Introducción al análisis de datos utilizando R

Björn Reu, Yovanny Duran, Zarith Villamizar, Silvia Ardila y Sergio Bolivar

2022-02-07

Contents

Preface

El curso 'Introducción al análisis de datos utilizando R' originó como electiva en el programa de Biología de la Universidad Industrial de Santander a partir del año 2016. Su consolidación y evolución didáctica ha sido posible a través del proyecto Análisis de datos utilizando el lenguaje de programación R: una experiencia de aprendizaje virtual en el aula invertida, de la convocatoría Iniciativas de innovación didáctica mediante el uso de tecnologías, INNOVA-TIC 2021 de la Vicerrectoría Académica de la Universidad Industrial de Santander, UIS.

Chapter 1

Introducción

El curso Introducción al análisis de datos utilizando R está principalmente dirigido a estudiantes de Biología y de Ingeniería Forestal en sus primeros semestres para familiarizarlos en el uso del lenguaje de programación R. Sin embargo, dado que la mayoría dell contenido se dedica a la introducción a la programación y análisis de datos se ajusta fácilmente a las necesidades de estudiantes de otras carreras.

R es un lenguaje de programación de entorno de software libre para computación estadística y gráfica (R Development Core Team, 2021). En años recientes, R se ha vuelto una herramienta indispensable para el análisis de datos (o ciencia de datos) en muchas áreas de las ciencias básicas y aplicadas (Tippmann, 2015) y en la industria (Gandrud, 2013). En lugar de aprender múltiples herramientas, los estudiantes pueden utilizar un entorno único para una gran diversidad de métodos de análisis de datos y la presentación de sus resultados. Más que un lenguaje de programación, R ofrece nuevas vías no solo para la análisis de datos si no también para la enseñanza de la estadística, debido a que los problemas estadísticos pueden ser desarrollados y analizados de manera transparente y reproducible, p.ej. usando R Markdown (Incerti et al., 2019). Esto permite un aprendizaje interactivo y orientado a resolver problemas, combinando la teoría y la práctica. De esta manera, se puede lograr una comprensión más profunda de análisis de datos de forma más sencilla, lo cual permitirá practicar análisis de datos utilizando todas las herramientas y posibilidades que les ofrece el lenguaje de programación R.

1.1 Sobre este curso

Este curso está diseñado bajo el concepto de aula invertida (flipped classroom) que promueve el autoaprendizaje y puede ser dictada en modalidad presencial,

virtual o mixto (p.ej en presencialidad remota, o híbrida). Esto significa que los participantes, de manera autodidacta, adquieren conocimiento previo a la clase sincrónica (i.e. interacción con los tutores), mediante vídeos y scripts de programación en R disponibles a través de la plataforma Moodle. Durante la clase sincrónica y bajo la orientación de un instructor se profundizan estos conocimientos a través de ejercicios prácticos que buscan resolver problemas reales en el análisis de datos.

Este curso está estructurado en tres unidades, cada una compuesta de 5 sesiones, la última dedicada a la evaluación. En la primera unidad, se da inicio con una introducción a la programación y generación de gráficos en R, teniendo en cuenta los diferentes tipos de datos. En la segunda unidad se profundiza en métodos más avanzados y eficientes en la selección, el cálculo y visualización de datos usando la colección de paquetes tidyverse y se dan los fundamentos para presentar flujos de trabajo transparentes y reproducibles usando R Markdown. En la tercera Unidad se explorarán los conceptos básicos de la estadística univariada a través de dos tipos de análisis ampliamente utilizados en el análisis de datos, análisis de varianza y regresión lineal simple.

Cada unidad incluye una sesión completa de práctica con ejercicios y retos en donde se profundiza el aspecto de análisis de datos con R. Las clases están enfocadas a maximizar el trabajo práctico a través de programar y resolver problemas y reducir la introducción teórica al mínimo necesario, con el fin de fortalecer las competencias en la selección, análisis y visualización de datos.

1.2 Lo que aprenderás

- Aprenderás la sintaxis básica del lenguaje R, la de la colección de paquetes Tidyverse y de R Markdown; es decir aprenderás programar, graficar y presentar informes utilizando R.
- Aprenderás sobre diferente tipos de datos y las funciones que se utilizan para la manipulación, análisis y visualización de datos en R
- Aprenderás los conceptos de analizar y graficar datos en R y presentar los resultados utilizando R Markdown
- Aprenderás a manipular y indexar las distintas estructuras de datos y extraer elementos o subconjuntos
- Aprenderás a aplicar métodos básicos en el análisis de datos en R y graficar los resultados acorde con lineamientos básicos
- Aprenderás a utilizar y valorar R como herramienta principal en la gestión, análisis y visualización de datos.
- Aprenderás a valorar las estrategias que permitan la transparencia y reproducibilidad durante el proceso de análisis de datos.

1.3 Plan de estudios

El tiempo de dedicación a este curso son 8 horas semanales; 4 horas semanales de trabajo con interacción con el tutor, y 4 horas de trabajo independiente para repasar el contenido y resolver ejercicios de profundización. La dinámica del curso está diseñado de la siguiente manera:

(1) Previo a la reunión sincrónica con el tutor debes revisar el material teórico e introductorio que estará disponible en moodle. Con esta introducción, trabajarás de manera independiente en un script de R que está disponible en la plataforma moodle antes de la clase. Durante este trabajo puedes anotar tus inquietudes y preguntas. Terminarás esta primera fase de trabajo independiente con un quizz sencillo de preguntas de selección múltiple disponible en la plataforma moodle.

(2) Durante la clase sincrónica (i.e. en presencialidad o a través de una plataforma, p.ej. Zoom) trabajarás de manera interactiva en R sobre el script (i.e. cada participante un su computador y compartiendo pantalla cuando sea necesario) y resuelvas las preguntas de manera interactiva. Además, durante la clase sincrónica practicarás individualmente o en equipo en resolver las problemas de los ejercicios bajo la orientación del tutor.

(3) Terminando la clase el tutor te facilitará e introducirá algunos retos, los cuales debes trabajar como tarea de trabajo independiente y resolverlas durante la próxima clase.

El curso se evalúa a través de quizes y ejercicios, los cuales evalúan las competencias adquiridas en cada sesión. Los quizes sobre los conceptos y técnicas de cada sesión serán considerados para la nota final del curso (en total 40% de la nota final). Además, al final de cada una de las tres Unidades del curso, desarrollarás ejercicios de análisis de datos que valdrán como cada uno 20% de la nota final.

1.4 ¿Cómo empezar?

Todo el material que acompaña el curso Introducción al análisis de datos utilizando R está disponible en la plataforma moodle de la Universidad; si no tienes acceso debes ponerte en contacto con el coordinador de tu programa. Asimismo la comunicación con el tutor, con los demás participantes, e incluso las actividades evaluativas se realizarán a través de la plataforma moodle; es decir debes siempre estar atento y consultarlo con frecuencia. Durante el curso trabajamos con el lenguaje de programación R y el interfaz gráfico RStudio. Se requiere instalar las dos, ayuda como instalar y descargarlos puedes encontrar aquí:

- https://www.r-project.org/ -> Download
- https://www.rstudio.com/ -> Download

1.5 Datos

1.5.1 SantanderBIO

El proyecto SantanderBio consistió en expediciones biológicas para documentar la biodiversidad del departamento de Santander en Colombia. SantanderBio se llevó a cabo por medio de expediciones en tres ecosistemas estratégicos del departamento, siendo estos las inmediaciones del Páramo de Almorzadero en el municipio de Santa Bárbara, la Serranía de los Yariguíes en el municipio El Carmen de Chucurí y la cuenca media del río Magdalena en el municipio de Cimitarra.

Como resultado las expediciones realizaron un aporte considerable al inventario de la biodiversidad en grupos biológicos como Plantas, Insectos, Peces, Anfibios, Reptiles, Aves y Mamíferos, con un rango de altura desde 74 hasta 3603 metros. Los datos estań estructurado como se puede acceder a través del Global Biodiversity Information Facility (GBIF).

1.5.2 ColTree

Estos datos hacen parte de la fundación COLTREE, la cual se dedica al monitoreo de parcelas permanentes de vegetación en Colombia. Los datos corresponden a la abundancia de todas las especies en parcelas de 20 por 20 metros y comprenden 148 parcelas ubicadas sobre el norte de la cordillera oriental, en los departamentos de Santander y Norte de Santander, desde el Magdalena medio hasta el páramo de Santurbán. Las parcelas abarcan un rango altitudinal entre 65 y 3410 msnm.

1.5.3 Páramos

1.5.4 Permanencia

Chapter 2

Introducción a la programación con R

Comenzaremos con las operaciones básicas ejecutadas en la línea de comandos en R o terminal. Este será nuestro espacio de trabajo en el cual ejecutaremos las ordenes cuyos resultados se mostrarán en la consola.

Todo lo que siga después de un numeral (#) será considerado como un comentario que al ejecutar será omitido. Para ejecutar el código en R podemos utilizar el botón "run" que disponemos arriba a la derecha, o utilizar un atajo:

- **Command + Enter (Mac)**
- **Control + Enter (Windows, Linux)**

2.0.1 La línea de comandos (Terminal)

Línea de comandos como calculadora:

```
2+5
- [1] 7
7-3
- [1] 4
(2+3)*5
- [1] 25
(2+3)/((2+3)*5)
- [1] 0.2
```

Se pueden ejecutar varias operaciones utilizando punto y coma (;):

```
2+5; 4+6
- [1] 7
- [1] 10
```

Dispone de diferentes funciones como raíz cuadrada o logaritmo:

```
?Arithmetic
sqrt(9) # *raíz cuadrada*
- [1] 3
log(1)  # logaritmo
- [1] 0
log10(10) # logaritmo base 10
- [1] 1
3^2 # potenciación (exponente 2)
- [1] 9
```

Las letras y variables más importantes estan disponibles en R:

```
?Constants
pi
- [1] 3.141593
letters
-  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t"
- [24] "x" "y" "z"
LETTERS
-  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S" "T"
- [24] "X" "Y" "Z"
```

2.0.2 Función ayuda

Existen tres formas de obtener ayuda:

- Área miscelánea -> “Help”
- Símbolo “?” antes de la función
- Tecla Tabulador (Autocompleta el nombre)

```
x <- c(1,2,3,4,5,6,7,8) # <- símbolo de asignación, permite agrupar los elementos dent
```

```
x = c(1,2,3,4,5,6,7,8) # "<-" o "=" significa lo mismo
```

```
mean(x)
- [1] 4.5
?sum
?rep
?sort
?order
```

Estructura de la función:

- Función: “mean()”
- Objeto: “x”
- Argumento (‘arg’): “na.rm=FALSE”

Ejercicio 1

1. Explique brevemente la función “order”. ¿Qué hace y que significan los argumentos?

2.0.3 Creando nuevas variables:

Para crear una nueva variable usted puede utilizar “<-” (símbolo de asignación) o “=”. Es aconsejable el uso de “<-” ya que se refiere a la indexación dentro de un vector.

Cada variable necesita un nombre que puede estar compuesto de letras, números, “.” o ”_” y nunca por números. Ejemplo: “my_data”.

R diferencia entre minúsculas y mayúsculas; “a” es diferente de “A”. Las variables pueden ser reescritas. Una vez escrita una variable podemos llamar al objeto:

```
x <- 3
x
- [1] 3
x <- 9
x
- [1] 9
resultado <- 3+9
resultado
- [1] 12
mode(resultado)
- [1] "numeric"
```

Además de las variables ('numeric'), R permite la definición de texto ('character') y de variables lógicas ('logic', ej: TRUE y FALSE). Los caracteres deben ser puestos entre comillas dobles ("…") para que R los identifique como tal.

Estos tipos de vectores se definen como vectores atómicos:

```
y <- "test"
y
- [1] "test"
mode(y)
- [1] "character"
a <- FALSE
a
- [1] FALSE
mode(a)
- [1] "logical"
mode(T) # T es igual a TRUE, y F es igual a FALSE
- [1] "logical"
```

¿Qué ocurre aquí?

```
s <- as.numeric(c(T,F,F))
mode(s)
- [1] "numeric"
```

2.0.4 El área de trabajo o memoria - Ventana arriba a la derecha 'Environment'

Todos los objetos creados durante una sesión son guardados en el área de trabajo. Para ver el área de trabajo ejecute ls(), para remover algún objeto de esta ejecute rm().

También puede utilizar los botones en el área de trabajo

```
ls()
- [1] "a"         "resultado" "s"         "x"         "y"
rm(s)
ls()
- [1] "a"         "resultado" "x"         "y"
```

2.0.5 Vamos a practicar

Puedes hacer cálculos simples con variables númericas

```
a <- 5
b <- 7
a+b+3
- [1] 15
```

¿Qué ocurre a continuación? Por favor explique.

```
b = 2
b == 2
- [1] TRUE
b == 3
- [1] FALSE
c = 7
d = -3
?Syntax
```

Ejercicio 2

1. Comprobar las siguientes afirmaciones lógicas:

- “c” mayor que “d”
- “c” menor o igual que “d”
- “c” menor o igual que “d”

2.0.6 R como calculadora - Operadores y funciones.

```
?Syntax

* <-               Asignar
* + - * / % ^      Aritméticas
* > >= < <= == !=  Relación (orden y comparación)
* ! & && | ||      lógicas
* $                lista indexada
* :                Crear una secuencia
```

Operadores lógicos para:

```
* !      NO
* &      Y
```

```
* |      O
* <     Menor que
* <=    Menor o igual que
* >     Mayor que
* >=    Mayor o igual que
* ==    IGUAL
* !=    NO IGUAL (diferente de)
* &&    AND with IF (y con si <condicional>)
* ||    OR with IF (o con si <condicional>)
```

2.0.7 Tipos de Datos y Objetos

- **"Logical"** :(FALSO/VERDADERO) / (FALSE/TRUE)
- **"Number"** : (Entero, Decimal, Complejo (e.g. 3i))
- **"Character"** : Letras y palabras ("", o '')

Otros tipos de datos son **"list"**, **"expression"**, **"name"**, **"symbol"** and **"function"**

Para operaciones más complejas necesitamos estructuras de datos más complejas. R ofrece más que solo objetos que contienen un elemento, como los vectores o las matrices.

2.0.8 Vectores I: Vectores sencillos

Para hacer un vector utilice la función concatenar "c()" . Los elementos sencillos de un vector están separados por ",".

```
x <- c (2, 5, 10, 14, 3, 1, 18, 24, 17)
x
- [1]  2  5 10 14  3  1 18 24 17
mode(x)
- [1] "numeric"
```

```
a <- c ("text", 2, 6, TRUE)
mode(a) ## ¿Qué ocurre aquí? Por qué "character"?
- [1] "character"
a <- c ( 2, 6, F)
mode(a) #y ahora?
- [1] "numeric"
```

Crear un vector vacío:

```
b <- numeric(20)
b
-  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Ejercicio 3

1. Crear un vector con nombre “vec” que contenga los números de 1 a 10.

Un vector también puede contener diferentes tipos de variables. Usted puede realizar cálculos con vectores que contengan solo elementos “numeric”

- **Suma**

```
sort(x)
- [1]  1  2  3  5 10 14 17 18 24
sum(x)
- [1] 94
```

- **Promedio**

```
mean(x)
- [1] 10.44444
```

- **Cuartil**

```
quantile(x)
-   0%  25%  50%  75% 100%
-    1    3   10   17   24
```

- **Longitud del vector**

```
length (x)
- [1] 9
length(letters)
- [1] 26
```

- **Ordenar**

```
x
- [1]  2  5 10 14  3  1 18 24 17
?sort(x)
sort(x, decreasing = F)
- [1]  1  2  3  5 10 14 17 18 24
sort(x, decreasing = FALSE)
- [1]  1  2  3  5 10 14 17 18 24
sort(x, decreasing = T)
- [1] 24 18 17 14 10  5  3  2  1
sort(x, decreasing = TRUE)
- [1] 24 18 17 14 10  5  3  2  1
```

¿Qué ocurre si colocamos "decreasing = TRUE"? Compruebe con "?sort"

"sort" y **"order"** realizan la misma función; sin embargo **"order"** puede ser aplicado a otro tipo de objetos diferentes a vectores, como los data frames.

- **Cálculo con vectores**

```
x
- [1]  2  5 10 14  3  1 18 24 17
length(x)
- [1] 9
#1
x + 10
- [1] 12 15 20 24 13 11 28 34 27
x
- [1]  2  5 10 14  3  1 18 24 17
#2
y <- c(1, 3, 5)
x + y
- [1]  3  8 15 15  6  6 19 27 22
x
- [1]  2  5 10 14  3  1 18 24 17
#3
y <- c (4, 2, 8, 5, 3, 9, 3, 10, 1)
xy <- x + y
xy
- [1]  6  7 18 19  6 10 21 34 18
x
- [1]  2  5 10 14  3  1 18 24 17
#4
y <- c(1, 3)
x + y
- Warning in x + y: longer object length is not a multiple of shorter object length
```

```
- [1]  3  8 11 17  4  4 19 27 18
x
- [1]  2  5 10 14  3  1 18 24 17
#5
y <- c (4, 2, 8, 5, 3, 9, 3, 10, 1)
sum(x)
- [1] 94
sum(x,y)
- [1] 139
sum(x + y)
- [1] 139
```

- **Etiquetas**

```
x <- c (2, 5, 10, 14, 3, 1, 18, 24, 17)
a <- c ("E1","E2","E3","E4","E5","E6","E7","E8","E9")
names (x) <- a
x
- E1 E2 E3 E4 E5 E6 E7 E8 E9
-  2  5 10 14  3  1 18 24 17
names(x) # Vector de los nombres del vector numérico x
- [1] "E1" "E2" "E3" "E4" "E5" "E6" "E7" "E8" "E9"
str (x) # sobre la estructura del vector x (muy útil!)
-  Named num [1:9] 2 5 10 14 3 1 18 24 17
-  - attr(*, "names")= chr [1:9] "E1" "E2" "E3" "E4" ...
summary (x) # indica los valores mínimos, máximos, etc.
-    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-    1.00    3.00   10.00   10.44   17.00   24.00
head (x)
- E1 E2 E3 E4 E5 E6
-  2  5 10 14  3  1
tail (x)
- E4 E5 E6 E7 E8 E9
- 14  3  1 18 24 17
```

Ejercicio 4:

- *Sumar los números enteros de 1 hasta 5.*
- *Crear un variable v1 que contenga una letra.*
- *Copiar v1 a v2.*
- *Comparar los valores de v1 y v2*

- *Crear un vector de longitud 20 con el tipo de datos que usted quiera y muestre las primeras nueve entradas.*
- *Crear un vector que contenga los seis primeras letras del abecedario.*

2.0.9 Vectores II: Vectores más complejos

- **Secuencias**

```
z <- 3:10
z
- [1]  3  4  5  6  7  8  9 10
```

```
?seq
seq(from = 1, to = 6, by = 0.2) # El argumento "by" genera frecuencias #
-  [1] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0 3.2 3.4 3.6 3.8 4.0 4.2 4.4 4.6 4.8
- [24] 5.6 5.8 6.0
seq(from = 1, to = 6, length.out = 20)  # ¿Cuál es la diferencia con respecto al anter
-  [1] 1.000000 1.263158 1.526316 1.789474 2.052632 2.315789 2.578947 2.842105 3.10526
- [11] 3.631579 3.894737 4.157895 4.421053 4.684211 4.947368 5.210526 5.473684 5.73684
```

La secuncia puede iniciar desde cualquier punto

```
seq(from = 530, to = 620, by = 30)
- [1] 530 560 590 620
```

```
z <- c(1, 3, 9)
z
- [1] 1 3 9
?rep
rep (z, times = 10) # ¿Qué ocurre aqui?
-  [1] 1 3 9 1 3 9 1 3 9 1 3 9 1 3 9 1 3 9 1 3 9 1 3 9 1 3 9 1 3 9
```

La secuencia de elementos del vector z se repite 10 veces

```
rep (z, each = 10) # ¿Cuál es la diferencia con la línea anterior?
-  [1] 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 9 9 9 9 9 9 9 9 9 9
```

Aquí cada uno de los elementos del vector z se repite 10 veces.

2.0.10 Indexar

Para extraer uno o varios elementos de un vector utilice "[]". Por favor observe y explique que ocurre en las siguientes líneas:

```
x
- E1 E2 E3 E4 E5 E6 E7 E8 E9
-  2  5 10 14  3  1 18 24 17
x[5]
- E5
-  3
x[4]
- E4
- 14
x[3:5]
- E3 E4 E5
- 10 14  3
x[c(3,5)]
- E3 E5
- 10  3
id<-c(1, 9, 2)
x[id]
- E1 E9 E2
-  2 17  5
x[c (1, 9, 2)]
- E1 E9 E2
-  2 17  5
x[-c (1, 9, 2)]
- E3 E4 E5 E6 E7 E8
- 10 14  3  1 18 24
```

Ejercicio 5:

- *Crear un vector "x" con los números de 1 a 100.*
- *Extraer del vector el elemento numero 87.*
- *Extrer todos los elementos excepto el 87 para crear un nuevo vector "z".*
- *Verificar la longitud del nuevo vector.*

PLUS: Extraer cada segundo elemento del vector

- *Crear un vector vacío de longitud 10 y asignar a su tercer elemento el valor que tiene la suma de x.*

Ejercicio 5:

- *Crear un vector vacío de cuatro elementos*
- *En el vector vacío indexe para cada posición la primera inicial de sus nombres y apellidos. (Si tiene un solo nombre indexe NA).*

2.0.11 NA´s

```
?NA
e <- c(122, 324, 34, NA, 234)
mean(e) # Por qué da este resultado? Con NA no se puede calcular!
- [1] NA
?mean
mean(e, na.rm = T) # Ahora sin tener en cuenta el NA
- [1] 178.5
is.na(e)        # Me indica cuales son NA
- [1] FALSE FALSE FALSE  TRUE FALSE
which(is.na(e)) # Me indica la posición del NA
- [1] 4
```

2.0.12 Matrices

La matriz es un conjunto de elementos que son del mismo tipo, ya sea numéricos, de carácter o lógico, distribuidos en dos dimensiones, filas y columnas.

- [1,] indica la primera fila
- [,1] indica la primera columna

```
mat <- matrix(data = 1:12, nrow = 3, ncol = 4, byrow = T)
mat
-      [,1] [,2] [,3] [,4]
- [1,]    1    2    3    4
- [2,]    5    6    7    8
- [3,]    9   10   11   12
```

```
mat[,1]
- [1] 1 5 9
mat[2,2]
- [1] 6
mat[1:3,1:3]
-      [,1] [,2] [,3]
- [1,]    1    2    3
- [2,]    5    6    7
- [3,]    9   10   11
mat[,1]
- [1] 1 5 9
mat[1,]
- [1] 1 2 3 4
```

- **Transponer**

```
mat2 <- t(mat)
mat2
-      [,1] [,2] [,3]
- [1,]    1    5    9
- [2,]    2    6   10
- [3,]    3    7   11
- [4,]    4    8   12
mat
-      [,1] [,2] [,3] [,4]
- [1,]    1    2    3    4
- [2,]    5    6    7    8
- [3,]    9   10   11   12
colnames(mat2) <- c("A","B","C")
mat2
-      A B  C
- [1,] 1 5  9
- [2,] 2 6 10
- [3,] 3 7 11
- [4,] 4 8 12
rownames(mat2) <-c ("sp1","sp2","sp3","sp4")
mat2
-     A B  C
- sp1 1 5  9
- sp2 2 6 10
- sp3 3 7 11
- sp4 4 8 12
class(mat2[,1])
- [1] "integer"
```

```
class(mat2[2,1])
- [1] "integer"
```

Ejercicio 6:

- *Realice una matriz con 11 columnas y 11 filas.*
- *Incluya en la matriz el número 8 en cada esquina y en la mitad de la matriz.*

- **cbind y rbind**

```
mat
-      [,1] [,2] [,3] [,4]
- [1,]    1    2    3    4
- [2,]    5    6    7    8
- [3,]    9   10   11   12
c5 <- c(5,5,5)
cbind(mat, c5)
-                  c5
- [1,] 1  2  3  4  5
- [2,] 5  6  7  8  5
- [3,] 9 10 11 12  5
r4 <- c(4,4,4,4)
rbind(mat, r4)
-    [,1] [,2] [,3] [,4]
-       1    2    3    4
-       5    6    7    8
-       9   10   11   12
- r4    4    4    4    4
```

- **rownames y colnames**

```
rownames(mat) <- c("a", "b", "c")
mat
-   [,1] [,2] [,3] [,4]
- a    1    2    3    4
- b    5    6    7    8
- c    9   10   11   12
colnames(mat) <- c("E1", "E2","E3","E4")
mat
```

```
-   E1 E2 E3 E4
- a  1  2  3  4
- b  5  6  7  8
- c  9 10 11 12
```

Chapter 3

Visualización de datos en R

En esta sesión vamos a explorar las herramientas en R para realizar gráficos a partir de datos, además revisaremos los diferentes elementos que permiten personalizar un gráfico.

Esta información puede ser complementada visulizando el video #3 del curso.

3.1 Gráficos y diagrama de dispersión simple

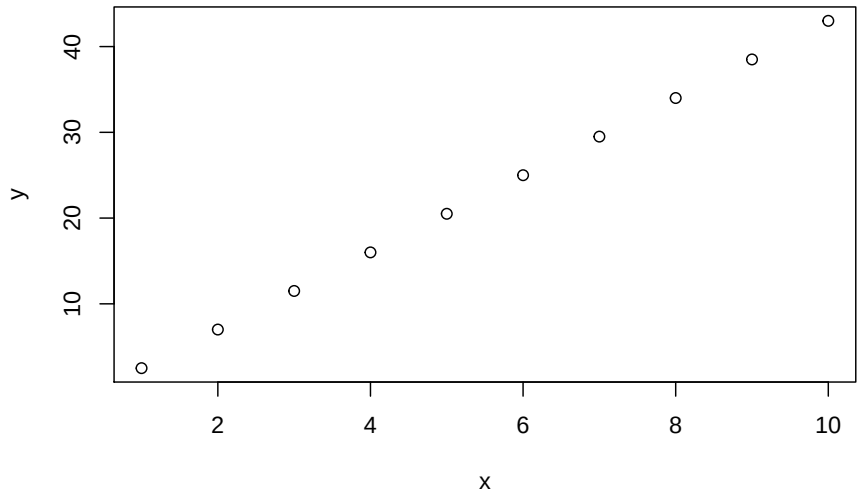
3.1.1 Diagrama de dispersión: función plot()

comenzamos generando algunos datos a graficar:

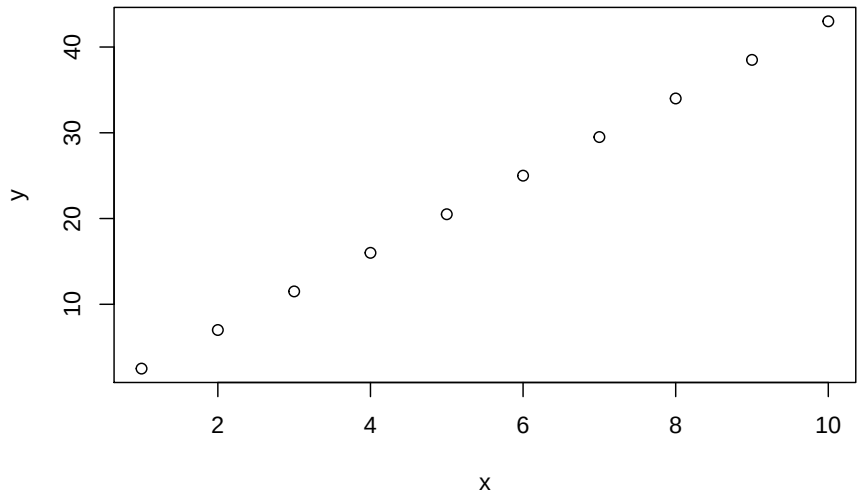
```
x <- 1:10
y <- 1.5 * x + seq(from = 1, to = 30,  by = 3) #
```

El diagrama de dispersión puede ser escrito de dos formas:

```
plot(x, y)
```

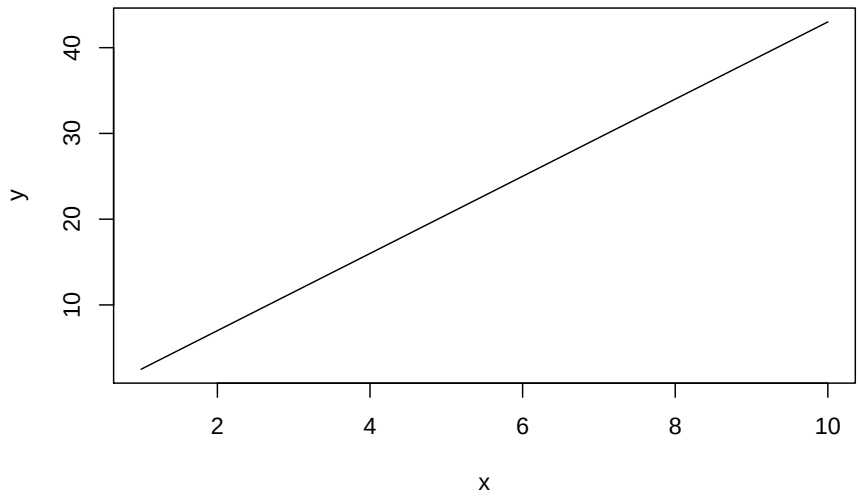



```
plot(y~x)
```

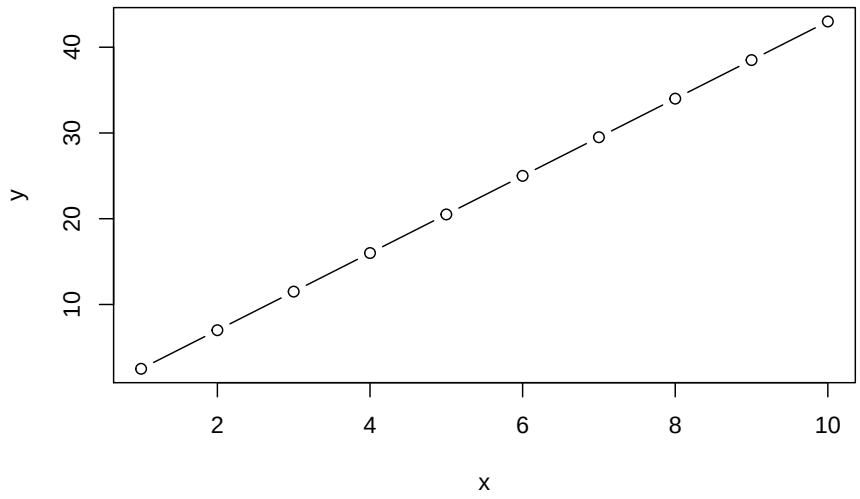


Con el argumento "type" es posible cambiar la disposición del diagrama de dispersión:

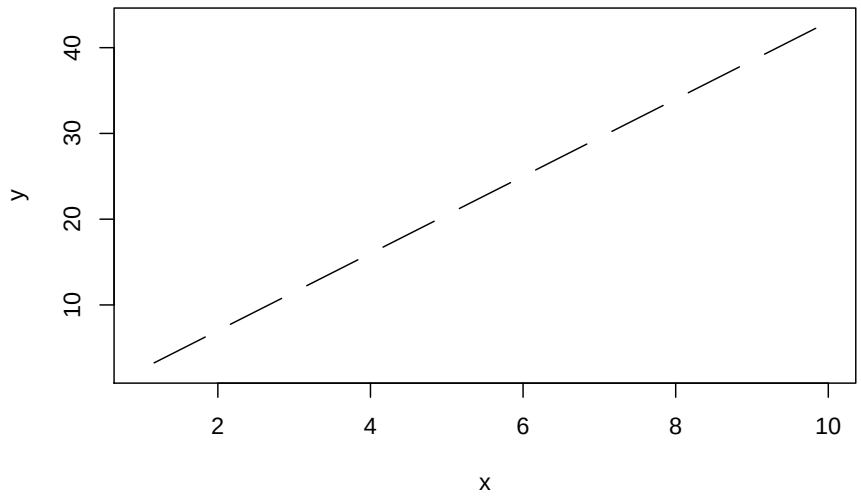
```
plot(x, y, type="l")
```



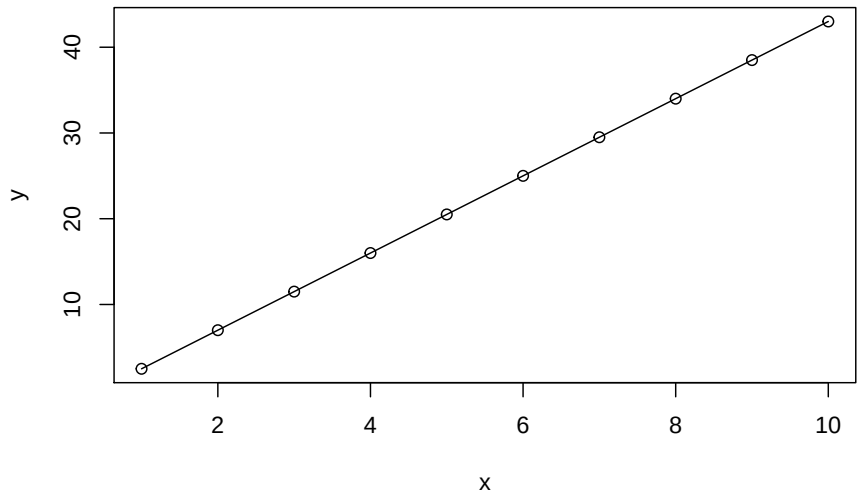

