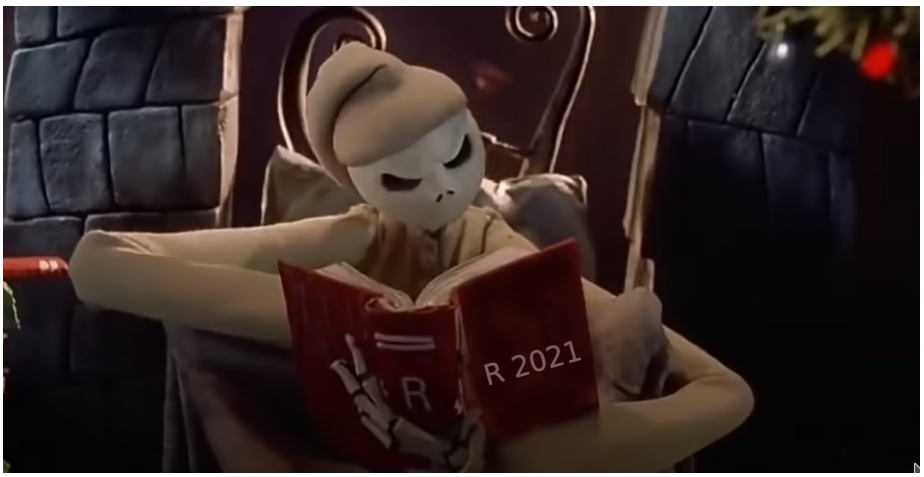
Fundamentos de R: Prácticas

Patricia Dell'Arciprete y Julio Lancelotti

2021-07-23

Contents


R
R 2021

Breve introducción a Ejercicios

Los ejercicios están diseñados para que se resuelvan utilizndo herramientas y conceptos vistos durante las clases teórico/prácticas. el objetivo de los mismos es que los alumnos se expongan situaciones de procesamiento de datos que probablemente requieran cuando trabajen con sus datos. Esta guía de ejercicios sigue el orden de las clases.

Usaremos los siguientes códigos para indicar el grado de dificultad de cada ejercicio. Las definiciones corresponden a transcripciones textuales de un diccionario web, pero son una selección totalmente arbitraria de las opciones disponibles en el mismo

(R) Regalo: Lo que se da a alguien sin esperar nada a cambio, como muestra de afecto o agradecimiento

(RR) Regular: Ordenado y sin excesos

(RRR) Razonable: Bastante, suficiente en cantidad o calidad

(RRRR) Ríspido: Que es difícil o costoso de realizar o soportar, por su dureza o violencia, porque requiere mucho esfuerzo o porque causa padecimiento

NOTA: a lo largo de todo este documento lo que esté con color de fondo es código, mientras que con fondo blanco se muestra el resultado que debería verse en consola al ejecutar el/los comando/s

RO1 Consola de R

0.1 Pimeras líneas de comando (R)

Abrir el programa R (NO el Rstudio), pegar el siguiente código y explicar que hace cada uno de los tres comandos recién ejecutados

```
1:40
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

```
x <- 1:40
x
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

R02 Rstudio

0.2 Crear un archivo de script (R)

Desde el RStudio abrir un nuevo archivo de texto (file/new file/Text File) al que llamaremos script_RO2.txt

0.3 (R)

En el script recién creado pegar el siguiente código:

```
# Este es mi primer script de ejercicios en R
x <- 5:10
x
mean(x)
```

¿Se ven las líneas de código con diferentes colores?

¿Se puede usar la combinación de teclas CTR + R (los usuarios linux probablemente deban usar CTR + enter) para enviar código a la consola?

Ahora guardar el **script_R02.txt**, como **script_R02.R** (File/Save as)
¿Se aprecia algún cambio en los colores del código?
¿Se puede usar la combinación de teclas CTR + R (los usuarios linux probablemente deban usar CTR + enter) para enviar código a la consola?
Si funciona en la consola debería verse:

```
## [1]  5  6  7  8  9 10
```

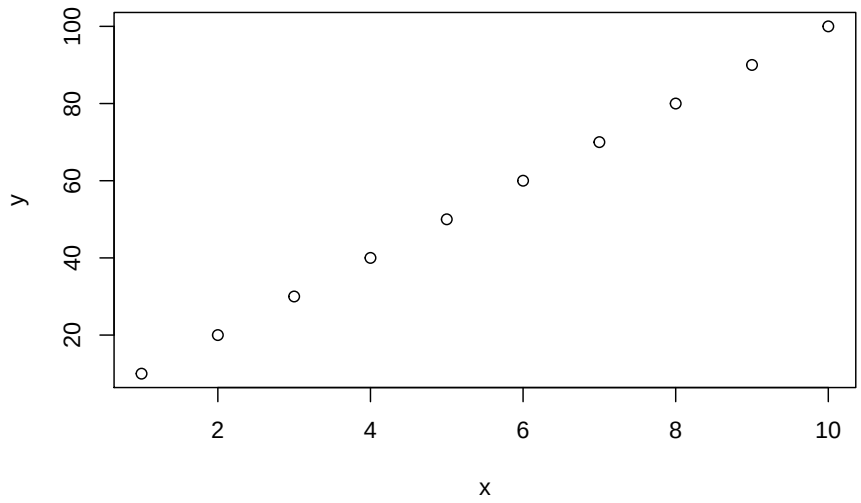
```
## [1] 7.5
```

¿Que se puede concluir acerca de la diferencia entre guardar un script con extensión txt o r/R?

0.4 interacción con R (RR)

Ejecutar las siguientes líneas de comando. Por ahora no es importante que identifiquen que es lo que se hace en cada línea. El objetivo es familiarizarse con las ventanas y la interacción RStudio y R

```
# mi grafiquito
x <- 1:10
y <- x*10
plot(x, y)
```



¿Cuál de los siguientes pasos siguieron para realizar la tarea?

Tipeando en la consola

Tipeando en el script y enviando el código a consola (CTRL + R)

Copiando y pegando en consola

Copiando y pegando en el script y enviando el código a consola (CTRL + R)

R03 Básicos

0.5 Ayudas 1 (RR)

Trabajaremos en este punto con la función ***median()*** (mediana). Analizar las diferencias entre
* ??median

- ?median
- apropos("median")

0.6 Ayudas 2 (RRR)

Averiguar qué hace la siguiente función

```
sample(1:10, 20, replace=TRUE)
```

0.7 Cómputos (RRR)

1. Calcular en R:

$$\frac{2^{pi}}{2^7} - 1$$

RTA: -0.9310549

NOTA: podemos usar el valor de pi (3.141593) o utilizar la constante pi que trae el R (si es necesario acceder a la ayuda)

2. Calcular en R:

$$\frac{3 - cos(pi)}{\sqrt[2]{7}}$$

RTA: 1.5118579

0.8 Comandos básicos (RRR)

Resolver los siguientes puntos

Listar los objetos del workspace

Asignar a un objeto que llamaremos $\boldsymbol{y}$ un vector que contenga los números 2, 5, 7, y 9

Aplicar la función **max()** para obtener el valor máximo del elemento $\boldsymbol{y}$

Reasignar el valor obtenido en el punto anterior al objeto llamado $\boldsymbol{y}$ (es decir reescribir $\boldsymbol{y}$) ¿Cuántos elementos tendrá ahora $\boldsymbol{y}$?

0.9 Creando vectores (RR)

Asignar a un objeto que llamaremos $\boldsymbol{x}$ los valores 90, 3, -5

Asignar a un objeto que llamaremos $\boldsymbol{y}$ los elementos 90, 3, "A". Comparar visualmente los objetos $\boldsymbol{y}$ y $\boldsymbol{x}$ ¿Se ve alguna diferencia entre ambos objetos?

0.10 Nombres de objetos (RR)

Identificar cuáles de las siguientes opciones podrían ser utilizadas como nombres de objetos en R. En los casos que sea necesario, proponer cambios para que esos nombres sean aceptados por el programa (lo más parecido posible a los nombres originales)

- RecorCholis
- tamaño máximo de célula
- SUperficie_del:Area
- 4min
- site#7
- sitio?3

0.11 Operando con vectores (RR)

Crear un objeto llamado ***zx*** de 100 valores en el **rango** 100-200. Verificar que el vector creado tenga 100 elementos

Crear un objeto llamado ***zy*** de dos elementos, con valores 1 y 2

Realizar ***zx/zy*** ¿Qué resultado arroja? ¿Aparece algún mensaje?

Repetir el inciso anterior, pero ***zy*** tendrán valores 1, 2 y 3

¿Y ahora apareció algún mensaje?

¿Se ejecutó la línea?

Seguramente a esta altura ya han recibido algún triste mensaje de ***error***, ¿qué diferencia hay entre un mensaje ***warning*** y un mensaje ***error***?

0.12 Dar nombre a los elementos de un vector (RRR)

Dado el siguiente vector:

```
miX <- 1:15
```

¿Qué resultado dará **names(miX)**?

```
names(miX)
```

Sabiendo la respuesta, dar nombres a los elementos de miX, donde a cada elemento se le asignará la letra del abecedario correspondiente a su posición

miX debería verse así:

```
 A  B  C  D  E  F  G  H  I  J  K  L  M  N  O
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

0.13 Aplicando funciones a vectores (RR)

Crear un vector ***y*** con los valores -5, 6, 0, 0, 0, 90, -38. Aplicar una función para convertir todos los valores de ***y*** a números positivos

Aplicar una función para obtener la mediana de ***y***

0.14 Pruebas y operaciones lógicas (RRR)

(tomado de swirl(); swirlstats.com)

1 ¿Cuáles de las siguientes expresiones se evaluarán como FALSE?

TRUE & TRUE

TRUE & FALSE

FALSE & TRUE

FALSE & FALSE

TRUE | TRUE

TRUE | FALSE

FALSE | TRUE

FALSE | FALSE

2 ¿Cuál será el resultado y porqué?

TRUE & c(TRUE, FALSE, FALSE)

TRUE && c(TRUE, FALSE, FALSE)

0.15 Pruebas lógicas

Dados los siguientes vectores

```
x <- c(1,2,3,4,5)
y <- c(1,3,3,3,5)
z <- c(T,T,T,F,F)
k <- c(T,F,T,F,T)
```

En cada uno de los siguientes casos analizar cual va a ser el resultado y de cuantos elementos estará compuesto el mismo. Luego correr en consola para verificar

```
x > 3
x == y
x != y
x[1] == y[1]
x && y
z & k
z && k
z == k
(x > 3 & y > 3) | x == 1
x > 3 & y > 3 & z
```

```
x > 3 & y > 3 | z
any(x > 4)
any (y > 4)
any(x > 15 | y > 4)
any (x > 15 | y > 5)
all(x[1] == y[1])
all(x[1] != y[1])
```

Debugging, detectando errores

Es importante **LEER LOS MENSAJES DE ERROR** y **WARNINGS** , en la mayoría de los casos son bastante informativos y ayudan a encontrar fallas en el script

Para cada una de las siguientes líneas de código explicar porqué se obtiene un mensaje de error, warnning, o un resultado no esperado.

```
rm( list = ls() ) #eliminemos primero todos los objetos
```

1.
```
y = c(1:7, 33/(2 + 5)
y
```

2.
```
mi_X
```

3.
```
myZ <- (1,4,5,7)
```

4.
```
mean(C(x,y))
```

5.
```
var-3 <- mean(x)
```

6.
```
var@ <- mean(x)
```

7.
```
sample(letters, 100)
```

R04 Indexación y Filtros

0.16 Vector LETTERS (R)

Explorar el objeto LETTERS que viene definido con el paquete básico de R ¿cuántos elementos tiene?

