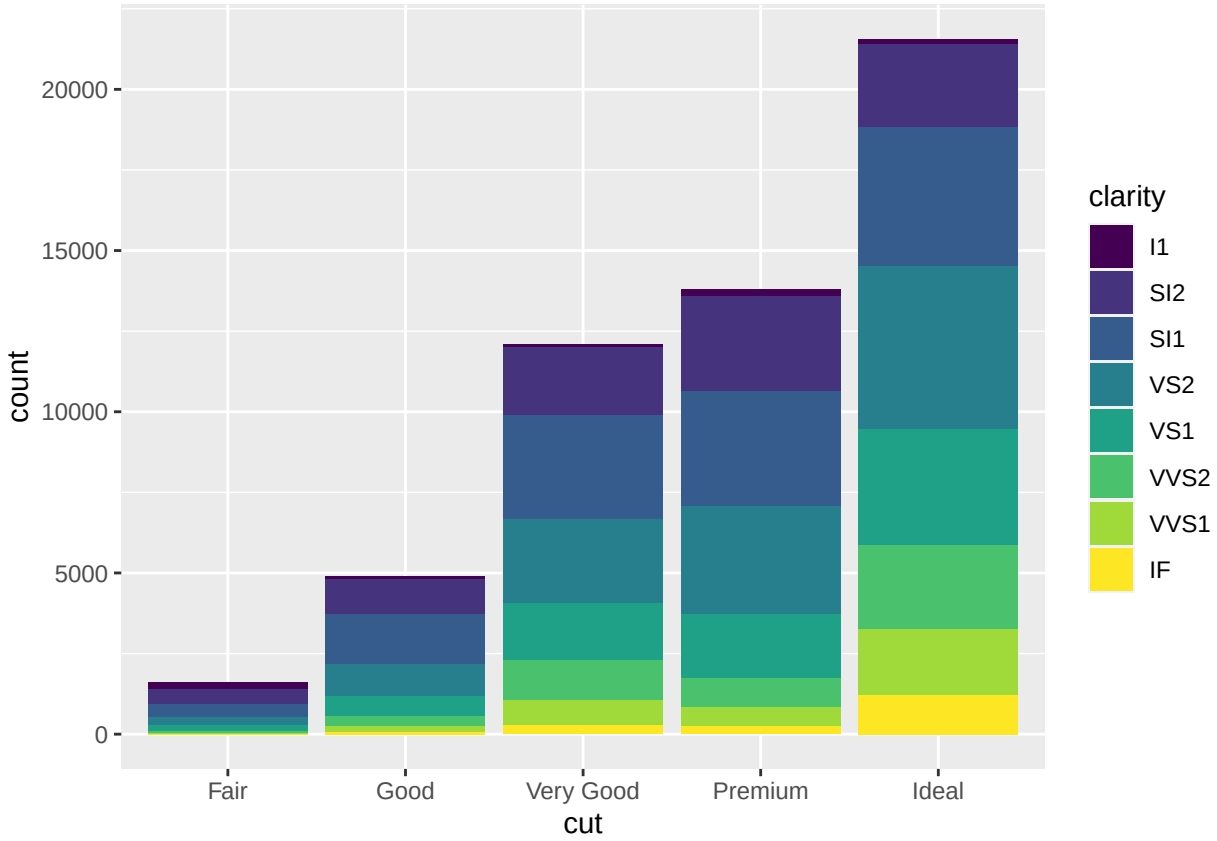
```
ggplot(data=diamonds) +
  geom_bar(mapping = aes(x=cut, fill=cut))
```




En este ultimo ejemplo producimos un grafico de barras con relleno de colores, pero es importante hacer la observacion de que aplicamos la estetica fill sobre la misma variable que estamos mapeando. Si por el contrario, indicamos como variable de aplicacion de la estetica fill una diferente a la especificada produciremos un grafico de barras apiladas.

Apliquemos fill a variable clarity en vez de cut

```
ggplot(data=diamonds) +
  geom_bar(mapping = aes(x=cut, fill=clarity))
```




Ahora nuestro grafico muestra 40 combinaciones diferentes de corte y claridad, 5 categorias por 8 niveles diferentes de claridad. Cada combinacion tiene su propio bloque

ggplot2 tiene muchas mas figuras geometricas que se pueden utilizar, para conocerlas se puede consultar la documentacion de ggplot

### Mas sobre el suavizado

A veces, puede ser difícil comprender tendencias en tus datos solo a través de diagramas de dispersión. El suavizado permite detectar una tendencia de datos aun cuando no puedes notar con facilidad una tendencia en los puntos de datos graficados. La funcionalidad de suavizado de ggplot2 es útil porque suma una l ínea de suavizado como otra capa en un diagrama; la línea de suavizado ayuda a ue un observador casual entienda el sentido de los datos.