```
plot(x, y, type="b")
```



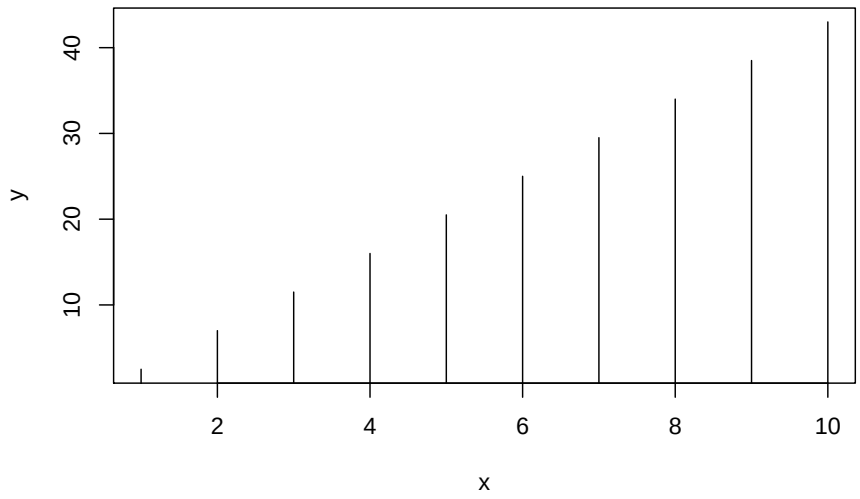
```
plot(x, y, type="c")
```



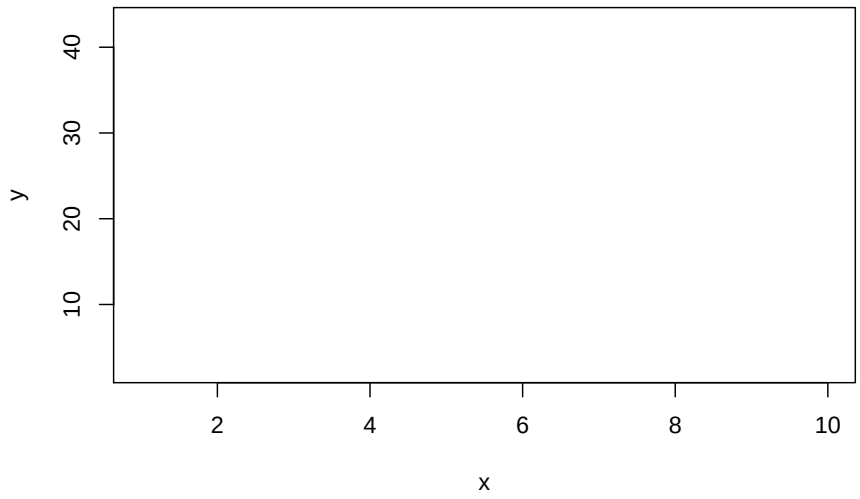

```
plot(x, y, type="o")
```



```
plot(x, y, type="h")
```

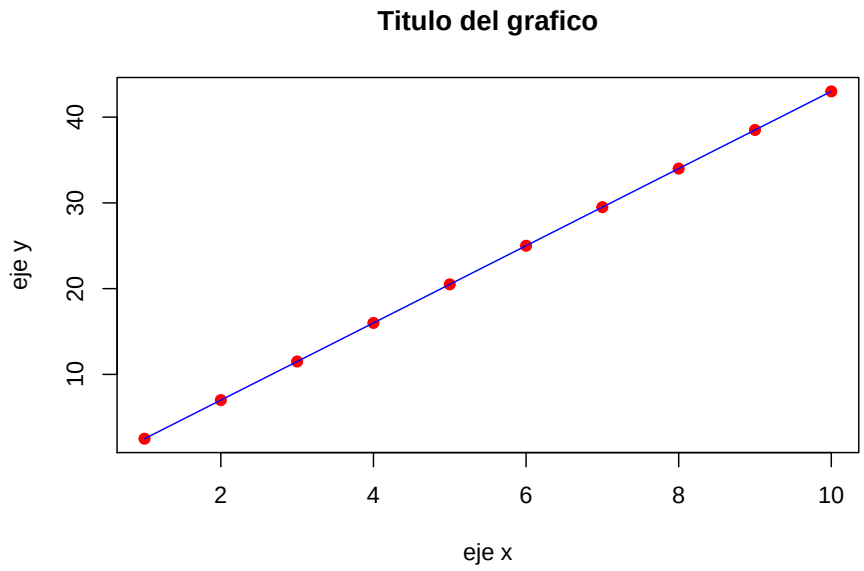



```
plot(x, y, type="n")
```



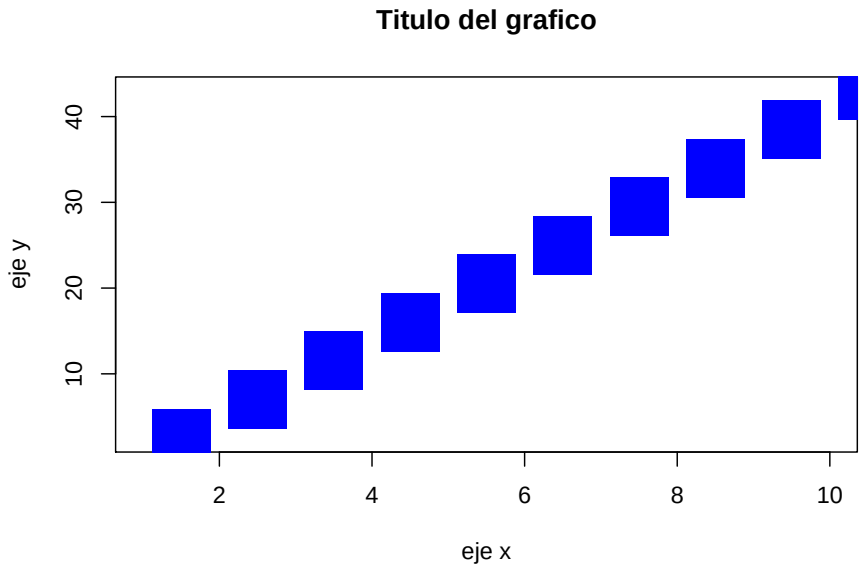
3.1.2 Etiquetas de la gráfica: xlab, ylab, main

```
plot(x, y, xlab = "eje x", ylab = "eje y", main = "Titulo del grafico", type = "n")
points(x , y, col = "red", pch = 19)
lines(x, y, col = "blue")
```



¿Qué ocurre aquí? Por favor explique:

```
plot(x, y, xlab = "eje x", ylab = "eje y", main = "Titulo del grafico", type = "n")
points(x + 0.5, y, col = "blue", pch = 15, cex = 5.5)
```



¿Para que son los argumentos “pch” y “cex”?

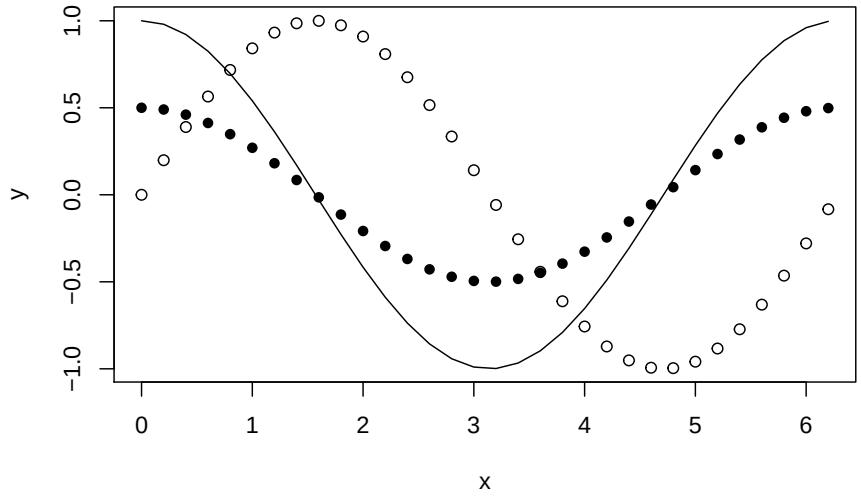
```
?points
```

Ejercicio 1

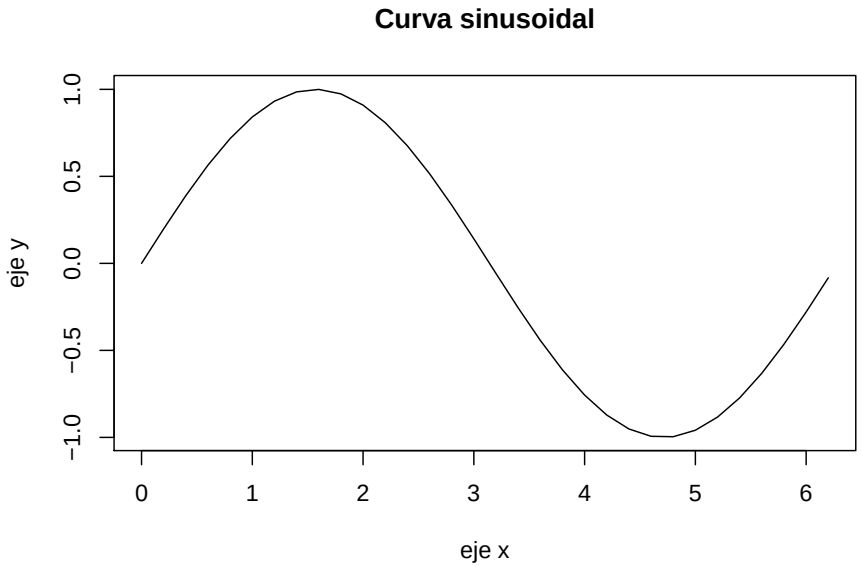
1. Haga una linea en zigzag con puntos de colores en los puntos de giro
2. Haga una carita (:|)

Más gráficos: {-}

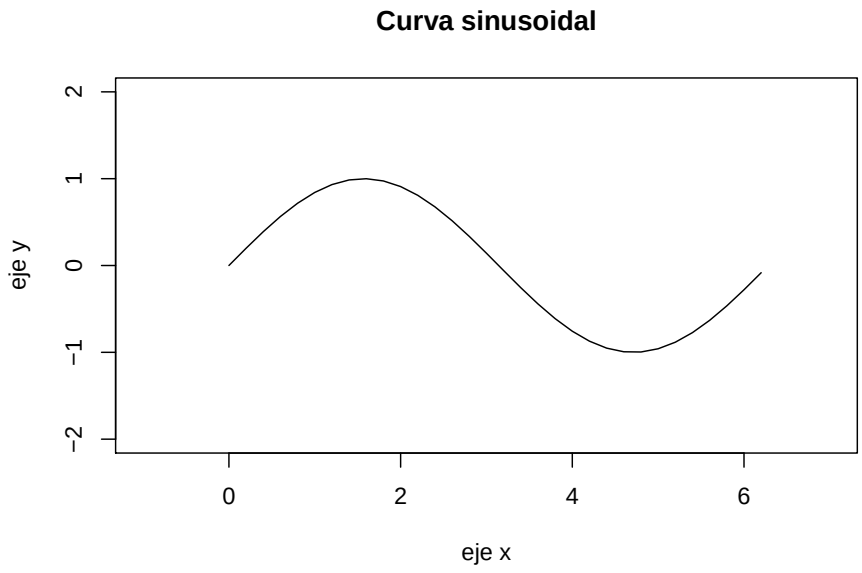
```
x <- seq(from = 0, to = 2 * pi, by = 0.2)
y <- sin(x) #Función seno
plot(x, y)

z <- cos(x) #Función coseno
lines(x, z)  # Agregando líneas al gráfico anterior
points(x, z * 0.5, pch = 16) # Agregando puntos al gráfico anterior
```



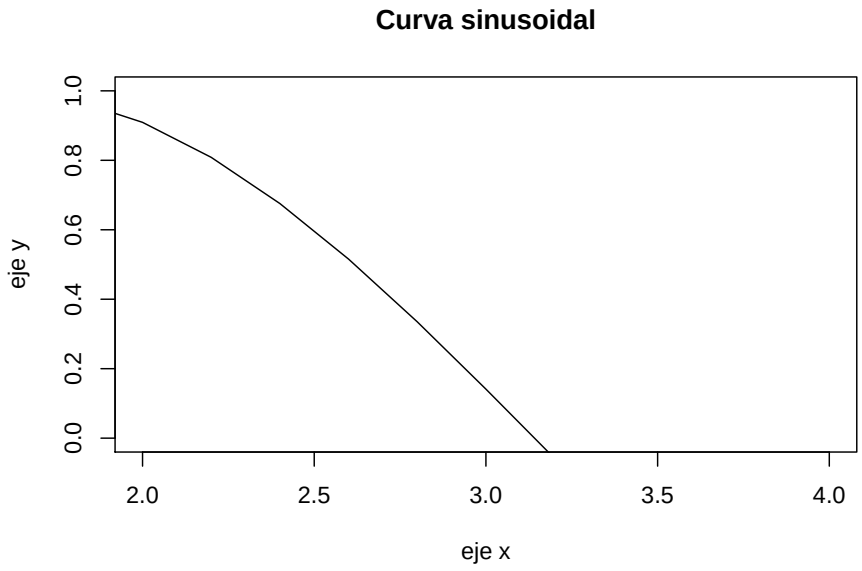

```
## De nuevo, eje, etiquetas y título
plot(x, y, xlab = "eje x", type = "l", ylab = "eje y", main = "Curva sinusoidal")
```



```r
## Cambiando la extensión de los ejes X y Y usando "xlim" y "ylim"
plot(x, y, xlab = "eje x", type = "l", ylab = "eje y", main = "Curva sinusoidal",
     xlim = c(-1, 7), ylim = c(-2, 2))
```




```r
plot(x, y, xlab = "eje x", type = "l", ylab = "eje y", main = "Curva sinusoidal",
     xlim = c(2, 4), ylim = c(0, 1))
```

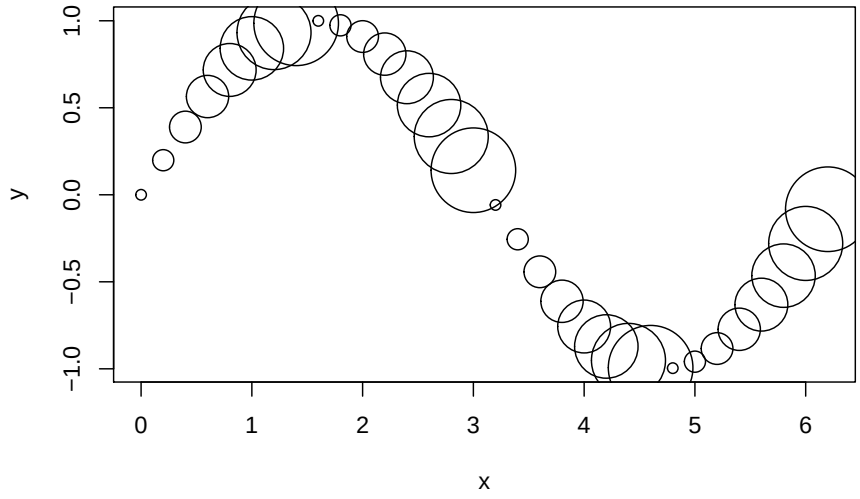


3.2 Demostración de la versatilidad de la gráfica

La representación gráfica en R es muy flexible. Vamos a cambiar mas parametros gráficos:

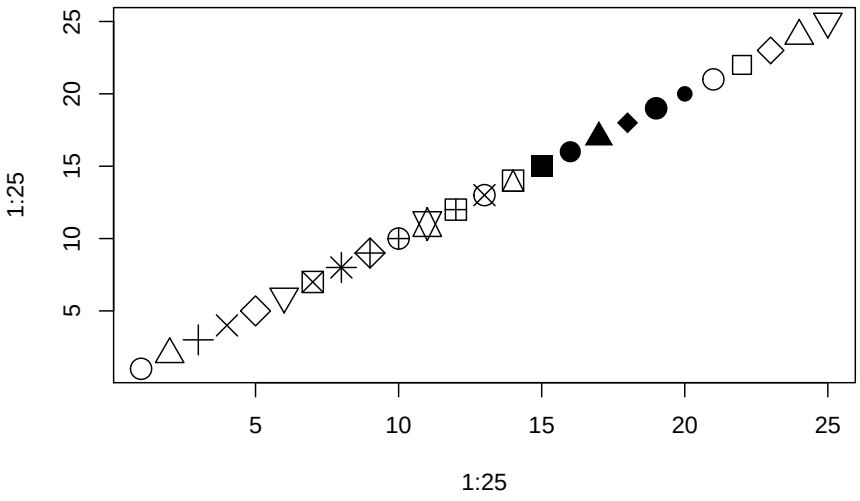
- 'cex': escala de simbolos (tamaño)

```
plot(x, y, cex = rep(1:8, times = 4))
```



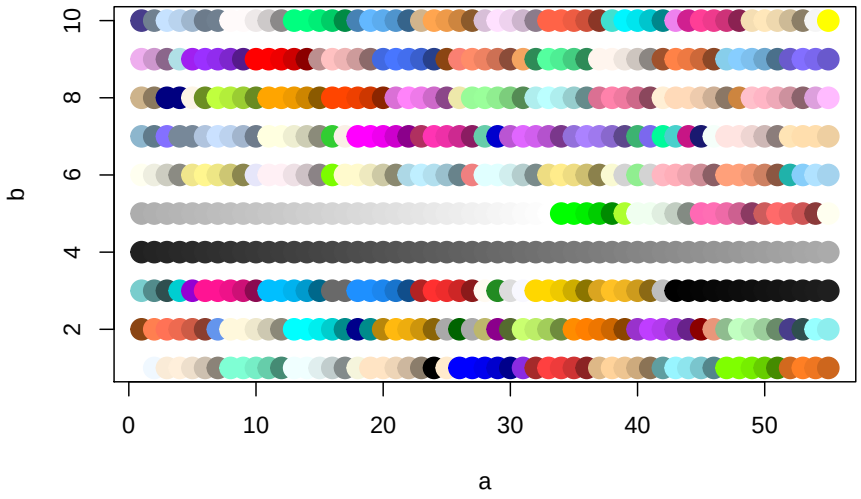
- 'pch': cambia el simbolo de los puntos

```
plot(1:25, 1:25, cex = 2, pch = 1:25)
```



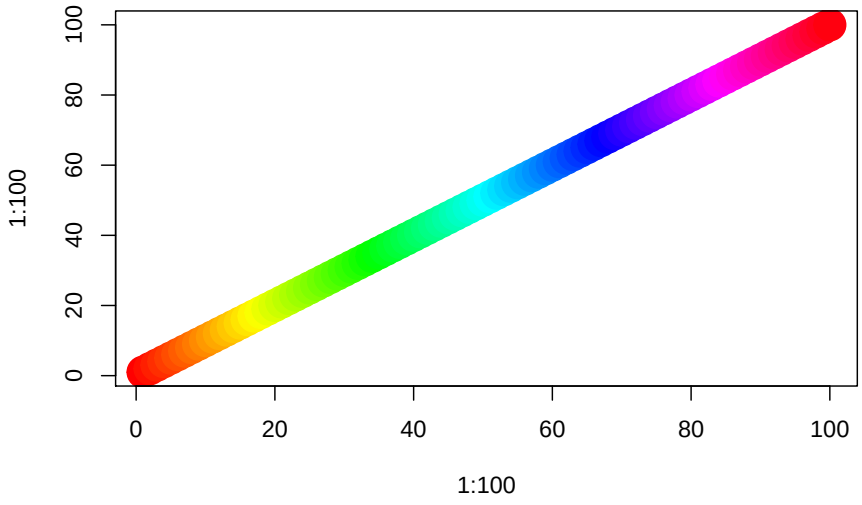
- 'col': colores de los puntos. En R se encuentran disponibles mas de 600 colores

```
a <- rep(1:55, times = 10); b <- rep(1:10, each = 55)
plot(a, b, pch = 19, cex = 2, col = colors()[-c(260:361, 653:657)])
```



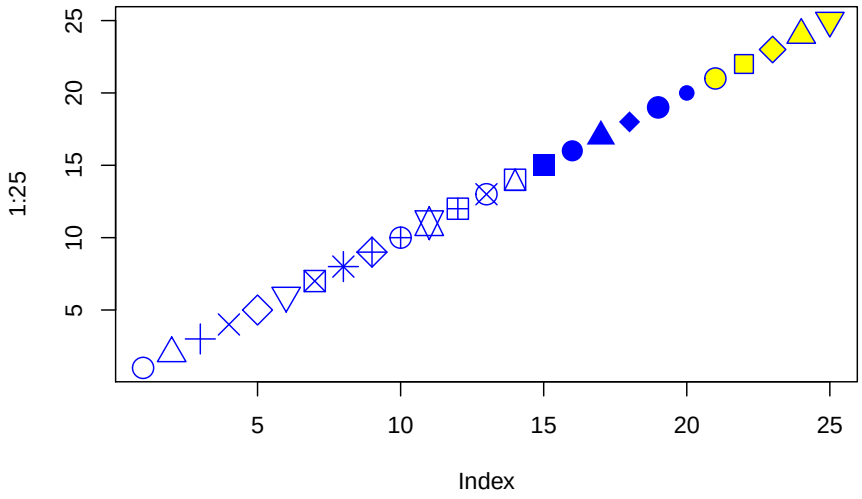
Los gradientes de colores tambien son posibles: ?rainbow

```
plot(1:100, 1:100, pch = 19, cex = 3, col = rainbow(100))
```



Los símbolos del 21 al 25 (?points) pueden tener diferentes colores de relleno

```
plot (1:25, cex=2, pch=1:25, col="blue", bg="yellow")
```



Ejercicio 2

1. Consulte la pagina web de Quick-R para ver ejemplos de diagramas de dispersion (http://www.statmethods.net/graphs/scatterplot.html) y elija uno que explique a sus colegas.

3.3 Barras, cajas e histogramas

Algunas de las gráficas más populares para presentar y explorar datos son las barras (conteo de observaciones), cajas (estadísticos de nuestros datos) e histogramas (frecuencia de valores)

En R, estas gráficas se realizan mediante las siguientes funciones:

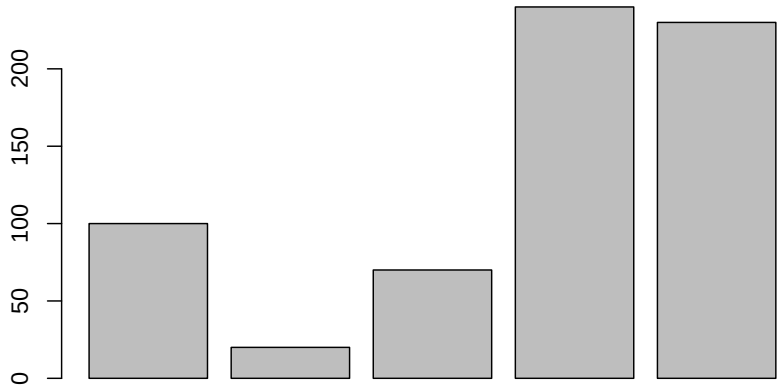
?barplot() ?boxplot() ?hist()

Vamos a explorar estas funciones.

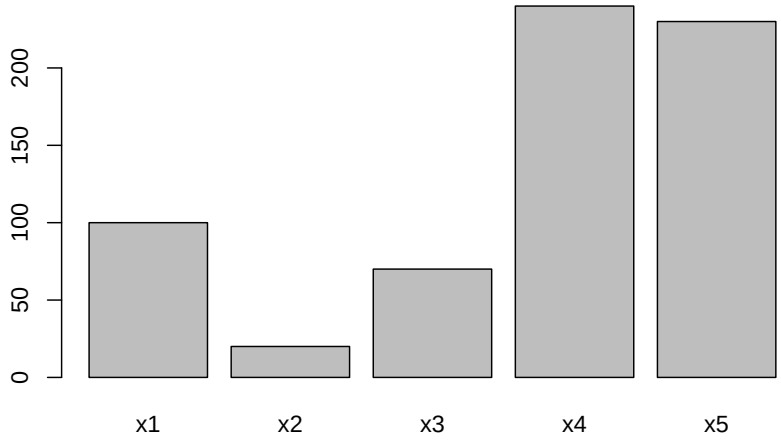
3.3.0.1 Barplot:

Si queremos observar el número de registros de nuestras variables, podemos hacerlo mediante un gráfico de barras. Para poder graficarlo necesitamos un vector o matriz que contenga valores numéricos

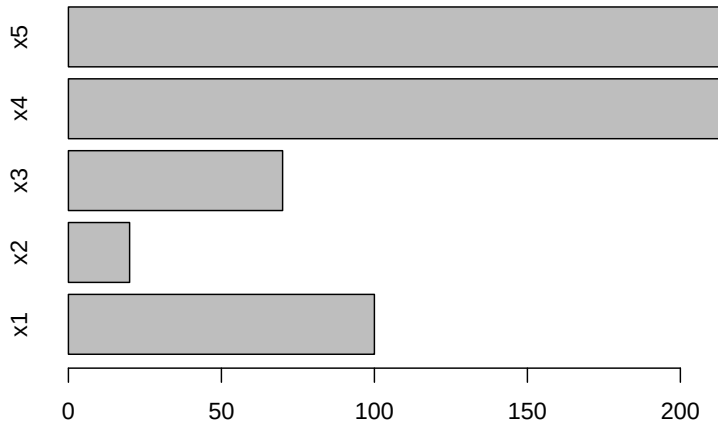
```
x <- c(100,20,70,240,230)

barplot(x)
```



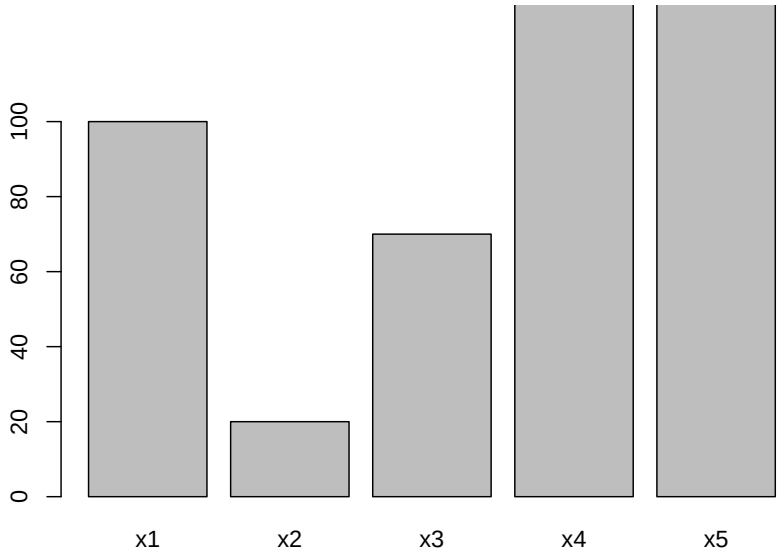

```r
barplot(x, names.arg = c("x1", "x2", "x3", "x4", "x5")) #Asignamos nombres a cada barra
```



```r
barplot(x, names.arg = c("x1", "x2", "x3", "x4", "x5"), horiz = TRUE) #Podemos inverti
```




```r
barplot(x, names.arg = c("x1", "x2", "x3", "x4", "x5"), ylim = c(0,100)) #Podemos limi
```



Ejercicio: Agregar colores diferentes a cada barra, nombres a los ejes y un título a la gráfica anterior

3.3.0.2 Boxplot:

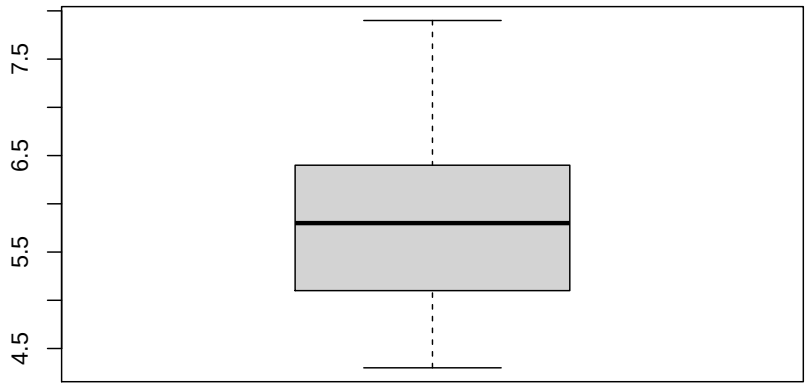
Mediante boxplot podemos hacer gráficos sencillos que nos muestren los estadísticos de nuestros datos.

```
data(iris)
```

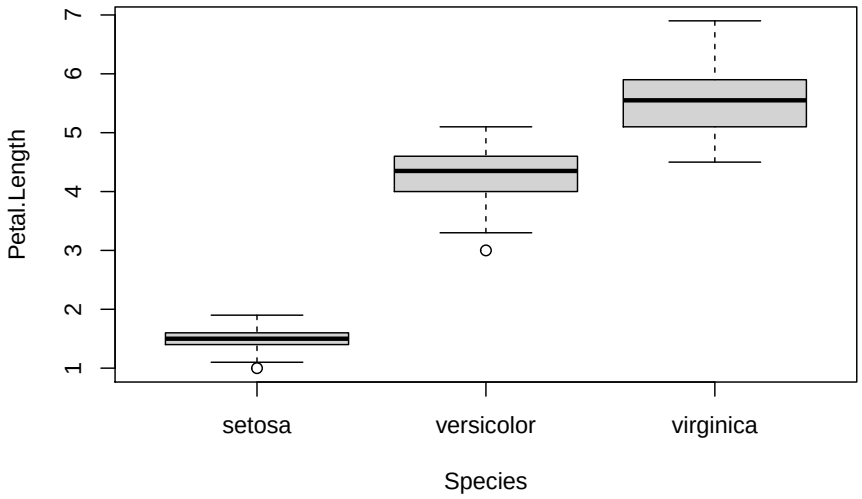
Cargamos un set de datos predeterminado en R y observamos su contenido.

Este conjunto de datos de iris da las medidas en centímetros de las variables longitud y anchura de los sépalos y longitud y anchura de los pétalos, respectivamente, para 50 flores de cada una de las 3 especies de iris. Las especies son Iris setosa, versicolor y virginica.

```
boxplot(iris$Sepal.Length) # En este boxplot podemos ver la mediana (linea negra central), los cu
```




```
boxplot(Petal.Length ~ Species, data = iris) #Podemos graficar los estadisticos para cada grupo u
```

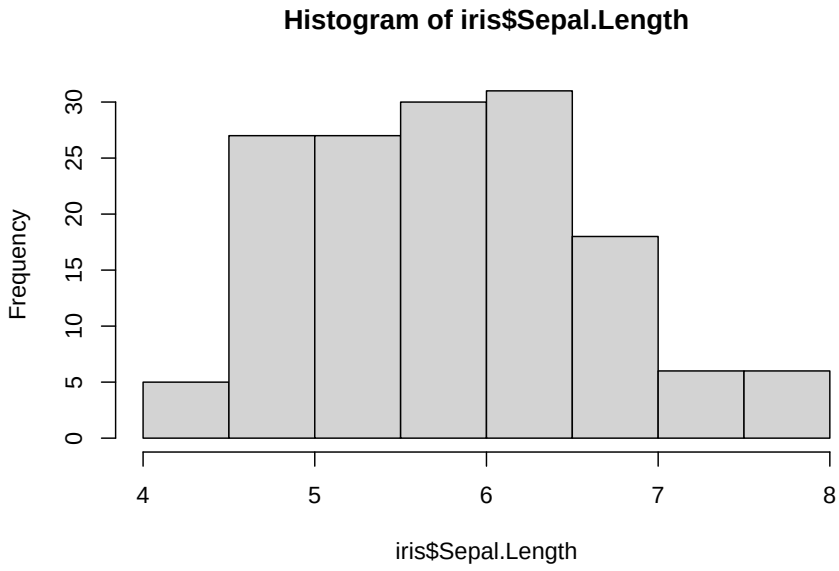


¿Qué pasa si invertimos las variables?

3.3.0.3 Histograma:

Finalmente, mediante un histograma podemos explorar frecuencias de valores

```
hist(iris$Sepal.Length) # En el gráfico, observamos la frecuencia (eje Y) y el rango d
```




```
hist(iris$Petal.Width, col = "yellow")
```

*Ejercicio:*Explore los diferentes argumentos dentro de cada función.

3.4 Función par()

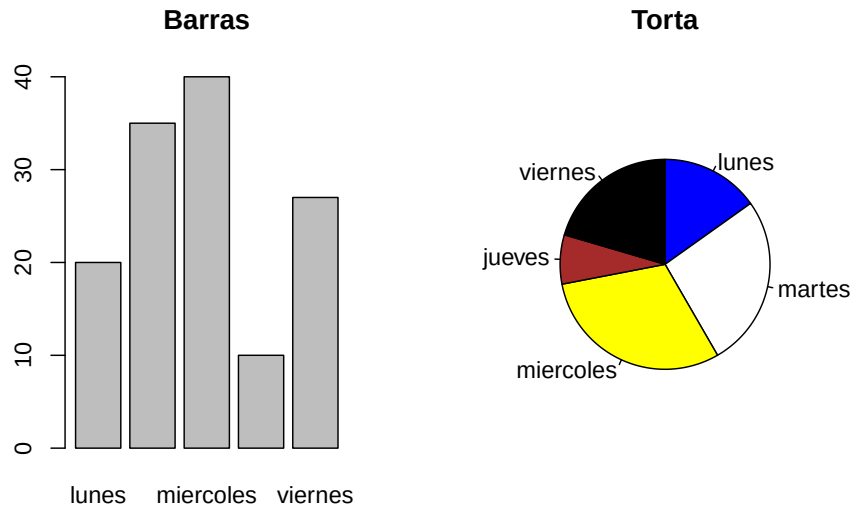
Por medio de la función par() podemos ajustar distintos parámetros de nuestra gráfica, así como presentar más de una gráfica en el mismo espacio, modificar colores, texto y muchas otras características.

?par

```
opar <- par() #Antes de empezar a utilizar par, guardamos la configuración por defecto

dev.off() #O podemos cerrar los plots creados anteriormente para reestablecer la confi
- null device
-           1
dat_dias <- data.frame(
  dias = c("lunes", "martes", "miercoles", "jueves", "viernes"),
  ganancias = c(20,35,40,10,27))
```

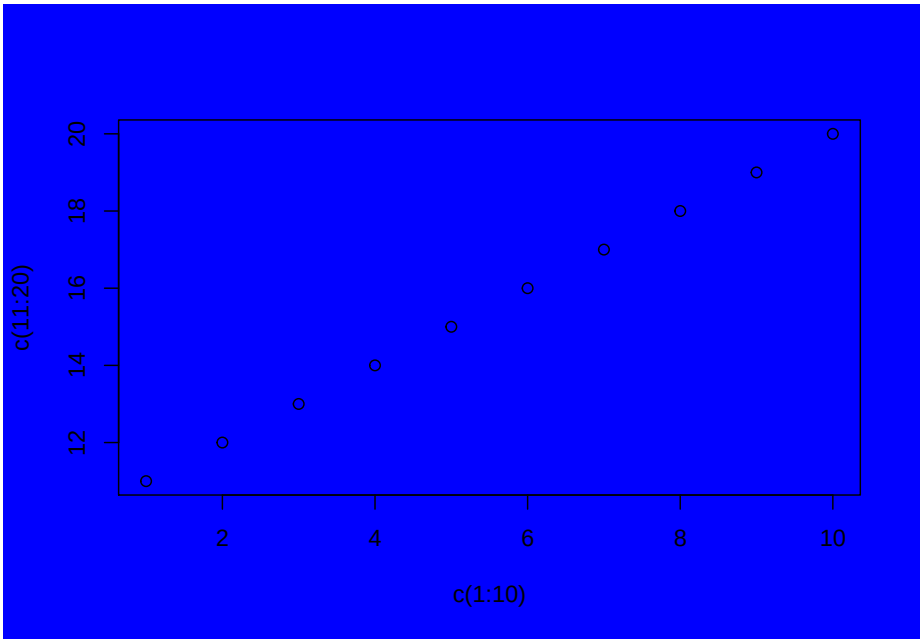
Visualizaremos dos gráficas en el mismo espacio mediante el argumento "mfrow".