```
LETTERS
```

```
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
[20] "T" "U" "V" "W" "X" "Y" "Z"
```

0.17 Indexación de vectores (RR)

Obtener los 10 primeros elementos de LETTERS

```
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"
```

0.18 Indexación de vectores (RRR)

Explicar que hacen las siguientes líneas de comando

```
miX <- 10:1
names(miX )
```

```
NULL
```

```
names(miX ) <- LETTERS[1:10]
```

```
miX
```

```
##  A  B  C  D  E  F  G  H  I  J
## 10  9  8  7  6  5  4  3  2  1
```

0.19 Indexación de vectores (RRR)

Dado el siguiente vector miX

```
 A  B  C  D  E  F  G  H  I  J
10  9  8  7  6  5  4  3  2  1
```

Obtener los cinco primeros elementos de ***miX*** mediante:

Un vector numérico

Un vector con los **nombres** correspondientes a esos elementos

Un vector lógico

0.20 Indexación de vectores (RRR)

Explorar el objeto ***x*** que se da a continuación (Por ahora no importan los comandos que utilizamos para generarlo)

```
set.seed(300)
x <- sample(1:100, 40, replace=T)
```

Obtener los ***valores*** de los ***elementos*** ubicados en las posiciones 5, 10, 15 y 20 de este vector

Generar un nuevo objeto llamado ***miY***, repitiendo cuatro veces el primer elemento de ***x***, tres veces el segundo elemento y dos veces el tercer elemento. ***miY*** debería verse así:

```
## [1] 78 78 78 78 66 66 66 89 89
```

0.21 Indexación: índices lógicos vs which (RRR)

Explorar el objeto ***x*** que se da a continuación

```
set.seed(300)
x <- sample(1:100, 40, replace= T)
```

Generar un vector lógico, que llamaremos ***idx1***. El mismo tomará valores T para aquellos valores de ***x*** mayores o iguales a 20 y F para los valores de ***x*** menores a 20

Utilizar el objeto ***idx1*** recién generado para obtener los **valores** del vector ***x*** mayores o iguales a 20

Utilizando la función **which()**, obtener la **posición** de los elementos de ***x*** cuyos valores sean mayores o iguales a 20. Asignar la salida de la función a un objeto que llamaremos ***idx2***

Utilizar el objeto ***idx2*** recién generado para obtener los **valores** del vector ***x*** mayores o iguales a 20

El resultado obtenido fue el mismo cuando indexamos con el vector lógico (***idx1***) y cuando usamos la función which (***idx2***). Sin embargo, ***idx1*** tiene 40 elementos, mientras que ***idx2*** tiene 33 elementos ¿por qué?

0.22 Indexación en vectores ejemplo colors() (RRR)

Escribir **colors()** en consola para examinar el objeto

¿Qué diferencia estructural hay entre los objetos colors y letters? Recurrir a la ayuda si fuera necesario

Volveremos a utilizar este objeto en varias ocasiones durante el curso. Por ahora simplemente lo usaremos como ejemplo.

```
colors()
```

Asignar el objeto **colors()** a un nuevo objeto que llamaremos ***misColores***

Generar un objeto, al que llamaremos ***amarillos***, con aquellos elementos de ***misColores*** que contengan en alguna parte del texto la palabra "yellow"

¿Cuántos elementos tiene el objeto ***amarillos***?

Para generar el objeto ***amarillos*** ¿era necesario generar el objeto ***misColores***?

Debugging, detectando errores

Para cada una de las siguientes líneas de código explicar porqué se obtiene un mensaje de error, warnning, o un resultado no esperado

```
rm( list = ls() ) #eliminemos primero todos los objetos
```

1.

```
k <- letters(1)
```

2.

```
month.name(1:10)
```

3.

```
y <- c("A", "B", "C", "D")
txt <- letters[y]
```

R05 Workspace y RHistory

0.23 Responder con Verdadero o Falso (RRR)

El working directory siempre es el directorio donde tenemos guardado el script

Todos los scripts deben estar guardados en la misma ubicación que el directorio de trabajo

El Rstudio tiene definido un directorio de trabajo por defecto

En una sesión de R sólo puede haber un directorio de trabajo. Sin embargo en un script se puede re-definir el directorio de trabajo tantas veces como uno quiera

Hice una simulación que tardó 3 hs en correr y no guardé los objetos generados. No me preocupo porque el resultado siempre se guarda en el Rhistory

Hice una simulación que tardó 3 hs en correr y no guardé los objetos generados, pero no me preocupo porque el resultado siempre se guarda en el RData

Por defecto (default) R genera el RHistory, donde guarda los comandos y objetos generados en la última sesión

En el RHistory quedan registrados los pasos en el mismo orden que aparecen en el script

Podemos selectivamente guardar un único objeto y cargarlo cuando lo necesitemos

El comando "ctrl + l" aplicado a la consola (clear console, limpiar consola) tiene el mismo efecto que remover los objetos rm(list= ls())

0.24 Salvar y cargar objetos (RR)

Correr la siguiente línea para remover todos los objetos creados durante esta sesión

```
rm( list = ls() )
```

Comprobar que no hay objetos en nuestra sesión

```
ls()
```

```
## character(0)
```

Dados los siguientes objetos

```
y <- rnorm(100)
x <- 1:10
z <- letters
```

Guardar la sesión de trabajo

Guardar solamente los objetos x, y

Remover todos los objetos y volver a cargar solamente x

Pedir un listado de todos los objetos

0.25 Debugging, detectando errores

Para cada una de las siguientes líneas de código explicar porqué se obtiene un mensaje de error, warnning, o un resultado no esperado

```
rm( list = ls() ) #eliminemos primero todos los objetos
```

1.

```
setwd("C:\R2021/R_Archivos/")
```

2.

```
setwd(C:/R2021/R_Archivos/)
```

R06 Importar y Exportar

0.26 Responder con Verdadero o Falso (RRR)

Los archivos que se quieren cargar (read.table) deben estar en el mismo directorio de trabajo que el script que los carga

Los campos de un archivo a importar siempre deben estar separados por comas

Para importar un archivo el separador decimal siempre tiene que ser el punto (".")

Tanto los archivos de entrada como los de salida tienen que tener definidos los nombres de filas y columnas

Al importar una tabla los cambios generados en R no modificarán los datos contenidos en el archivo de origen

Una ruta absoluta es aquella que especifica la ubicación de una carpeta o archivo desde su directorio raíz

Para ser importados a R los datos siempre deben estar ordenados en un sistema de filas y columnas

Para importar una tabla con encabezados, todas las columnas deben estar nombradas

Si importamos una tabla con nombres de encabezado repetidos R renombrará cada encabezado para que éstos no tengan nombres repetidos

Siempre hay que ingresar la ruta absoluta del archivo a importar o a exportar

0.27 read.table y write.table

Con un editor de texto (ej bloc de notas, o el propio Rstudio) abrir los archivos "MiTrayect1.csv" y "MiTrayect2.csv", alojados en la carpeta "4-Datos". Identificar el separador de campos y el símbolo decimal. Los datos son posiciones geográficas de una determinada trayectoria

Importar a R (***read.table()***) ambos arvhivos, asignando el resultado de la importación a dos objetos que llamaremos respectivamente *tray1* y *tray2*

Mostrar en consola ambos objetos para ver su estructura

Generar un objeto al que llamaremos "tray3" combinando los dos objetos recién creados. El nuevo objeto deberá tener dos columnas (dlon y dlat) y tantas filas como la suma de las filas de los dos archivos recién incorporados.

Usar la función **rbind()** (row binding, juntar por filas) para "juntar" ambas series de datos en una sola tabla. La función se usa:

```
rbind (x, y)
```

Guardar en la carpeta "Resultados" (***write.table()***) un archivo de texto con el contenido de ***tray3*** con el nombre "Trayect1+2.csv". Asegurarse que el archivo de salida tenga el mismo formato que los archivos originales (nombres de columnas, separadores de campos, símbolo decimal, sin nombre de filas, etc.)

0.28 Debugging, detectando errores

Para cada una de las siguientes líneas de código explicar porqué se obtiene un mensaje de error, warnning, o un resultado no esperado

```
rm( list = ls() ) #eliminemos primero todos los objetos
```

```
#Generamos un objeto de prueba
x <- 1:100
```

1.

```
read.table(c:/R2015/R_Archivos/4-Datos/MiTrayect1.csv, sep= ",", dec= ".", header= T )
```

2.

```
read.table("C:/R2015/R_Archivos/4-Datos/MiTrayect1.csv", sep= ",", dec= ".", header= "
```

3.

```
write.table("C:/R2015/R_Archivos/Resultados/nombre.csv", sep= ",", dec= ".")
```

4.

```
write.table(x, "C:/R2015/R_Archivos/Resultados/nombre.csv", sep= ",", dec= ".", header=
```

5.

```
write.table(x "C:/R2015/R_Archivos/Resultados/nombre.csv", sep= ",", dec= ".")
```

R07 Objetos 1

0.29 Vectores 1 (RR)

1. Generar cinco vectores de **10 elementos** cada uno con los nombres y características que se dan a continuación. Utilizar las funciones **seq()** y **rep()** en los casos pertinentes.

- texto: un vector texto con las 10 primeras letras del abecedario

```
texto
```

```
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"
```

- enteros: primeros 10 números impares

```
enteros
```

```
 [1]  1  3  5  7  9 11 13 15 17 19
```

- decimales: valores entre 1 y 1.9 en intervalos de 0.1

```
decimales
```

```
 [1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9
```

- lógico: 5 valores T seguidos de 5 F

```
logico
```

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

- factor1: un vector tipo factor con las 10 primeras letras del abecedario

```
factor1
```

```
 [1] a b c d e f g h i j
Levels: a b c d e f g h i j
```

- factor2: el mismo objeto que el punto anterior pero con orden de jerarquía, donde a > b > c. . . . >j

```
factor2
```

```
 [1] a b c d e f g h i j
Levels: j < i < h < g < f < e < d < c < b < a
```

2. Comparar la salida en consola de cada uno de los objetos

0.30 Vectores 2 (RR)

¿Cuál es el error en las siguientes lineas de comando?

```
miX <- ("uno", "dos", "tres")

miY <- c(uno, dos, tres)

miZ <- "a":"Z"
```

R08 Objetos2

0.31 Responder con verdadero o falso (RRR)

1. Tanto las matrices como los data.frame son arreglos rectangulares de datos. Lo que significa que están formados por uno o más vectores de iguales dimensiones y todos los elementos que los componen tienen datos o valores NA
2. A diferencia de los data.frame, las matrices tienen el mismo número de filas que de columnas
3. Todos los vectores que componen una matriz deben tener el mismo formato (numérico, texto o lógico), aunque también pueden incluir NA's o ser de NA's únicamente
4. Los data.frame permiten que un mismo vector-columna contenga datos con más de un tipo de formato
5. Un array **puede pensarse** como un vector con n elementos, donde cada elemento es una matriz y todas las matrices tienen las mismas dimensiones
6. Las listas, arrays, matrices y data.frames tienen las mismas dimensiones
7. El objeto par() es una lista
8. El siguiente objeto x es un data frame de una fila y 10 columnas

```
x <- 1:10
    names(x) <- letters[1:10]
        x
```

```
 a  b  c  d  e  f  g  h  i  j
 1  2  3  4  5  6  7  8  9 10
```

9. La siguiente es una matriz compuesta por una columna de datos con formato numérico y una columna de texto

```
matrix(c(1:3,"a", "b","c"), ncol=2)
```

```
     [,1] [,2]
[1,] "1"  "a"
[2,] "2"  "b"
[3,] "3"  "c"
```

10. Las dimensiones de un data.frame difieren si al mismo se le agrega nombre de filas (rownames)

0.32 Generar un data.frame 1 (R)

Con los siguientes vectores crear un objeto del tipo data.frame, al que llamaremos *MiDf*.