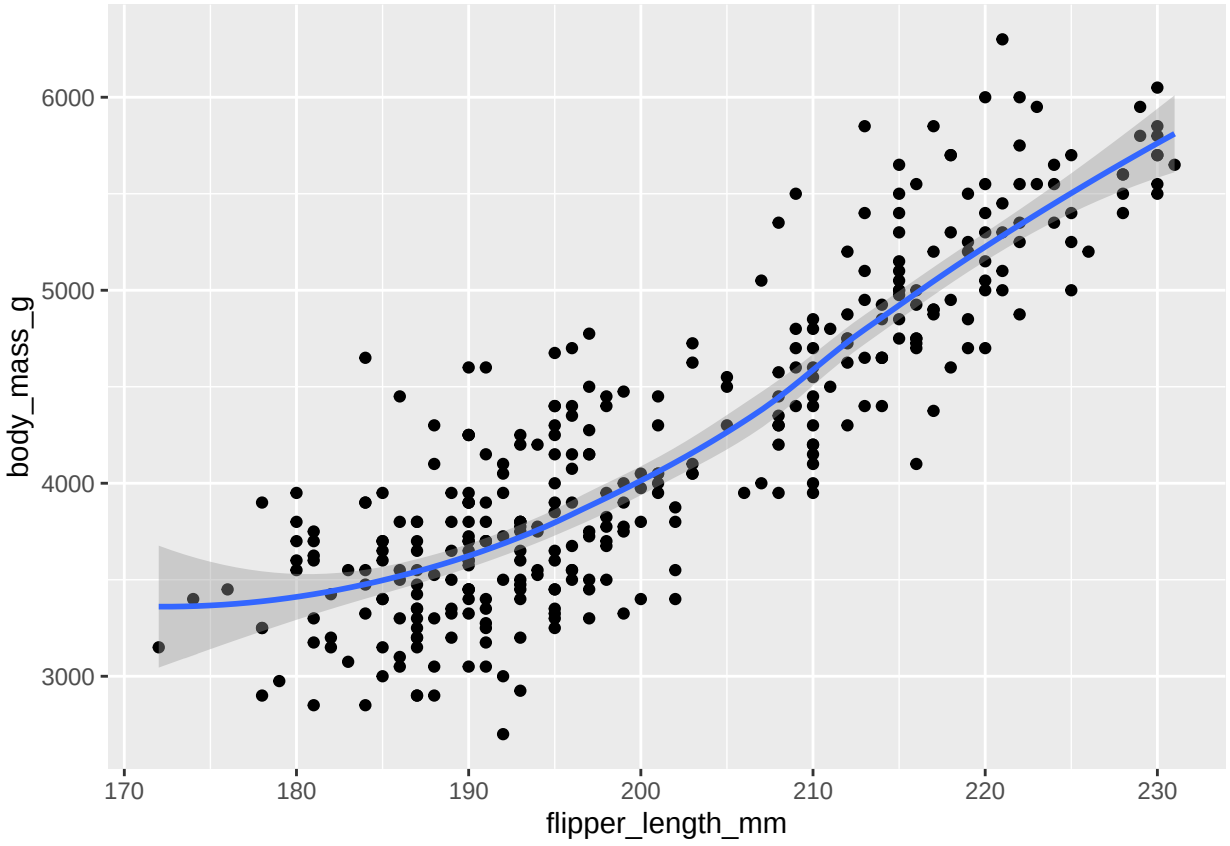
**Suavizado loess** Óptimo para suavizar diagramas con menos de 1000 puntos