```
par(mfrow = c(1,2))
barplot(dat_dias$ganancias, main = "Barras", names.arg = dat_dias$dias)
pie(dat_dias$ganancias, labels = dat_dias$dias, main = "Torta", clockwise = T, col = c
```



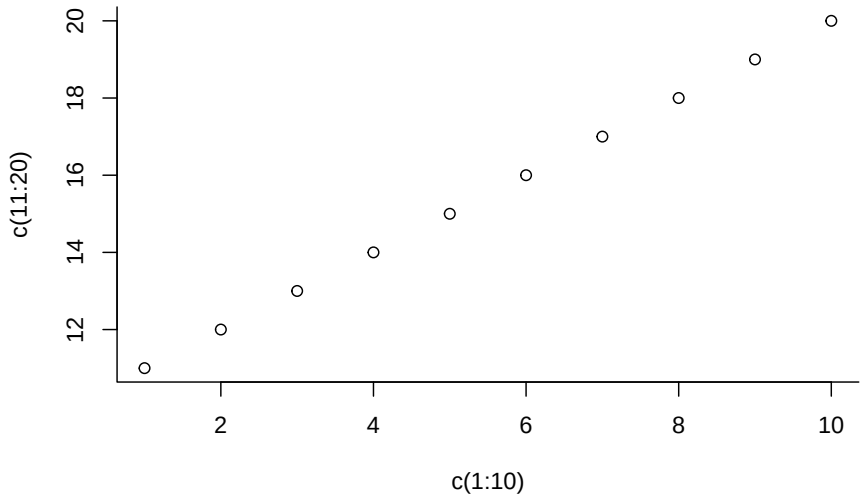
Ejercicio: Utilice el argumento "bg" para cambiar el color de fondo del siguiente plot

```
plot(x = c(1:10), y = c(11:20))
par(mfrow = c(1,1), bg = "blue")
plot(x = c(1:10), y = c(11:20))
```



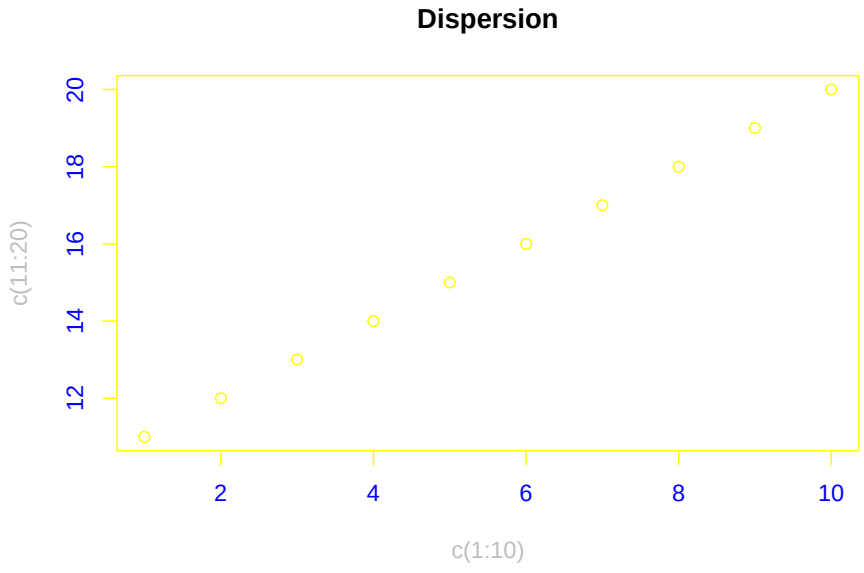
Con “bty” podemos cambiar la forma de la caja del plot:

```
par(mfrow = c(1,1), bg = "white", bty = "l") #Pruebe cambiando "l" por o,7,c,u,],n
plot(x = c(1:10), y = c(11:20))
```



Podemos agregar colores a los diferentes elementos del gráfico

```
par(mfrow = c(1,1), bg = "white", col.axis = "blue", col.lab = "gray", col.main = "bla
plot(x = c(1:10), y = c(11:20), main = "Dispersion")
```



Para cambiar la fuente del texto:

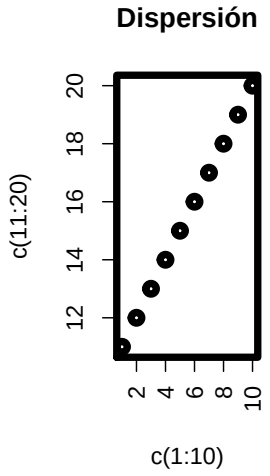
- font.axis (valor numérico)
- font.lab
- font.main
- font.sub

Ejercicio 3

1. Explore los siguientes argumentos disponibles en par(), explique su función y utilícelas en una gráfica

- las
- lty
- lwd
- fig
- mai
- pin
- new

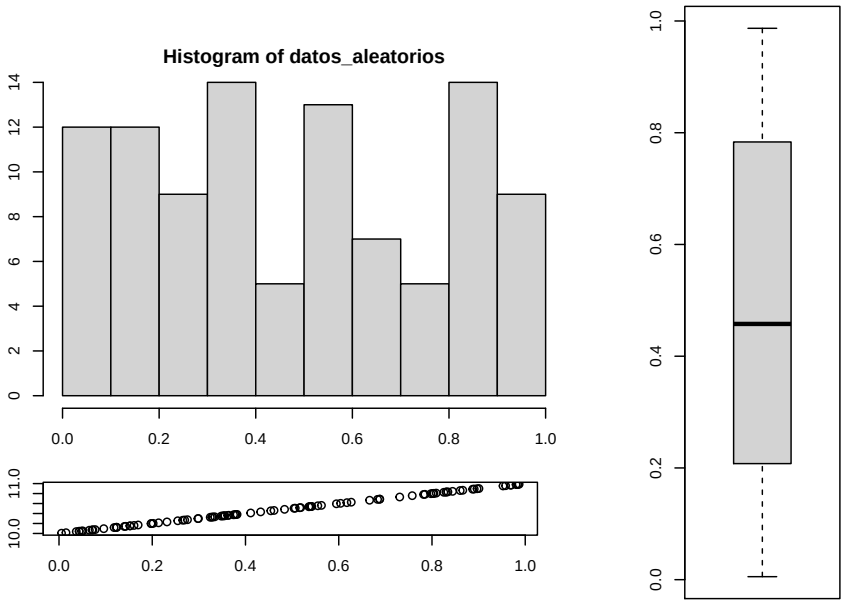
```
par(mfrow = c(1,1), bg = "white", las = 3, lty = 3, lwd = 5, pin = c(1,2))
plot(x = c(1:10), y = c(11:20), main = "Dispersión")
```



3.4.1 Gráficos compuestos

Usando algunas de las funciones anteriores, podemos ajustar varias figuras en un plot

```
datos_aleatorios <- runif(100)

par(cex=0.7, mai=c(0.1,0.1,0.2,0.1), fig=c(0.1,0.7,0.3,0.9))
hist(datos_aleatorios) # Configuramos el tamaño de nuestro histograma y lo graficamos

par(fig=c(0.8,1,0,1), new=TRUE) # Configuramos la posición de un boxplot y agregamos n
boxplot(datos_aleatorios)

par(fig=c(0.1,0.67,0.1,0.25), new=TRUE)
plot(x = datos_aleatorios, y = datos_aleatorios + 10) # Y añadimos un gráfico de dispe
```



Chapter 4

Leyendo y manejando datos en R

Estableciendo el directorio de trabajo para poder leer datos desde un archivo

Antes de empezar, debemos crear nuestro sitio de trabajo mediante la función setwd(), en el cual debemos poner entre comillas la dirección de la carpeta en la que tenemos guardado este script y el archivo con los datos del proyecto Santander BIO "Santander_BIO_data.csv"

```
setwd("~/r tic/sesion 2") # Note que empieza con ~ ya que esto nos dirige automáticamente a la ca
```

En RStudio se recomienda utilizar: Session -> Set Working Directory

---

4.1 Data frame

Al igual que una matriz, los data frames son un conjunto de elementos ordenados en dos dimensiones. Filas (observaciones) y columnas (variables). Sin embargo, en un data frame se pueden almacenar diferentes tipos de datos en diferentes columnas.

Una de las formas más comunes en la que vamos a encontrar datos, es en formato Excel o .csv. Estos son datos tabulados que constan de dos dimensiones, filas y columnas, que pueden almacenar datos numéricos y caracteres.

4.1.1 Leer archivo desde carpeta de trabajo

Para leer con facilidad en R los archivos .csv (coma seperated values), es posible importarlos de la siguiente forma:

```
dat <-read.csv("SantanderBIO.csv") # Note que los nombres de los documentos van entre
```

Ya hemos cargado nuestros datos en un data frame, el cual se trata de información sobre inventarios de la biodiversidad en el marco del proyecto Santander BIO. Nuestro data frame cuenta con información taxonómica (Nombre de la especie, familia, reino, etc.) e información sobre los sitios y la altura a la que se encuentran estas especies.

4.1.2 Evaluar la estructura del data frame

Ahora vamos a explorar nuestros datos aprendiendo paso a paso algunas funciones importantes

```
dim(dat) # Dimensiones de los datos: 1255 filas (observaciones) y 11 columnas (variabl
- [1] 1255   10
colnames(dat) #Nombre de las columnas
-  [1] "species"      "locality"     "municipality" "kingdom"      "phylum"       "clas
-  [7] "order"        "family"       "taxonRank"    "elevation"
#str(dat) Estructura de los datos
```

Tambien es posible preguntar por otras condiciones en nuestro data frame que no sean necesariamente numéricas o que su resultado no sea una posición. Por ejemplo, un resultado tipo lógico, mediante la función is.numeric/character()

```
is.numeric(dat$species)
- [1] FALSE
is.character(dat$species)
- [1] TRUE
is.numeric(dat$elevation)
- [1] TRUE
is.character(dat$elevation)
- [1] FALSE
```

4.2 Acceder variables dentro del data frame

Para seleccionar una variable o columna del data frame utilizaremos el caracter "$" seguido del nombre asignado asignado a nuestros datos, después colocaremos el nombre de la variable deseada o podremos desplegar todas las opciones con tabulador

```
ele <- dat$elevation
head(ele)#La función head devuelve por defecto los seis primeros valores de un vector
- [1] 158 158 159 158 158 158
tail(ele)#En contraste, la función tail devuelve los seis últimos valores
- [1] 2575 2525 2402 2390 2525 2525
min(dat$elevation) #Con min obtenemos el valor mínimo de una variable deseada (en este caso altu
- [1] 74
max(dat$elevation) #Y con max el valor máximo.
- [1] 3603
```

Ejercicio: Explorar con las funciones mean() y sd().

Sabemos que hay distintos lugares en donde se colecto la información y que cada lugar puede aparecer más de una vez porque es probable que cuente con más de una especie colectada, también es probable que una especie se repita debido a que se puede encontrar en más de un lugar. Entonces, vamos a averiguar el número de especies registradas en total.

```
head(dat$species)
- [1] "Aciotis circaeifolia"   "Adiantum obliquum"      "Adiantum pulverulentum"
- [4] "Adiantum wilsonii"      "Aechmea longicuspis"    "Aegiphila cordata"
length(dat$species)# Con la longitud del vector podemos obtener el número total de registros
- [1] 1255
species <- unique(dat$species) #Con la función "unique" obtendremos los nombres de las especies s

length(species) #Hay 1000 especies en total
- [1] 1000
```

Ahora averiguaremos cuantas especie únicas puede haber en un municipio aleatorio. Para esto, vamos a crear un vector que contenga solo los datos del municipio deseado, aunque esto no es obligatorio, puede ser una forma ordenada de escribir nuestro código.

```
unique(dat$municipio) #Nombres de los municipios de colecta
- NULL
m_Cimitarra <- dat[dat$municipality=="Cimitarra",] #Usamos los corchetes [] para seleccionar las

sp_unicas_cimitarra <- unique(m_Cimitarra$species)
```

```
length(sp_unicas_cimitarra) #En el municipio de Cimitarra se registraron 397 especies
- [1] 397
```

Ejercicio 1

1. Extraer la siguiente información del municipio de Cimitarra:

- Elevacion máxima y mínima de los registros en SantanderBIO
- Elevación máxima y mínima de los registros de aves
- Número de localidades únicas

2. Extraer la altura máxima y mínima de la que se tiene registro en cada reino (Plantae, Animalia) por localidad (Vereda El Aguila, Vereda Guineales, Vereda Locacion, Vereda Riveras de San Juan)
3. Calcular el promedio de la altura por municipio

4.3 Acceder observaciones mas complejos

Podemos especificar más de una condición para crear un set de datos más pequeño

```
or_operador <- dat[dat$municipality=="Cimitarra" | dat$municipality=="Santa Barbara",]

or_operador <- dat[dat$municipality=="Cimitarra" | dat$municipality=="Santa",] # En es

#Si queremos elegir todas las observaciones menos una en particular, usamos el operado

plantas_no <- dat[!dat$kingdom=="Plantae",] # Seleccionamos solo observaciones con reg
```

Para seleccionar varias condiciones usamos “&”

```
alt_0_100 <- dat[dat$elevation > 0 & dat$elevation < 100,] # Seleccionar alturas entre

length(alt_0_100[,1]) # Existen 16 observaciones registradas en el rango de altura de
- [1] 16
```

¿Cómo podemos verificar que obtuvimos los datos deseados?

```
min(alt_0_100$elevation)
- [1] 74
max(alt_0_100$elevation)
- [1] 97
```

¿Cómo podemos averiguar el número de observaciones de anfibios por municipio?

```
amphi_cimitarra <- dat[dat$municipality=="Cimitarra" & dat$class=="Amphibia",]
length(amphi_cimitarra[,1])
- [1] 33
```

Ejericio: Calcular también para El carmen de Chucurí y Santa Barbara.

4.4 Agregar columnas al data frame para apoyar el análisis

En el ejercicio anterior observamos una diferencia en el número de registros de anfibios en cada municipio. Vamos a ver el número de registros totales en cada municipio

```
dat$conteo <- 1 #Podemos añadir una nueva columna que tenga el número 1 en cada observación y así

sum(dat[dat$municipality=="Cimitarra", "conteo"]) #En Cimitarra existen 551 registros
- [1] 551
sum(dat[dat$municipality=="El Carmen de Chucuri", "conteo"]) #En el Carmen de Chucuri existen 372
- [1] 372
sum(dat[dat$municipality=="Santa Barbara", "conteo"]) #En Santa Barbara existen 332 registros
- [1] 332
```

Finalmente, podemos eliminar la columna de conteo

```
dat <- dat[,-11]
```

4.4.1 crear rangos en una variable continua:

¿Cuál es el rango de altura en la que registraron los anfibios del ejercicio anterior?

Para esto usaremos la función cut()

?cut

```
dat$cut <- cut(dat$elevation, 4)

dat$cut <- cut(dat$elevation, 4, labels = c("a", "b", "c", "d"))
```

Explique que sucede en las líneas de código anterior, luego defina 4 rangos de altura que empiece desde 0 y termine en 4000 metros y colóqueles una etiqueta. Finalmente responda la pregunta anterior.

```
dat$cut <- cut(dat$elevation, breaks = c(0,1000,2000,3000,4000), labels = c("Cero a mi

levels(dat$cut)
- [1] "Cero a mil"            "Mil a dos mil"         "Dos mil a tres mil"
- [4] "Tres mil a cuatro mil"
```

Ejercicio 2

1. Calcular el número de registros por cada rango de altura

- Elevacion máxima y mínima de los registros en SantanderBIO
- Elevación máxima y mínima de los registros de aves
- Número de localidades únicas

2. Calcular el número de especies por cada rango de altura
3. Calcular el número de especies de plantas por cada rango de altura

4.5 Indexar y reemplazar

En R podemos preguntar por una condición en específico y el resultado será la posición de este en el data frame. Esto se hace mediante la función which().

?which

Podemos preguntar por ejemplo por los registros que se encuentren a una altura menor de 80 metros

```
which(dat$elevation<80) #Existen 6 registros que se encuentran a una altura menor de 8
- [1] 1144 1147 1165 1173 1179 1180
dat[1144,] #El resultado de la fila 1144 corresponde la especie Curimata mivartii regi
-                species            locality municipality  kingdom   phylum          cl
- 1144 Curimata mivartii Riveras de San Juan    Cimitarra Animalia Chordata Actinoptery
-              order      family taxonRank elevation        cut
- 1144 Characiformes Curimatidae   SPECIES        74 Cero a mil
```

Si guardamos las posiciones del resultado anterior, podemos obtener todos los resultados en un nuevo data frame de manera rápida:

```
posiciones <- which(dat$elevation<80)

nuevo_dat <- dat[posiciones,] #Podemos escribir el nombre del vector que contiene los valores de
nuevo_dat
-                             species            locality municipality  kingdom   phylum
- 1144              Curimata mivartii Riveras de San Juan    Cimitarra Animalia Chordata
- 1147      Dasyloricaria filamentosa Riveras de San Juan    Cimitarra Animalia Chordata
- 1165        Megaleporinus muyscorum Riveras de San Juan    Cimitarra Animalia Chordata
- 1173 Pimelodella floridablancaensis Riveras de San Juan    Cimitarra Animalia Chordata
- 1179      Rhinoclemmys melanosterna Riveras de San Juan    Cimitarra Animalia Chordata
- 1180       Rineloricaria magdalenae Riveras de San Juan    Cimitarra Animalia Chordata
-               class         order        family taxonRank elevation        cut
- 1144 Actinopterygii Characiformes   Curimatidae   SPECIES        74 Cero a mil
- 1147 Actinopterygii  Siluriformes  Loricariidae   SPECIES        74 Cero a mil
- 1165 Actinopterygii Characiformes   Anostomidae   SPECIES        74 Cero a mil
- 1173 Actinopterygii  Siluriformes Heptapteridae   SPECIES        74 Cero a mil
- 1179       Reptilia    Testudines   Geoemydidae   SPECIES        74 Cero a mil
- 1180 Actinopterygii  Siluriformes  Loricariidae   SPECIES        74 Cero a mil
```

Ademas de extraer información de un data frame, podemos editarlo de forma sencilla. Supongamos que encontramos un error asociado a un dato en específico, como el nombre de una familia. En nuestro caso hipotético, supongamos que nuestros datos tienen un error de digitación, y el nombre de la familia Acanthaceae está mal escrito, por lo que necesitamos modificarlo.

```
dat2 <- dat #Copiamos el data frame en un nuevo vector

nombre_incorrecto <- which(dat2$family=="Acanthaceae") #Guardamos la posición de las filas en las

dat2[nombre_incorrecto, 8] <- "Nombre correcto" #Cambiamos el nombre por el que deseamos
```

Otra alternativa: Utilice la siguiente función para cambiar el nombre de la familia Viperidae a “Notviperidae” y verifique que el cambio se hizo correctamente

?replace

```
dat2$family <- replace(dat2$family ,c(dat2$family=="Viperidae"), "Notviperidae")
verificacion <- dat2[dat2$family=="Notviperidae",] #Y finalmente verificamos que el cambio fue re
```

Ejercicio 3

1. Realizar un gráfico de barras del número de registros totales por cada municipio

2. Realizar un gráfico del número de registros de insectos en cada localidad
3. Realizar un boxplot de la elevación por clases
4. Realizar un histograma de la elevación

Chapter 5

Manejando datos utilizando tidyverse

En esta sesión vamos a trabajar los datos de la sesión anterior de una manera diferente. Para esto vamos a cargar el conjunto de paquetes Tidyverse.

install.packages("tidyverse")

Los paquetes que ofrece Tidyverse están orientados a facilitar la manipulación, importación, exploración y visualización de datos, permitiendo que el proceso sea eficiente y que los scripts puedan ser reproducibles entre usuarios. En esta sesión vamos a trabajar principalmente con el paquete dplyr y algunas generalidades de otros paquetes para trabajar con datos tidy.

Los "datos tidy" se caracterizan por lo siguiente:

- Cada columna es una variable
- Cada fila es una observación
- Cada celda es un valor único

Estas 3 propiedades las veremos con frecuencia a lo largo de esta sesión.

5.1 Readr

El paquete Readr es una alternativa para leer datos rectangulares como un .csv. Las principales ventajas de usar readr para leer un .csv son la velocidad con la que se importan los datos y el "parse" realizado sobre estos. Esto último quiere

decir que la función analizará automaticamente el tipo de dato que está siendo importando y no producirá cambios inesperados. Con frecuencia, la función base de R puede convertir vectores de caracteres a factores. Además, Readr lee automáticamente formatos de fechas.

```
dat <- read_csv("SantanderBIO.csv") #Como resultado devuelve el número de filas y colu
- Parsed with column specification:
- cols(
-   species = col_character(),
-   locality = col_character(),
-   municipality = col_character(),
-   kingdom = col_character(),
-   phylum = col_character(),
-   class = col_character(),
-   order = col_character(),
-   family = col_character(),
-   taxonRank = col_character(),
-   elevation = col_double()
- )
```

Recordemos que nuestro data frame contiene información sobre inventarios de la biodiversidad en el marco del proyecto Santander BIO. Nuestro data frame cuenta con información taxonómica (Nombre de la especie, familia, reino, etc) e información sobre los sitios y la altura a la que se encuentran estas especies.

Podemos cambiar el tipo de dato al importarlo, por ejemplo convertir algunas variables en factores

```
dat <- read_csv("SantanderBIO.csv", col_types =
                  list(
                    species = col_character(),
                    locality = col_factor(),
                    municipality = col_factor(),
                    kingdom = col_factor(),
                    phylum = col_character(),
                    class = col_character(),
                    order = col_character(),
                    family = col_character(),
                    taxonRank = col_character(),
                    elevation = col_double()
                  )
)
```

5.2 Tibble

Tibble es una forma moderna de data frame que ha probado ser más eficiente. Los tibbles no cambian los nombres de las variables o el tipo de dato.

```
class(dat) #Si miramos de que clase son los objetos cargados con el paquete Readr, nos damos cuen
- [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
dat_conv <-read.csv("SantanderBIO.csv")

class(dat_conv)
- [1] "data.frame"
dat_tibble <- as_tibble(dat_conv)

class(dat_tibble)
- [1] "tbl_df"     "tbl"        "data.frame"
```

Hay dos diferencias entre un tibble y un data frame convencional: el método de impresión o print() y la forma de obtener subconjuntos.

```
dat_tibble #Tibble nos muestra hasta 10 observaciones para mantener la consola menos saturada. En
- # A tibble: 1,255 x 10
-    species     locality  municipality kingdom  phylum  class  order  family  taxonRank elevatio
-    <chr>       <chr>     <chr>        <chr>    <chr>   <chr>  <chr>  <chr>   <chr>          <int
-  1 Aciotis ci~ El Aguila Cimitarra    Plantae  Trache~ Magno~ Myrta~ Melast~ SPECIES          15
-  2 Adiantum o~ El Aguila Cimitarra    Plantae  Trache~ Polyp~ Polyp~ Pterid~ SPECIES          15
-  3 Adiantum p~ El Aguila Cimitarra    Plantae  Trache~ Polyp~ Polyp~ Pterid~ SPECIES          15
-  4 Adiantum w~ El Aguila Cimitarra    Plantae  Trache~ Polyp~ Polyp~ Pterid~ SPECIES          15
-  5 Aechmea lo~ El Aguila Cimitarra    Plantae  Trache~ Lilio~ Poales Bromel~ SPECIES          15
-  6 Aegiphila ~ El Aguila Cimitarra    Plantae  Trache~ Magno~ Lamia~ Lamiac~ SPECIES          15
-  7 Aegiphila ~ El Aguila Cimitarra    Plantae  Trache~ Magno~ Lamia~ Lamiac~ SPECIES          15
-  8 Allobates ~ El Aguila Cimitarra    Animalia Chorda~ Amphi~ Anura  Aromob~ SPECIES          15
-  9 Amaioua gu~ El Aguila Cimitarra    Plantae  Trache~ Magno~ Genti~ Rubiac~ SPECIES          15
- 10 Andinoacar~ El Aguila Cimitarra    Animalia Chorda~ Actin~ Perci~ Cichli~ SPECIES          12
- # ... with 1,245 more rows
dat_tibble %>% print(n = 20)
- # A tibble: 1,255 x 10
-    species     locality  municipality kingdom  phylum  class  order  family  taxonRank elevatio
-    <chr>       <chr>     <chr>        <chr>    <chr>   <chr>  <chr>  <chr>   <chr>          <int
-  1 Aciotis ci~ El Aguila Cimitarra    Plantae  Trache~ Magno~ Myrta~ Melast~ SPECIES          15
-  2 Adiantum o~ El Aguila Cimitarra    Plantae  Trache~ Polyp~ Polyp~ Pterid~ SPECIES          15
-  3 Adiantum p~ El Aguila Cimitarra    Plantae  Trache~ Polyp~ Polyp~ Pterid~ SPECIES          15
-  4 Adiantum w~ El Aguila Cimitarra    Plantae  Trache~ Polyp~ Polyp~ Pterid~ SPECIES          15
-  5 Aechmea lo~ El Aguila Cimitarra    Plantae  Trache~ Lilio~ Poales Bromel~ SPECIES          15
-  6 Aegiphila ~ El Aguila Cimitarra    Plantae  Trache~ Magno~ Lamia~ Lamiac~ SPECIES          15
-  7 Aegiphila ~ El Aguila Cimitarra    Plantae  Trache~ Magno~ Lamia~ Lamiac~ SPECIES          15
```

```
-  8 Allobates ~ El Aguila Cimitarra    Animalia Chorda~ Amphi~ Anura  Aromob~ SPECIES
-  9 Amaioua gu~ El Aguila Cimitarra    Plantae  Trache~ Magno~ Genti~ Rubiac~ SPECIES
- 10 Andinoacar~ El Aguila Cimitarra    Animalia Chorda~ Actin~ Perci~ Cichli~ SPECIES
- 11 Annona gla~ El Aguila Cimitarra    Plantae  Trache~ Magno~ Magno~ Annona~ SPECIES
- 12 Annona muc~ El Aguila Cimitarra    Plantae  Trache~ Magno~ Magno~ Annona~ SPECIES
- 13 Anolis aur~ El Aguila Cimitarra    Animalia Chorda~ Repti~ Squam~ Dactyl~ SPECIES
- 14 Anolis tro~ El Aguila Cimitarra    Animalia Chorda~ Repti~ Squam~ Dactyl~ SPECIES
- 15 Apeiba tib~ El Aguila Cimitarra    Plantae  Trache~ Magno~ Malva~ Malvac~ SPECIES
- 16 Apteronotu~ El Aguila Cimitarra    Animalia Chorda~ Actin~ Gymno~ Aptero~ SPECIES
- 17 Astyanax f~ El Aguila Cimitarra    Animalia Chorda~ Actin~ Chara~ Charac~ SPECIES
- 18 Bactris pi~ El Aguila Cimitarra    Plantae  Trache~ Lilio~ Areca~ Arecac~ SPECIES
- 19 Basiliscus~ El Aguila Cimitarra    Animalia Chorda~ Repti~ Squam~ Coryto~ SPECIES
- 20 Boana pugn~ El Aguila Cimitarra    Animalia Chorda~ Amphi~ Anura  Hylidae SPECIES
- # ... with 1,235 more rows
```

Extraer datos de un tibble se hace de una manera más estricta, ya que estos no realizan coincidencia parcial. Necesitará escribir el nombre completo de la variable.