```
letras <- LETTERS[1:10]
enteros <- round( runif(10, 0, 10), 0 )
decimales <- round( runif(10, 0, 10), 3 )
logicos <- sample( c(T,F), 10, replace=T )
```

```
MiDf
```

```
   letras enteros decimales logicos
1       A       4     6.638    TRUE
2       B       1     7.662   FALSE
3       C       3     0.350    TRUE
4       D       9     0.390    TRUE
5       E       0     9.803   FALSE
6       F       3     2.716   FALSE
7       G       6     0.483   FALSE
8       H       8     6.413   FALSE
9       I       4     4.885    TRUE
10      J       2     2.722   FALSE
```

0.33 Agregar columnas a un data.frame (RR)

Agregar al data frame del punto anterior una columna con valores NA's. A esta columna la llamaremos *sin.dato*

```
MiDf
```

```
   letras enteros decimales logicos sin.dato
1       A       4     6.638    TRUE       NA
2       B       1     7.662   FALSE       NA
3       C       3     0.350    TRUE       NA
4       D       9     0.390    TRUE       NA
5       E       0     9.803   FALSE       NA
6       F       3     2.716   FALSE       NA
7       G       6     0.483   FALSE       NA
8       H       8     6.413   FALSE       NA
9       I       4     4.885    TRUE       NA
10      J       2     2.722   FALSE       NA
```

0.34 Generar un data.frame 2: seq(), rep(), paste() (RRR)

Utilizar una combinación de funciones **rep()**, **paste()**, **seq()** y **data.frame()** para generar una tabla similar a la que se muestra a continuación:

```
     tratamiento dosis resultado
1  tratamiento 1     0        NA
2  tratamiento 1     1        NA
3  tratamiento 1    10        NA
4  tratamiento 1   100        NA
5  tratamiento 1  1000        NA
6  tratamiento 3     0        NA
7  tratamiento 3     1        NA
8  tratamiento 3    10        NA
9  tratamiento 3   100        NA
10 tratamiento 3  1000        NA
11 tratamiento 5     0        NA
12 tratamiento 5     1        NA
13 tratamiento 5    10        NA
14 tratamiento 5   100        NA
15 tratamiento 5  1000        NA
16 tratamiento 7     0        NA
17 tratamiento 7     1        NA
18 tratamiento 7    10        NA
19 tratamiento 7   100        NA
20 tratamiento 7  1000        NA
21 tratamiento 9     0        NA
22 tratamiento 9     1        NA
23 tratamiento 9    10        NA
24 tratamiento 9   100        NA
25 tratamiento 9  1000        NA
```

Nota 1: los valores 1 a 15 no forman parte de la tabla, son los indicadores del nro de fila que da la consola de R Nota 2: la secuencia dosis puede generarse a partir del siguiente vector

```
dosis <- c(0,1,10,100,1000)
```

Nota 3: es conveniente aprovechar la regla de reciclado de vectores

0.35 sort vs order (RR)

Dado el siguiente vector

```
miX <- 15:1
names(miX) <- letters[1:15]
miX
```

```
##  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o
## 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
```

Ordenar en forma ascendente el vector ***miX*** utilizando:
* La función **sort()**
* La función **order()**

```
 o  n  m  l  k  j  i  h  g  f  e  d  c  b  a
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

0.36 Ordenar un data.frame (RRR)

La utilidad de ***order()*** es más evidente cuando queremos ordenar un data.frame. Dado el siguiente data.frame

```
trat <- paste("tratamiento", rep( seq(1, 9, 2), each=5))
dosis <- c(0,1,10,100,1000)
tabla <- data.frame(tratamiento= trat, dosis= dosis, resultado= NA)
tabla
```

```
##      tratamiento dosis resultado
## 1  tratamiento 1     0        NA
## 2  tratamiento 1     1        NA
## 3  tratamiento 1    10        NA
## 4  tratamiento 1   100        NA
## 5  tratamiento 1  1000        NA
## 6  tratamiento 3     0        NA
## 7  tratamiento 3     1        NA
## 8  tratamiento 3    10        NA
## 9  tratamiento 3   100        NA
## 10 tratamiento 3  1000        NA
## 11 tratamiento 5     0        NA
## 12 tratamiento 5     1        NA
## 13 tratamiento 5    10        NA
## 14 tratamiento 5   100        NA
## 15 tratamiento 5  1000        NA
## 16 tratamiento 7     0        NA
## 17 tratamiento 7     1        NA
## 18 tratamiento 7    10        NA
## 19 tratamiento 7   100        NA
## 20 tratamiento 7  1000        NA
## 21 tratamiento 9     0        NA
```

```
## 22 tratamiento 9     1        NA
## 23 tratamiento 9    10        NA
## 24 tratamiento 9   100        NA
## 25 tratamiento 9  1000        NA
```

Utilizar la función order() para ordenar el objeto *tabla* en orden creciente de la variable *dosis* como se muestra a continuación

```
     tratamiento dosis resultado
1  tratamiento 1     0        NA
6  tratamiento 3     0        NA
11 tratamiento 5     0        NA
16 tratamiento 7     0        NA
21 tratamiento 9     0        NA
2  tratamiento 1     1        NA
7  tratamiento 3     1        NA
12 tratamiento 5     1        NA
17 tratamiento 7     1        NA
22 tratamiento 9     1        NA
3  tratamiento 1    10        NA
8  tratamiento 3    10        NA
13 tratamiento 5    10        NA
18 tratamiento 7    10        NA
23 tratamiento 9    10        NA
4  tratamiento 1   100        NA
9  tratamiento 3   100        NA
14 tratamiento 5   100        NA
19 tratamiento 7   100        NA
24 tratamiento 9   100        NA
5  tratamiento 1  1000        NA
10 tratamiento 3  1000        NA
15 tratamiento 5  1000        NA
20 tratamiento 7  1000        NA
25 tratamiento 9  1000        NA
```

0.37 Generar matrices (RR)

1. Generar dos matrices (*MiMz1* y *MiMz2*) de dos columnas y 100 filas con números del 1 al 200. En la primera los números deberán llenarse por filas y en la segunda por columnas

```
head(MiMz1)
```

```
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

```
[3,]    5    6
[4,]    7    8
[5,]    9   10
[6,]   11   12
```

```
head(MiMz2)
```

```
     [,1] [,2]
[1,]    1  101
[2,]    2  102
[3,]    3  103
[4,]    4  104
[5,]    5  105
[6,]    6  106
```

2. Ahora generar una matriz (MNa) de 10 columnas cada una con 100 elementos con valores NA

```
head(MNa)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA
[2,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA
[3,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA
[4,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA
[5,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA
[6,]   NA   NA   NA   NA   NA   NA   NA   NA   NA    NA
```

0.38 Combinar matrices (RR)

Generar una tercer matriz ($M3$) combinando las tres matrices anteriores

```
head(M3[,1:7]) # Notar que solamente mostramos las primeras 7 columnas
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## [1,]    1    2    1  101   NA   NA   NA
## [2,]    3    4    2  102   NA   NA   NA
## [3,]    5    6    3  103   NA   NA   NA
## [4,]    7    8    4  104   NA   NA   NA
## [5,]    9   10    5  105   NA   NA   NA
## [6,]   11   12    6  106   NA   NA   NA
```

0.39 Nombrar columnas de matrices (RR)

Dar nombres a las columnas de la matriz con el texto *var.1* a var.n_, donde *n* es el número de columnas

```
     var_ 1 var_ 2 var_ 3 var_ 4 var_ 5 var_ 6 var_ 7
[1,]      1      2      1    101     NA     NA     NA
[2,]      3      4      2    102     NA     NA     NA
[3,]      5      6      3    103     NA     NA     NA
[4,]      7      8      4    104     NA     NA     NA
[5,]      9     10      5    105     NA     NA     NA
[6,]     11     12      6    106     NA     NA     NA
```

0.40 Generar listas (RR)

1. Generar una lista – que llamaremos ***MiLista*** – con los siguientes objetos

```
letras <- letters
midf <- data.frame(sitio= 1:50, valor= NA)
mimz <- matrix(NA, ncol= 2, nrow= 3)
```

0.41 Función split (RRR)

1. Importar el archivo DoubsEnv.csv que se encuentra en la carpeta 4-Datos (la descripción de la base de datos se encuentra en el archivo datos_eer.html) y asignar la tabla a un objeto que llamaremos ***env***
2. Utilizando la función **split()** generar una lista donde cada elemento de la misma corresponderá a los datos de cada una de las secciones del río Doubs.

```
## $lower
##    Site lon lat   das alt pen   deb  pH dur  pho  nit  amm oxy  dbo   sec
## 23   23 192 228 304.3 246 1.2 28.80 8.1  97 2.60 3.50 1.15 6.3 16.4 lower
## 24   24 179 233 314.7 241 0.3 29.76 8.0  99 1.40 2.50 0.60 5.2 12.3 lower
## 25   25 145 217 327.8 231 0.5 38.70 7.9 100 4.22 6.20 1.80 4.1 16.7 lower
## 26   26  91 187 357.9 214 0.5 39.10 7.9  94 1.43 3.00 0.30 6.2  8.9 lower
## 27   27  65 174 373.2 206 1.2 39.60 8.1  90 0.58 3.00 0.26 7.2  6.3 lower
## 28   28  49 164 394.7 195 0.3 43.20 8.3 100 0.74 4.00 0.30 8.1  4.5 lower
## 29   29  27 151 422.0 183 0.6 67.70 7.8 110 0.45 1.62 0.10 9.0  4.2 lower
## 30   30   8 133 453.0 172 0.2 69.00 8.2 109 0.65 1.60 0.10 8.2  4.4 lower
##
## $middle
##    Site lon lat   das alt pen  deb  pH dur  pho  nit  amm  oxy dbo   sec
```

```
## 11   11 186 130 123.4 483 4.1 19.9 8.1  96 0.30 1.60 0.00 11.5 2.7 middle
## 12   12 205 145 132.4 477 1.6 20.0 7.9  86 0.04 0.50 0.00 12.2 3.0 middle
## 13   13 222 167 143.6 450 2.1 21.1 8.1  98 0.06 0.52 0.00 12.4 2.4 middle
## 14   14 228 182 152.2 434 1.2 21.2 8.3  98 0.27 1.23 0.00 12.3 3.8 middle
## 15   15 252 190 164.5 415 0.5 23.0 8.6  86 0.40 1.00 0.00 11.7 2.1 middle
## 16   16 266 209 185.9 375 2.0 16.1 8.0  88 0.20 2.00 0.05 10.3 2.7 middle
## 17   17 245 203 198.5 348 0.5 24.3 8.0  92 0.20 2.50 0.20 10.2 4.6 middle
## 18   18 225 200 211.0 332 0.8 25.0 8.0  90 0.50 2.20 0.20 10.3 2.8 middle
## 19   19 206 194 224.6 310 0.5 25.9 8.1  84 0.60 2.20 0.15 10.6 3.3 middle
## 20   20 189 193 247.7 286 0.8 26.8 8.0  86 0.30 3.00 0.30 10.3 2.8 middle
## 21   21 187 201 281.2 262 1.0 27.2 7.9  85 0.20 2.20 0.10  9.0 4.1 middle
## 22   22 192 212 294.0 254 1.4 27.9 8.1  88 0.20 1.62 0.07  9.1 4.8 middle
##
## $upper
##    Site lon lat  das alt  pen   deb  pH dur  pho  nit  amm  oxy dbo   sec
## 1     1  88   7  0.3 934 48.0  0.84 7.9  45 0.01 0.20 0.00 12.2 2.7 upper
## 2     2  94  14  2.2 932  3.0  1.00 8.0  40 0.02 0.20 0.10 10.3 1.9 upper
## 3     3 102  18 10.2 914  3.7  1.80 8.3  52 0.05 0.22 0.05 10.5 3.5 upper
## 4     4 100  28 18.5 854  3.2  2.53 8.0  72 0.10 0.21 0.00 11.0 1.3 upper
## 5     5 106  39 21.5 849  2.3  2.64 8.1  84 0.38 0.52 0.20  8.0 6.2 upper
## 6     6 112  51 32.4 846  3.2  2.86 7.9  60 0.20 0.15 0.00 10.2 5.3 upper
## 7     7 114  61 36.8 841  6.6  4.00 8.1  88 0.07 0.15 0.00 11.1 2.2 upper
## 8     8 110  76 49.1 792  2.5  1.30 8.1  94 0.20 0.41 0.12  7.0 8.1 upper
## 9     9 136 100 70.5 752  1.2  4.80 8.0  90 0.30 0.82 0.12  7.2 5.2 upper
## 10   10 168 112 99.0 617  9.9 10.00 7.7  82 0.06 0.75 0.01 10.0 4.3 upper
```

0.42 Generar arrays (RRR)

Dadas las siguientes matrices