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

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

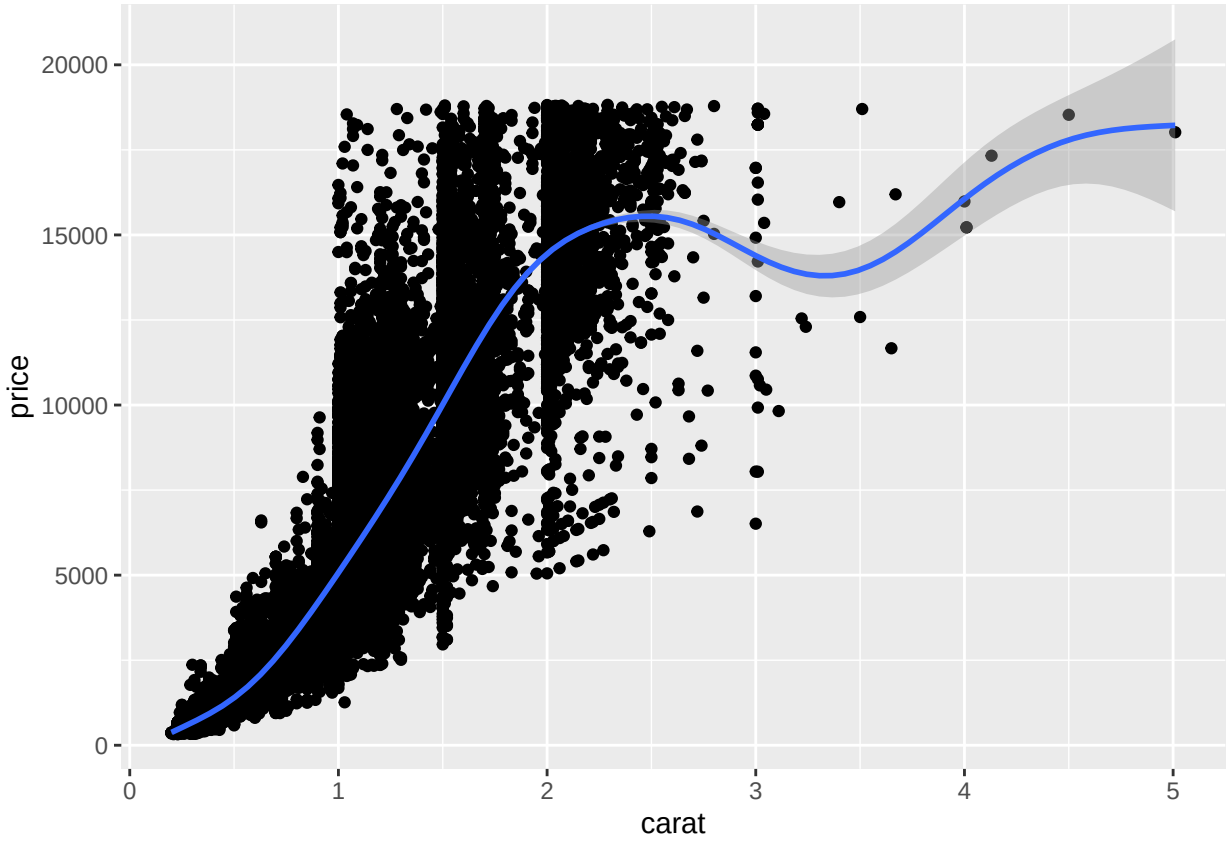
```
## Warning: Removed 2 rows containing non-finite values (`stat_smooth()`).
```

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




**Suavizado gam** El suavizado con modelos aditivos generalizados, es útil para suavizar diagramas con un gran número de puntos

```
ggplot(data=diamonds) +
  geom_point(mapping = aes(x=carat,y=price)) +
  geom_smooth(mapping=aes(x=carat,y=price),method = "gam",formula = y ~s(x))
```




La funcionalidad de suavizado en ggplot2 ayuda a que los diagramas de datos sean más legibles, para que puedas reconocer mejor las tendencias de datos y sacar conclusiones clave.

El método GAM (Generalized Additive Model) es una técnica de modelado estadístico que extiende los modelos lineales generalizados (GLM) permitiendo que las relaciones entre las variables predictoras y la variable de respuesta sean no lineales. Los modelos aditivos generalizados se construyen sumando funciones suavizadas de las variables predictoras.

En el código que proporcionaste, `geom_smooth` se utiliza con el método "gam" para realizar un suavizado de los datos. La función `s(x)` en la fórmula específica cómo se suaviza la relación entre las variables. Aquí hay una explicación más detallada: - **Método "gam"**: Indica que se está utilizando un modelo aditivo generalizado para realizar el suavizado. Este método es útil cuando se sospecha que la relación entre las variables no es estrictamente lineal y puede tener patrones más complejos.

- **`formula = y ~ s(x)`**: La fórmula especifica la relación entre la variable de respuesta (`y`) y la variable predictora (`x`). En este caso, se está utilizando una función suavizadora (`s()`) en la variable `x`. La función `s()` es responsable de suavizar la relación, permitiendo así que la relación sea no lineal.