```
dat_conv$lo[1:5] #De forma convencional, al escribir el nombre incompleto de una varia
- [1] "El Aguila" "El Aguila" "El Aguila" "El Aguila" "El Aguila"
dat_tibble$lo #Mientras que en tibble resultará en un error
- Warning: Unknown or uninitialised column: `lo`.
- NULL
dat$lo #Lo mismo sucederá con Readr
- Warning: Unknown or uninitialised column: `lo`.
- NULL
```

Finalmente, usted puede crear tibbles fila por fila o por columnas

```
tibble(x = 1:5, y = 1, z = x ^ 2 + y) #Por columnas
- # A tibble: 5 x 3
-       x     y     z
-   <int> <dbl> <dbl>
- 1     1     1     2
- 2     2     1     5
- 3     3     1    10
- 4     4     1    17
- 5     5     1    26
tribble(
  ~x, ~y,  ~z,
  "a", 2,  3.6,
  "b", 1,  8.5
)#Por fila
- # A tibble: 2 x 3
```

```
-   x         y     z
-   <chr> <dbl> <dbl>
- 1 a         2   3.6
- 2 b         1   8.5
```

5.3 Dplyr

```
dat <- read_csv("SantanderBIO.csv")
```

```
## Parsed with column specification:
## cols(
##   species = col_character(),
##   locality = col_character(),
##   municipality = col_character(),
##   kingdom = col_character(),
##   phylum = col_character(),
##   class = col_character(),
##   order = col_character(),
##   family = col_character(),
##   taxonRank = col_character(),
##   elevation = col_double()
## )
```

Dplyr es uno de los paquetes más utiles para la manipulación de datos. Dentro de sus funciones más utiles se encuentran:

mutate() Añade nuevas variables en función de variables existentes select() Selecciona variables de acuerdo a su nombre filter() Selcciona observaciones de acuerdo a sus valores summarise() Resume cada grupo en menos filas arrange() Cambia el orden de las observaciones

La forma simple para seleccionar una variable en un data frame común es mediante el símbolo $

dat$species

5.3.1 Select

Mediante el paquete dplyr podemos seleccionar variables con la función select()
?select()

```
select(dat, species) #Seleccionamos el data frame y seguido una variable
- # A tibble: 1,255 x 1
-    species
-    <chr>
-  1 Aciotis circaeifolia
-  2 Adiantum obliquum
-  3 Adiantum pulverulentum
-  4 Adiantum wilsonii
-  5 Aechmea longicuspis
-  6 Aegiphila cordata
-  7 Aegiphila panamensis
-  8 Allobates niputidea
-  9 Amaioua guianensis
- 10 Andinoacara latifrons
- # ... with 1,245 more rows
select(dat, species, locality, municipality) #O más de una variable
- # A tibble: 1,255 x 3
-    species                locality  municipality
-    <chr>                  <chr>     <chr>
-  1 Aciotis circaeifolia   El Aguila Cimitarra
-  2 Adiantum obliquum      El Aguila Cimitarra
-  3 Adiantum pulverulentum El Aguila Cimitarra
-  4 Adiantum wilsonii      El Aguila Cimitarra
-  5 Aechmea longicuspis    El Aguila Cimitarra
-  6 Aegiphila cordata      El Aguila Cimitarra
-  7 Aegiphila panamensis   El Aguila Cimitarra
-  8 Allobates niputidea    El Aguila Cimitarra
-  9 Amaioua guianensis     El Aguila Cimitarra
- 10 Andinoacara latifrons  El Aguila Cimitarra
- # ... with 1,245 more rows
```

Podemos encadenar varias funciones mediante "pipes" o %>% (se puede generar mediante el atajo control+shift+m). Esto es muy útil ya que nos permite realizar tareas con menos línea de código

Empezamos cargando primero los datos por fuera de cualquier función, seguido de un pipe

```
dat %>% select(species)
- # A tibble: 1,255 x 1
-    species
```

```
-    <chr>
-  1 Aciotis circaeifolia
-  2 Adiantum obliquum
-  3 Adiantum pulverulentum
-  4 Adiantum wilsonii
-  5 Aechmea longicuspis
-  6 Aegiphila cordata
-  7 Aegiphila panamensis
-  8 Allobates niputidea
-  9 Amaioua guianensis
- 10 Andinoacara latifrons
- # ... with 1,245 more rows
```

Ya que tenemos nuestra variable seleccionada, podemos encadenar funciones que trabajen sobre estos datos. Si la función no requiere parametros adicionales, la función se escribe en su forma básica: funcion()

```
dat %>% select(species) %>% unique() #Nos muestra los valores únicos de esta variable
- # A tibble: 1,000 x 1
-    species
-    <chr>
-  1 Aciotis circaeifolia
-  2 Adiantum obliquum
-  3 Adiantum pulverulentum
-  4 Adiantum wilsonii
-  5 Aechmea longicuspis
-  6 Aegiphila cordata
-  7 Aegiphila panamensis
-  8 Allobates niputidea
-  9 Amaioua guianensis
- 10 Andinoacara latifrons
- # ... with 990 more rows
```

La estructura básica de este proceso es escoger nuestro set de datos, filtrar y seleccionar las variables que necesitamos y aplicar una función. La complejidad del código dependerá del resultado deseado. A continuación vamos a realizar una función simple en un conjunto de datos específico de nuestro set de datos.

5.3.2 Filter

?filter()

```
dat %>% filter(municipality == "El Carmen de Chucuri") #Hasta este paso filtramos nues
- # A tibble: 372 x 10
-    species     locality  municipality   kingdom phylum class  order  family  taxonRa
-    <chr>       <chr>     <chr>          <chr>   <chr>  <chr>  <chr>  <chr>   <chr>
-  1 Aciotis ac~ La Belle~ El Carmen de ~ Plantae Trach~ Magno~ Myrta~ Melast~ SPECIES
-  2 Aciotis pu~ La Belle~ El Carmen de ~ Plantae Trach~ Magno~ Myrta~ Melast~ SPECIES
-  3 Adelobotry~ La Belle~ El Carmen de ~ Plantae Trach~ Magno~ Myrta~ Melast~ SPECIES
-  4 Aechmea ti~ La Belle~ El Carmen de ~ Plantae Trach~ Lilio~ Poales Bromel~ SPECIES
-  5 Aerenea im~ La Belle~ El Carmen de ~ Animal~ Arthr~ Insec~ Coleo~ Ceramb~ SPECIES
-  6 Alchornea ~ La Belle~ El Carmen de ~ Plantae Trach~ Magno~ Malpi~ Euphor~ SPECIES
-  7 Allomaieta~ La Belle~ El Carmen de ~ Plantae Trach~ Magno~ Myrta~ Melast~ SPECIES
-  8 Allomarkgr~ La Belle~ El Carmen de ~ Plantae Trach~ Magno~ Genti~ Apocyn~ SPECIES
-  9 Allophylus~ La Belle~ El Carmen de ~ Plantae Trach~ Magno~ Sapin~ Sapind~ SPECIES
- 10 Amazilia a~ La Belle~ El Carmen de ~ Animal~ Chord~ Aves   Apodi~ Trochi~ SPECIES
- # ... with 362 more rows
dat %>% filter(municipality == "El Carmen de Chucuri") %>% select(species) #Seleccionar
- # A tibble: 372 x 1
-    species
-    <chr>
-  1 Aciotis acuminifolia
-  2 Aciotis purpurascens
-  3 Adelobotrys adscendens
-  4 Aechmea tillandsioides
-  5 Aerenea impetiginosa
-  6 Alchornea grandiflora
-  7 Allomaieta zenufanasana
-  8 Allomarkgrafia plumeriiflora
-  9 Allophylus excelsus
- 10 Amazilia amabilis
- # ... with 362 more rows
dat %>% filter(municipality == "El Carmen de Chucuri") %>% select(species) %>% unique()
- # A tibble: 357 x 1
-    species
-    <chr>
-  1 Aciotis acuminifolia
-  2 Aciotis purpurascens
-  3 Adelobotrys adscendens
-  4 Aechmea tillandsioides
-  5 Aerenea impetiginosa
-  6 Alchornea grandiflora
-  7 Allomaieta zenufanasana
-  8 Allomarkgrafia plumeriiflora
-  9 Allophylus excelsus
- 10 Amazilia amabilis
- # ... with 347 more rows
```

```
El_Carmen <- dat %>%
  filter(municipality == "El Carmen de Chucuri") %>%
  select(species) %>%
  unique() #Esto puede ser guardado en un vector y separado de una forma mas elegante

length(El_Carmen$species) #Para El Carmen de Chucurí se registraron 357 especies
- [1] 357
```

¿Cuántas clases hay en ese mismo municipio?

```
class_carmen <- dat %>%
  filter(municipality == "El Carmen de Chucuri") %>%
  select(class) %>%
  unique()

length(class_carmen$class) #En El Carmen de Chucurí se registraron 9 clases
- [1] 9
```

5.3.3 Summarise

Podemos realizar resumenes estadísticos de nuestros datos y crear un nuevo data frame con los resultados usando la función summarise()

?summarise()

Vamos a promediar los valores de elevación

```
ele_mean <- dat %>%
  summarise(mean = mean(elevation), n = n()) #El argumento n nos muestra el tamaño del grupo y lo
```

Explique el resultado y lo que se muestra en cada columna

Podemos agrupar los resultados

```
conteo_clase <- dat %>%
  group_by(class) %>%
  summarise(n = n()) #Realizamos un conteo de los registros de especies en cada clase
- `summarise()` ungrouping output (override with `.groups` argument)
ele_mean_group <- dat %>%
  group_by(municipality) %>%
  summarise(mean = mean(elevation), n = n())
- `summarise()` ungrouping output (override with `.groups` argument)
```

Es posible realizar más de una operación simplemente añadiendo una "," y escribiéndola dentro de summarise. Calcule el valor promedio, mínimo y máximo de la elevación por municipio.

```
ele_mean_group <- dat %>%
  group_by(municipality) %>%
  summarise(mean = mean(elevation), min(elevation), max(elevation), n = n())
- `summarise()` ungrouping output (override with `.groups` argument)
```

Podemos etiquetar nuestros datos o incluso organizarlos dentro de un mismo data frame

```
#Creamos una etiqueta para cada clase

clase_etiqueta <- dat %>%
  group_by(class) %>%
  summarise(cur_group_id()) #Esto podra ser utilizado mas adelante con mutate() y crea
- `summarise()` ungrouping output (override with `.groups` argument)
```

Utilizamos las funciones cur_ para crear subconjuntos de datos ordenados por grupo. Los nuevos subconjuntos serán una lista de tibbles.

?context

Las funciones cur_ devuelven información sobre el grupo seleccionado en listas de tibles que puede contener una o mas variables. Esto dependerá de la función cur_ que usemos:

Mediante cur_data obtenemos el conjunto de variables asociadas a los factores de una variable deseada. El vector resultante será un data frame de dos variables, una columna del grupo (o variable) seleccionada y otra columna con una lista de las variables asociadas a cada factor:

```
clase_datos <- dat %>%
  group_by(class) %>%
  summarise(data = list(cur_data()))
- `summarise()` ungrouping output (override with `.groups` argument)
```

Ya que nuestro data frame contiene listas dentro, podemos acceder a ella mediante corchetes []:

Podemos extraer las variables mediante $

```
clase_datos$class
-  [1] "Actinopterygii" "Amphibia"       "Aves"           "Cycadopsida"    "Elasmobran
-  [6] "Insecta"        "Liliopsida"     "Lycopodiopsida" "Magnoliopsida"  "Mammalia"
- [11] "Pinopsida"      "Polypodiopsida" "Reptilia"
#clase_datos$data
```

Para acceder a los elementos de una variable, usamos corchetes dobles. Y finalmente, usamos nuevamente $ para acceder a una variable ya que estamos dentro de otro data.frame

```
#clase_datos$data[[1]]$species Seleccionamos el la columna "species" del tible de la primera clas
```

Para especificar una clase, las funciones de dplyr:

```
Aves <- clase_datos %>%
  filter(class == "Aves") %>%
  select(data)
```

cur_data elimina la variable seleccionada con los diferentes tibles que crea, si queremos mantenerla, usamos cur_data_all

```
clase_datos_completo <- dat %>%
  group_by(class) %>%
  summarise(data = list(cur_data_all()))
- `summarise()` ungrouping output (override with `.groups` argument)
```

Ejercicio 1

1. Utilizar “summarise” y “filter” para averiguar cuál es la familia con mayor registro de especies
2. Averiguar cuál es la elevacion máxima en la que fue registrada una planta y un animal

---

5.3.4 Mutate

Ahora vamos a ver como se modifican columnas en dplyr mediante la función mutate().

?mutate()

Vamos a crear una nueva varible que etiquete las observaciones de a cuerdo a una variable, en este caso, dependiendo de a que clase pertenecen

```
da <- dat %>%
  group_by(class) %>%
  mutate(id = cur_group_id())
```

En mutate, las nuevas variables se crean a partir de las variables existentes.

```
new_var <- dat %>%
  select(species, elevation) %>% #Nos quedamos con 2 variables
  mutate(
    doble_elevacion = elevation * 2, #Creamos una nueva variable a partir de los datos
    doble_elevacion_logaritmo = log(doble_elevacion) #Creamos una segunda variable a p
  )
```

Tambien es posible remover o modificar variables existentes. Vamos a eliminar la variable order, y modificar la variable de elevación

```
nuevo_dat <- dat %>%
  mutate(
    order = NULL,
    elevation = elevation/2
  )
```

Podemos modificar multiples columnas usando “across” dentro de “mutate”

?across

```
across_data <- dat %>%
  mutate(across(.cols = everything(), as.factor)) #Convertimos todas las columnas en f
```

Ya que la elevación es una variable numérica, debemos evitar tenerla como factor

```
across_data <- dat %>%
  mutate(across(!elevation, as.factor)) #Con el signo de admiración antes de la variab
```

Ejercicio 2

1. Crear dos columnas nuevas, una en donde le sume 1000 a cada observacion de elevación y otra donde le reste 1000. Ademas, mantenga la variable de especies, municipio y de elevación orginal.
2. Explorar el argumento “starts_with()” para seleccionar las variables que empiezan con una secuencia de caracteres en especifico.

5.3.5 Arrange

Finalmente con arrange() podemos cambiar el orden de las observaciones o filas

?arrange

Vamos a ordenar nuestra variable numérica de elevación

```
ordenado <- dat %>% arrange(elevation) #Ordena las observaciones de menor a mayor
```

Ordenelo de mayor a menor usando desc() en la funcion arrange()

```
ordenado <- dat %>% arrange(desc(elevation))
```

Tambien es posible ordenarlo por categorias de una variable. Intente crear un data frame ordenado por su género

En nuestro set de datos, no existe una columna para el género. Vamos a intentar crearla a partir de la variable species, la cual contiene el epíteto genérico y crear una nueva variable con estos caracteres.

5.3.6 Separate()

Podemos separar characteres con separate()

?separate()

```
separado <- dat %>% separate(species, c("genus", "species"))
- Warning: Expected 2 pieces. Additional pieces discarded in 3 rows [226, 720, 950].
```

separate() se encarga de separar caracteres cuando encuentra un valor diferente a una letra o numero, como el espacio. En este caso, separará los caracteres en dos columnas llamadas “specie” y “genus”. Vemos que luego de ejecutar, obtenemos una advertencia con las posiciones de filas en las que encontraron problemas. Revisamos las posiciones en el data frame original

```
dat$species[c(226, 720, 950)] #Vemos que el epíteto de estas especies está acompañado de otra pal
- [1] "Aniba puchury-minor"      "Aniba puchury-minor"      "Peperomia albert-smithii"
separado <- dat %>% separate(species, c("specie", "genus", "otro"))
- Warning: Expected 3 pieces. Missing pieces filled with `NA` in 1252 rows [1, 2, 3, 4, 5, 6, 7,
- 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ...].
```

Si usamos NA en vez del nombre de la nueva columna omitiremos esa columna

```
# separado <- dat %>% separate(genus, c(NA, "species"))
```

O podemos especificar el comportamiento de la separación

```
separado <- dat %>%
  separate(species, c("genus", "species"), " ", extra = "merge")#Le indicamos a la fun
```

Revisamos la observacion 226

```
separado$genus[226]
- [1] "Aniba"
```

También podemos realizar esto mediante expresiones regulares que veremos mas adelante en la sesion

```
separado <- dat %>%
  separate(species, c("specie", "genus"), sep = " ")
```

¡Ahora podemos ordenar nuestras observaciones de acuerdo al genero!

```
ordenado_grupo <- separado %>% arrange(genus)
```

5.4 Aggregate

Una alternativa clásica en R a la función de summarise() en dplyr es aggregate, aunque es recomendable usar los paquetes de tidiverse, conocer la sintaxis de otras funciones es recomendable para desarrollar habilidades en la programación

Al igual que summarise, aggregate() permite calcular resúmenes estadísticos de subconjuntos de datos

?aggregate

Vamos a promediar la elevación a través de algunos grupos en nuestros datos

```
aggre_prom <- aggregate(elevation ~ class, data = dat, FUN = mean) #Aggregate tiene 3
```

Esta es la forma que utilizabamos en dplyr para realizar el mismo proceso

```
sum_prom <- dat %>%
  group_by(class) %>%
  summarise(mean = mean(elevation))
- `summarise()` ungrouping output (override with `.groups` argument)
```

Adicionalmente, podemos obtener resultados que dependan de mas de una variable. Si necesitamos obtener los valores promedio de elevación por clase y al mismo tiempo por municipio, agregamos el signo "+" de la siguiente manera:

```
aggre_prom <- aggregate(elevation ~ class+municipality, data = dat, FUN = mean) #De esta forma, c

sum_prom <- dat %>%
  group_by(class, municipality) %>%
  summarise(mean = mean(elevation)) #Esto funciona de igual forma en summarise añadiendo otra var
- `summarise()` regrouping output by 'class' (override with `.groups` argument)
```

Ejercicio 3

1. ¿Qué pasa al cambiar el orden de las variables en aggregate y en summarise? ¿Son los resultados iguales? Explique.
2. Utilice la función aggregate para calcular el número de observaciones de especies por orden utilizando la funcion length. Realice el mismo proceso usando el paquete dplyr
3. Calcule la desviacion estándar, el valor mínimo y máximo de la elevación para cada familia, phylum y municipio con aggregate y dplyr.

5.5 Stringr

El paquete stringr brinda herramientas útiles para trabajar con expresiones regulares y caracteres. Las expresiones regulares son un lenguaje que describen patrones de texto. Esto es de gran ayuda cuando necesitamos preparar y limpiar nuestros datos. Ejecute las siguientes funciones y describa lo que esta pasando.

```
x <- fruit[1:5]

str_detect(x, "aeiou")
- [1] FALSE FALSE FALSE FALSE FALSE
str_detect(x, "[aeiou]")
- [1] TRUE TRUE TRUE TRUE TRUE
str_count(x, "[aeiou]")
- [1] 2 3 4 3 3
str_subset(x, "[aeiou]")
- [1] "apple"       "apricot"     "avocado"     "banana"      "bell pepper"
str_locate(x, "[aeiou]")
-      start end
- [1,]     1   1
```

```
- [2,]     1   1
- [3,]     1   1
- [4,]     2   2
- [5,]     2   2
str_extract(x, "[aeiou]")
- [1] "a" "a" "a" "a" "e"
str_match(x, "(.)[aeiou](.)")
-      [,1]  [,2] [,3]
- [1,] NA    NA   NA
- [2,] "ric" "r"  "c"
- [3,] "voc" "v"  "c"
- [4,] "ban" "b"  "n"
- [5,] "bel" "b"  "l"
str_replace(x, "[aeiou]", "?")
- [1] "?pple"       "?pricot"     "?vocado"     "b?nana"      "b?ll pepper"
str_split(c("a,b", "c,d,e"), ",")
- [[1]]
- [1] "a" "b"
-
- [[2]]
- [1] "c" "d" "e"
```

Todas las funciones de stringr comienzan con str_, y las funciones anteriores son las 7 principales del paquete.

Aplicando estas funciones a nuestro set de datos, podemos realizar busquedas de caracteres de nuestros registros

```
which(str_detect(dat$species, "Clusia hammeliana")=="TRUE") #Buscamos la posición de u
- [1] 36
```

Sin embargo, si no estamos seguro, podemos hacer una busqueda con algunos caracteres que recordemos

```
which(str_detect(dat$species, "sia hamm")=="TRUE")
- [1] 36
```

También podemos contar el numero de registros por localidad o municipio

```
sum(str_count(dat$municipality, "Cimitarra"))
- [1] 551
```

En stringr podemos hacer busquedas mas generalizadas que no veremos en esta sesión, pero esto es un ejemplo de lo que podemos hacer conociendo algunas clases de caracteres

```
# str_subset(dat$species, "[:space:][aeiou]")
#Extraemos todas las especies en la que su epíteto específico comienza con una vocal. Observe que
```

¿Cómo podemos averiguar si hay un número en los nombres de especies?

```
#which(str_detect(dat$species, "123456789"))
#which(str_detect(dat$species, "[:digit:])) La clase de caracter para encontrar cualquier número
#str_detect(dat$elevation, "[:digit:]")
```

Explore las clases de caracteres pre-creados

- Puntuación: [:punct:]
- Letras: [:alpha:]
- Letras minúsculas: [:lower:]
- Letras mayúsculas: [:upper:]
- Dígitos: [:digit:]
- Dígitos hexadecimales: [:xdigit:]
- Letras y núMeros: [:alnum:]
- Caracteres de control: [:cntrl:]
- Letras, números y puntuación: [:graph:]
- Letras, números, puntuación y espacio en blanco: [:print:]
- Caracter de espacio: [:space:]
- Espacio y tabulador: [:blank:]

Chapter 6

Visualización de datos utilizando ggplot2

El paquete ggplot2 ofrece herramientas que ayudan a visualizar datos tidy en data frames

```
- Parsed with column specification:
- cols(
-   species = col_character(),
-   locality = col_character(),
-   municipality = col_character(),
-   kingdom = col_character(),
-   phylum = col_character(),
-   class = col_character(),
-   order = col_character(),
-   family = col_character(),
-   taxonRank = col_character(),
-   elevation = col_double()
- )
```

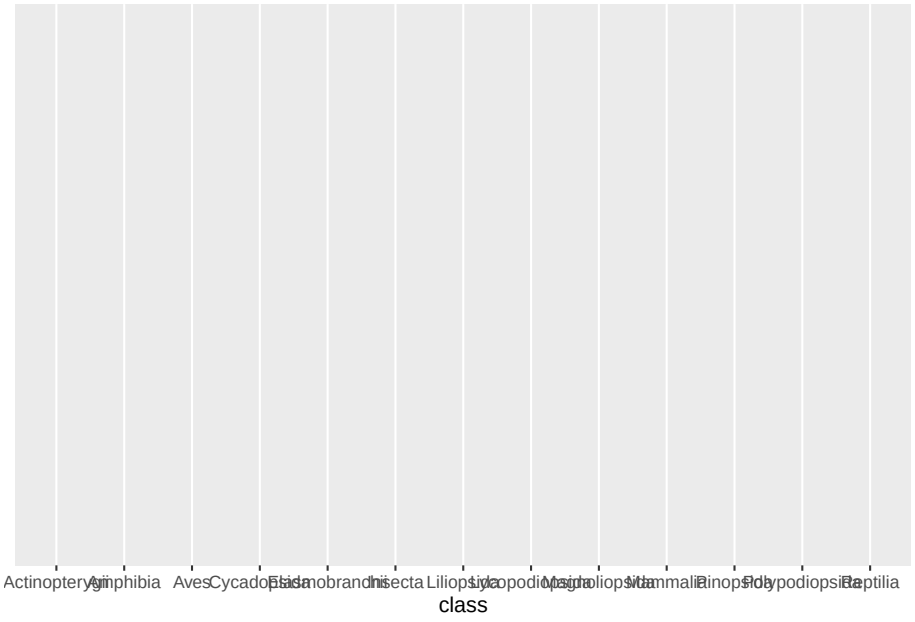
El paquete ggplot2 ofrece herramientas que ayudan a visualizar datos tidy en data frames de forma organizada y sencilla, basado en la gramática de las gráficas. En la grámatica de las gráficas, la idea es que puedas construir gráficas a partir de los mismos componentes: un set de datos; un sistema de coordenadas ("X" y "Y") y aspectos del gráfico; y formas o elementos geométricos que representan a los datos (puntos, lineas circulos, etc)

En ggplot, cada componente es un capa que se va añadiendo una tras otra usando el símbolo "+"

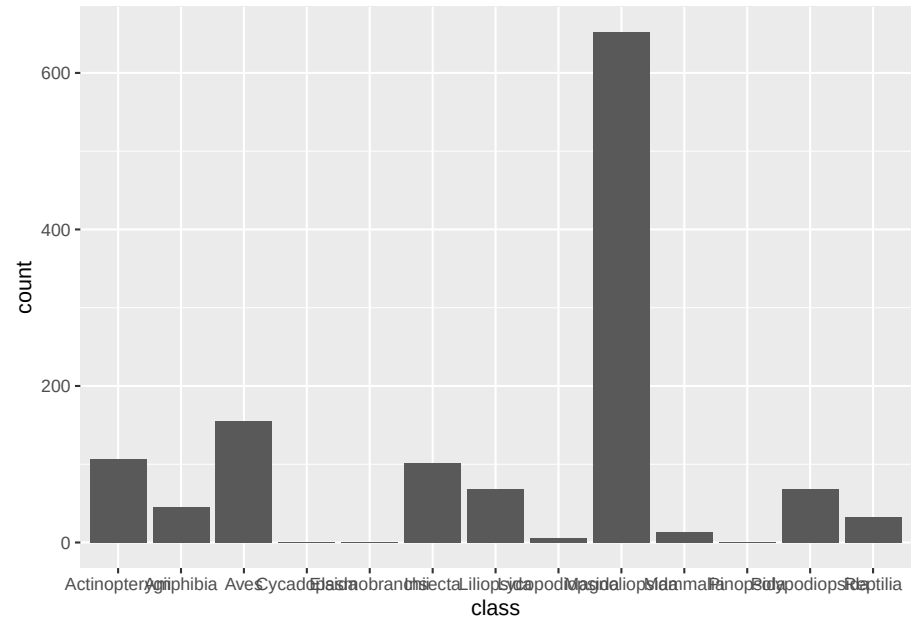
```
ggplot(data = dat) #La  primera capa de un ggplot es el conjunto de datos que proviene
```



```
ggplot(data = dat) +
  aes(x = class) #Luego se añade el sistema de coordenas o relación entre las variable
```



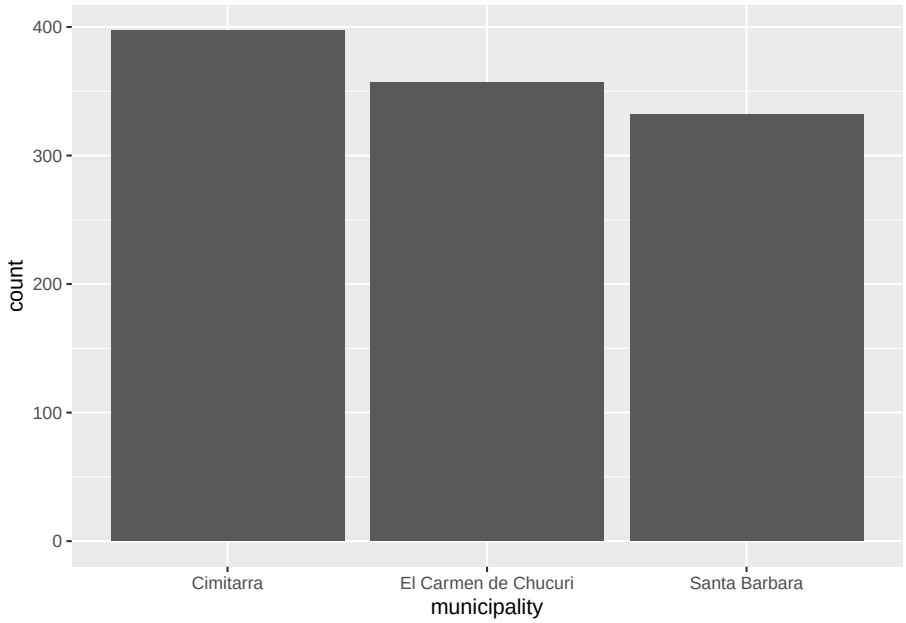

```
ggplot(data = dat) +
  aes(x = class) +
  geom_bar() #Y añadimos un geom_ para representar geometricamente nuestros datos. Hay que tener
```



Estos son los 3 componentes principales para elaborar una gráfica en ggplot. Es posible seguir añadiendo capas para mejorar nuestra grafica y personalizarla de muchas maneras, sin embargo, si no se especifican, ggplot las establecerá por defecto

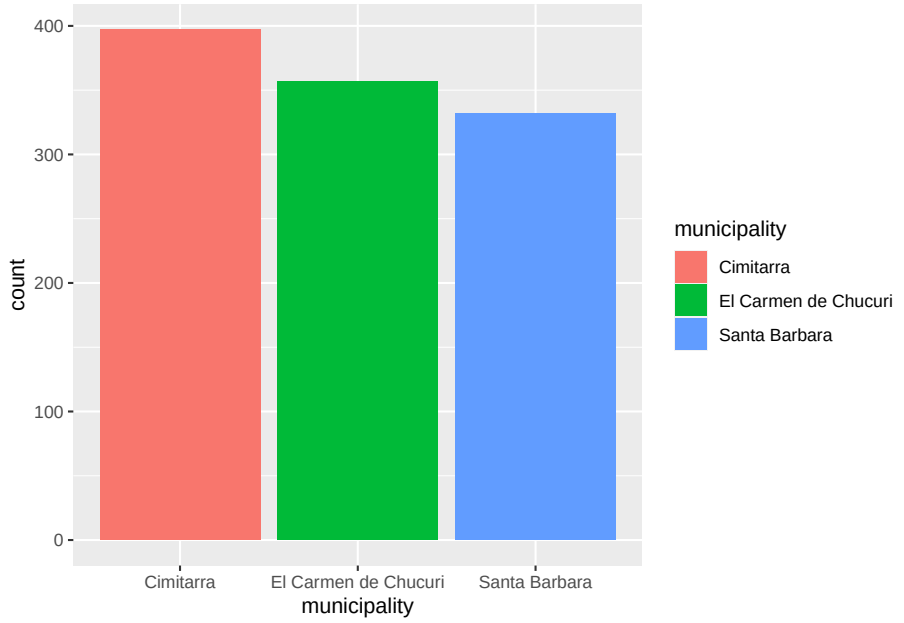
Al hacer parte del mundo de tidyverse, es posible encadenar la función de ggplot con las funciones vistas anteriormente y así crear un gráfico de forma directa. Vamos a crear una gráfico que nos muestre el número de especies únicas por municipio

```
dat %>%
  group_by(municipality) %>%
  summarise(unicos = unique(species)) %>%
  ggplot(aes(x = municipality)) +
  geom_bar()
- `summarise()` regrouping output by 'municipality' (override with `.groups` argument)
```



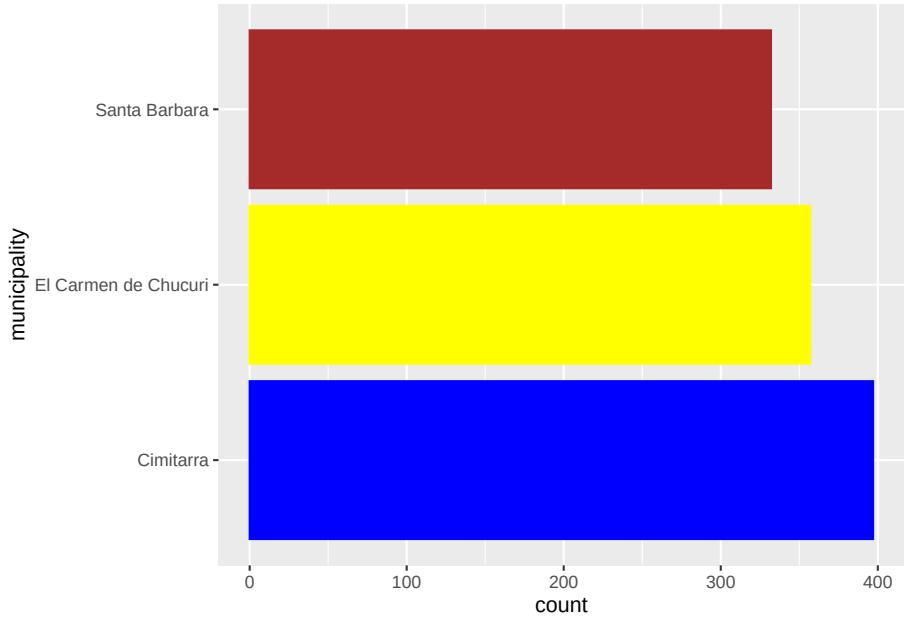
Ggplot también establece colores y leyenda por defecto a partir de los factores en el elemento aes() con el argumento fill

```
dat %>%
  group_by(municipality) %>%
  summarise(unicos = unique(species)) %>%
  ggplot(aes(x = municipality, fill = municipality)) +
  geom_bar()
- `summarise()` regrouping output by 'municipality' (override with `.groups` argument)
```



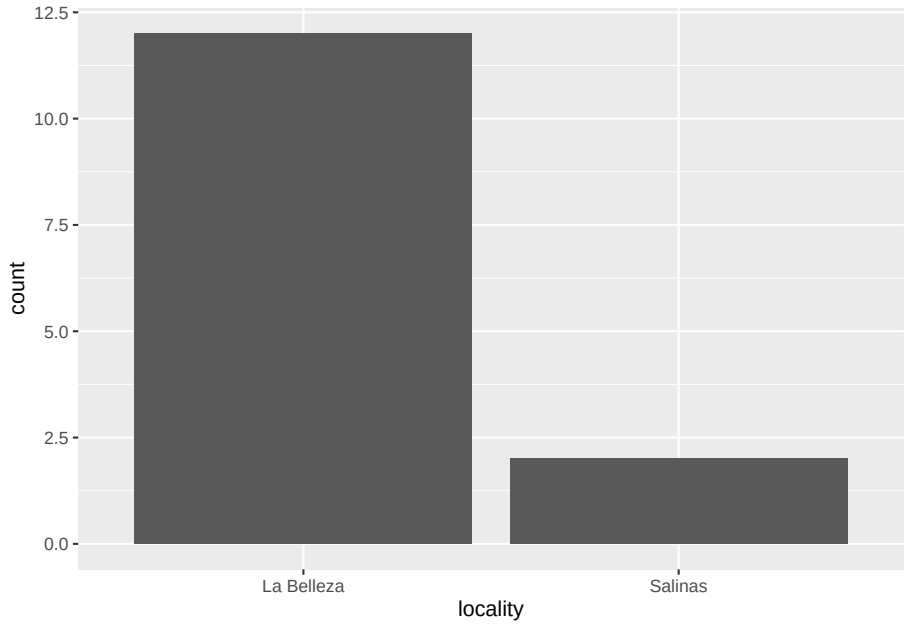
Podemos cambiar el orden de las barras cambiando el eje x a eje y. Si deseamos usar colores específicos para cada barra lo hacemos en el geom_:

```
dat %>%
  group_by(municipality) %>%
  summarise(unicos = unique(species)) %>%
  ggplot(aes(y = municipality)) +
  geom_bar(fill = c("blue", "yellow", "brown"), col = c("blue", "yellow", "brown")) # ¿Cuál es lo
- `summarise()` regrouping output by 'municipality' (override with `.groups` argument)
```



Ejercicio: Realice una grafica con ggplot de los registros de mamiferos por cada localidad

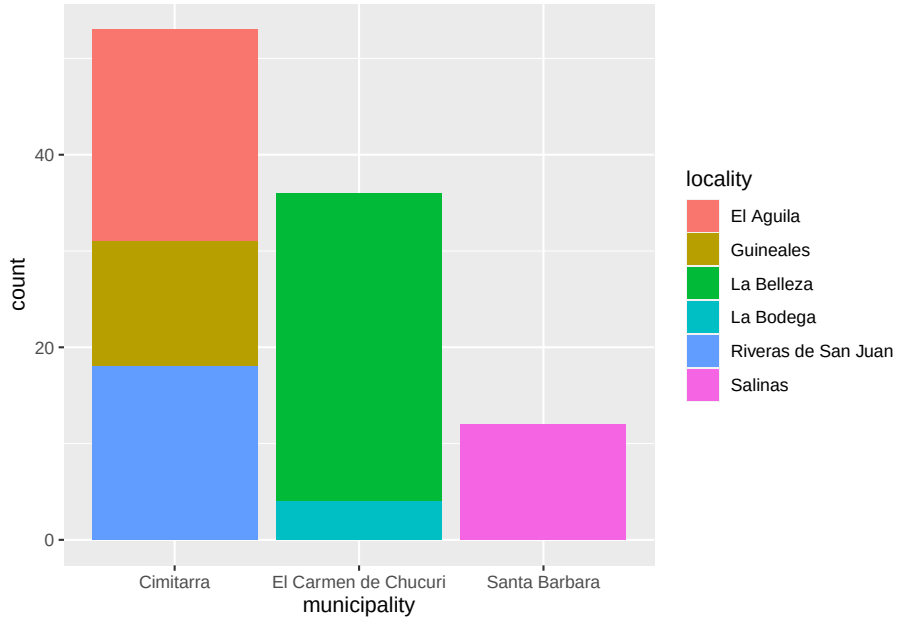
```
dat %>%
  filter(class == "Mammalia") %>%
  ggplot(aes(x = locality)) +
  geom_bar()
```



Ejercicio: Grafique los registros del orden coleoptera en las diferentes localidades de cada municipio. Para diferenciar a que municipio pertenece cada localidad, usel el argumento fill de aes()

Podemos agrupar las barras

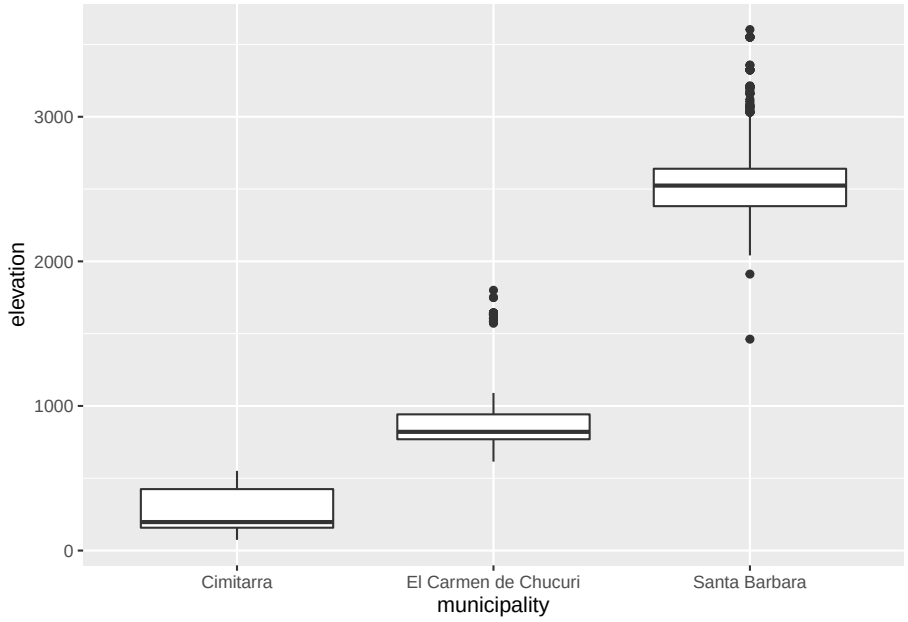
```
dat %>%
  filter(order == "Coleoptera") %>%
  group_by(locality) %>%
  ggplot(aes(x = municipality, fill = locality)) +
  geom_bar(position =  "stack")
```



También podemos hacer graficos de cajas e histogramas cambiando de geom_

?geom_boxplot