```
MiMz1 <- matrix(1:200, nrow= 100, byrow = T )
MiMz2 <- matrix(1:200, nrow= 100, byrow = F )
```

1. Generar un array al que llamaremos ***MiArray*** cuyas dos únicas matrices serán ***MiMz1*** y ***MiMz2***.

- Opción 1: generar primero un array de dos matrices con NA's, de las mismas dimensiones que *MiMz1* y luego reemplazar esas matrices por los objetos *MiMz1* y *MiMz2*
- Opción 2: generar el array a partir de las dos matrices combinadas mediante cbind

0.43 indexar data frame sencillo(RR)

1. Asignar el objeto iris (que viene con el paquete base) a un objeto que llamaremos ***midf***

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
```

2. Mediante indexación obtener las primeras 10 filas de ***midf*** de las columnas Sepal.Width Petal.Length y Species

0.44 indexar data.frame (RRR)

1. Generar un objeto al que llamaremos ***Env*** a partir del archivo DoubsEnv.csv de la carpeta 4-Datos
2. Seleccionar todas las columnas de ***Env***, pero solamente las filas pares utilizando:

- un vector lógico
- un vector numérico

3. Selecionar los registros de env que cumplan simultáneamente con las siguientes condiciones

- Que sean de la sección alta del río (cateogría *upper* de la variable *sec*)
- Que los valores de la variable altura (*alt*) esté en el rango 700-900
- Que la pendiente (pen) sea menor que 10

El resultado debería verse como la tabla mostrada a continuación

```
##   Site lon lat  das alt pen  deb  pH dur  pho  nit  amm  oxy dbo   sec
## 4    4 100  28 18.5 854 3.2 2.53 8.0  72 0.10 0.21 0.00 11.0 1.3 upper
## 5    5 106  39 21.5 849 2.3 2.64 8.1  84 0.38 0.52 0.20  8.0 6.2 upper
## 6    6 112  51 32.4 846 3.2 2.86 7.9  60 0.20 0.15 0.00 10.2 5.3 upper
## 7    7 114  61 36.8 841 6.6 4.00 8.1  88 0.07 0.15 0.00 11.1 2.2 upper
## 8    8 110  76 49.1 792 2.5 1.30 8.1  94 0.20 0.41 0.12  7.0 8.1 upper
## 9    9 136 100 70.5 752 1.2 4.80 8.0  90 0.30 0.82 0.12  7.2 5.2 upper
```

0.45 Eliminando duplicados

Las siguientes líneas de comando generan una tabla con registros que a propósito están repetidos (una o más veces). El objetivo es utilizar una combinación de

comandos, eliminar las repeticiones de los registros repetidos. Propongo utilizar las funciones unique() y match() para realizar la tarea

```
set.seed(200)
sitio <-  paste("sitio", 1:20, sep= " ")
humedad <- sample(20:100, 2)
datos <- data.frame(sitio, humedad)
datos <- datos[sample(1:20, 30, replace=T), ]
rownames(datos) <- NULL
datos
```

```
##       sitio humedad
## 1  sitio 18      63
## 2  sitio 12      63
## 3  sitio 20      63
## 4   sitio 8      63
## 5   sitio 4      63
## 6   sitio 6      63
## 7  sitio 13      57
## 8   sitio 6      63
## 9   sitio 6      63
## 10  sitio 3      57
## 11 sitio 11      57
## 12  sitio 3      57
## 13  sitio 5      57
## 14  sitio 1      57
## 15 sitio 11      57
## 16  sitio 8      63
## 17  sitio 7      57
## 18 sitio 18      63
## 19  sitio 4      63
## 20  sitio 6      63
## 21  sitio 4      63
## 22 sitio 19      57
## 23 sitio 20      63
## 24 sitio 19      57
## 25 sitio 14      63
## 26  sitio 2      63
## 27  sitio 9      57
## 28  sitio 1      57
## 29 sitio 13      57
## 30 sitio 16      63
```

```
##       sitio humedad
## 1  sitio 18      63
## 2  sitio 12      63
## 3  sitio 20      63
```

```
## 4   sitio 8      63
## 5   sitio 4      63
## 6   sitio 6      63
## 7  sitio 13      57
## 10  sitio 3      57
## 11 sitio 11      57
## 13  sitio 5      57
## 14  sitio 1      57
## 17  sitio 7      57
## 22 sitio 19      57
## 25 sitio 14      63
## 26  sitio 2      63
## 27  sitio 9      57
## 30 sitio 16      63
```

0.46 indexar una matriz(RR)

1. Generar una matríz a la que llamaremos *mtz* de 20 filas por 5 columnas, con números consecutivos de 100 a 1 y que se complete por filas

2. obtener los elementos de las posiciones de las filas 5 a 10 de las columnas 2 y 4. Asignar el resultado a un objeto que llamaremos *mtz2*.

calcular el valor prmoedio de los valores de *mtz2* (el promedio de toda la matríz)

0.47 indexar una lista 1 (RR)

1. Generar una lista (***MiLista2***) cuyos dos únicos elementos serán los objetos ***iris*** (incluido en el paquete base) y ***crabs*** del paquete *MASS*.

2. Asegurarse que los dos elementos de la lista tengan nombres (***iris*** y ***crabs***), ya sea porque fuera definido en la creación de la lista o mediante la función names()

2. Seleccionar el segundo elemento de esta lista mediante los tres métodos que se listan a continuación

- El operador *$*
- Un vector numérico (un solo valor)
- El nombre del elemento (*crabs*)

0.48 Indexar una lista 2 (RRR)

A partir de la lista generada en el punto anterior obtener las filas pares de ***crabs***, pero solamente de la quinta y sexta columna, utilizando para seleccionar las columnas:

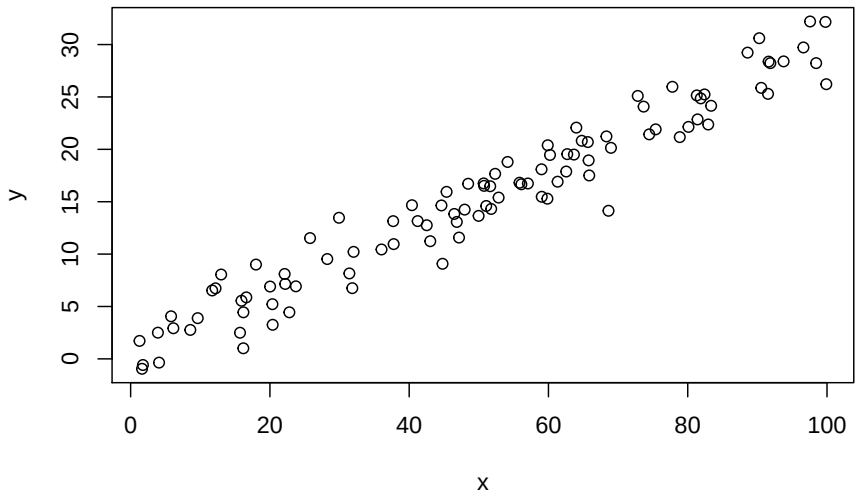
- el nombre de las columnas
- un vector numérico

0.49 Obtener valores de una salida de prueba estadística (RRR)

Vamos a ejemplificar el uso de algunos de los resultados obtenidos de una prueba estadística, que en este cas será una regresión lineal.

Las siguientes líneas generan datos simulados de una variable independiente (x) y de una variable dependiente (y). La variable está simulada con un error normal aditivo, con un intercepto = 0.2 y una pendiente = 0.3.

```
x <- runif(100, 1, 100) ## Valores de x
y <- 0.2 + 0.3 * x + rnorm(100, 0, 2) ## valores de y agregando un error normal aditiv
plot(x, y)
```



```
modelo <- lm(y ~ x) # modelo lineal simple (distribución normal) de y en función de x
```

1. ¿Qué tipo de objeto es modelo? Si la respuesta no es informativa recurrir a la ayuda de lm y leer la sección **value**

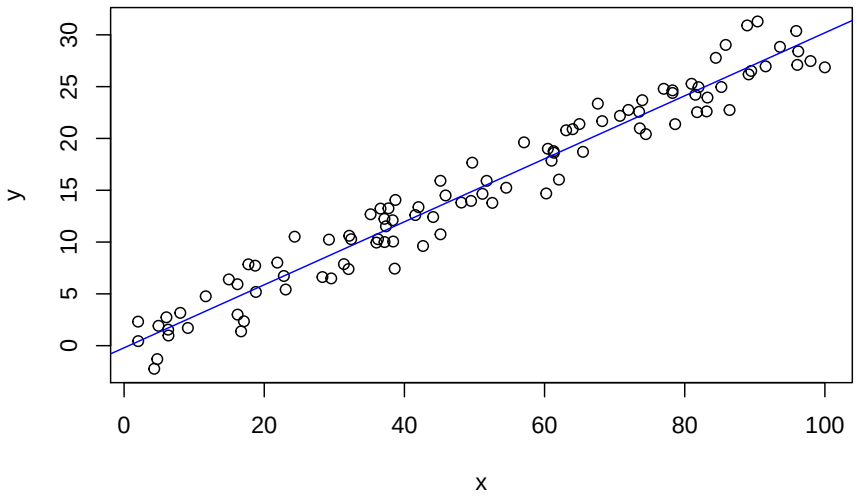
2.¿Cuántos elementos tiene modelo?