En resumen, el código está creando un gráfico de dispersión (`geom_point`) de los datos diamonds con las variables `carat` en el eje x y `price` en el eje y. Luego, utiliza `geom_smooth` con el método "gam" y una fórmula que incluye una función suavizadora `s(x)` para mostrar una línea suavizada que representa la relación entre `carat` y `price`. Esto ayuda a visualizar tendencias más suaves y a identificar patrones en los datos que pueden no ser evidentes en el gráfico de dispersión original.

El código está creando un gráfico de dispersión (geom_point) de los datos diamonds con las variables carat en el eje x y price en el eje y. Luego, utiliza geom_smooth con el método "gam" y una fórmula que incluye una función suavizadora s(x) para mostrar una línea suavizada que representa la relación entre carat y price. Esto ayuda a visualizar tendencias más suaves y a identificar patrones en los datos que pueden no ser evidentes en el gráfico de dispersión original.

### Estetica y facetas

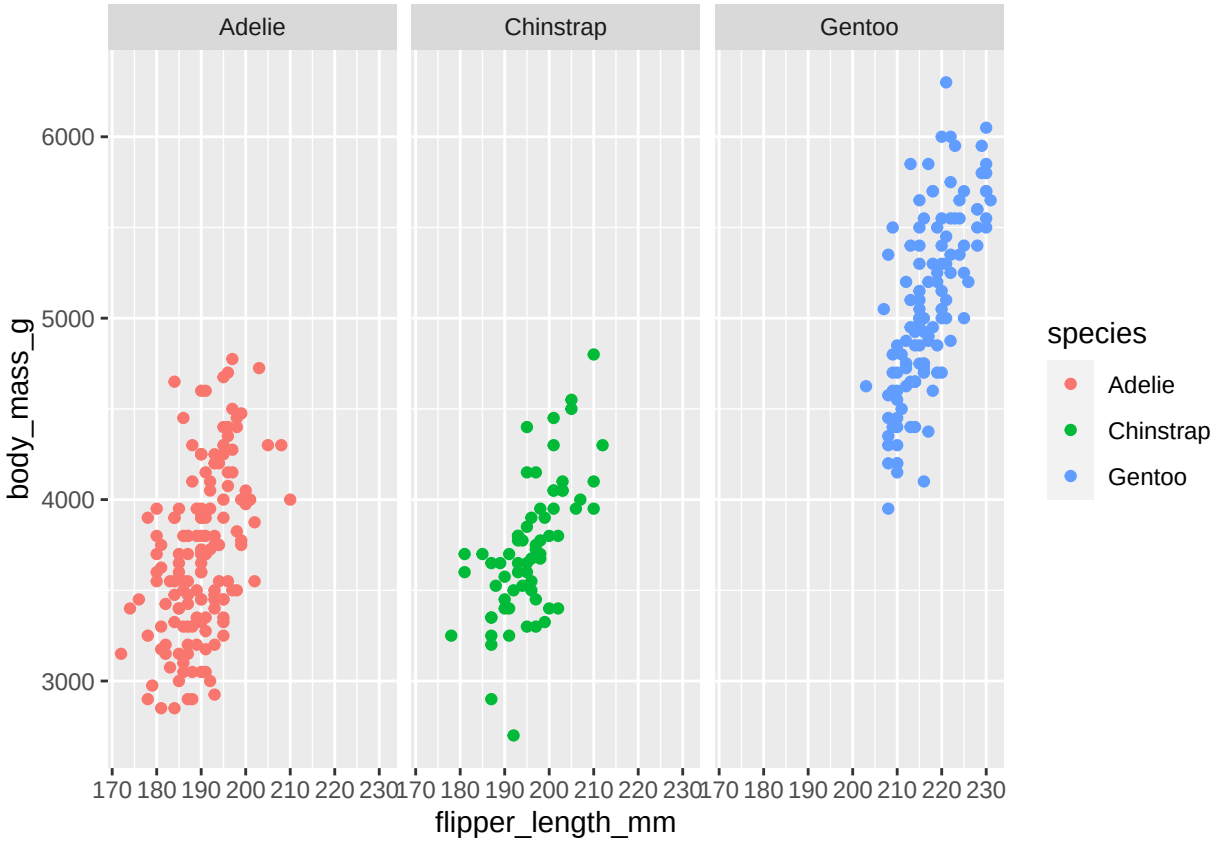
Las facetas permite crear grupos mas pequeños o subconjutnos de datos. Las facetas muestran diferentes caras de los datos. El uso de facetas ayuda a descubrir nuevos patrones y enfocarte en relaciones entre diferentes variables.

ggplot2 tiene dos funciones para facetas: - facet_wrap() - facet_grid()

**facet_wrap()** Nos sirve para crear subconjuntos a traves de una sola variable. Esto se ejemplifica con el conjunto de datos de pinguinos, en donde podriamos querer crear un diagrama por separado para cada especie.