```
dat %>%
  ggplot(aes(x = municipality, y = elevation)) + #Grafico de cajas de la elevacion por
  geom_boxplot()
```



Ejercicio 1

1. Realice un gráfico de cajas sobre la distribución de las alturas a la que fueron registradas las aves en cada localidad utilizando las funciones tidy
2. ¿Cómo es la distribución de elevacion de las familias del orden Polypodiales?
3. Realice un histograma de la elevación total y la elevación de ambos reinos

Experimente graficando la elevacion de diferentes grupos (reinos, clase, familia etc) utilizando las funcioens tidy

6.1 Theme

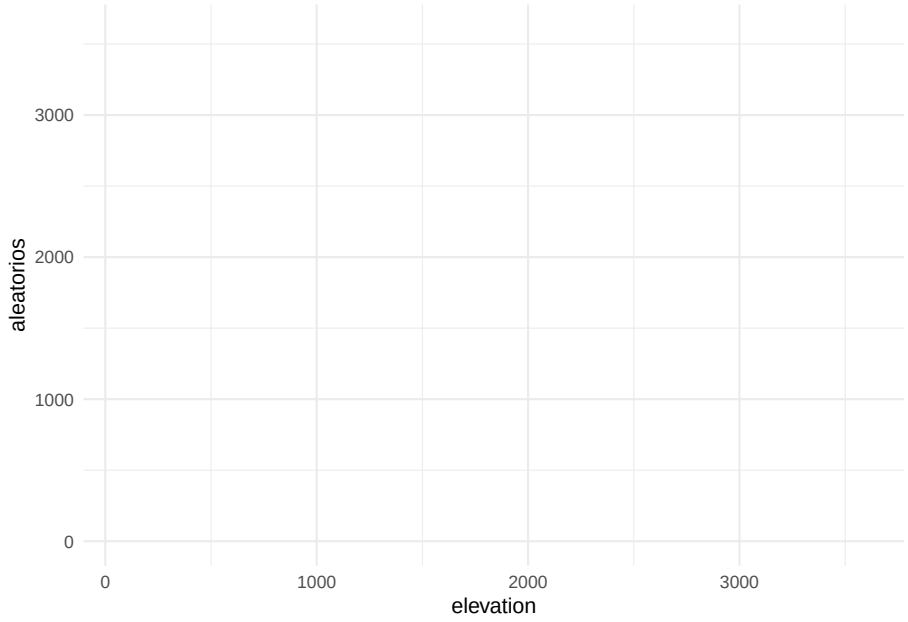
Igual que en la función básica de plot(), en ggplot podemos tener mas de una gráfica y personalizar cada elemento mediante theme()

```
as <- dat %>% mutate(elevation_2 = elevation+10)
```

Ggplot cuenta con temas predeterminados que podemos cargar mediante theme_

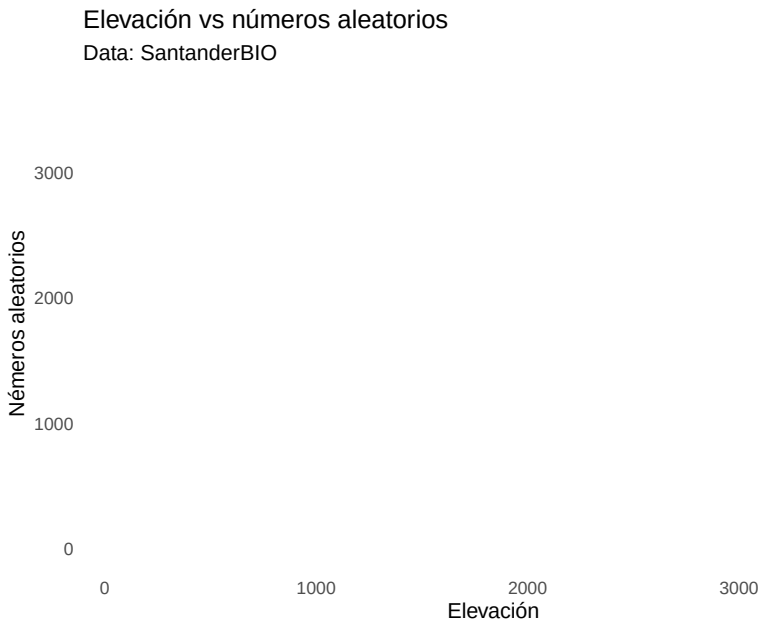
?theme()

```
dat %>% mutate(aleatorios = runif(1255, min = 0, max = 3600)) %>%
  ggplot(aes(x=elevation, y=aleatorios)) + #establecemos los datos y las variables "x"
  theme_minimal() #tema del grafico para personalizar color de fondo, bordes, cuadrícu
```



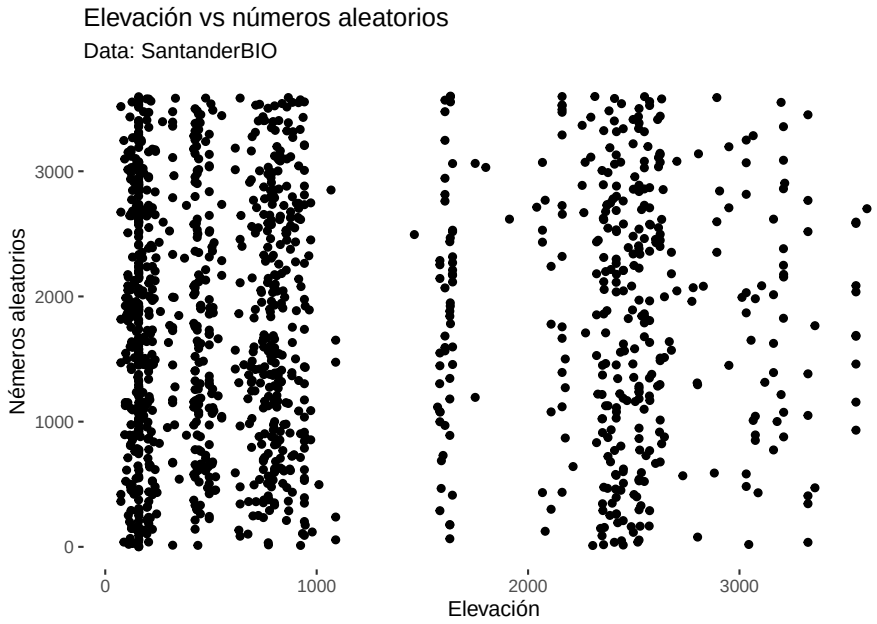
Dentro de theme() modificamos elementos como líneas, colores, ejes, entre otros

```
dat %>% mutate(aleatorios = runif(1255, min = 0, max = 3600)) %>%
  ggplot(aes(x=elevation, y=aleatorios)) +
  theme_minimal() + theme(panel.border = element_blank(),
                          panel.grid.major = element_blank(),
                          panel.grid.minor = element_blank(),
                          axis.line = element_line(colour = "white")) +#definimos fond
  labs(title = "Elevación vs números aleatorios", #título
       subtitle = "Data: SantanderBIO") + #subtítulo
  labs(x = "Elevación", y = "Némeros aleatorios")#nombres de los ejes
```



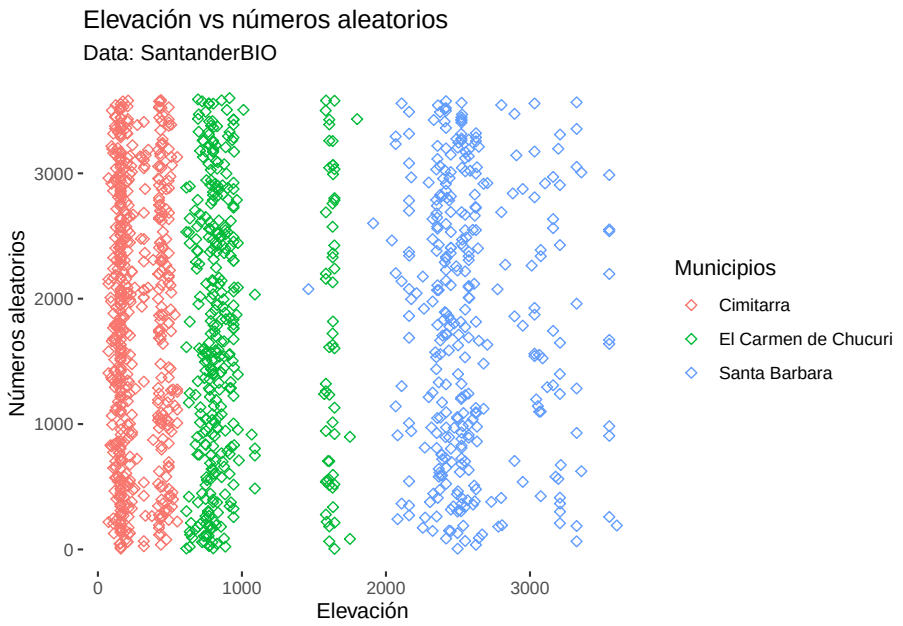
Añadimos un geom_ de puntos

```
dat %>% mutate(aleatorios = runif(1255, min = 0, max = 3600)) %>%
  ggplot(aes(x=elevation, y=aleatorios)) + #Grafico base
  geom_point() + # geometria que corresponde a los puntos
  theme_bw() + theme(panel.border = element_blank(),
                     panel.grid.major = element_blank(),
                     panel.grid.minor = element_blank(),
                     axis.line = element_line(colour = "white")) +
  labs(title = "Elevación vs números aleatorios",
       subtitle = "Data: SantanderBIO") +
  labs(x = "Elevación", y = "Némeros aleatorios")
```



Coloreamos los puntos y modificamos el título de la leyenda generada

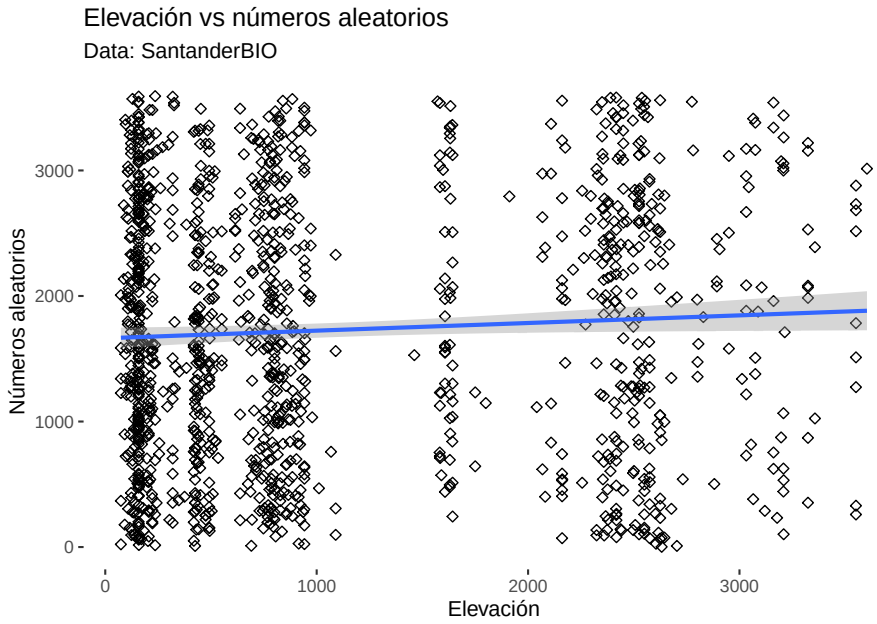
```
dat %>% mutate(aleatorios = runif(1255, min = 0, max = 3600)) %>%
  ggplot(aes(x=elevation, y=aleatorios, color = municipality)) + #Gráfico base con col
  geom_point(shape=5) +  # Geom que corresponde a los puntos y shape para modificar la
  theme_bw() + theme(panel.border = element_blank(),
                     panel.grid.major = element_blank(),
                     panel.grid.minor = element_blank(),
                     axis.line = element_line(colour = "white"))+#definimos fondo y bo
  scale_colour_discrete(name = "Municipios") + #Establecemos el título de la leyenda
  labs(title = "Elevación vs números aleatorios",
       subtitle = "Data: SantanderBIO") +
  labs(x = "Elevación", y = "Números aleatorios")
```



Podemos incluir en la gráfica algunos métodos estadísticos como la relación lineal entre los puntos

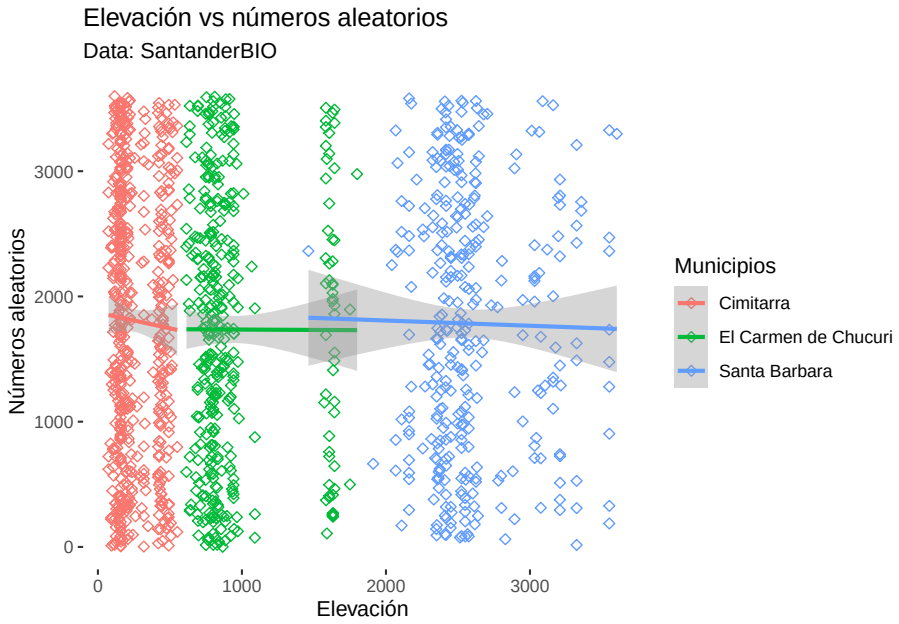
?geom_smooth

```
dat %>% mutate(aleatorios = runif(1255, min = 0, max = 3600)) %>%
  ggplot(aes(x=elevation, y=aleatorios)) +
  geom_point(shape=5) +
  geom_smooth(method = "lm", se = TRUE) + #método lm e intervalo de confianza
  theme_bw() + theme(panel.border = element_blank(),
                     panel.grid.major = element_blank(),
                     panel.grid.minor = element_blank(),
                     axis.line = element_line(colour = "white")) +
  scale_colour_discrete(name = "Municipios") +
  labs(title = "Elevación vs números aleatorios",
       subtitle = "Data: SantanderBIO") +
  labs(x = "Elevación", y = "Números aleatorios")
- `geom_smooth()` using formula 'y ~ x'
```



Y por cada grupo

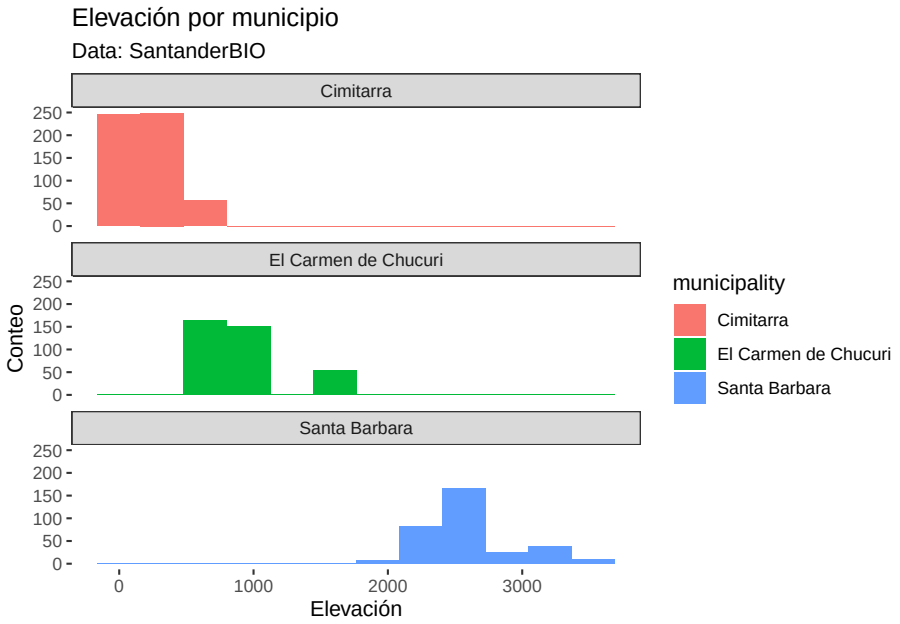
```
dat %>% mutate(aleatorios = runif(1255, min = 0, max = 3600)) %>%
  ggplot(aes(x=elevation, y=aleatorios, color = municipality)) +
  geom_point(shape=5) +
  geom_smooth(method = "lm", se = TRUE) +
  theme_bw() + theme(panel.border = element_blank(),
                     panel.grid.major = element_blank(),
                     panel.grid.minor = element_blank(),
                     axis.line = element_line(colour = "white")) +
  scale_colour_discrete(name = "Municipios") +
  labs(title = "Elevación vs números aleatorios",
       subtitle = "Data: SantanderBIO") +
  labs(x = "Elevación", y = "Números aleatorios")
- `geom_smooth()` using formula 'y ~ x'
```



Para poder mostrar mas de una gráfica en el mismo plot se utiliza face_wrap. Revise la función y realice el histograma anterior pero ubique en un mismo plot 3 gráficas para cada municipio

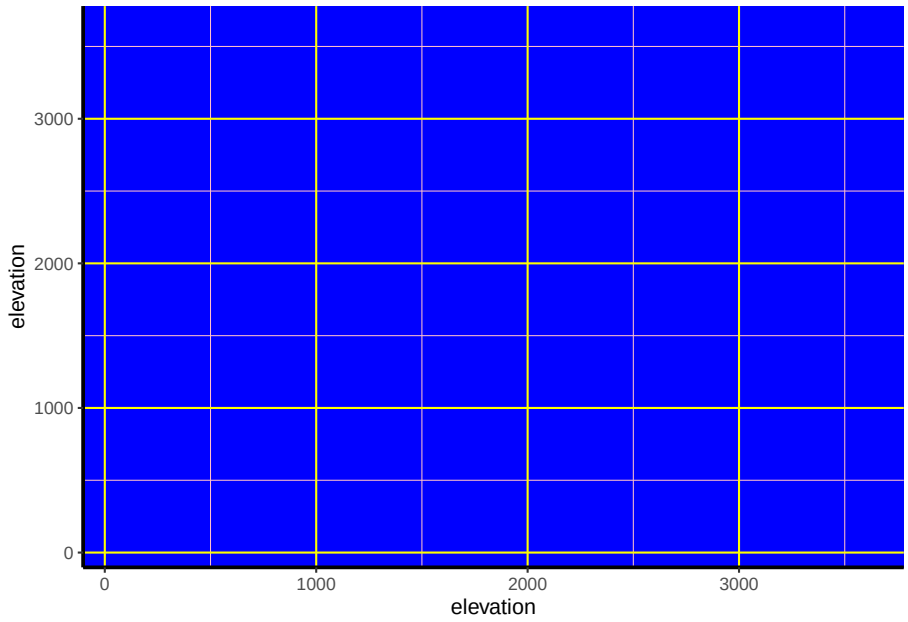
?facet_wrap

```
ggplot(data = dat) +
  geom_histogram(aes(x = elevation, fill = municipality), bins = 12) +
  facet_wrap(~municipality, ncol = 1)+ #para cada especie, realice tres histogramas en una colum
  theme_bw() + theme(panel.border = element_blank(),
                     panel.grid.major = element_blank(),
                     panel.grid.minor = element_blank(),
                     axis.line = element_line(colour = "white")) +
  labs(title = "Elevación por municipio",
       subtitle = "Data: SantanderBIO") +
  labs(x = "Elevación", y = "Conteo")
```



Todas las características de un tema pueden ser guardadas en un vector para evitar escribir el tema cada vez que se grafique

```
mitema <- theme(panel.grid.major = element_line(colour = "yellow"),
                panel.grid.minor = element_line(colour = "pink"),
                panel.background = element_rect(fill = "blue"),
                panel.border = element_blank(),axis.line = element_line(size = 0.9, li
ggplot(data = dat, aes(x = elevation, y = elevation))+
  mitema
```

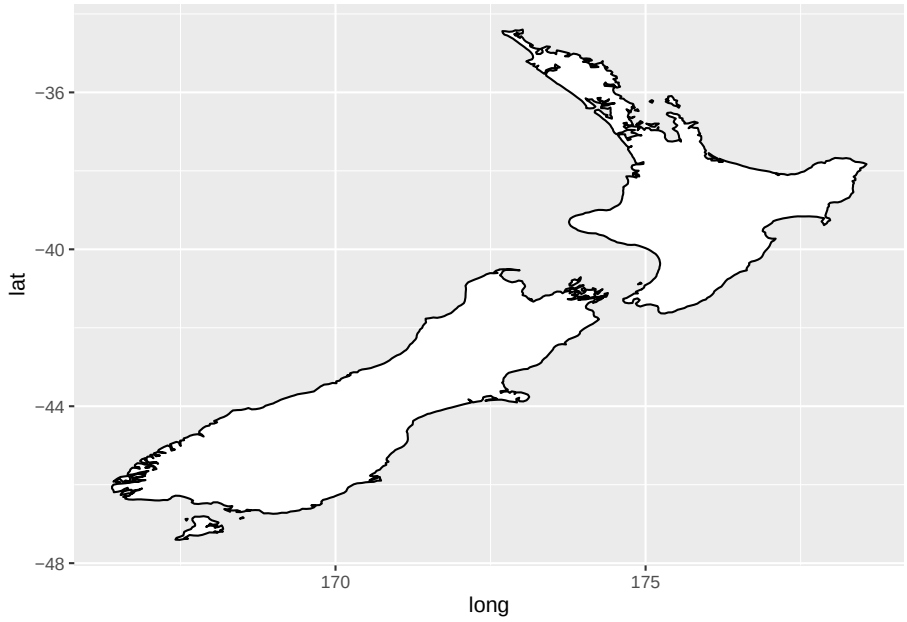


Ejercicio 2

1.Contruir un tema personalizado 2. Crear una gráfica de barras de el número de individuos por cada clase, utilizando un color diferente para cada barra y el tema creado anteriomente. 3.Realizar el gráfico anterior pero generando una gráfica separada para cada clase en el mismo plot 4. Conviertir la gráfica de barras del punto 2 en una gráfica de torta mediante coord_polar()

Finalmente, con el sistema de coordenadas “x” y “y” es posible hacer increíbles mapas mediante ggplot

```
nz <- map_data("nz") #Cargamos un set de datos que contienede coordenadas longitud y latitud, y c
ggplot(nz, aes(long, lat, group = group)) +
  geom_polygon(fill = "white", colour = "black")
```

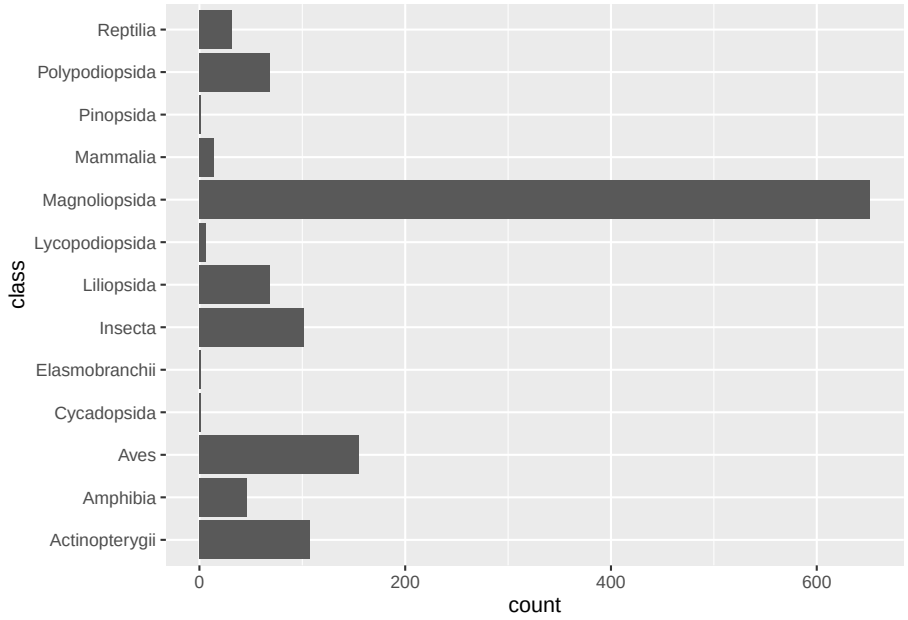


6.2 Forcats

El paquete forcats tiene como objetivo brindar herramientas de ayuda para manejar variables categóricas.

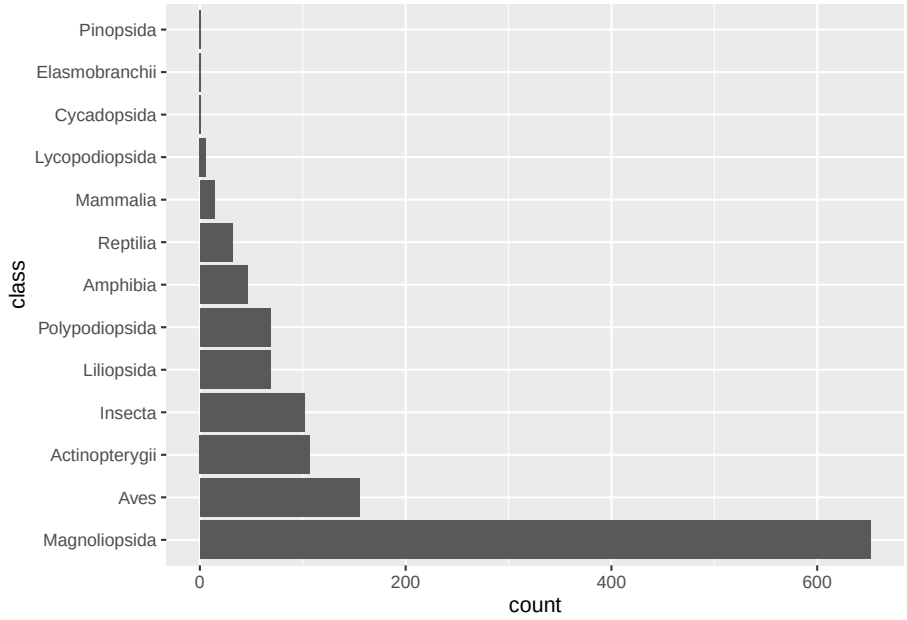
Convirtamos una variable en un factor

```
dat$class <- as.factor(dat$class)
levels(dat$class) #Revisamos las categorías
-  [1] "Actinopterygii" "Amphibia"       "Aves"           "Cycadopsida"    "Elasmobran
-  [6] "Insecta"        "Liliopsida"     "Lycopodiopsida" "Magnoliopsida"  "Mammalia"
- [11] "Pinopsida"      "Polypodiopsida" "Reptilia"
dat %>%
  ggplot(aes(x = class)) +
  geom_bar() +
  coord_flip()
```



Vemos que se grafica un conteo de la variable "class" por cada factor, ahora vamos a ordenar esta gráfica con forcats

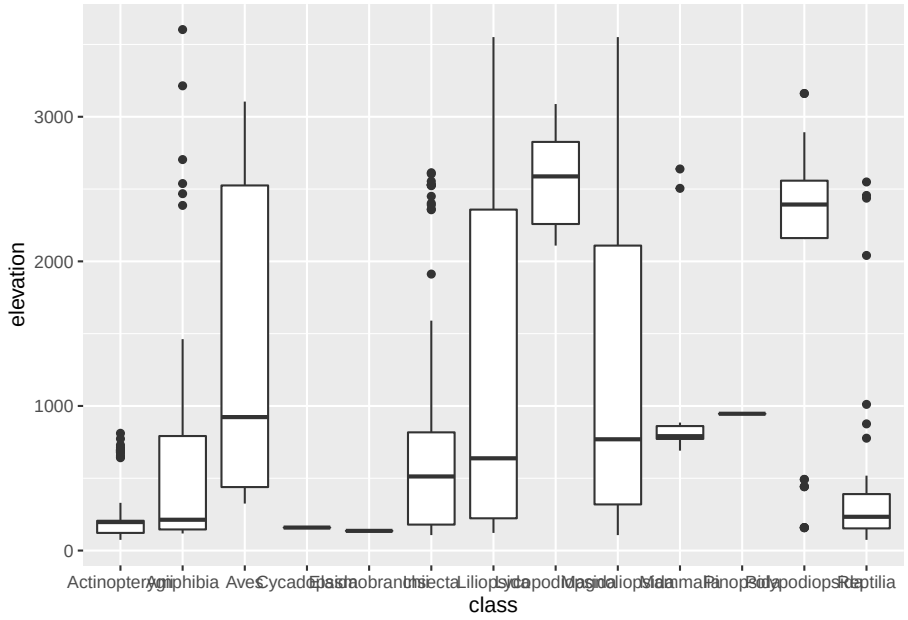
```
dat %>%
  mutate(class = fct_infreq(class)) %>%
  ggplot(aes(x = class)) +
  geom_bar() +
  coord_flip()
```



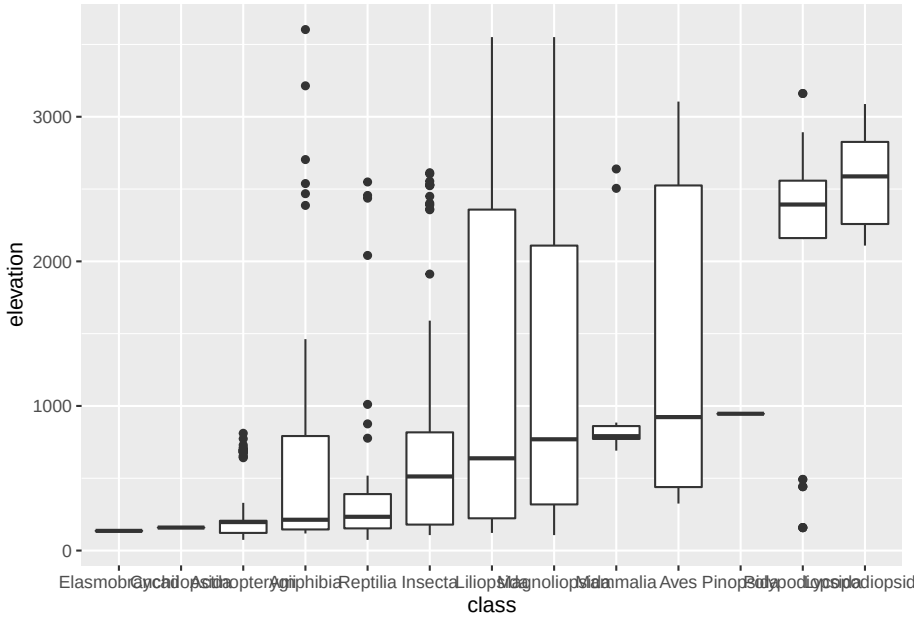

```
#Ahora tenemos los cada categoria ordenada por su frecuencia
```

Podemos ordenar un factor por otra variable