3. Utilizar los coeffientes del modelo (mediante indexación) para obtener el valor esperado de x= 3

```
## (Intercept)
##    1.377057
```

0.50 Valores esperados lm (RRR)

1. En base a los valores de los coeficientes (a, b) y utilizando la función ***abline()*** agregar los valores esperados a la figura del punto anterior



0.51 Debugging, detectando errores

Correr las siguientes líneas de comando para generar datos de prueba. Examinar cada uno de los objetos

```
rm( list = ls() ) #eliminemos primero todos los objetos
```

```
  x <- runif(100, 10, 100)
        y <- rnorm(100, 10 ,100)
        z <- seq(0, 200, 2)
        g  <- sample(letters, 100, T)

        tabla <- data.frame(x, y, g)
        matriz <- matrix( c(x, y, g ), ncol=3)
        lista <- list(tabla, matriz)
```

Para cada una de las siguientes líneas de código explicar por que se obtiene un mensaje de error, warnning, o un resultado no esperado

1.

```
dim( data.frame(x, y, z) )
```

2.

```
dim(X)
```

3.

```
dim(y)
```

4.

```
cbind(x, y, z)
```

5.

```
matrix(1:10, ncol=2, nrow=3)
```

7.

```
matrix( c(x, y, z), ncol=3 )
```

8.

```
matriz$x
```

9.

```
names( lista[1] )
```

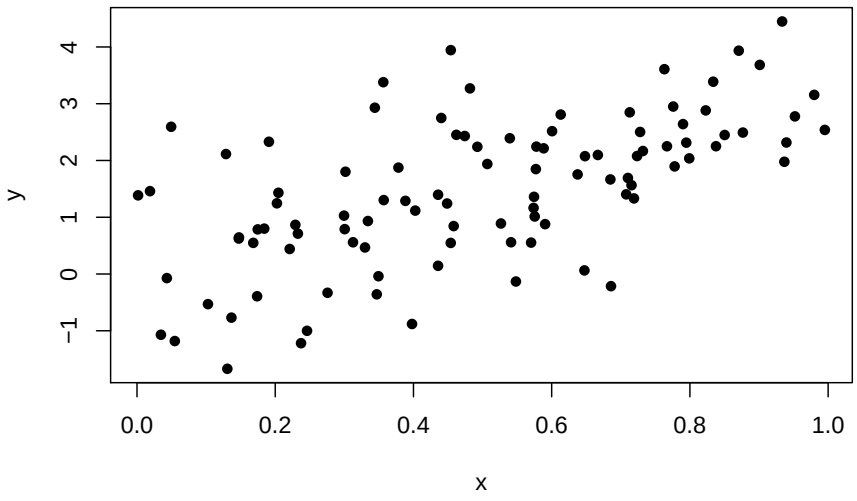
R09 Gráficos 1

0.52 Figura sencilla (R)

Dados los siguientes objetos:

```
set.seed(330)
x <- runif(100)
set.seed(100)
y <- 3*x + rnorm(100)
```

Obtener una figura como se muestra a continuación



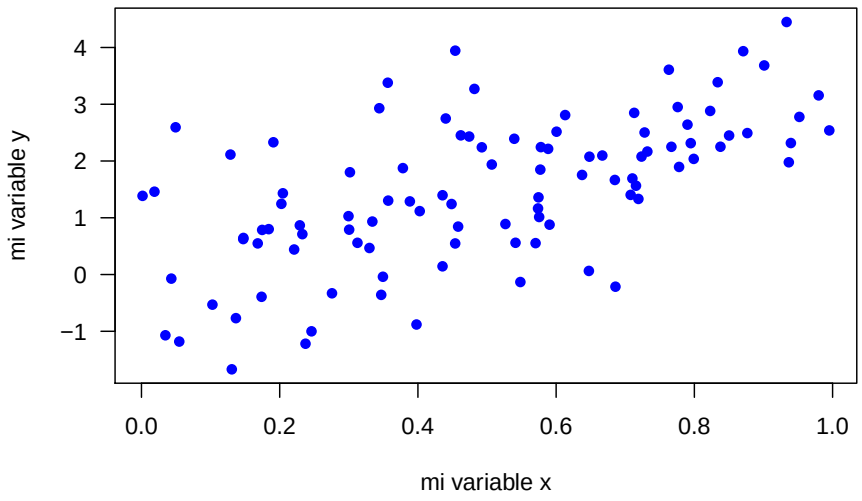
0.53 Configurar figuras (RR)

Modificar la figura del punto anterior con las siguientes características

- El color de los puntos debe ser azul
- La leyenda de los ejes y y x deberán decir respectivamente "mi variable y" y "mi variable x"

0.54 axis (RRR)

Modificar el script para rotar la leyenda de los valores del eje y, de la figura anterior, como se muestra en la siguiente figura



Para hacerlo examinar las opciones del parámetro ***las***

0.55 Figura sencilla (RR)

1. Importar el archivo DoubsEnv.csv que se encuentra en la carpeta 4-Datos (la descripción de la base de datos se encuentra en el archivo datos_eer.html)

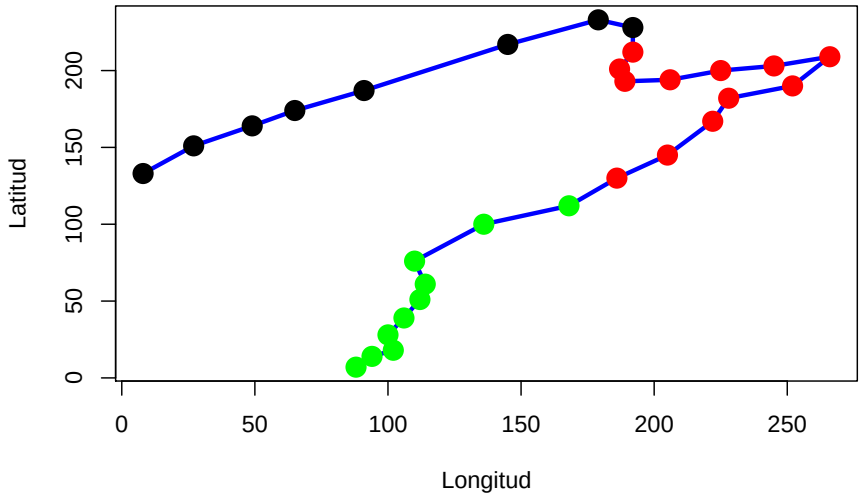
```
  Site lon lat  das alt  pen  deb  pH dur  pho  nit  amm  oxy dbo   sec
1    1  88   7  0.3 934 48.0 0.84 7.9  45 0.01 0.20 0.00 12.2 2.7 upper
```

```
2     2  94  14  2.2 932  3.0 1.00 8.0  40 0.02 0.20 0.10 10.3 1.9 upper
3     3 102  18 10.2 914  3.7 1.80 8.3  52 0.05 0.22 0.05 10.5 3.5 upper
4     4 100  28 18.5 854  3.2 2.53 8.0  72 0.10 0.21 0.00 11.0 1.3 upper
5     5 106  39 21.5 849  2.3 2.64 8.1  84 0.38 0.52 0.20  8.0 6.2 upper
6     6 112  51 32.4 846  3.2 2.86 7.9  60 0.20 0.15 0.00 10.2 5.3 upper
```

2. Graficar mediante gráfico de dispersión (el default de plot) la relación entre las variables nit (x) y amm (y). Rotular los ejes x e y como se muestra en la figura

0.56 Posiciones Doubs (RRR)

1. Importar el archivo DoubsEnv.csv que se encuentra en la carpeta 4-Datos (la descripción de la base de datos se encuentra en el archivo datos_eer. html)
2. Obtener la siguiente figura de las posiciones de los sitios de muestreo (Site, de la tabla DoubsEnv). Los colores negro, rojo y verde corresponden a las secciones (sec, de la tabla DoubsEnv) lower, middle y upper respectivamente.



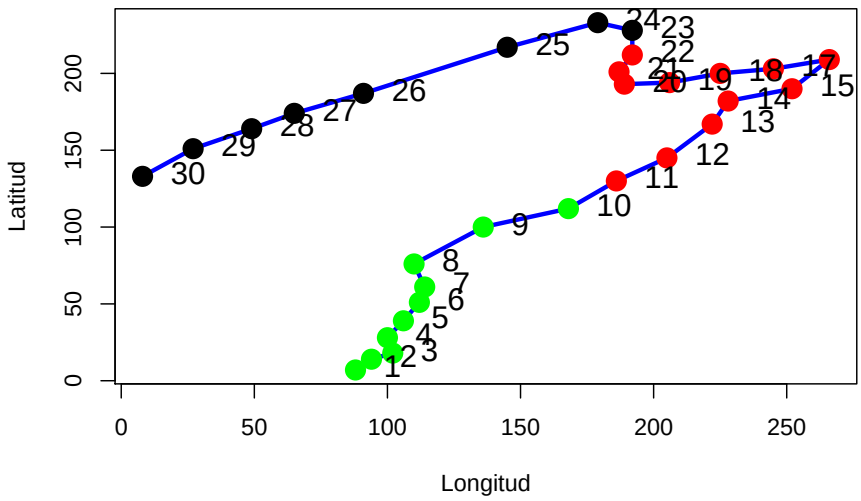
Pasos sugeridos

- Generar una figura con toda la serie de datos (x= lon, y= lat), con lty= "l" (tipo de figura línea)

- Generar tres objetos, cada uno con los registros correspondientes a las tres secciones del río (los objetos podrían llamares lower, middle y upper)
- mediante points() agregar a la figura los puntos correspondientes a cada una de las secciones, utilizando los objetos recién generados

0.57 Rotulando puntos (RRR)

1. Importar el archivo DoubsEnv.csv que se encuentra en la carpeta 4-Datos (la descripción de la base de datos se encuentra en el archivo datos_eer.html)
2. Obtener la siguiente figura que corresponde a las posiciones de las estaciones de muestreo (variables lon y lat) y cuyo color corresponde a la sección (negro, rojo y verde para las secciones lower, middle y upper respectivamente). Donde cada número se refiere al ID del sitio (variable Site). Este paso puede hacerse mediante la función text(), donde los valores de $\boldsymbol{x}$ y $\boldsymbol{y}$ son los mismos usados en la figura

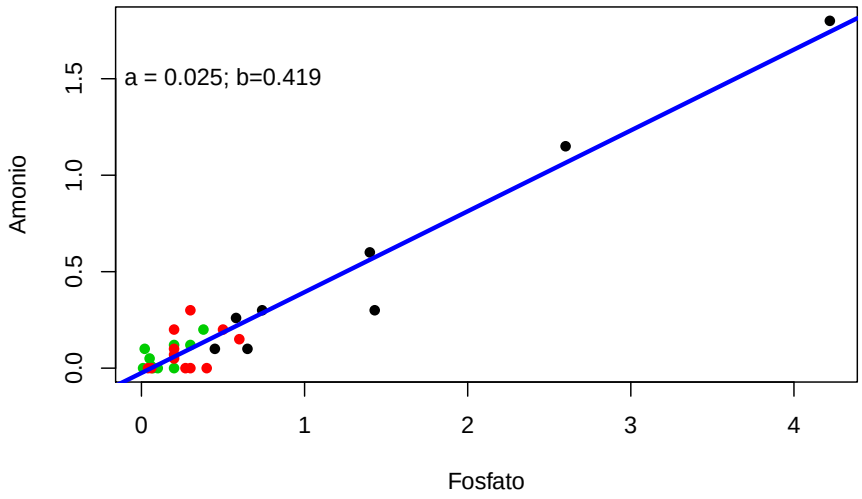


0.58 Regresión (RRRR)

1. Importar el archivo DoubsEnv.csv que se encuentra en la carpeta 4-Datos (la descripción de la base de datos se encuentra en el archivo datos_eer.

html)

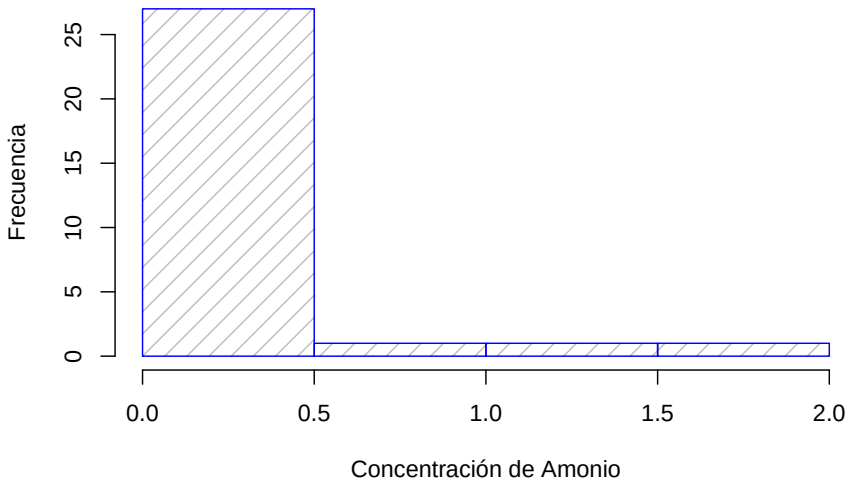
2. Obtener la siguiente figura tomando las variables amm y pho como ***y*** y ***x*** respectivamente. Para agregar la línea de regresión utilizar la función ***abline()***, tomando los valores de pendiente e intercepto que se muestran en la figura. Los colores negro, rojo y verde corresponden a las secciones lower, middle y upper respectivamente.