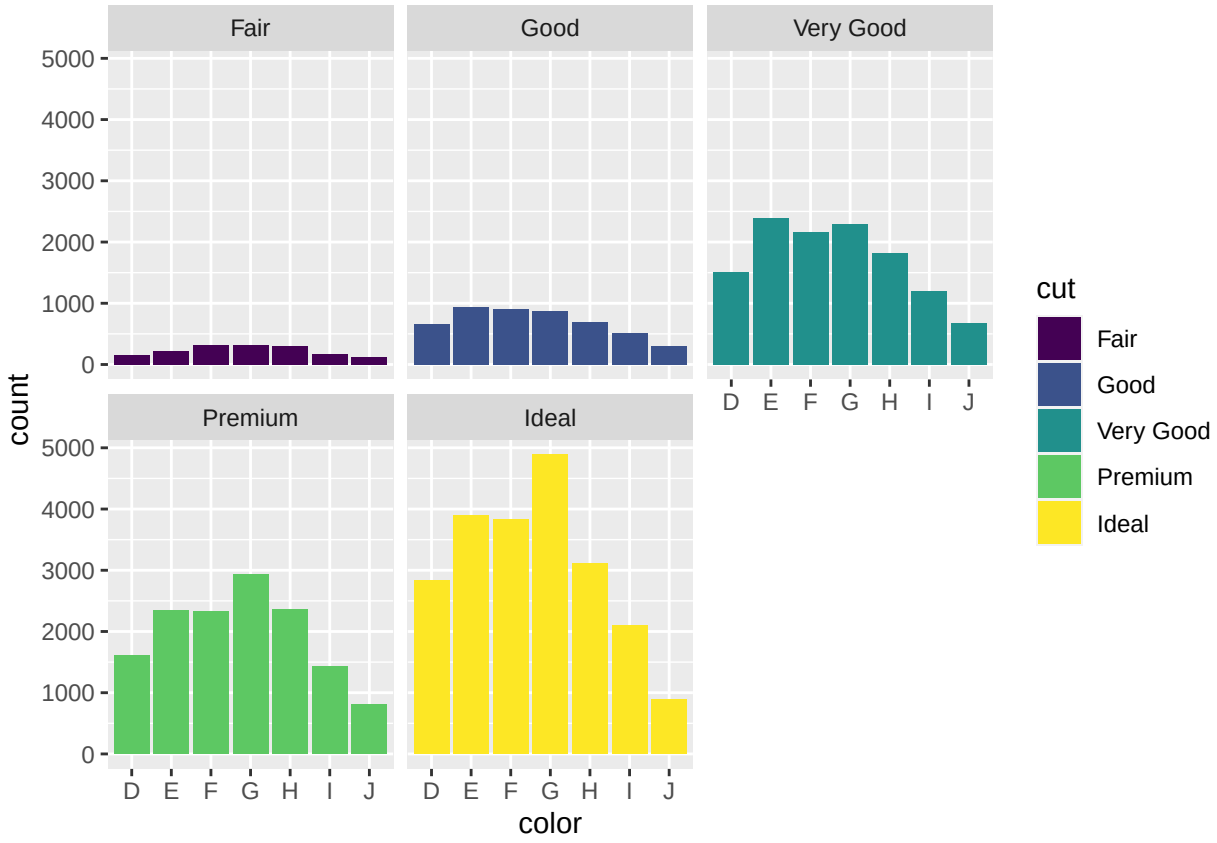
```
ggplot(data=penguins) +
  geom_point(mapping = aes(x=flipper_length_mm,
                           y=body_mass_g, color=species)) +
  facet_wrap(~species)
```

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




Ejemplo con el conjunto de datos diamons

```
ggplot(data=diamonds) +
  geom_bar(mapping = aes(x=color, fill=cut)) +
  facet_wrap(~cut)
```