```
dat %>%
  ggplot(aes(x = class, y = elevation)) +
  geom_boxplot()
```



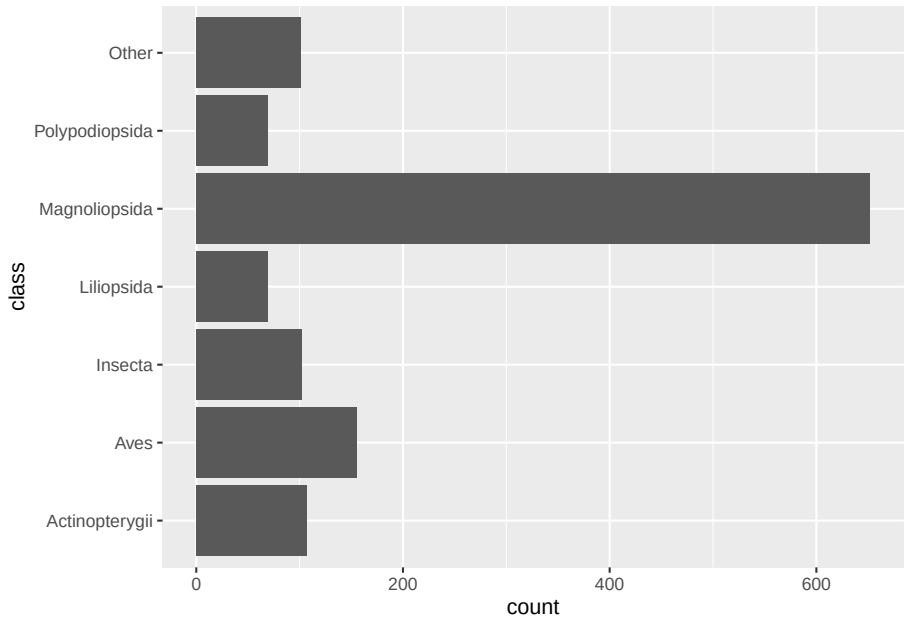

```
#Bloxplot de la distribución del a altura por cada clase
dat %>%
  mutate(class = fct_reorder(class, elevation)) %>%
  ggplot(aes(x = class, y = elevation)) +
  geom_boxplot()
```



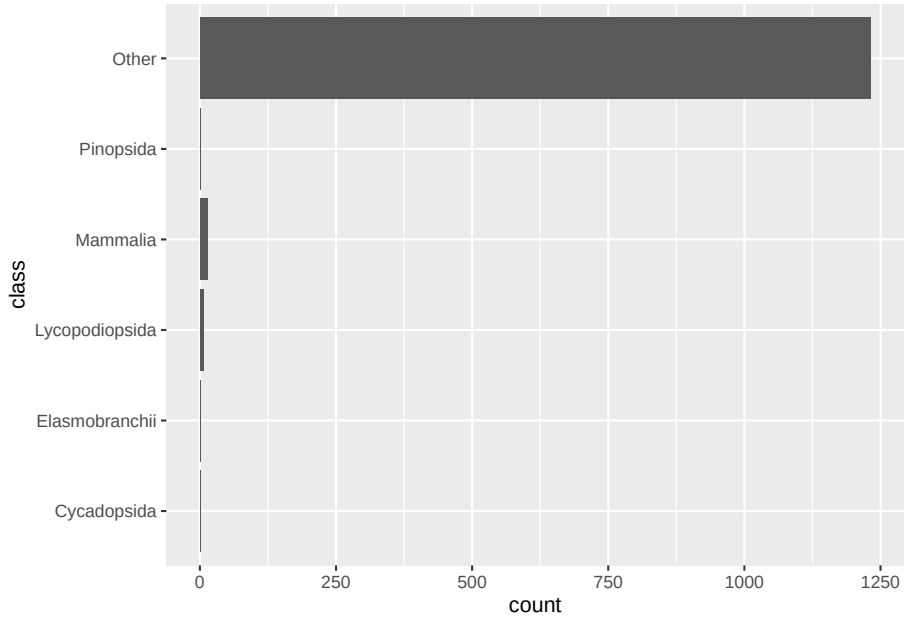

```
#En este caso ordenamos las clases por la mediana de sus elevaciones
```

Si tenemos clases con pocas o muchas observaciones podemos agruparlos en un nuevo grupo llamado "other". Explique la diferencia entre las dos gráficas

```
dat %>%
  mutate(class = fct_lump(class, n = 5)) %>%
  ggplot(aes(x = class)) +
  geom_bar() +
  coord_flip()
```




```
dat %>%
  mutate(class = fct_lump(class, n = -5)) %>%
  ggplot(aes(x = class)) +
  geom_bar() +
  coord_flip()
```



forcats tambien nos permite ordenar las categorias a mano

```
f <- as.factor(dat$phylum)
levels(f)
- [1] "Arthropoda"   "Chordata"     "Tracheophyta"
# fct_relevel(f, "Tracheophyta", "Arthropoda", "Chordata")
```

Chapter 7

Introducción a Rmarkdown

En esta sesión vamos a aprender a crear un documento markdown. Este es un formato para escribir reportes reproducibles y dinámicos en R. Usted puede exportar un markdown como PDF, HTML O Word, en donde puede incluir código de R y resultados como gráficas o texto.

La sintaxis de Markdown es igual a la usada en latex, un software en el cual se procesa texto. Por esto, antes de empezar, debemos instalar el paquete "tinytex".

R Markdown cuenta con una sintaxis para escribir texto plano o código. Comenzando por el texto plano, vamos a revisar los distintos formatos de texto:

7.0.1 Sintaxis de texto plano:

Para seleccionar un título de primer nivel

Título 1: #

Título 2: ##

Título 3: ###

Título 4: ####

Texto en itálica o *Texto en itálica*

Texto en negrilla o **Texto en negrilla**

Equaciones: $A = \pi * r^2$

enlace

> Cita en bloque

Título de la tabla	Segundo título
Celda 1	Celda 2
Celda 3	Celda 4

- Lista
  - Sub item de lista

1. Lista numérica
   - Sub item de la lista

Separador:

7.0.2 Sintaxis de código:

Para incluir código en Markdown, debemos incluirlo dentro de la siguiente estructura denominada **"chunk"**:

```
1+2
```

```
## [1] 3
```

Podemos nombrar un chunk y ajustar la forma en la que se muestran los resultados mediante distintos argumentos definidos con el operado lógico TRUE o FALSE:

7.0.2.0.0.1 *Nota: dos chunks no pueden tener el mismo nombre*

- echo: Nos permite decidir si el código debe ser mostrado en el documento

```
## [1] 22
```

- eval: Nos da la opción si ejecutar el código y mostrar su resultado o no

```
5/2
```

- warning: Controla los mensajes de warning que pueden resultar al ejecutar el código

```
warning("Este es un mensaje de waning")
```

```
## Warning: Este es un mensaje de waning
```

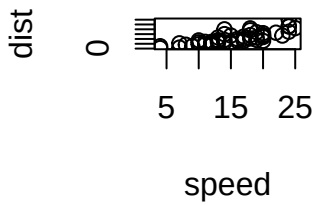
- error: Controla los mensajes de errores que pueden resultar al ejecutar el código

```
2*!
```

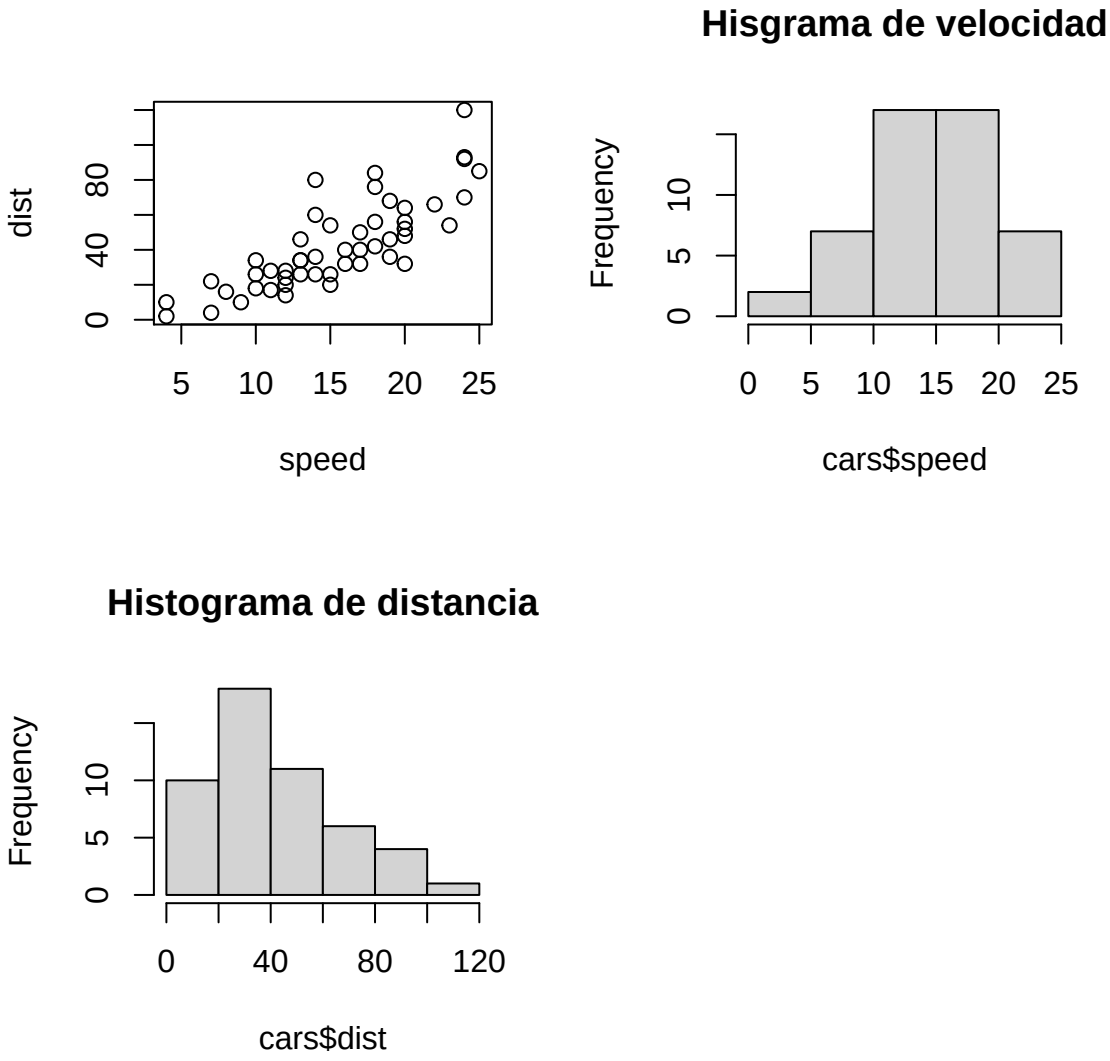
```
## Error: <text>:2:0: unexpected end of input
## 1: 2*!
##    ^
```

Podemos realizar gráficas y configurar el tamaño mostrado en el documento final:

```
plot(cars)
```



Es posible ajustar mas de una gráfica a la vez:



7.0.2.0.0.2 Mayor información sobre la configuración de chunks Finalmente, para generar nuestro documento, debemos usar el botón **Knit** que se encuentra en la partde superior de R. Allí escogeremos el documento de salida.

Chapter 8

De números aleatorios, distribuciones y probabilidades

En R, existen funciones para la generación de números aleatorios, es decir, podemos generar cualquier número de 0 a infinito de forma automática. Para esto, debemos especificar la distribución de probabilidad de la cual queremos obtener estos números aleatorios.

Una distribución de probabilidad describe la gama de resultados que podemos obtener. De esta manera, la probabilidad de que obtengamos un valor dependerá de la distribución de probabilidad.

Ejemplo:

8.0.0.1 La probabilidad de los dados:

Un dado convencional se caracteriza por poseer 6 lados, y cada lado representa un valor de 1 al 6. Frecuentemente en juegos de mesa como el parqués, mientras más alto sea el número, más podremos avanzar y ganar. Al lanzar el dado para poder avanzar, existe una probabilidad fija de obtener cualquiera de los 6 lados. Ya que esta probabilidad es la misma para cada lado (1/6), podríamos decir que la distribución de los números de un dado pertenece a una distribución uniforme.

8.0.1 Distribución uniforme

En nuestro ejemplo del dado, planteamos que la distribución de los números de cada lado del dado tiene una distribucion uniforme, debido a que la probabilidad

de de obtener cada lado es la misma. Vamos a explorar un poco esto en R.

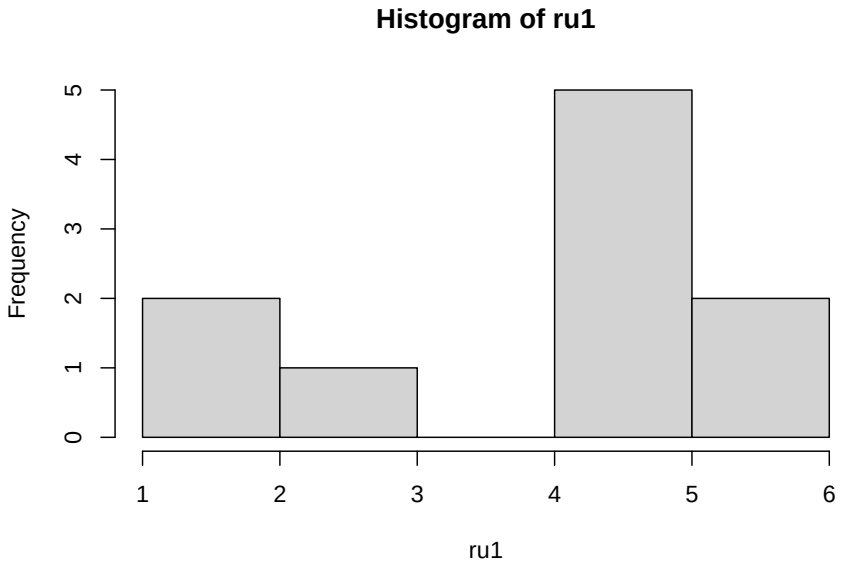
En nuestro ejemplo del dado, planteamos que la distribución de los números de cada lado del dado es uniforme, debido a que la probabilidad de obtener cada lado es la misma. Vamos a explorar un poco esto en R.

Para obtener un número aleatorio "n" de una distribución uniforme utilizamos runif()

?runif

Vamos a escoger 10 números aleatorios del 1 al 6 provenientes de una distribución uniforme

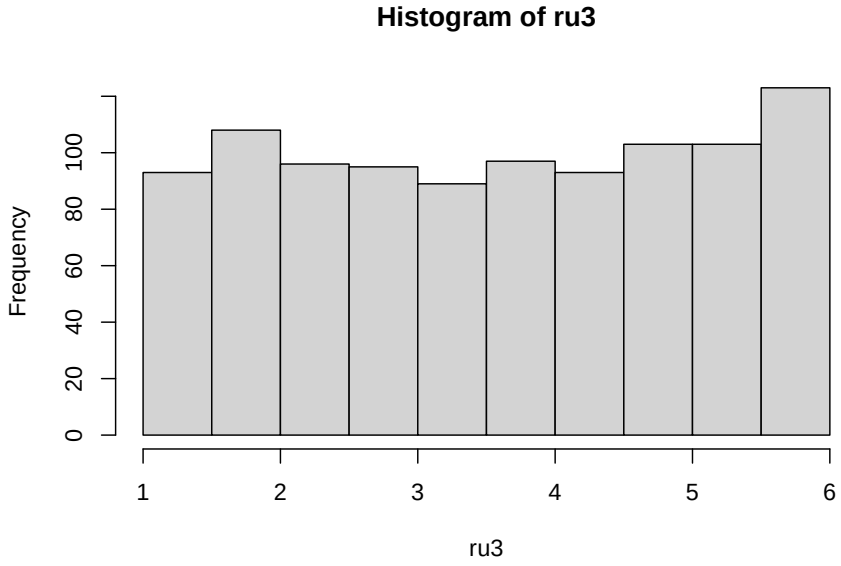
```
ru1 <- runif(n = 10, min = 1, max = 6) #Los números que obtendremos tienen decimales

hist(ru1)
```



Según nuestro histograma, y dependiendo de los números que se seleccionen, puede que obtengamos más veces un número que otro, ¿entonces la probabilidad no es la misma?

```
ru2 <- runif(n =100, min = 1, max = 6)
hist(ru2)
```


```
ru3 <- runif(n =1000, min = 1, max = 6)
hist(ru3)
```



¡Ahora nuestro histograma parece más uniforme!. Cuando trabajamos con

datos, siempre es una buena practica tener una muestra grande, con esto evitamos el sesgo en nuestros analisis.

8.0.2 Distribución normal

La distribución normal es una de las más importantes, ya que se ajusta a muchos datos que representan procesos en la vida real. Su representación se asemeja a la de una campana, la campana de Gauss.

Para obtener un número aleatorio "n" de una distribución normal utilizamos rnorm()

?rnorm

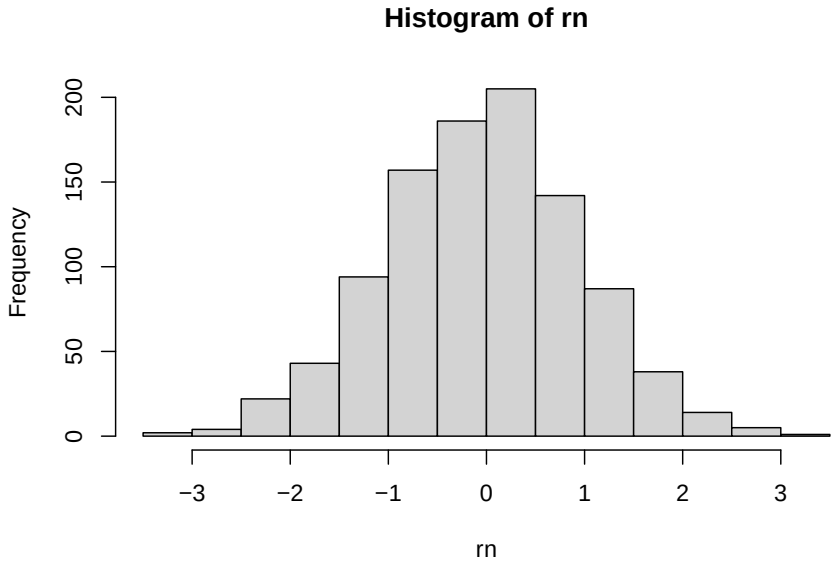
```
rnorm(n = 4, mean = 100, sd = 5)
- [1]  96.33674  98.45675 102.87299  99.18282
```

?set.seed para reproducir los mismos números Utilizamos ?set.seed para reproducir el mismo grupo de números cada vez

```
set.seed(6)
rnorm(n = 10, mean = 5, sd = 5)
-  [1]  6.348030  1.850073  9.343299 13.635978  5.120938  6.840126 -1.546021  8.693110
- [10] -0.241986
set.seed(6)
rnorm(n = 10, mean = 5, sd = 5)
-  [1]  6.348030  1.850073  9.343299 13.635978  5.120938  6.840126 -1.546021  8.693110
- [10] -0.241986
set.seed(4)
rnorm(n = 10, mean = 5, sd = 5)
-  [1]  6.083774  2.287537  9.455723  7.979903 13.178090  8.446377 -1.406233  3.934277
- [10] 13.884316
```

Distribución normal estándar

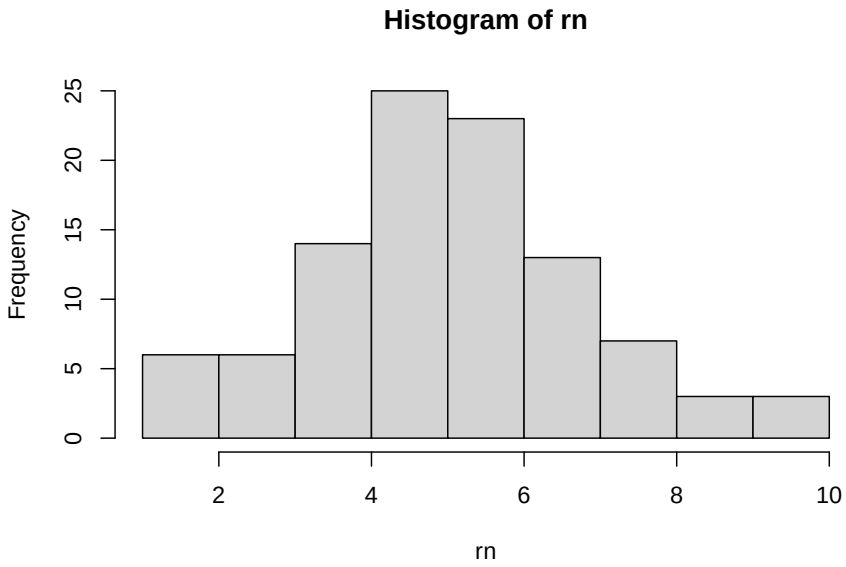
```
rn<-rnorm(n = 100)
rn<-rnorm(n = 1000, mean = 0, sd = 1)
hist(rn)
```




```
mean(rn)
- [1] -0.03971125
sd(rn)
- [1] 0.9865829
```

Distribución normal con media de 5 y desviación estándar de 2

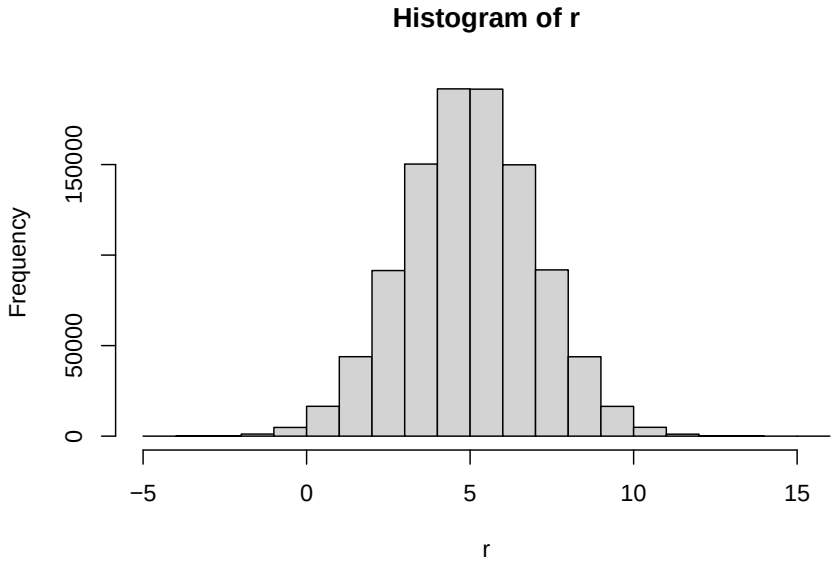
```
rn<-rnorm(n = 100, mean = 5, sd = 2)
hist(rn)
```




```
mean(rn)
- [1] 4.988073
sd(rn)
- [1] 1.812704
```

Aumente n: 1000, 10000, 100000

```
r<-rnorm(1000000, mean=5, sd=2)
hist(r)
```

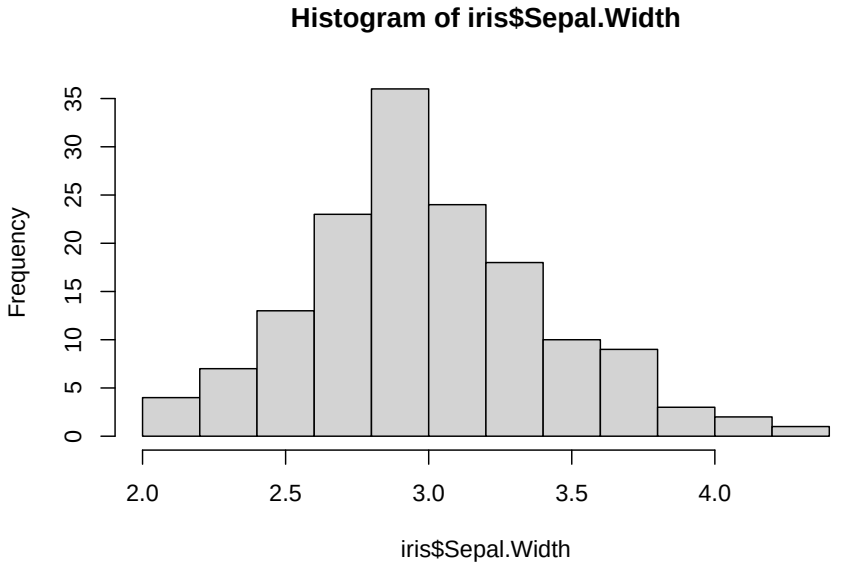



```
mean(r)
- [1] 4.999839
sd(r)
- [1] 1.997826
```

Vamos a cargar un conjunto de datos que contiene medidas del sépalo y pétalo de diferentes especies del género Iris.

```
data(iris)
hist(iris$Sepal.Length)
```


```
hist(iris$Sepal.Width)
```

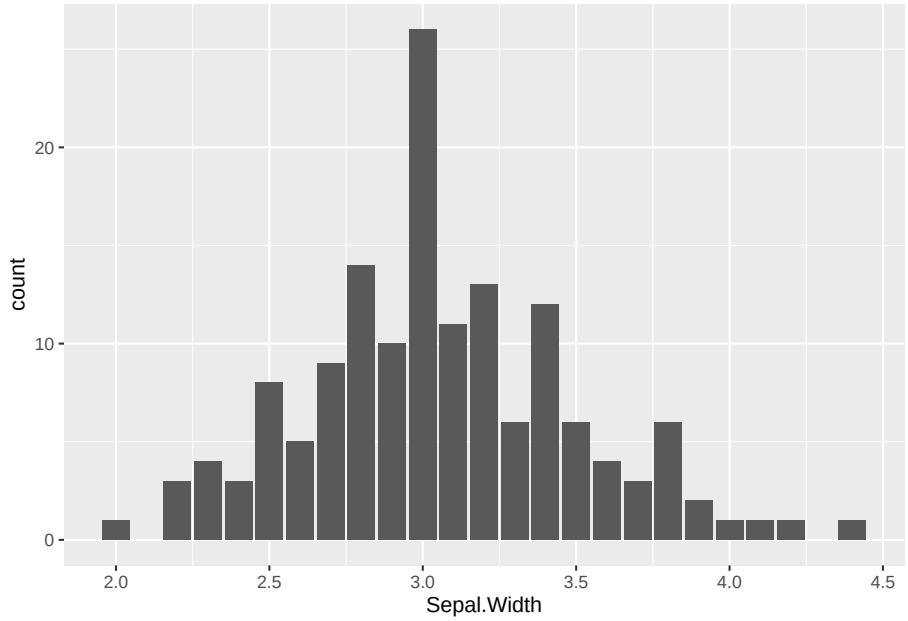


```r
hist(iris$Petal.Length)
```


```r
hist(iris$Petal.Width)
```

8.0.3 Tidy

```
library(ggplot2)

ggplot(iris, aes(Sepal.Width)) +
  geom_bar(stat = "count")
```



¿Cuál de las 4 variables (Sepal.Length, Sepal.Width, Petal.Length, Petal.Width) representa mejor una distribución normal?

Ejercicio 1

1. Crear un vector v1 que contenga una muestra de 1000 números aleatorios provenientes de una distribución normal con media de 112 y desviación estándar de 35. Graficar los valores en un histograma.

2. Crear un vector v2 que contenga una muestra de 1000 números aleatorios provenientes de una distribución normal con media de 34 y desviación estándar de 3. Graficar los valores en un histograma.

3. 3. Multiplicar v1 con v2 y graficar el resutado en un histograma. ¿Cuál es la diferencia de este histograma con los anteriores?

4. Crear un vector "x" que contenga 15 números entre 1 y 50 que vengan de una distribución uniforme. Crear otro vector "y" que contenga los valores

de “x” pero dos veces mas grande. Sumar a cada valor de “y” un valor constante de 7. Graficar mostrando “y” sobre “x” : plot(x,y)

8.0.4 Probabilidad en la distribución normal

Para calcular la probabilidad de obtener un rango de valores que pertenezcan a una distribución normal, utilizamos la función pnorm()

?pnorm

Podemos calcular la probabilidad de obtener un valor menor de -2 que venga de una distribución normal (mean = 2, sd = 2)

```
pnorm(q = -2, mean = 2, sd = 2)
- [1] 0.02275013
```

O también podemos calcular la probabilidad de obtener un valor mayor que -2 que venga de la distribución normal mencionada anteriormente (mean = 2, sd = 2)

```
1-pnorm(-2,mean=2,sd=2)
- [1] 0.9772499
```

Ejercicio 2

1. ¿Cuál es la probabilidad de obtener un valor menor que 5 proveniente de la distribución normal mencionada anteriormente (mean = 5, sd = 2)?
2. ¿Cuál es la probabilidad de obtener un valor entre 2 y 8?

Chapter 9

Análisis de varianza - ANOVA

En esta sesión vamos a aprender a realizar el test de análisis de varianza o ANOVA. Mediante esta prueba, se busca encontrar si hay diferencias significativas entre las medias de dos grupos. Para poder calcularla, se tendrán en cuenta dos tipos de varianza: 1) La varianza entre grupos (los valores promedio de cada grupo se comparan con la media total de los datos); y la varianza dentro del grupo (la varianza de cada dato en un grupo con el promedio dentro del mismo grupo).

Vamos a aprender primero como realizar la anova a mano paso a paso y luego utilizaremos la función de ANOVA en R.

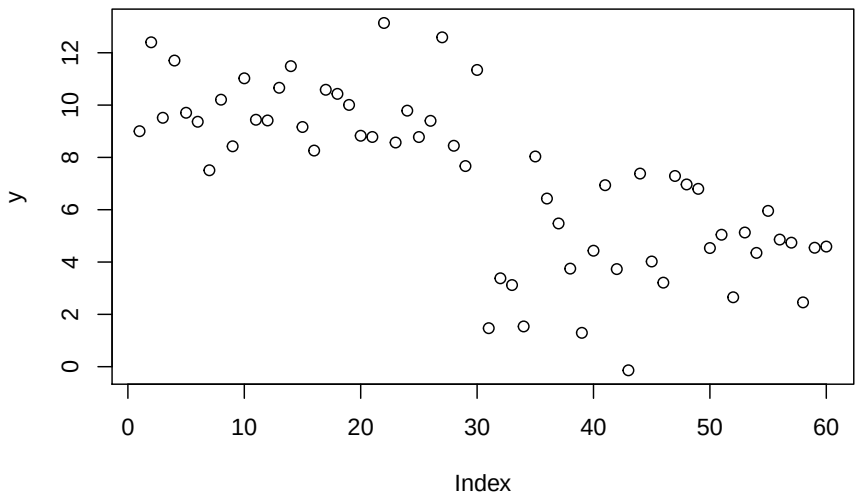
9.0.1 Anova a mano

Vamos a generar nuestros datos para hacer una Anova a mano. Supongamos que estamos experimentando con un tratamiento que permita a las plantulas de una planta crecer más rápido. Para esto tendremos dos grupos de plantas de una misma especie, unas a las que se les provea un compuesto "z" el cuál debería acelerar el crecimiento y otros a los que no. Llamaremos al grupo con el tratamiento "z" g1 y el otro grupo g2.

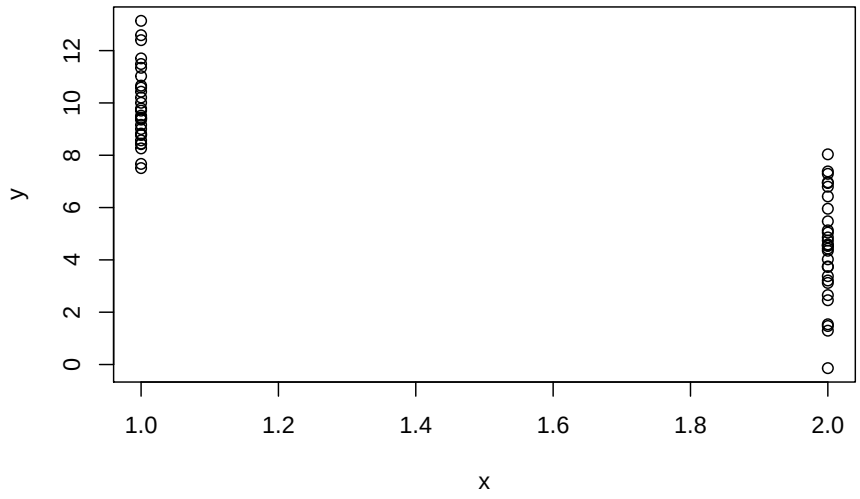
Generamos datos aleatorios del crecimiento de las plantas, la cual es nuestra variable dependiente

```
g1 <- rnorm(30, mean=10, sd=2) # 30 observaciones (replicas) en donde el crecimiento en promedio

g2 <- rnorm(30, mean=5, sd=2) # 30 observaciones (replicas) en donde el crecimiento en promedio e
```

```
y <- c(g1, g2) # Los concatenamos en un mismo vector

x <- rep(c(1, 2), each = 30) # Creamos la variable de nuestros grupos o variable indep

plot(y)
```



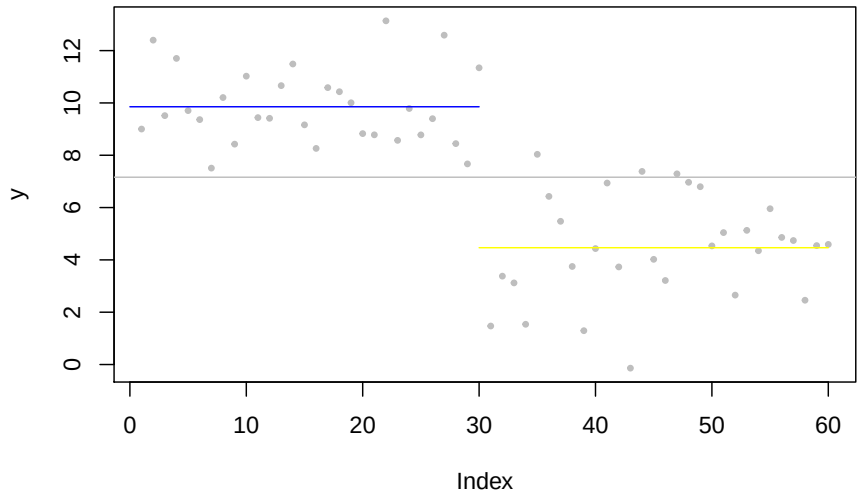

```
plot(x, y) # En este punto parecen haber diferencia entre el crecimiento del grupo 1 y
```



Ahora empezamos a realizar una ANOVA paso a paso. Para esto, hallaremos la suma de cuadrados y los grados de libertad, pues esto es igual a la varianza

Obtenemos los promedios de cada agrupo y el total, los cuales necesitaremos para hallar la suma de los residuales al cuadrado (suma total de la distancia entre cada observación con la media elevada al cuadrado)