0.59 Histograma (RRR)

1. Importar el archivo DoubsEnv.csv que se encuentra en la carpeta 4-Datos (la descripción de la base de datos se encuentra en el archivo datos_eer. html)

Obtener la siguiente figura en base a la variable amm de DoubsEnv. Prestar atención a leyendas, tamaño de texto, colores y patrones de colores. Recurrir a la ayuda de hist()

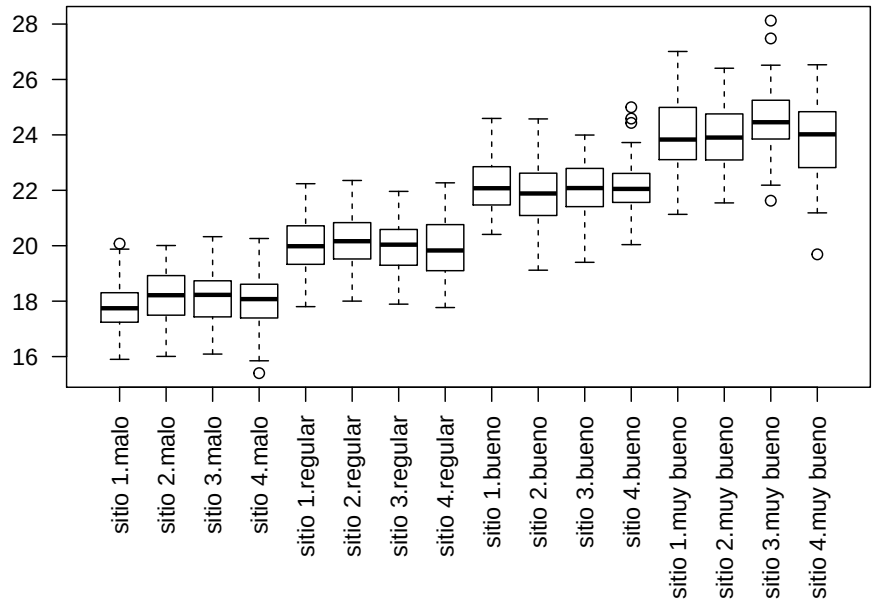


0.60 Factores para ordenar figuras (RRR)

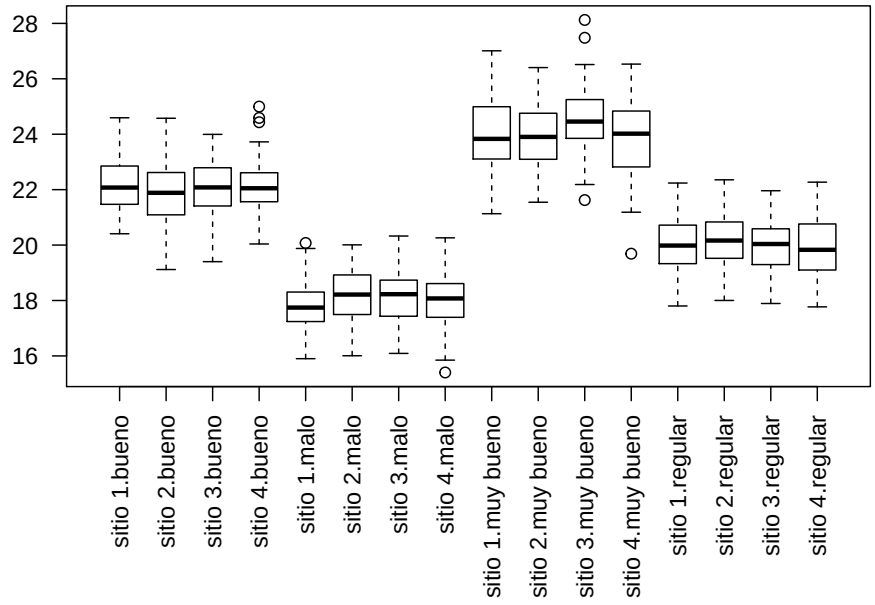
El archivo soil.RData contiene datos simulados de un experimento hipotético. En el experimento se tomaron valores de una variable hipotética (var) en cuatro sitios. En cada sitio se seleccionaron condiciones de suelo clasificadas en cuatro categorías en función de su calidad (fact).

1. Explorar visualmente el objeto x. Utilizar str() para ver las características de sus variables

2. Se quiere realizar una figura como la que se muestra a continuación para evaluar si hay diferencias entre sitios y los tipos de suelo. ** NOTA ** Por ahora no nos vamos a preocupar por embellecer la figura, simplemente nos interesa que se muestre en el orden adecuado. Como se muestra a continuación



Sin embargo boxplot() ordena a las variables por un índice alfabético, y por lo tanto con los datos recién generados la figura quedaría como se muestra a continuación (correr los siguientes comandos para confirmarlo).



Si bien los sitios aparecen ordenados nos gustaría que la figura también muestre

los boxplot de la variable considerando el órden de las calidades de suelo (fact) como se muestra en la primera figura.

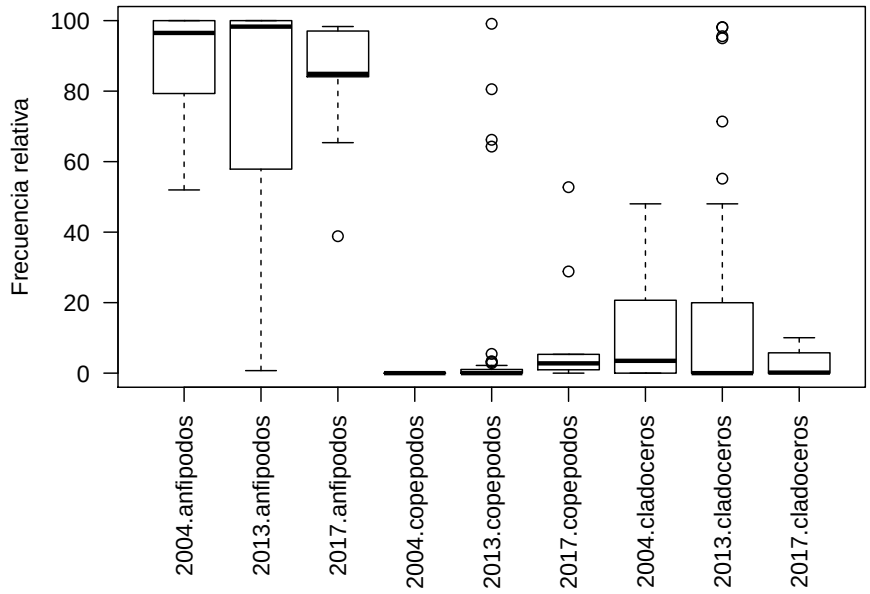
¿Cómo podríamos hacerlo? Este es uno de los casos donde los factores vienen a ayudarnos. La variable fact ya es un factor, pero éste no está ordenado!!! ** Ver la clase 07_Objetos1.r la sección de factores **

0.61 Usando reshape para generar boxplots (RRR)

El objeto *lago*, que se encuentra en el archivo *4-Datos/lago.RData*, contiene datos de frecuencia relativa de presas en contenidos estomacales de truchas.

```
            ID year anfipodos  copepodos cladoceros ostracodos parabroteas
1 str01022017-2 2017  38.84514 52.7559055  3.6745407          0           0
2   str030217-2 2017  84.44976  0.9569378 10.0478469          0           0
3   str050207-1 2017  97.05015  0.2949853  0.1474926          0           0
4   str050217-2 2017  84.89583  4.6875000  6.7708333          0           0
5   str050217-3 2017  65.38462 28.8461538  5.7692308          0           0
6 str07022017-1 2017  84.07080  0.0000000  0.0000000          0           0
    larvas  insectos
1 0.000000  4.724409
2 0.000000  4.545455
3 0.000000  2.507375
4 1.041667  2.604167
5 0.000000  0.000000
6 0.000000 15.929204
```

Se desea generar un gráfico boxplot mostrando variaciones en el contenido estomacal de truchas en tres años de muestreo, incluyendo solamente las tres presas más importantes en la dieta, como muestra la siguiente figura



Sin embargo, habrán notado que el objeto *lago* está en formato ancho, mientras que la función boxplot requiere que los datos estén en formato largo

Seguir los siguientes pasos para generar la figura 1. Generar un nuevo objeto x con las variables "year", "anfipodos","copepodos" y "cladoceros"

2. Usando la función adecuada del paquete _reshape2, cambiar de formato ancho a formato largo y definir value.name= "frec"

```
head(x); tail(x)
```

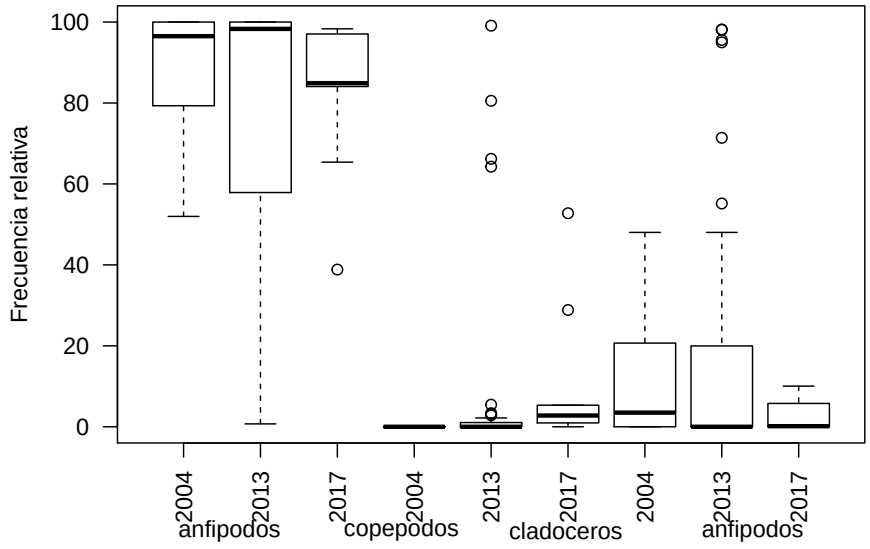
```
##   year  variable     frec
## 1 2017 anfipodos 38.84514
## 2 2017 anfipodos 84.44976
## 3 2017 anfipodos 97.05015
## 4 2017 anfipodos 84.89583
## 5 2017 anfipodos 65.38462
## 6 2017 anfipodos 84.07080
```

```
##     year   variable frec
## 193 2013 cladoceros    0
## 194 2013 cladoceros  NaN
## 195 2013 cladoceros  NaN
## 196 2013 cladoceros    0
## 197 2017 cladoceros    0
## 198 2017 cladoceros  NaN
```