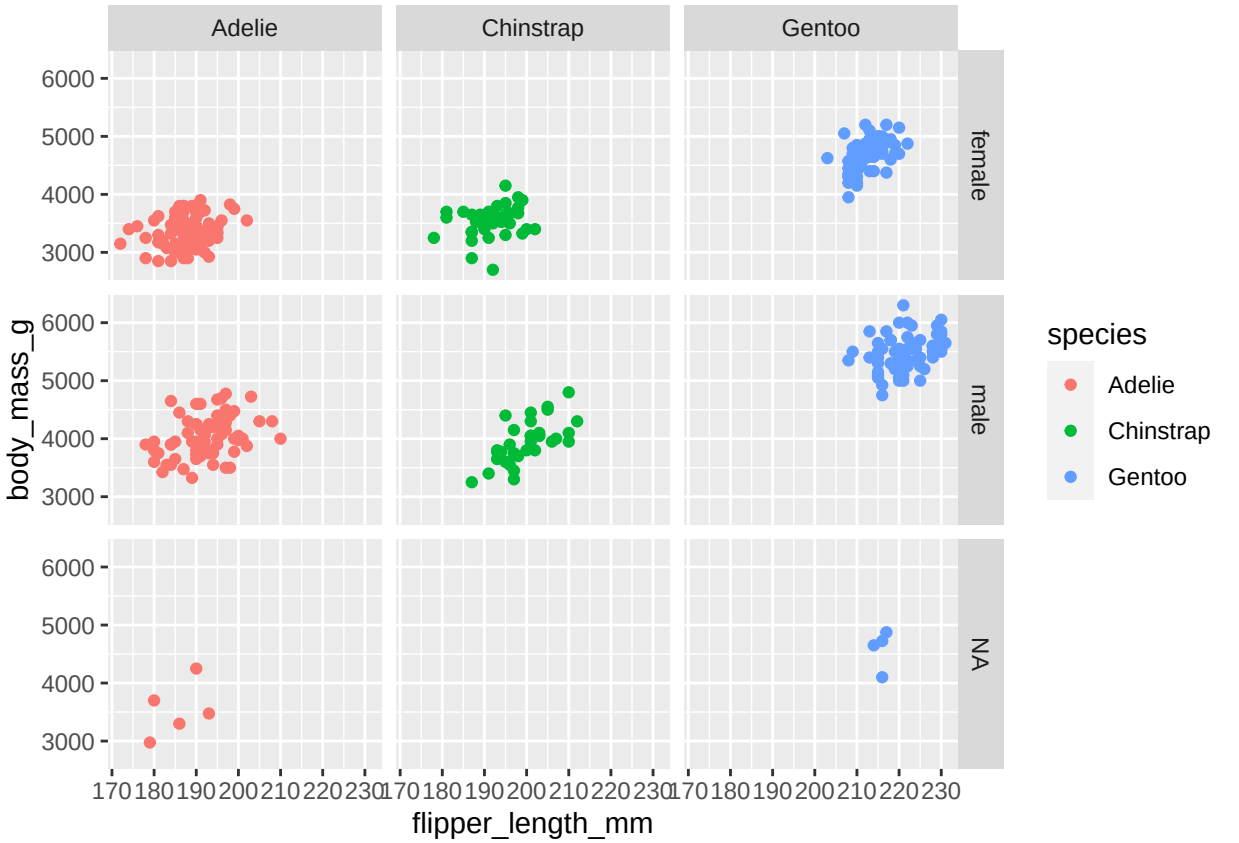



**facet_grid()** Si deseamos facetar nuestros datos por mas de una variable utilizamos facet_grid(), esto separa los diagramas en facetas verticalmente según los valores de la primera variable y horizontalmente según los valores de la segunda variable.

Por ejemplo podemos tomar nuestro diagrama de pinguinos y usar facet_grid con las variables, sex y species

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

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




El codigo anterior crea 9 gráficos diferentes, hay 3 especies y 3 generos. Y cada grafico representa la relacion body_mass_g vs flipper_length_mm. con facet_grid() podemos reorganizar y mostrar con rapidez datos complejos y hace que sea más fácil ver realciones entre diferentes grupos.