```
tot.mean <- mean(y) # Obtenemos el promedio de todas la observaciones

# en la grafica:
plot(y, pch=19, col="grey", cex=0.5)
abline(tot.mean, 0, col="grey")
?abline
# a = intercepto, b = pendiente, para una línea horizontal la pendiente (slope) = 0

g1.mean <- mean(g1)
g2.mean <- mean(g2)
lines(c(0, 30), c(g1.mean, g1.mean) , col = "blue") # media del grupo 1
lines(c(30, 60), c(g2.mean, g2.mean) , col = "yellow") # media del grupo 2
```



Calculamos la suma de cuadrados de los residuos, los cuales se dividen en suma de cuadrados totales (SST); el error de la suma de cuadrados (SSE), que se refiere a la varianza dentro de cada grupo que no se puede explicar; y la suma de cuadrados del tratamiento (SSA) o varianza entre grupos.

Suma de cuadrados total: SST = SSE + SSA

```
SST <- sum((y - tot.mean) ^2) #Hallamos el SST sumando la resta entre la media de todo
```

Suma de cuadrados residual

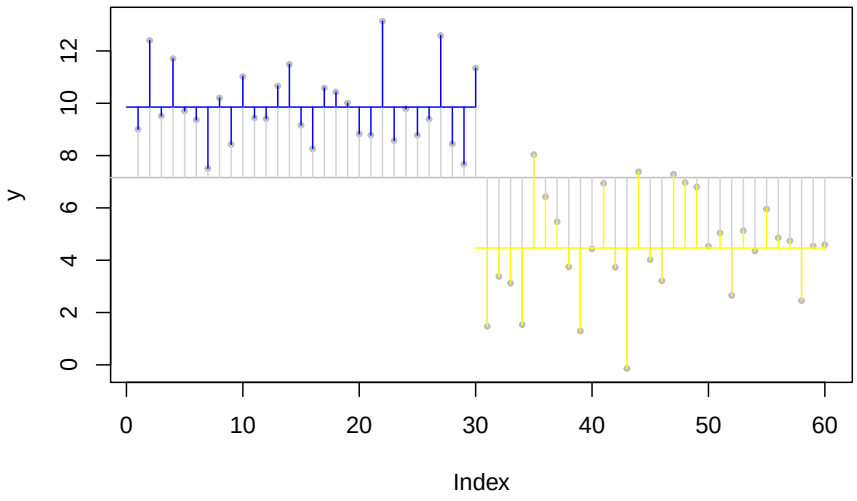
```
SSE.g1 <- sum((g1 - g1.mean) ^2) # Realizamos el mismo paso para calcular el SST pero

SSE.g2 <- sum((g2 - g2.mean) ^2)

SSE <- SSE.g1 + SSE.g2 #Sumamos los SSE de cada grupo
```

SSA

```
SSA <- SST - SSE
```

En la gráfica: Se dibujan los residuos en vez de las sumas de los cuadrados:

```
plot(y, pch=19, col="grey", cex=0.5)
abline(tot.mean, 0, col="grey")
lines(c(0, 30), c(g1.mean, g1.mean) , col = "blue") # media del grupo 1
lines(c(30, 60), c(g2.mean, g2.mean) , col = "yellow") # media del grupo 2

segments(1:60, tot.mean, 1:60, y, col="lightgrey")
segments(1:30, g1.mean, 1:30, g1, col="blue")
segments(31:60, g2.mean, 31:60, g2, col="yellow")
```



A partir de acá, vamos a calcular el estadístico F, el promedio de los cuadrados (MSA Y MSE), los grados de libertad y el valor de p (valor de significancia) para determinar si nuestros grupos son estadísticamente diferentes

MSA = *SSA/k-1* Donde k-1 es igual a los grados de libertad

```
MSA <- SSA/1 #siendo k igual al número de grupos, que en nuestro caso es 2
```

MSE = *SSE/k(n-1)* Donde k(n-1) es igual a los grados de libertad

```
MSE <- SSE/58 #en este caso, n es igual a 30 o número de observaciones por grupo
```

9.0.2 Estadístico F

```
Fval <- MSA/MSE
Fval
- [1] 142.5467
```

Valor de significancia pf = qf(Fval, k-1, k(n-1))

```
sig <- 1-pf(Fval, 1, 58)
sig
- [1] 0
```

Usualmente, valores pequeños de p provienen de valores de F grandes, y estos a su vez sugieren una varianza mas grande entre grupos que dentro de cada grupo. En nuestro experimento, se puede concluir que la sustancia "z" está teniendo afectando el crecimiento de las plántulas de manera positiva.

Finalmente, podemos usar el R2 para determinar cuanta varianza en nuestros datos (y) está siendo explicada por la variable independiente (x)

```
R2 = 1 - (SSE/SST)
R2 # es la varianza que puede ser explicada por las diferencias en las medias
- [1] 0.7107905
```

Felicitaciones, hizo una ANOVA por usted mismo(a), paso a paso. De ahora en adelante, puede usar las funciones de R!

Ejercicio 1

1. Crear tres vectores de datos aleatorios, cada uno de 40 observaciones que se diferencien por sus valores medias: entre g1 y g2 por un valor de 2 y entre g2 y g3 de un valor de 5, las cuales vienen de una distribución normal con una sd=2. Concatenar ambos vectores.
2. Crear un vector "y" en el cuál se repitan tres letras (A,B,C), cada una igual al número de observaciones de cada vector creado en el punto 1.
3. Realizar una anova con los vectores creados anteriormente e identificar si la varianza entre los grupos es significativamente diferente.

9.0.3 Anova con funciones de R

En R, usamos la función aov() para hacer una anova rápidamente

?aov()

```
aov(y ~ x)
- Call:
-    aov(formula = y ~ x)
-
- Terms:
-                         x Residuals
- Sum of Squares  435.4493  177.1775
- Deg. of Freedom        1        58
-
- Residual standard error: 1.747794
- Estimated effects may be unbalanced
```

Si quieremos ver el valor de F y el valor de p, utilizamos la función "summary"

```
summary(aov(y ~ x))
-             Df Sum Sq Mean Sq F value Pr(>F)
- x            1  435.4   435.4   142.5 <2e-16 ***
- Residuals   58  177.2     3.1
- ---
- Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Vemos que el valor de F es igual al que calculamos anteriormente, y también el valor de p (Pr(>F)) es significativo y recahzamos la hipótesis nula H0.

Hasta este punto, hemos realizado un anova de una vía, ya que solo determinamos el efecto de una sola variable independiente, lo que es igual a realizar una regresión lineal simple, ya que Anova es un método que ocurre dentro del modelo de la regresión lineal.

```
summary(lm(y ~ x))
-
- Call:
- lm(formula = y ~ x)
-
- Residuals:
-     Min      1Q  Median      3Q     Max
- -4.6043 -1.0785 -0.0917  1.0488  3.5696
-
- Coefficients:
-             Estimate Std. Error t value Pr(>|t|)
- (Intercept)  15.2418     0.7135   21.36   <2e-16 ***
- x            -5.3879     0.4513  -11.94   <2e-16 ***
- ---
- Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
-
```

```
- Residual standard error: 1.748 on 58 degrees of freedom
- Multiple R-squared:  0.7108,  Adjusted R-squared:  0.7058
- F-statistic: 142.5 on 1 and 58 DF,  p-value: < 2.2e-16
```

9.0.4 Anova con datos reales

```
library(tidyverse)
```

En esta sección vamos a explorar el análisis de anova con un set de datos que contiene información taxonómica de especies vegetales, abundancia e información del diametro promedio de cada especie colectada en diferentes parcelas. Estos datos hacen parte de la fundación COLTREE, la cual se dedica al monitoreo de parcelas permanentes de vegetacion a nivel nacional, con el proposito de comprender el funcionamiento de los bosques en terminos de su productividad vegetal y su relacion con el ciclo del carbono.

```
dat <- read_csv("Species_Coltree.csv")
- Parsed with column specification:
- cols(
-   Parcela = col_double(),
-   N_Familia = col_character(),
-   N_cientifico = col_character(),
-   Genero = col_character(),
-   Epiteto = col_character(),
-   Cod_HaCr = col_character(),
-   DAP = col_double()
- )
```

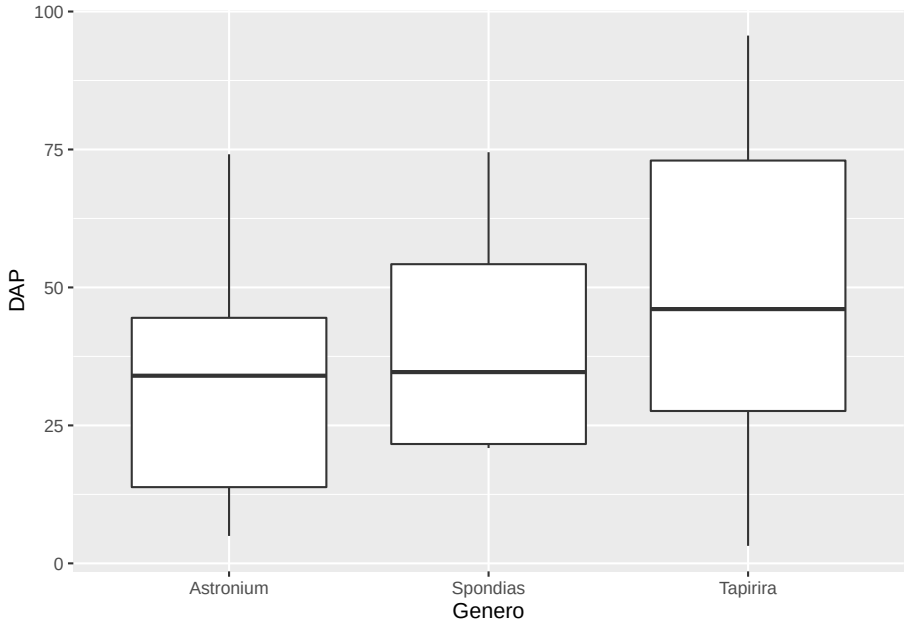
Utilizando la función de Anova en R, vamos a evaluar si existen diferencias en el diametro entre los géneros de plantas de una misma familia. En este ejemplo, vamos a trabajar con mas de dos grupos ya que es más frecuente utilizar otra prueba (T-Test) para comparar solo dos grupos.

Escogemos la familia Anacardiaceae

```
familia <- dat %>%
  filter(N_Familia == "ANACARDIACEAE")
```

Visualizamos las medias de los géneros

```
familia %>%
  ggplot(aes(x = Genero, y = DAP)) +
  geom_boxplot()
```



Realizamos la Anova

```
anova <- aov(familia$DAP ~ familia$Genero)

summary(anova)
-                 Df Sum Sq Mean Sq F value Pr(>F)
- familia$Genero   2   3847  1923.4   2.746 0.0668 .
- Residuals      184 128863   700.3
- ---
- Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

En este ejemplo, vemos que tenemos un valor de F bajo, y un valor de p>0.05, por lo que concluimos que el diametro entre los géneros de esta familia no difieren entré si significativamente. Sin embargo, con summary no obtenemos información sobre cuáles grupos se están diferenciando o no. Para esto, hacemos uso de una prueba post-hoc o "Tukey's Honest Significant Differences" (HSD) para ayudarnos a cuantificar cuales grupos difieren entre ellos.

?TukeyHSD()

```
TukeyHSD(anova)
-   Tukey multiple comparisons of means
-     95% family-wise confidence level
-
- Fit: aov(formula = familia$DAP ~ familia$Genero)
```

```
-
- $`familia$Genero`
-                          diff         lwr      upr     p adj
- Spondias-Astronium  7.390421 -27.5649622 42.34580 0.8716087
- Tapirira-Astronium 15.816341  -0.5479194 32.18060 0.0606142
- Tapirira-Spondias   8.425920 -23.2113415 40.06318 0.8042146
```

Podemos ver un valor de p asociado a la interacción entre grupos. Vemos que los géneros que más difieren son Tapirira y Astronium, sin embargo las medias no son significativamente diferente

Ejercicio 2

1. Realizar un Anova para evaluar si existen diferencias en el diametro entre las familias de dos parcelas diferentes y realice una prueba post hoc

Chapter 10

Regresión lineal simple

En esta sesión vamos a conocer los fundamentos básicos para realizar una regresión lineal y como se realiza en R.

10.0.1 Regresión lineal

Mediante la regresión lineal, podemos predecir el valor de una variable depentiente "y" en función de una variable independiente "x" mediante la ecuación de la recta (yi= + *xi+ i). Además, la regresión lineal nos sirve como análisis exploratorio para observar si dos o mas variables se relacionan linealmente (si 'x' cambia entonces 'y' también cambiará en una magnitud similar)

A diferencia de la Anova, nuestra variable independiente "x" es usualmente una variable contínua y no una variable categórica, pero su notación es la misma:

$lm(y \sim x)$

Como vimos anteriormente, utilizamos la función lm (linear model)

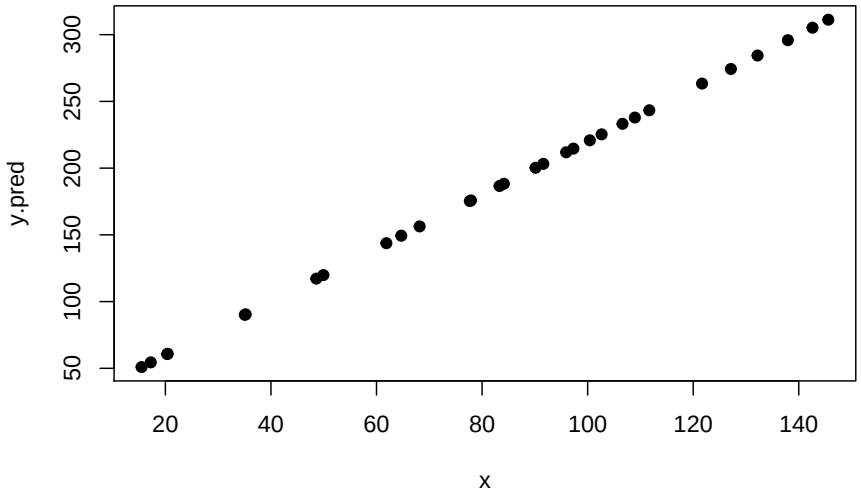
Vamos a crear datos para realizar una regesión lineal simple. Para esto, debemos crear 3 componentes: respuesta (y) = parte deterministico (x) + parte estocástico (error).

Creamos la variable deterministica, es decir, datos definidos los cuales conocemos (30 valores aleatorios entre 15 y 150)

```
x <- runif(30,15,150)
x
-  [1]  77.89806  49.93597 100.41606  91.60555 132.19549  48.59789  17.23751  35.21655  20.44315
- [10] 106.60689  61.87017  68.17134  95.93525 121.67831  20.34498  97.30020  83.30630  77.68414
- [19] 145.59537  90.13274  35.05001 108.94690 102.65441 142.60856  64.68734  84.13185 111.67959
- [28]  15.48433 137.93079 127.14179
```

Creamos la variable de respuesta (parte deterministica)

```
y.pred <- 20 + 2 * x
plot(x, y.pred, pch = 19)
```




```
summary(lm(y.pred ~ x)) # Obtenemos un R2 de 1 (ajuste perfecto), valor de p significa
- Warning in summary.lm(lm(y.pred ~ x)): essentially perfect fit: summary may be unreli
-
- Call:
- lm(formula = y.pred ~ x)
-
- Residuals:
-        Min          1Q      Median          3Q         Max
- -1.762e-13 -9.750e-16  1.705e-15  6.667e-15  1.085e-13
-
- Coefficients:
-              Estimate Std. Error   t value Pr(>|t|)
- (Intercept) 2.000e+01  1.708e-14 1.171e+15   <2e-16 ***
- x           2.000e+00  1.879e-16 1.065e+16   <2e-16 ***
- ---
- Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
-
- Residual standard error: 3.951e-14 on 28 degrees of freedom
- Multiple R-squared:      1,   Adjusted R-squared:      1
- F-statistic: 1.133e+32 on 1 and 28 DF,  p-value: < 2.2e-16
```

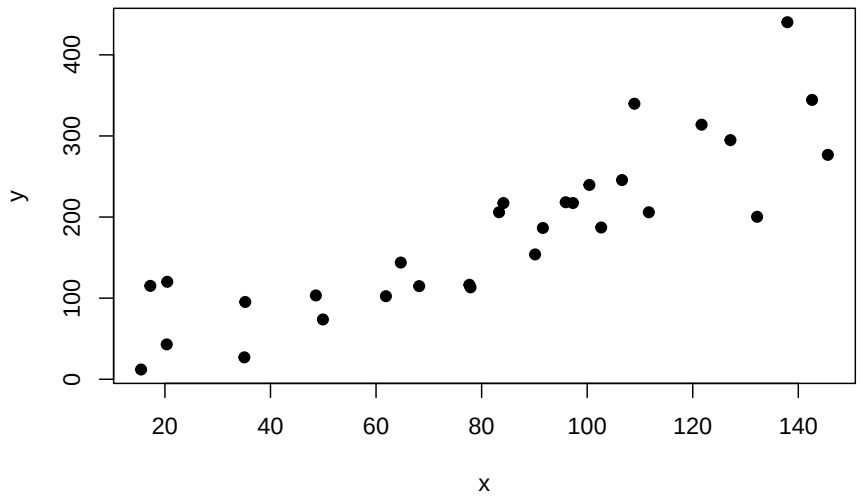
Cuando trabajamos con datos reales, casi nunca encontramos un ajuste perfecto porque hay un error asociado a los datos que tiene su origin en otras factores que influyen la relación entre y y x (incluso errores en la medicion)

Ahora añadimos el componente estocástico (el error) a la variable respuesta:

```
y <- y.pred + rnorm(30,0,50) # el error viene de una distribucion normal con una media de 0
```

Observamos mediante un gráfico como se relacionan ahora nuestras variables

```
plot(y ~ x, pch = 19)
```




```
summary(lm(y ~ x)) # Ahora el ajuste de la regresión es menor debido al error en y
-
- Call:
- lm(formula = y ~ x)
-
- Residuals:
-     Min      1Q  Median      3Q     Max
- -93.860 -39.382   0.865  21.011 133.171
-
- Coefficients:
```

```
-             Estimate Std. Error t value Pr(>|t|)
- (Intercept)   -3.007     22.276  -0.135    0.894
- x              2.248      0.245   9.176  6.2e-10 ***
- ---
- Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
-
- Residual standard error: 51.52 on 28 degrees of freedom
- Multiple R-squared:  0.7504,  Adjusted R-squared:  0.7415
- F-statistic:  84.2 on 1 and 28 DF,  p-value: 6.2e-10
```

10.0.2 Regresión a mano

Igual que en un anova, podemos calcular los parámetros de la regresión a mano mediante el método de la suma de cuadrados y la información del promedio de los datos.

Recordemos que hacemos uso de la ecuación de la recta = yi= + *xi+ i, en los que debemos calcular algunos parámetros con el método de la suma de cuadrados. En esta ecuación se nos presentan varios componentes:

- *yi* o variable respuesta
- o intercepto, el cual es el valor de y cuando x = 0
- o pendiente (el valor de p que obtenemos nos dice si la pendiente es significativa diferente de 0 o no)
- *xi* o variable predictora
- *i* o el error. Todos los modelos estadísticos tienen una varianza residual de información observada no tenida en cuenta en nuestros predictores y está direcamente relacionada con la variable respuesta a través de una distribución (típicamente una distribución normal, por lo menos para cumplir los supuestos de la regresión lineal). Para tener un mejor ajuste de nuestro modelo, debemos entender como se genero nuestra variable dependiente. De esta manera, nos aseguramos de escoger una distribución que describa nuestros datos y que capture el error residual correctamente (funcion rnorm())

Vamos a calcular la suma de cuadrados de nuestros datos

```
SSY <- sum((y - mean(y))^2) # sumas de cuadrados corregidas en y

SSX <- sum((x - mean(x))^2) # sumas de cuadrados corregidas en x

SSXY <- sum((x - mean(x)) * (y - mean(y))) # suma corregida de productos (covarianza d
```

Pendiente:

```
b <- SSXY/SSX # máxima probabilidad de pendiente b
b
- [1] 2.248206
```

Intercepto:

```
a <- mean(y) - b * mean(x) # Usamos la ecuación y valores promedio para calcular el intercepto
a
- [1] -3.006865
```

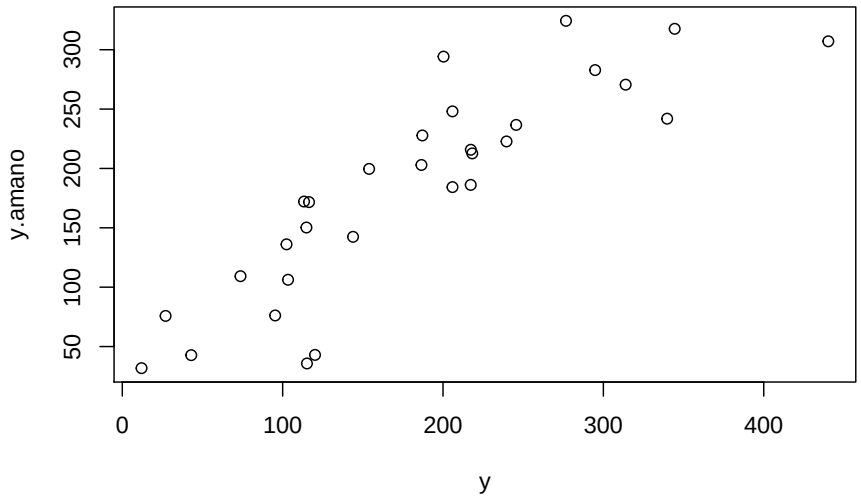
Comprobamos:

```
summary(lm(y ~ x))
-
- Call:
- lm(formula = y ~ x)
-
- Residuals:
-     Min      1Q  Median      3Q     Max
- -93.860 -39.382   0.865  21.011 133.171
-
- Coefficients:
-             Estimate Std. Error t value Pr(>|t|)
- (Intercept)   -3.007     22.276  -0.135    0.894
- x              2.248      0.245   9.176  6.2e-10 ***
- ---
- Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
-
- Residual standard error: 51.52 on 28 degrees of freedom
- Multiple R-squared:  0.7504,  Adjusted R-squared:  0.7415
- F-statistic:  84.2 on 1 and 28 DF,  p-value: 6.2e-10
```

De esta forma hemos calculado los parámetros que desconociamos en la ecuación de la recta. De esta forma tenemos lo siguiente:

y = + *x Esto significa que cuando x = 0, "y" es igual al valor en , y la variable respuesta "y" aumenta por cada unidad en x

```
y.amano = a + b*x
plot(y, y.amano)
```



Si queremos predecir el valor de y cuando x = 100 utilizando nuestra ecuación:

```
y.result <-  a + b*100
```

Ejercicio 1

1. Utilizando los siguientes datos calcular el intercepto y la pendiente a mano mediante la suma de cuadrados:

x <- runif(20, 1, 20) y.pred <- 20 + 2 * x y <- y.pred + + rnorm(20,0,5)

2. Realizar una interpretación de los valores de la pentiente e intercepto teniendo en cuenta la siguiente información: Asuma que la variable “y” son datos del peso de roedores y que la variable “x” es información del tamaño de estos roedores

3. Utilizar la función de R para realizar la regresión lineal e interpretar el valor de R2 y valor de p

10.0.3 Indices de diversidad y regresión lineal

Ahora que conocemos como realizar una regresión lineal, vamos a utilizar los datos de la fundación COLTREE para explorar algunos índices que miden la biodiversidad y reaizar regresiones lineales con estos resultados.

En adición al set de datos taxonómicos, trabajaremos con datos medioambientales de cada parcela

Cargamos los datos

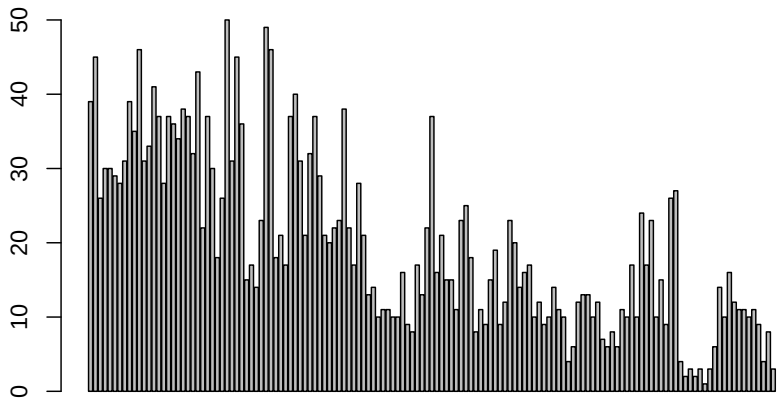
```
library(tidyverse)
dat_spec <- read_csv("Species_Coltree.csv")
dat_env <- read_csv("Env_Coltree.csv")
```

Algunas de las principales variables medioambientales

- Temp_media = Temperatura media anual
- Rango_diurno = Rango diurno promedio
- Est_T = Estacionalidad de la temperatura
- MaxTemp = Temperatura maxima del mes mas calido
- MinTemp = Temperatura minima del mes mas frio
- Rango_anual_T = Rango anual de temperatura
- T_media_tmh = Temperatura media del trimestre mas humedo
- T_media_tms = Temperatura media del trimestre mas seco
- T_media_tmc = Temperatura media del trimestre mas calido
- T_media_tmf = Temperatura media del trimestre mas frio
- Precip_anual = Precipitacion anual
- Precip_max = Precipitacion del mes mas humedo
- Precip_min = Precipitacion del mes mas seco
- Est_P = Estacionalidad de la precipitacion (Coeficiente de Variacion)

Para calcular la riqueza de especies (diversidad), debemos contar el número de especies presentes en cada sitio (parcela en nuestro caso)

```
conteo_parcela <- dat_spec %>%
  group_by(Parcela) %>%
  select(N_cientifico) %>%
  unique() %>%
  summarise(n = n())
- Adding missing grouping variables: `Parcela`
- `summarise()` ungrouping output (override with `.groups` argument)
barplot(conteo_parcela$n)
```




```
riqueza <- conteo_parcela$n
```

En caso de contar con una matriz de abundancias, podemos convertir la matriz una matriz de presencia/ausencia y contar el número de especies; o utilizar el paquete vegan:

Convertimos nuestros datos a una matriz de abundancias:

```
dat_abun <- dat_spec %>%
  select(Parcela, N_cientifico) %>% #Seleccionamos las variables deseadas
  group_by(Parcela, N_cientifico) %>% # Agrupamos las observaciones por parcela y nomb
  summarise(freq = n()) %>% # contamos el número de individuos para obtener la abundan
  pivot_wider(names_from = N_cientifico, values_from = freq) # Mediante la función piv
- `summarise()` regrouping output by 'Parcela' (override with `.groups` argument)
```

Reemplazamos los NA por 0

```
dat_abun <- dat_abun %>%
  replace(is.na(dat_abun), 0)
```

10.0.4 Paquete vegan

install.packages("vegan")

```
library(vegan)
```

Dado que vegan trabaja con data.frames debemos transformar el tibble a un data.frame y seguir con programar en R (y no tidyverse)

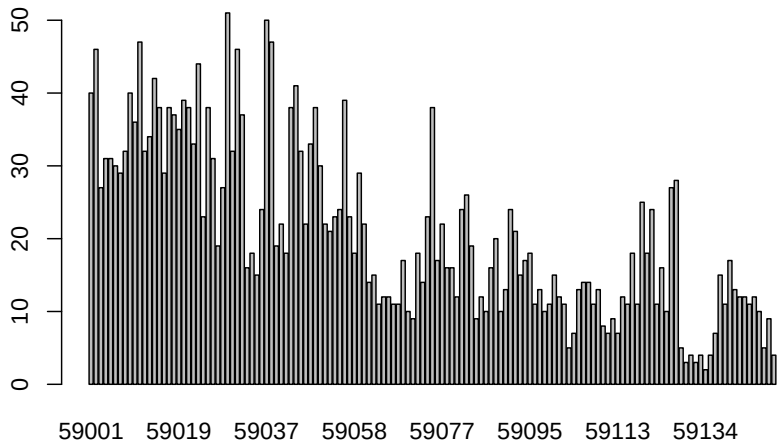
```
abd <- as.data.frame(dat_abun)
```

Asignar la primera columna (Parcela) como nombre de las filas

```
rownames(abd) <- abd[,1]
dat_abun <-dat_abun[,-1]
```

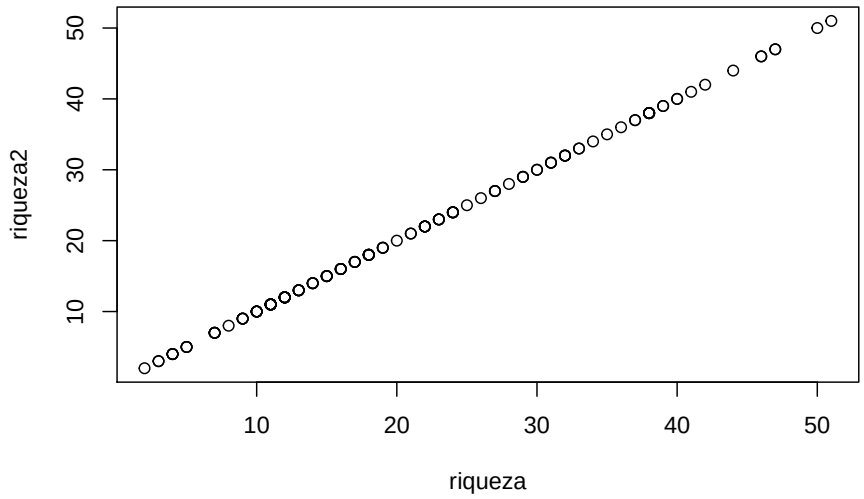
Contar especies con una matriz de presencia/ausencia

```
pa <- replace(abd, abd > 0, 1) # Convertimos la matriz de abundancias en una matriz de presencia/

riqueza <- apply(pa, 1, sum) # Sumamos las especies presentes en cada parcela para estimar su riq

barplot(riqueza)
```



Tambien podemos generar la matriz de abundancias de manera tradicional utilizando tabla de conteo:

```
abd2 <- as.data.frame.matrix(table(dat_spec$Parcela, dat_spec$N_cientifico))
riqueza2 <- specnumber(abd)

# verificamos si las dos vías conducen al mismo resultado
plot(riqueza, riqueza2)
```

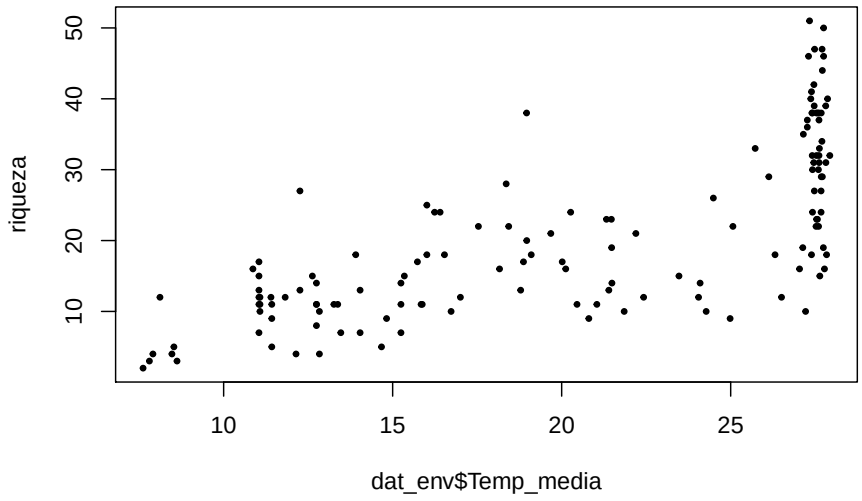


A partir de la matriz de abundacias, podemos calcular índices de diversidad como shannon o simpson mediante la función diversity() del paquete vegan. Utilice la función diversity() para calcular el indice de shannon.

```
div <- diversity(dat_abun, index = "shannon")
```

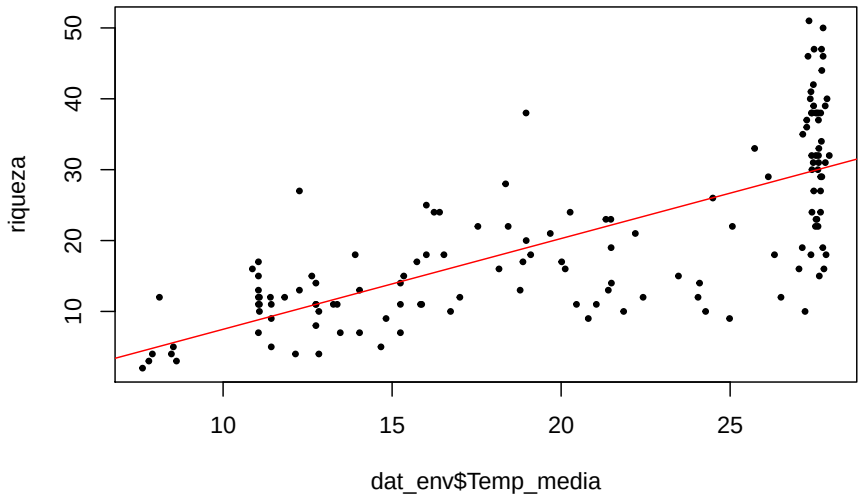
Ahora Vamos a relacionar la riqueza de especies (o diversidad) con algunos de los factores ambientales de las parcelas.