3. Correr las siguientes líneas de comando para realizar la figura. NO es

necesario endender cada uno de las líneas de comando utilizadas. La figura final tiene algunas modificaciones para lograr un formato más agradable. Si llamaron al objeto con otro nombre que no sea x, cambiar únicamente las partes de boxplot donde se haga referencia al objeto x

```
ley <- rep( c(2004, 2013, 2017), 3) ## un vector con los nombres del eje x
if (Sys.info()['sysname'] == "Windows") windows() else x11() #chequea el sistema opera

par(mar= c(6,5,1,1)) # define los valores de los márgenes de la figura
boxplot(x$frec ~ x$year + x$variable, names= ley, las= 2, ylab= "Frecuencia relativa",
  par(usr= c(0, 10, 0, 10) ) # redefine la escala de x e y de la figura
  text(x= c(1.5, 3.8, 6, 8.5), y= -2, c("anfipodos","copepodos", "cladoceros"), xpd= N
```




```
dev.off()
```

```
## pdf
##   2
```

R10 Gráficos 2

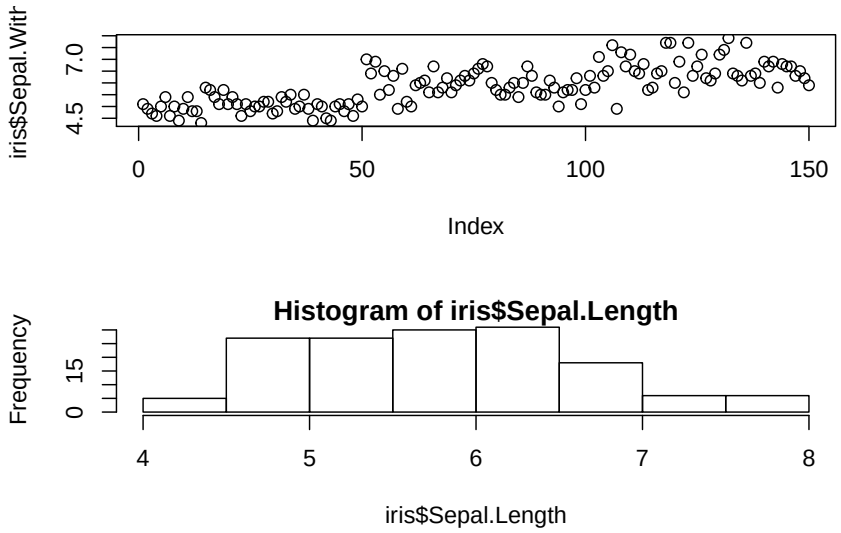
0.62 Figura múltiple simple (RR)

Trabajaremos con el objeto iris (del paquete base de R)

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Realizar una figura de dos paneles. En el panel superior graficar la relación entre las variables Sepal.Length y Sepal.With y en el panel inferior graficar un histograma de la variable Sepal.Length. En esta etapa no vamos a personalizar los ejes y leyendas, pero se espera que se modifiquen los parámetros ***mar*** y ***oma*** de ***par()***, para que la figura luzca similar a la que se muestra a continuación

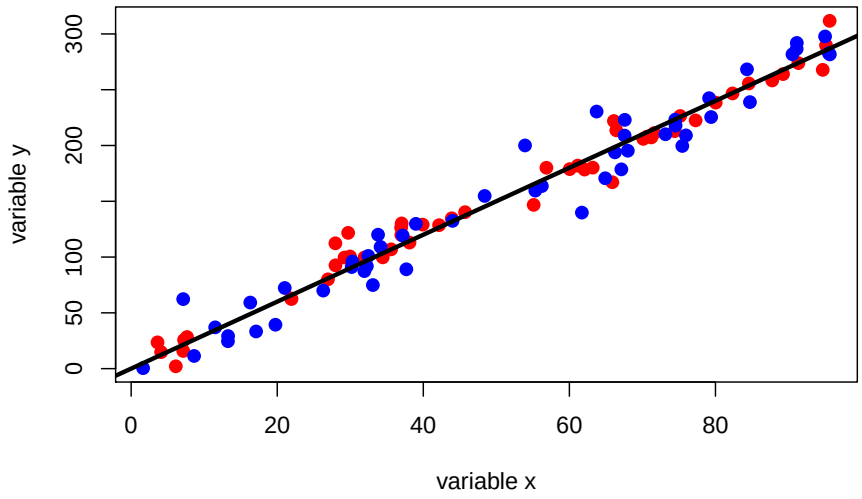


0.63 Dispersión 2 (RR)

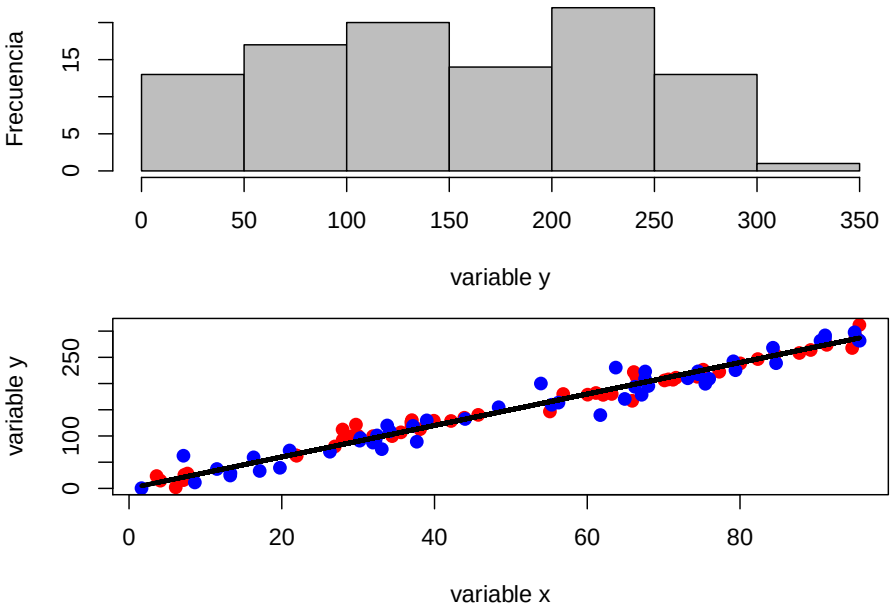
Dados los siguientes vectores (por ahora sólo importa el resultado y no como lo generamos)

Obtener la siguiente figura. Donde rojo corresponde a las hembras (H en el vector sexo) y azul a los machos (M en el vector sexo).
Ayuda: usar el vector sexo para indexar los vectores $\boldsymbol{x}$ y $\boldsymbol{y}$.

```
x <- runif(100, 0, 100)
y <- 0.03 + 3* rnorm(100, x, 5)
yest <- 0.03 + 3*x
sexo <- rep(c("H", "M"), each=50)
```



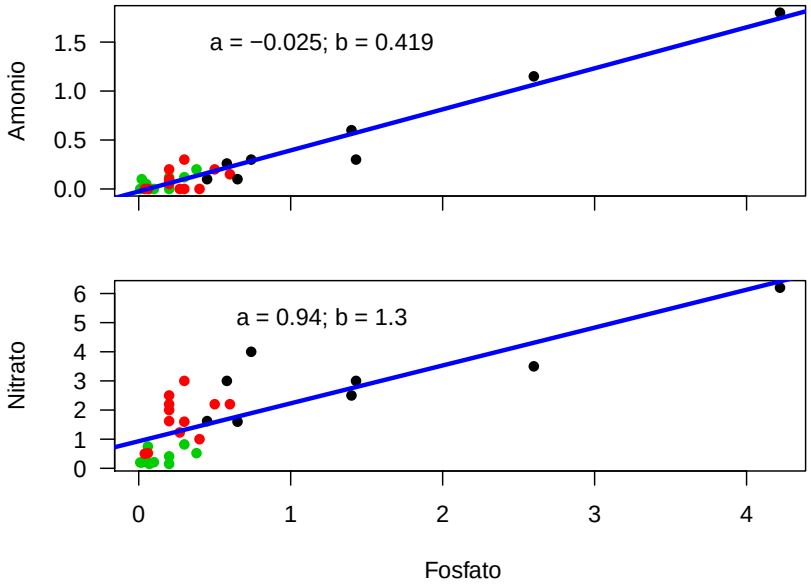
1.2 Obtener la siguiente figura. Prestar atención a leyendas, espacios entre figuras, tamaño de símbolos y texto y colores



0.64 Figura completa (RRRR)

1. Importar el archivo DoubsEnv.csv que se encuentra en la carpeta 4-Datos (la descripción de la base de datos se encuentra en el archivo datos_eer.html).

2. Generar una figura múltiple de dos paneles, mostrando dos gráfico de dispersión. El gráfico del panel superior corresponde a la relación amonio (variable amm) vs fosfato (variable pho) y el del panel inferior a nitrato (ni) vs fosfato (pho). Los colores negro, rojo y verde corresponden respectivamente a las secciones lower, middle y upper.



Lineamientos generales

- Definir la función ***par()*** para componer una figura con dos paneles (superior e inferior)

- Generar la primera figura, que no debe tener rótulos en el eje x, para lo cual hay que especificar en ***plot()*** que no queremos que lo grafique. Luego agregar un eje x mediante la funció ***axis()***
- Utilizar la función ***abline()*** para trazar la recta de regresión utilizando los valores que se muestran en la figura

- Agregar la leyenda con los valores de los parámetros a y b mediante la función ***text()***. Considerar la conveniencia de definir el parámetro ***usr*** dentro del ***par()*** (ver archivo de clase)

- Generar la segunda figura. Repetir los paso para la figura 1. En este caso el eje x tiene rótulos, pero queremos que la orientación de los rótulos de los ejes se muestren como en la figura de muestra. Por lo tanto, conviene volver a definir el eje utilizando ***axis()*** como en la primera figura.

- Utilizar los parámetros mar y oma para ajustar los márgenes internos y externos de la/las figuras

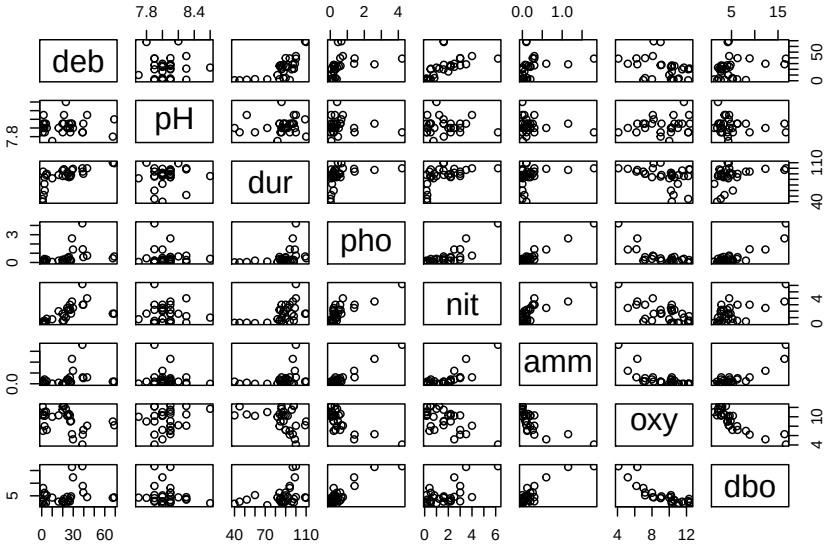
```
par( mfrow= c(), mar=c(), oma= c() )
plot()
axis()
abline()
par( usr= c() )
text()
#segunda figura
plot()
axis()
abline()
par( usr= c() )
text()
```

R11 Gráficos3

0.65 Pairs (RR)

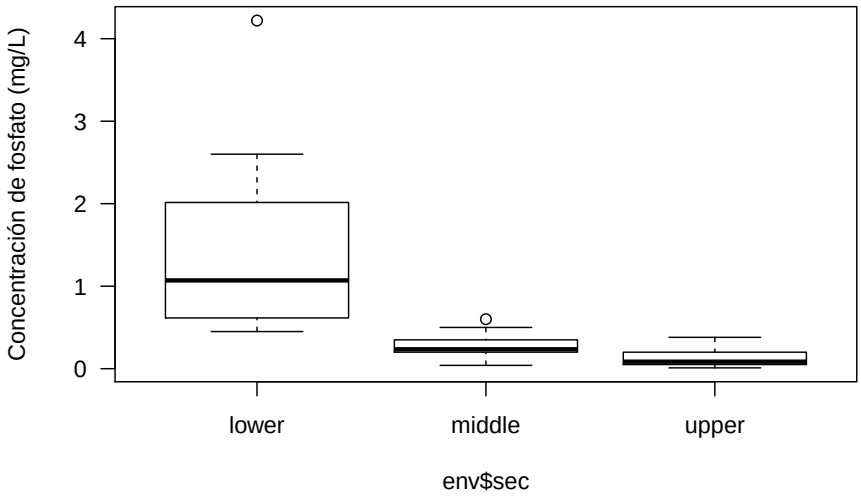
1. Asignar a un objeto que llamaremos ***env*** la tabla contenida en el archivo DoubsEnv.csv que se encuentra en la carpeta 4-Datos (la descripción de la base de datos se encuentra en el archivo datos_eer.html).