### Personalizando la apariencia de nuestros diagramas

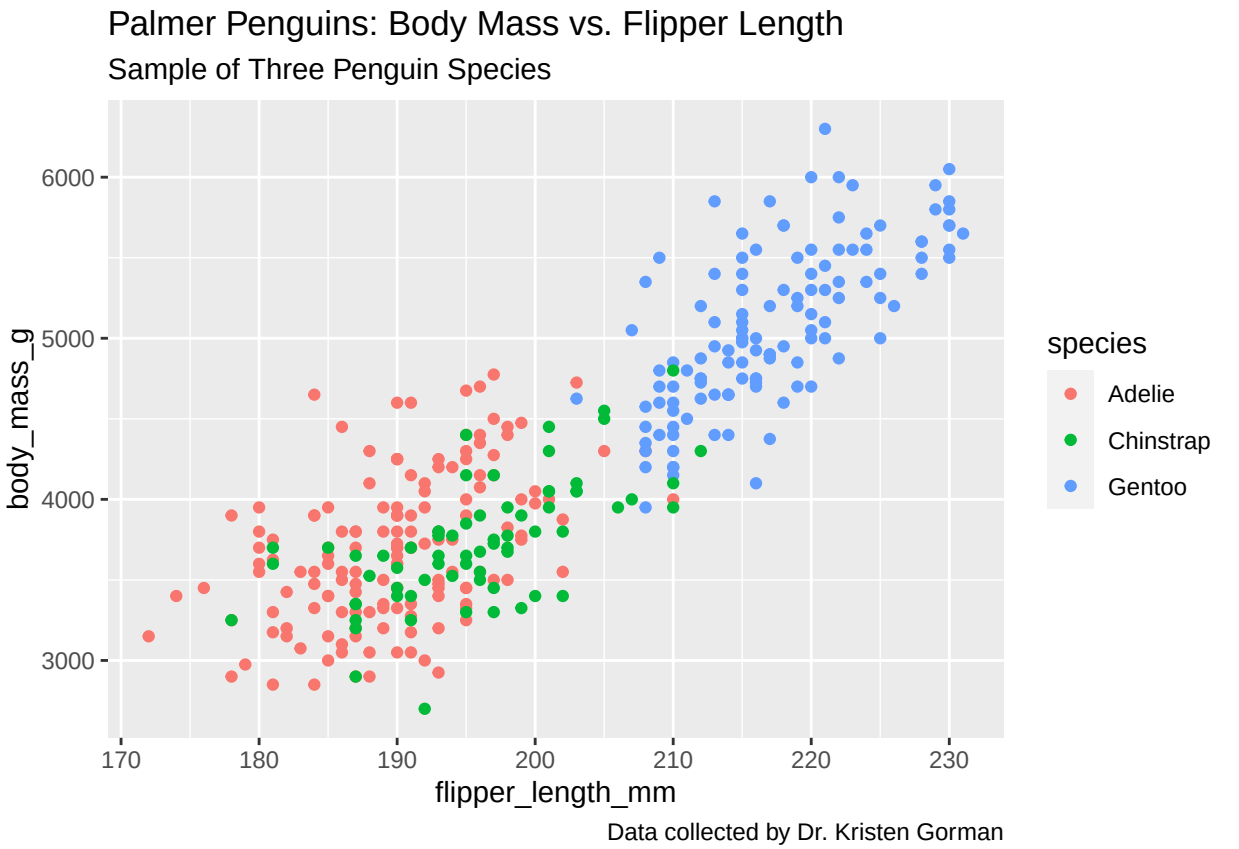
**Usando las funciones label y annotate** En ggplot2, agregar anotaciones a un diagrama puede ayudar a explicar el propósito del diagrama o a destacar datos importantes. Cuando presentas tus visualizaciones de datos a interesados, quizás no tengas mucho tiempo para reunirte con ellos. Las etiquetas y anotaciones dirigirán su atención a elementos clave y los ayudarán a comprender tu diagrama con rapidez.

**label()** Es util para agregar etiquetas informativas a un diagrama, como titulos, subtitulos y leyendas

Agregando un titulo

```
ggplot(data = penguins) +
  geom_point(mapping = aes(x=flipper_length_mm, y=body_mass_g, color=species))+
  labs(title="Palmer Penguins: Body Mass vs. Flipper Length",
       subtitle = "Sample of Three Penguin Species",
       caption = "Data collected by Dr. Kristen Gorman")
```