```
plot(riqueza ~ dat_env$Temp_media, pch=19, cex=0.5)
```



Riqueza ~ T_media

```
reg <- lm(riqueza ~ dat_env$Temp_media) # Ajustamos un modelo con la riqueza de especies y la tem
summary(reg)
-
- Call:
- lm(formula = riqueza ~ dat_env$Temp_media)
-
- Residuals:
-      Min       1Q   Median       3Q      Max
- -19.5299  -4.9216   0.0015   4.8653  21.3207
-
- Coefficients:
-                    Estimate Std. Error t value Pr(>|t|)
- (Intercept)         -5.3239     2.1833  -2.438    0.016 *
- dat_env$Temp_media   1.2806     0.1015  12.611   <2e-16 ***
- ---
- Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
-
- Residual standard error: 8.186 on 139 degrees of freedom
- Multiple R-squared:  0.5336,  Adjusted R-squared:  0.5303
- F-statistic:   159 on 1 and 139 DF,  p-value: < 2.2e-16
plot(riqueza ~ dat_env$Temp_media, pch=19, cex=0.5)
abline(reg, col = "red")
```



Ejercicio 2

1. Con base a los resultados de la regresión lineal, reponda las siguientes preguntas:

- ¿La riqueza de especies depende de la temperatura?
- ¿Cuál resultado de la regresión demuestra eso?
- ¿Cuánta varianza explica la temperatura en la variación de la riqueza de especies?

2. Añadir etiquetas a los ejes de la gráfica anterior e insertar una leyenda con el valor de R^2 y la ecuación de la línea de regresión.

Ejercicio 3

1. Realizar regresiones con las otras variables ambientales y determinar cuales de ellas pueden explicar la variabilidad en riqueza a traves del gradiente altitudinal.

2. Repetir el mismo analisis pero ahora con el Indice de Shannon como variable independiente. Los resultados son congruentes?

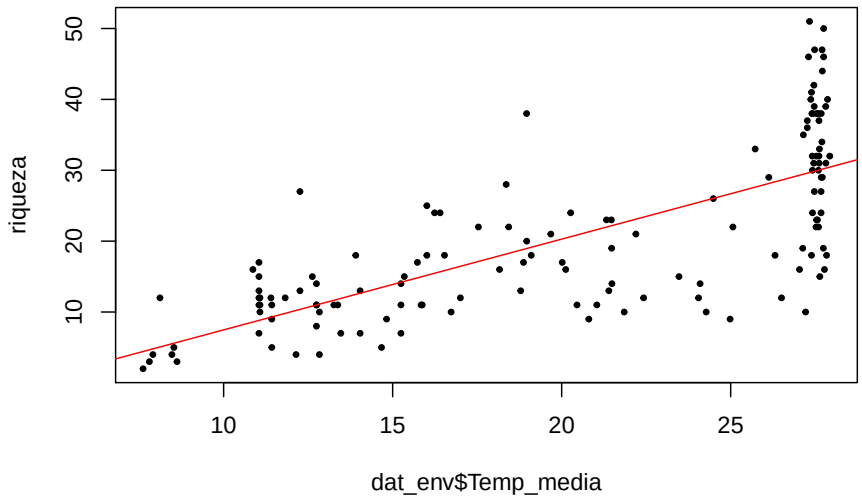
3. Mostrar los resultados de todos estos analisis en gráficos de dispersión con una línea de tendencia y su respectiva leyenda.

10.0.5 Supuestos de la regresión lineal

Para poder validar los resultados de nuestra regresión lineal, debemos saber si nuestro modelo cumple con los supuestos o requisitos asociados al modelo:

- *Linealidad:* Debido a que la ecuación de la recta que se emplea en la regresión lineal, la relación entre las variables dependientes e independientes debe ser lineal. Esto puede ser observado fácilmente mediante el diagrama de dispersión:

```
plot(riqueza ~ dat_env$Temp_media, pch=19, cex=0.5)
lm <- lm(riqueza ~ dat_env$Temp_media)
abline(lm, col="red")
```



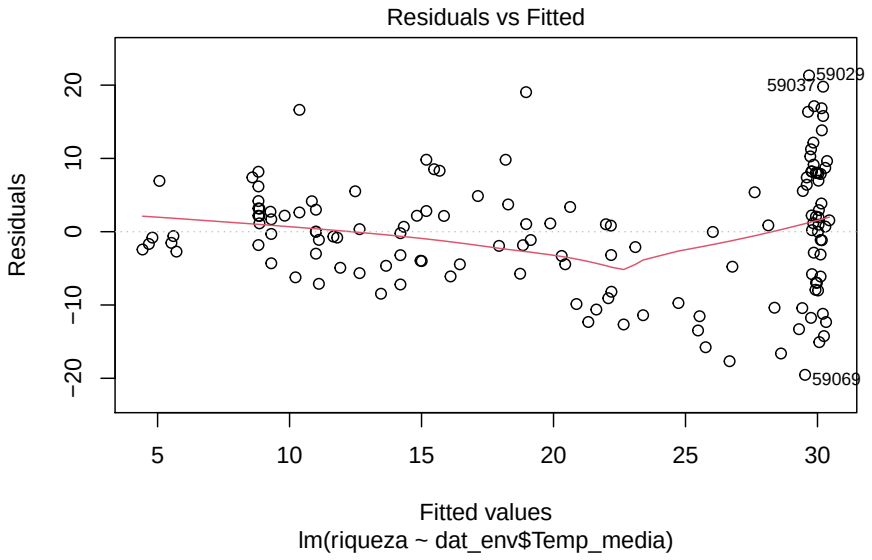
Podemos observar que es probable que los datos no se relacionan linealmente y ajustar una relación lineal podría resultar en resultados erroneos

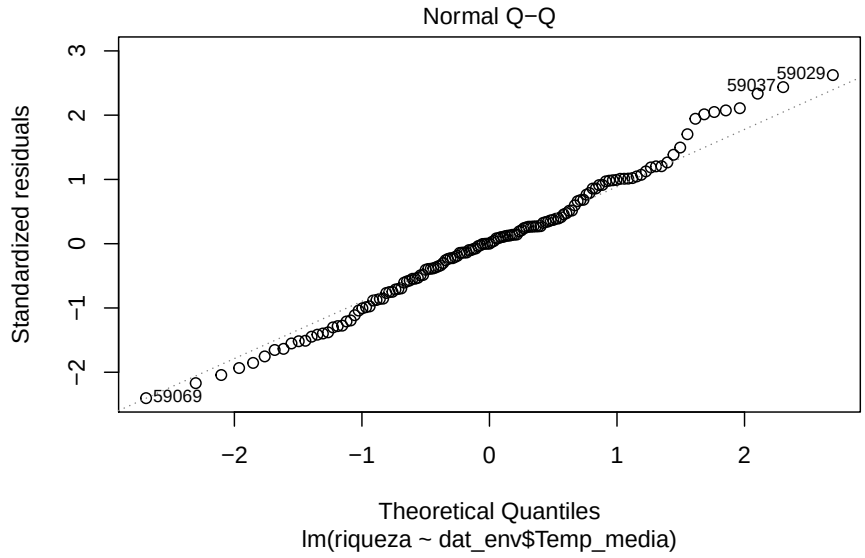
- *Independencia de los residuos:* Los residuos deben ser independientes entre sí y representan una variable aleatoria, es decir, que la predicción de un valor los residuos no es afectada por los residuos cercanos. Podemos probar este supuesto mediante el test de durbin-watson:

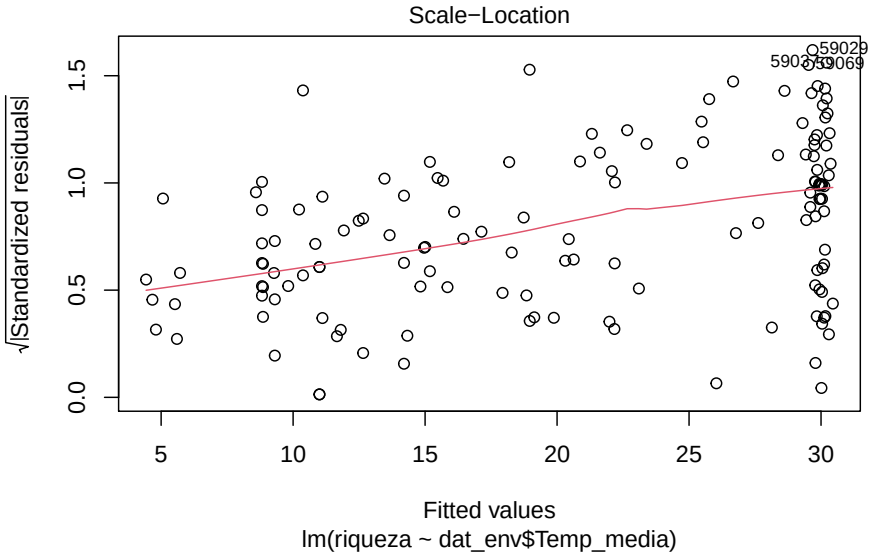
```
library(lmtest)

dwtest(lm) # En esta prueba, valores de DW alrededor de 2 y p significativos significa
-
-   Durbin-Watson test
-
- data:  lm
- DW = 1.2752, p-value = 5.06e-06
- alternative hypothesis: true autocorrelation is greater than 0
```

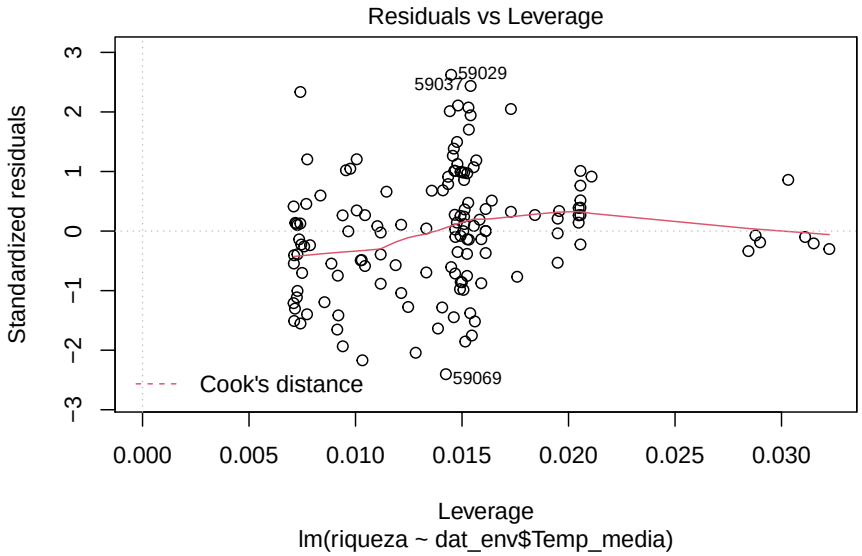
- *Residuos constantes "homocedasticidad":* Los residuos (diferencia entre los valores observados de la variable dependiente respecto a los valores predecidos mediante la ecuación de la recta) deben ser constantes. Esto quiere decir que los residuos no aumentan ni disminuyen a medida que se predicen valores más grandes o pequeños. Podemos observar este supuesto graficando nuestro vector de regresión lineal:

```
plot(lm) # Al graficar esto presionamos enter en la consola y debemos observar la prim
```



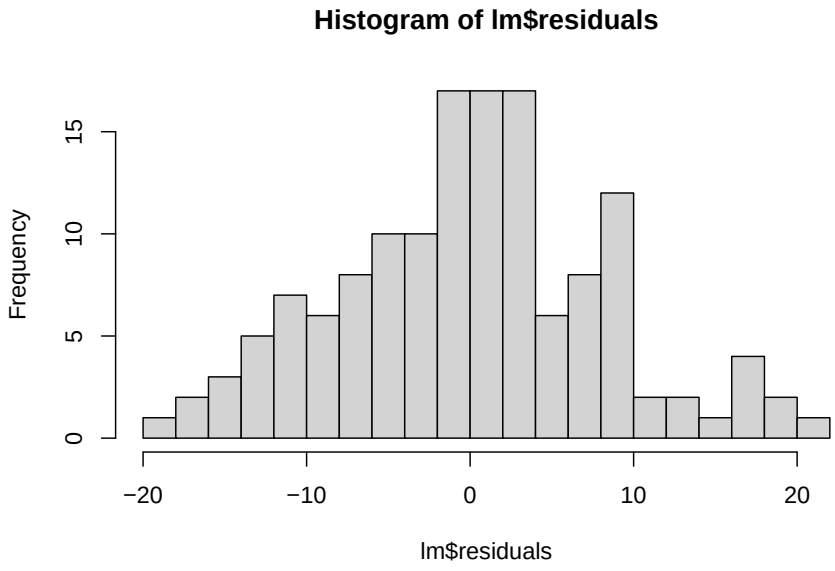

Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(riqueza ~ dat_env$Temp_media)
59029
59037
59069


Scale−Location
√|Standardized residuals|
Fitted values
lm(riqueza ~ dat_env$Temp_media)
59029
59069



Normalidad de residuos: Los residuos deben presentan una distribución normal. Mediante la gráfica de histograma y la prueba de shaphiro-wilk podemos probar este supuesto:

```
hist(lm$residuals, 15) # Los residuos parecen ajustarse a una distribución normal
```



?shapiro.test

```
shapiro.test(lm$residuals) # Con el valor de p de la prueba, aceptamos la hipótesis nula ya que p
-
-   Shapiro-Wilk normality test
-
- data:  lm$residuals
- W = 0.99037, p-value = 0.4452
```

Chapter 11

Regresión lineal múltiple

En esta sesión vamos explorar la regresión lineal múltiple, la colinealidad entre variables y la selección de modelos en R.

11.0.1 Regresión lineal múltiple

En la sesión pasada, vimos como ajustabamos la ecuación de la recta para observar la relación lineal entre una variable dependiente "y" y una única varible independiente "x". En la regresión lineal múltiple, el modelo lineal puede ajustarse con más de una variable indepentiente "x".

Cargamos nuevamente de la fundación COLTREE

```
library(tidyverse)

dat_spec <- read_csv("Species_Coltree.csv")

dat_env <- read_csv("Env_Coltree.csv")
```

Vamos a revisar nuevamente algunas variables ambientales que predicen mejor la riqueza de especies.

Riqueza de especies

```
conteo_parcela <- dat_spec %>%
  group_by(Parcela) %>%
  select(N_cientifico) %>%
  unique() %>%
  summarise(n = n())
- Adding missing grouping variables: `Parcela`
```

```
- `summarise()` ungrouping output (override with `.groups` argument)
riqueza <- conteo_parcela$n
```

Observemos la relación lineal simple con algunas de las variables ambientales

```
summary(lm(riqueza ~ Altitud, data = dat_env))
-
- Call:
- lm(formula = riqueza ~ Altitud, data = dat_env)
-
- Residuals:
-      Min       1Q   Median       3Q      Max
- -19.0114  -7.1204  -0.4536   5.0176  22.9118
-
- Coefficients:
-              Estimate Std. Error t value Pr(>|t|)
- (Intercept) 27.9343228  1.1143402   25.07   <2e-16 ***
- Altitud     -0.0070511  0.0007003  -10.07   <2e-16 ***
- ---
- Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
-
- Residual standard error: 9.115 on 139 degrees of freedom
- Multiple R-squared:  0.4217,  Adjusted R-squared:  0.4176
- F-statistic: 101.4 on 1 and 139 DF,  p-value: < 2.2e-16
summary(lm(riqueza ~ Temp_media, data = dat_env))
-
- Call:
- lm(formula = riqueza ~ Temp_media, data = dat_env)
-
- Residuals:
-      Min       1Q   Median       3Q      Max
- -19.5299  -4.9216   0.0015   4.8653  21.3207
-
- Coefficients:
-             Estimate Std. Error t value Pr(>|t|)
- (Intercept)  -6.3239     2.1833  -2.896  0.00439 **
- Temp_media    1.2806     0.1015  12.611  < 2e-16 ***
- ---
- Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
-
- Residual standard error: 8.186 on 139 degrees of freedom
- Multiple R-squared:  0.5336,  Adjusted R-squared:  0.5303
- F-statistic:   159 on 1 and 139 DF,  p-value: < 2.2e-16
summary(lm(riqueza ~ Prec_anual , data = dat_env))
-
```

```
- Call:
- lm(formula = riqueza ~ Prec_anual, data = dat_env)
-
- Residuals:
-     Min      1Q  Median      3Q     Max
- -20.655  -5.422  -0.453   5.479  20.215
-
- Coefficients:
-              Estimate Std. Error t value Pr(>|t|)
- (Intercept) -4.808152   1.910959  -2.516    0.013 *
- Prec_anual   0.012908   0.000941  13.717   <2e-16 ***
- ---
- Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
-
- Residual standard error: 7.813 on 139 degrees of freedom
- Multiple R-squared:  0.5751,  Adjusted R-squared:  0.5721
- F-statistic: 188.2 on 1 and 139 DF,  p-value: < 2.2e-16
summary(lm(riqueza ~ Prec_min, data = dat_env))
-
- Call:
- lm(formula = riqueza ~ Prec_min, data = dat_env)
-
- Residuals:
-     Min      1Q  Median      3Q     Max
- -25.529  -5.641  -1.679   4.446  31.146
-
- Coefficients:
-             Estimate Std. Error t value Pr(>|t|)
- (Intercept)  5.49132    1.70880   3.214  0.00163 **
- Prec_min     0.28751    0.03046   9.438  < 2e-16 ***
- ---
- Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
-
- Residual standard error: 9.357 on 139 degrees of freedom
- Multiple R-squared:  0.3905,  Adjusted R-squared:  0.3862
- F-statistic: 89.07 on 1 and 139 DF,  p-value: < 2.2e-16
summary(lm(riqueza ~ Est_Temp, data = dat_env))
-
- Call:
- lm(formula = riqueza ~ Est_Temp, data = dat_env)
-
- Residuals:
-     Min      1Q  Median      3Q     Max
- -20.778  -9.485  -1.801   7.867  28.351
-
```

```
- Coefficients:
-             Estimate Std. Error t value Pr(>|t|)
- (Intercept) 29.53475    3.27525   9.018 1.38e-15 ***
- Est_Temp    -0.28556    0.09172  -3.113  0.00225 **
- ---
- Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
-
- Residual standard error: 11.59 on 139 degrees of freedom
- Multiple R-squared:  0.06518, Adjusted R-squared:  0.05846
- F-statistic: 9.692 on 1 and 139 DF,  p-value: 0.002247
summary(lm(riqueza ~ Est_Prec, data = dat_env))
-
- Call:
- lm(formula = riqueza ~ Est_Prec, data = dat_env)
-
- Residuals:
-     Min      1Q  Median      3Q     Max
- -19.827  -7.250  -1.863   6.813  31.611
-
- Coefficients:
-             Estimate Std. Error t value Pr(>|t|)
- (Intercept)  55.1167     7.2338   7.619 3.58e-12 ***
- Est_Prec     -0.7351     0.1493  -4.923 2.38e-06 ***
- ---
- Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
-
- Residual standard error: 11.06 on 139 degrees of freedom
- Multiple R-squared:  0.1485,  Adjusted R-squared:  0.1423
- F-statistic: 24.24 on 1 and 139 DF,  p-value: 2.38e-06
```

Es probable que haya una combinacion de variables (por ejemplo, la interacción entre la temperatura y la precipitación) que predice mejor la riqueza de especies. Ahora, esta vez usaremos la regresión múltiple con lm():

```
reg <- lm(riqueza ~ . ,data = dat_env[,c(3:10)]) # el '.' se puede remplazar listando

summary(reg)
-
- Call:
- lm(formula = riqueza ~ ., data = dat_env[, c(3:10)])
-
- Residuals:
-      Min       1Q   Median       3Q      Max
- -22.2358  -4.5579  -0.5811   4.7145  21.1356
-
```

```
- Coefficients:
-                     Estimate Std. Error t value Pr(>|t|)
- (Intercept)        -4.030603  16.391021  -0.246   0.8061
- Altitud             0.001305   0.001461   0.893   0.3734
- Temp_media          0.403589   0.488264   0.827   0.4100
- Rango_diurno_Temp -14.397146   6.956396  -2.070   0.0404 *
- Est_Temp           -0.531457   0.283209  -1.877   0.0628 .
- Prec_anual          0.010430   0.004217   2.473   0.0147 *
- Rango_anual_Temp   14.081350   7.558751   1.863   0.0647 .
- Prec_min            0.009222   0.048913   0.189   0.8507
- Est_Prec           -0.030856   0.140044  -0.220   0.8260
- ---
- Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
-
- Residual standard error: 7.857 on 132 degrees of freedom
- Multiple R-squared:  0.592,   Adjusted R-squared:  0.5673
- F-statistic: 23.94 on 8 and 132 DF,  p-value: < 2.2e-16
```

Esta vez, en la regresión lineal múltiple existen menos predictores significativos, mientras que en las regresiones simples varios todos los predictores resultan significativos. Esto puede estar siendo causado por la colinealidad entre las variables independientes, es decir, que las variables predictoras deben ser independientes entre sí y no están linealmente relacionadas. Al igual que en la regresión simple, para validar nuestros resultados es necesario cumplir con los supuestos estadísticos, dentro de los cuales se incluye la no colinealidad.

11.0.2 Colinealidad

Es posible observar como resultado de la colinealidad, valores altos de r2 con predictores no significarivos. Podemos explorar los valores de correlación entre pares de variables para detectar posible colinealidad, sin embargo, a pesar de que dos variables tengan valores de correlación bajo, esto no asegura la no colinealidad entre ellas.

Correlación entre pares de variables mediante la cor():

Mediante la correlación determinamos la relación lineal entre dos variables, y a diferencia de la regresión lineal, no tiene que haber una variable dependiente y otra independiente.

?cor

```
cor(dat_env$Altitud, dat_env$Temp_media)
- [1] -0.8941604
```

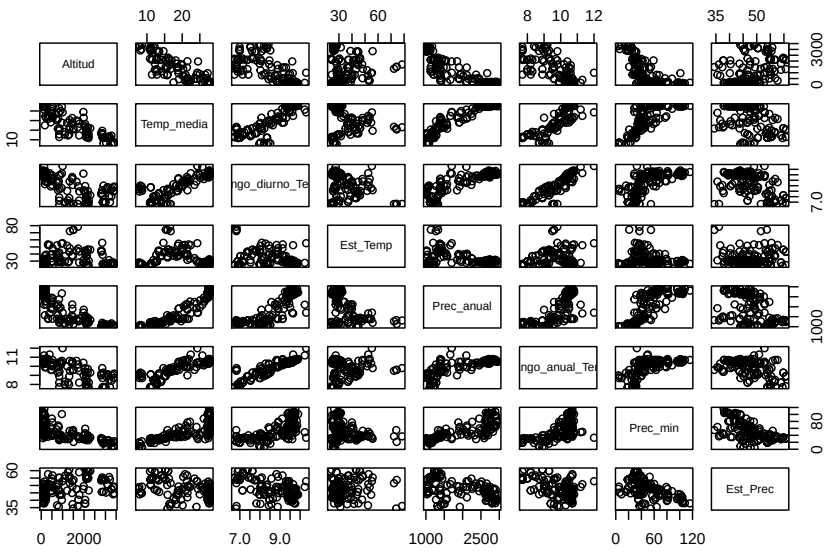
El coeficiente de relación "r" resultante puede variar entre -1 a 1. Mientras mas cerca este este valor a un extremo, más corelacionados estan las dos variables entre ellas, ya sea una correlación positiva (cuando una variable aumenta la igual aumenta de la igual manera), o correlación negativa (mientras que una variable disminuye la otra aumenta). Si obtenemos el cuadrado de r, podremos obtener el porcentaje de variación de una variable en relación a la varciación de la otra.

Vamos a realizar múltiples correlaciones

```
cor(dat_env[,3:10])
-                      Altitud Temp_media Rango_diurno_Temp    Est_Temp Prec_anual
- Altitud            1.0000000 -0.8941604        -0.7206622  0.10635396 -0.8506740
- Temp_media        -0.8941604  1.0000000         0.8738675 -0.17105184  0.9466035
- Rango_diurno_Temp -0.7206622  0.8738675         1.0000000 -0.28490057  0.8474429
- Est_Temp           0.1063540 -0.1710518        -0.2849006  1.00000000 -0.3632199
- Prec_anual        -0.8506740  0.9466035         0.8474429 -0.36321989  1.0000000
- Rango_anual_Temp  -0.7399973  0.8624162         0.9216675  0.09539631  0.7587150
- Prec_min          -0.7378962  0.7830482         0.6769452 -0.19008865  0.8165380
- Est_Prec           0.3953888 -0.4783876        -0.5153144  0.03317230 -0.4978007
-                   Rango_anual_Temp   Prec_min   Est_Prec
- Altitud                -0.73999728 -0.7378962  0.3953888
- Temp_media              0.86241624  0.7830482 -0.4783876
- Rango_diurno_Temp       0.92166747  0.6769452 -0.5153144
- Est_Temp                0.09539631 -0.1900886  0.0331723
- Prec_anual              0.75871496  0.8165380 -0.4978007
- Rango_anual_Temp        1.00000000  0.6452798 -0.5249194
- Prec_min                0.64527977  1.0000000 -0.5831187
- Est_Prec               -0.52491943 -0.5831187  1.0000000
```

Múltiples gráficas de correlaciones

```
pairs(dat_env[,3:10])
```



¿Cuáles de los predictores climáticos están fuertemente correlacionados?

Podemos hacer uso del Factor de Inflacion de la Varianza (VIF) para estimar cuanta varianza de un coeficiente de regresion es inflada debido a la colinearidad del modelo.

install.packages("car")

```
library(car)
?vif

v <- vif(reg)
v
-           Altitud        Temp_media Rango_diurno_Temp          Est_Temp        Prec_anual
-          5.854714         25.096705        120.481368         20.742716         19.860437
-  Rango_anual_Temp          Prec_min          Est_Prec
-        132.094409          3.657030          1.743197
```

Para entender los resultados de esta función debemos tener conocer como se calcula manualmente:

Primero debemos calcular el r2 de alguna variable frente a las demás:

```
rcuadrado <- summary(lm(dat_env$Altitud ~ ., data = dat_env[4:10]))$r.squared
rcuadrado
- [1] 0.8291975
```

```
1/(1-rcuadrado)
- [1] 5.854714
v[1]
-  Altitud
- 5.854714
```

VIF relaiza este proceso para cada variable independiente. En nuestro ejemplo, vemos que cuando la altura se utiliza como variable dependiente, el r2 del modelo es alto, lo que resulta en un valor de VIF cercano a 6. ¿Qué pasa entonces cuando el r2 es de 0.9?

```
1/(1-0.9)
- [1] 10
```

Generalmente valores de VIF de 10 indican alta colinealidad, sin embargo, valores de hasta 6 pueden reflejar una colinealidad en nuestros datos, por lo cual deberíamos seleccionar las variables con los valores de VIF más bajos.

```
v
-           Altitud        Temp_media Rango_diurno_Temp          Est_Temp        Prec_a
-          5.854714         25.096705        120.481368         20.742716         19.80
-  Rango_anual_Temp          Prec_min          Est_Prec
-        132.094409          3.657030          1.743197
fit1 <-  lm(riqueza ~ Altitud + Prec_min + Est_Prec, data = dat_env)

summary(fit1)
-
- Call:
- lm(formula = riqueza ~ Altitud + Prec_min + Est_Prec, data = dat_env)
-
- Residuals:
-      Min       1Q   Median       3Q      Max
- -19.1975  -5.9142  -0.4795   4.8080  25.5600
-
- Coefficients:
-              Estimate Std. Error t value Pr(>|t|)
- (Intercept) 23.415374   8.809118   2.658  0.00879 **
- Altitud     -0.004532   0.001002  -4.521 1.32e-05 ***
- Prec_min     0.131387   0.048022   2.736  0.00704 **
- Est_Prec    -0.102539   0.146332  -0.701  0.48466
- ---
- Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
-
- Residual standard error: 8.788 on 137 degrees of freedom
```

```
- Multiple R-squared:  0.4703,  Adjusted R-squared:  0.4587
- F-statistic: 40.54 on 3 and 137 DF,  p-value: < 2.2e-16
```

A comparación con el modelo que incluía todas las variables, aunque el r2 disminuyó, la significancia de las variables en el modelo aumentó. Para seleccionar variables que mejor se ajusten en el modelo, podemos hacer uso de otras funciones en R.

11.0.3 Selección de modelos

Podemos hacer una selección de predictores en R mediante la función step()

```
?step
step(reg)
- Start:  AIC=590
- riqueza ~ Altitud + Temp_media + Rango_diurno_Temp + Est_Temp +
-     Prec_anual + Rango_anual_Temp + Prec_min + Est_Prec
-
-                     Df Sum of Sq    RSS    AIC
- - Prec_min           1      2.19 8150.2 588.04
- - Est_Prec           1      3.00 8151.0 588.06
- - Temp_media         1     42.17 8190.1 588.73
- - Altitud            1     49.25 8197.2 588.85
- <none>                           8148.0 590.00
- - Rango_anual_Temp   1    214.22 8362.2 591.66
- - Est_Temp           1    217.37 8365.3 591.72
- - Rango_diurno_Temp  1    264.40 8412.4 592.51
- - Prec_anual         1    377.61 8525.6 594.39
-
- Step:  AIC=588.04
- riqueza ~ Altitud + Temp_media + Rango_diurno_Temp + Est_Temp +
-     Prec_anual + Rango_anual_Temp + Est_Prec
-
-                     Df Sum of Sq    RSS    AIC
- - Est_Prec           1      5.87 8156.0 586.14
- - Temp_media         1     41.92 8192.1 586.76
- - Altitud            1     47.27 8197.4 586.86
- <none>                           8150.2 588.04
- - Rango_anual_Temp   1    212.07 8362.2 589.66
- - Est_Temp           1    215.19 8365.3 589.72
- - Rango_diurno_Temp  1    262.39 8412.5 590.51
- - Prec_anual         1    441.98 8592.1 593.49
-
- Step:  AIC=586.14
```

```
- riqueza ~ Altitud + Temp_media + Rango_diurno_Temp + Est_Temp +
-     Prec_anual + Rango_anual_Temp
-
-                     Df Sum of Sq    RSS    AIC
- - Temp_media         1     37.59 8193.6 584.79
- - Altitud            1     49.84 8205.9 585.00
- <none>                           8156.0 586.14
- - Est_Temp           1    213.28 8369.3 587.78
- - Rango_anual_Temp   1    217.27 8373.3 587.85
- - Rango_diurno_Temp  1    262.86 8418.9 588.62
- - Prec_anual         1    513.14 8669.2 592.75
-
- Step:  AIC=584.79
- riqueza ~ Altitud + Rango_diurno_Temp + Est_Temp + Prec_anual +
-     Rango_anual_Temp
-
-                     Df Sum of Sq    RSS    AIC
- - Altitud            1     25.30 8218.9 583.23
- <none>                           8193.6 584.79
- - Est_Temp           1    217.36 8411.0 586.48
- - Rango_anual_Temp   1    259.31 8452.9 587.18
- - Rango_diurno_Temp  1    278.37 8472.0 587.50
- - Prec_anual         1   1512.78 9706.4 606.68
-
- Step:  AIC=583.23
- riqueza ~ Rango_diurno_Temp + Est_Temp + Prec_anual + Rango_anual_Temp
-
-                     Df Sum of Sq     RSS    AIC
- <none>                           8218.9 583.23
- - Est_Temp           1    197.53  8416.4 584.57
- - Rango_anual_Temp   1    234.06  8453.0 585.19
- - Rango_diurno_Temp  1    253.07  8472.0 585.50
- - Prec_anual         1   2110.46 10329.4 613.45
-
- Call:
- lm(formula = riqueza ~ Rango_diurno_Temp + Est_Temp + Prec_anual +
-     Rango_anual_Temp, data = dat_env[, c(3:10)])
-
- Coefficients:
-       (Intercept)  Rango_diurno_Temp           Est_Temp         Prec_anual   Rango_a
-          -5.16482          -13.38481           -0.49457            0.01211
```

Mediante step, podemos observar el valor del criterio de akaike (AIC) de diferentes combinaciones de predictores. AIC representa la calidad relativa de un modelo estadístico, lo que nos permite hacer selección de un modelo. Usual-

mente se escoge aquel con el AIC más bajo.

```
fit2 <- lm(riqueza ~ Rango_diurno_Temp + Est_Temp + Prec_anual + Rango_anual_Temp, data = dat_env

summary(fit2)
-
- Call:
- lm(formula = riqueza ~ Rango_diurno_Temp + Est_Temp + Prec_anual +
-     Rango_anual_Temp, data = dat_env)
-
- Residuals:
-      Min       1Q   Median       3Q      Max
- -20.8322  -4.7917  -0.8338   5.0426  20.7800
-
- Coefficients:
-                    Estimate Std. Error t value Pr(>|t|)
- (Intercept)        -5.164817   8.210273  -0.629   0.5304
- Rango_diurno_Temp -13.384812   6.540779  -2.046   0.0426 *
- Est_Temp           -0.494574   0.273561  -1.808   0.0728 .
- Prec_anual          0.012111   0.002049   5.910 2.62e-08 ***
- Rango_anual_Temp   13.739870   6.981699   1.968   0.0511 .
- ---
- Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
-
- Residual standard error: 7.774 on 136 degrees of freedom
- Multiple R-squared:  0.5884,  Adjusted R-squared:  0.5763
- F-statistic: 48.61 on 4 and 136 DF,  p-value: < 2.2e-16
```

Ejercicio 1

1. Usar la función AIC() para escoger cual de los dos modelos fit1 y fit2 es mejor en predecir la riqueza de especies.