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



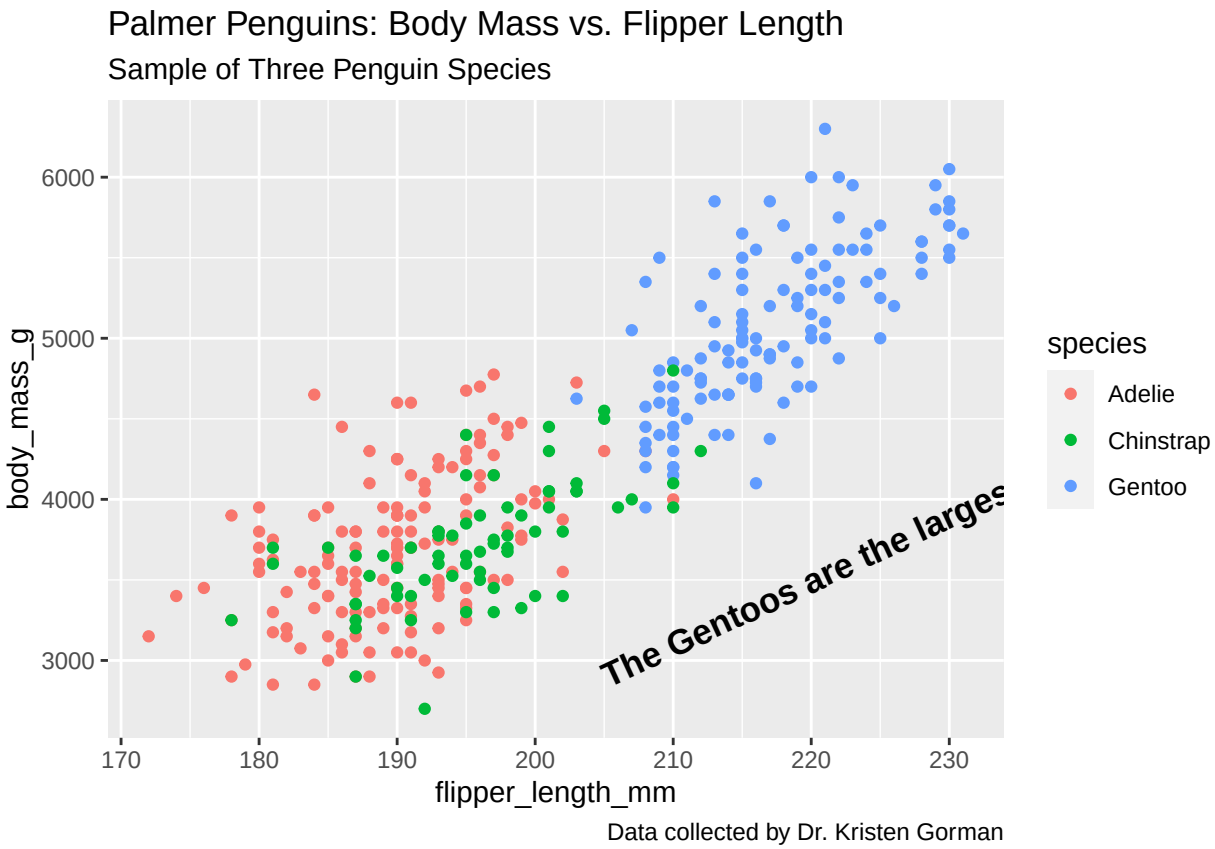

Si queres ingresar texto dentro de la cuadricula para destacar puntos de datos especificos, utilzamos la siguiente funcion:

**annotate()** Este texco comunica con claridad lo que muestra el diagrama y reforzará una parte importante de nuestros datos.

Los argumentos de la función annotate incluyen informacion sobre el tipo de etiqueta, coordenadas x e y donde se ubicara la etiqueta y el texto que incluirá la etiqueta.

```
ggplot(data = penguins) +
  geom_point(mapping = aes(x=flipper_length_mm, y=body_mass_g, color=species))+
  labs(title="Palmer Penguins: Body Mass vs. Flipper Length",
       subtitle = "Sample of Three Penguin Species",
       caption = "Data collected by Dr. Kristen Gorman") +
  annotate("text",
           x=220, y=3500,
           label= "The Gentoos are the largest",
           color="black",
           fontface="bold",size=4.5,angle=25)
```

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




**Guardar visualizaciones** Poder reproducir y compartir nuestro trabajo es importante en el analisis de datos. Para lograr guardar nuestras visualizaciones utilizamos la opción exportar en la pestaña de daigramas de RStudio o la funcion ggsave() que ofrece el paquete ggplot2

**Usando ggsave()** ggsave() es una funcion util para guardar un diagrama. De manera determinada guarda el último diagrama que ejecutaste y lo guarda del tamaño del dispositivo gráfico actual.