2. Utilizar la función *pairs()* para analizar la relación entre las variables correspondientes a las columnas 7 a 14 de ***env***.



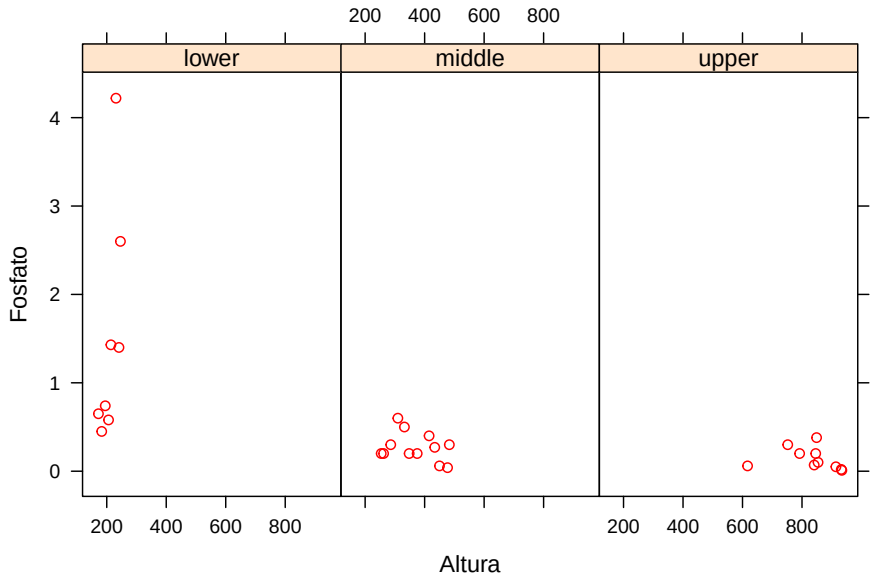
0.66 Boxplot

Generar una figura boxplot de la variable fosfato (pho) discriminando por sección del río Doubs. Notar la orientación de la leyenda de los ejes. Recurrir a la función ***axis()***.



0.67 Gráficos condicionales xyplot

Utilizando las función gráfica ***xyplot()*** del paquete ***lattice*** generar una figura múltiple como la mostrada a continuación

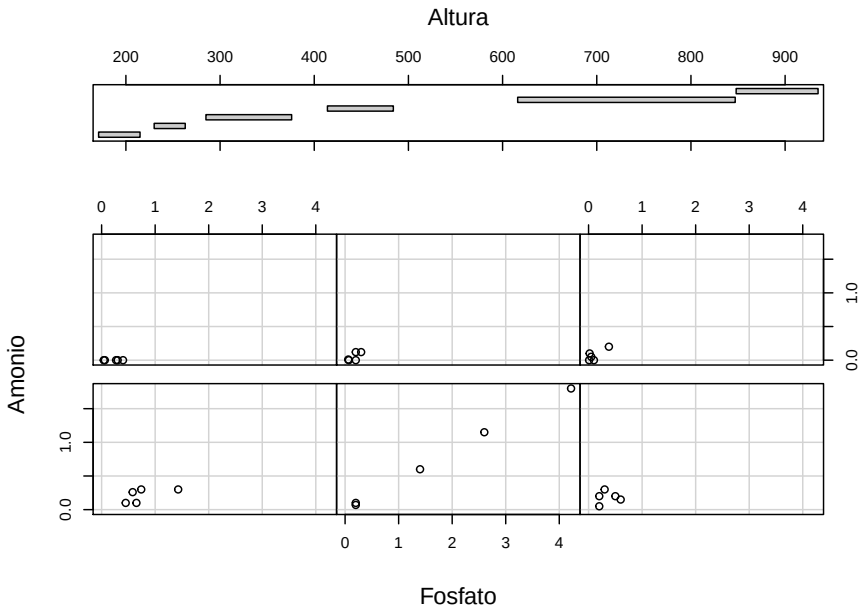



```
## X11cairo
##        3
```

La figura muestra la relación entre las variables fosfato (pho) y altura (alt) condicional a las tres secciones del río

0.68 Gráficos condicionales coplot

Utilizar la función coplot del paquete lattice para generar la siguiente figura




```
## pdf
##   2
```

La figura muestra la relación entre las variables amonio (amm) y fosfato (pho) condicional a rangos de valores de altura (alt), con parámetro de _**overlap= 0.1.**
NOTA** el argumento xlab es un vector con dos elementos para los ejes x inferior (primer elemento) y superior (segundo elemento). Entonces definir xlab= c("Fosfato", "Altura").

0.69 Debugging, detectando errores

Correr las siguientes líneas de comando para generar datos de prueba. Examinar cada uno de los objetos

```
rm( list = ls() ) #eliminemos primero todos los objetos
```

```
    x <- runif(100,10, 100)
    y <- rnorm(100, 10 ,100)
    z <- seq(0,200, 2)
        g  <- sample(letters,100, T)

        tabla <- data.frame(x,y,g)
        matriz <- matrix(c(x,y,g), ncol=3)
        lista <- list(tabla, matriz)
```

Para cada una de las siguientes líneas de código explicar por que se obtiene un mensaje de error, warnning, o un resultado no esperado

1.
```
plot(x,y, pch = 16, col = blue)
```

2.
```
mhist <- hist(x))
```

3.
```
plot(x, y, pch = 16, label = T, col = "orange", main="mifigura")
```

4.
```
plot(x, y, ylim(1,100))
```

5.
```
barplot(g)
```

6.
```
plot(x, y, col=c("red", "green" "blue"))
```

7.
```
plot(x, y, pch = 16, col = "orange", main="""")
```

R12 Resumir

0.70 summary (R)

Asignar a un objeto que llamaremos ***env*** la tabla contenida en el archivos DoubsEnv.csv, que se encuentra en la carpeta 4-Datos (la descripción de la base de datos se encuentra en el archivo datos_eer.html).

Asignar la salida de la función ***summary(env)*** a un objeto que llamaremos ***resumen*** * ¿Qué clase de objeto es ***resumen***?

0.71 table (R)

Aplicar la función ***table()*** a la variable sec ¿qué información nos da esta salida?

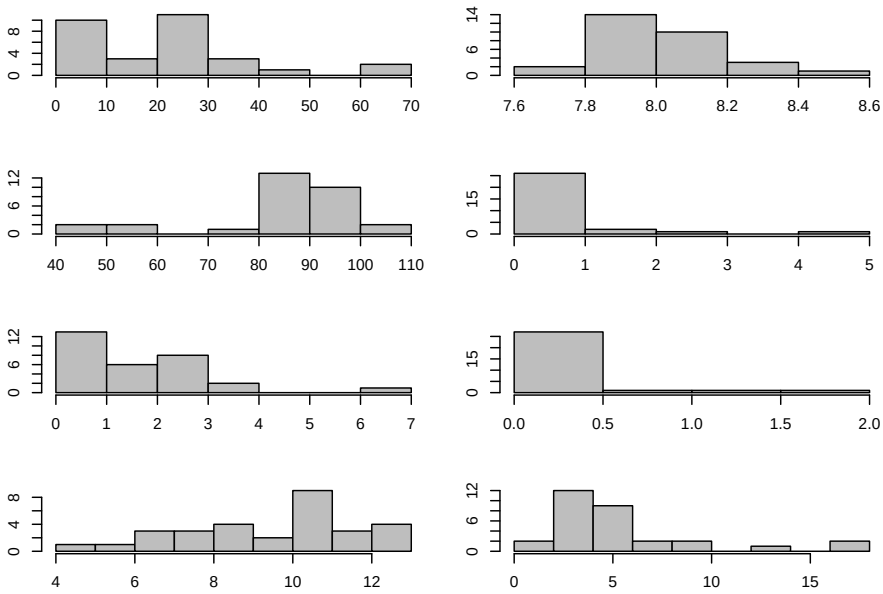
0.72 apply (RR)

1. Asignar a un objeto que llamaremos ***env*** la tabla contenida en el archivos DoubsEnv.csv que se encuentra en la carpeta 4-Datos (la descripción de la base de datos se encuentra en el archivo datos_eer.html).
2. Utilizando la función apply() calcular el rango de valores de cada una de las variables de campo de env (variables 4 a 14)

0.73 apply (RRR)

1. Asignar a un objeto que llamaremos ***env*** la tabla contenida en el archivos DoubsEnv.csv que se encuentra en la carpeta 4-Datos (la descripción de la base de datos se encuentra en el archivo datos_eer.html).

2. Utilizando la función ***apply()*** generar una figura múltiple de 8 paneles (4 filas y dos columnas) utilizando las columnas 7 a 14 de env. La figura se muestra a continuación



Estructura

```
par(mfrow=c(4,2), mar= c(3,2,2,2))
apply(COMPLETAR)
```

0.74 tapply (RRR)

Asignar a un objeto que llamaremos ***env*** la tabla contenida en el archivos DoubsEnv.csv que se encuentra en la carpeta 4-Datos (la descripción de la base de datos se encuentra en el archivo datos_eer.html).

Utilizar la función tapply para estimar los cuantiles 5 y 95 de la variable pho del objeto ***env*** categorizando por sección del río (sec). Asignar la salida a un objeto que llamaremos ***tabla***. La función para necesaria es quantile (recurrir a la ayuda)

- ¿Qué clase de objeto es ***tabla*** ?

0.75 Múltiples variables (RR)

1. Utilizar una función de la que hemos visto en la clase R12_Resumir para calcular el valor de la mediana de las variables ambientales del objeto ***env*** para cada una de las secciones del río

0.76 grep() which() resumir (RRRR)

El archivo "SP_aves_BI.csv", de la carpeta 4-Datos, contiene un listado de aves de Sudamérica categorizadas de acuerdo a su estado de conservación (BirdLife International Las especies fueron clasificadas en las siguientes categorías:

Threatened Species

CR - Critically Endangered

CR (PE) - Critically Endangered (Possibly Extinct)

CR (PEW) - Critically Endangered (Possibly Extinct In The Wild)

EN - Endangered

VU - Vulnerable

NT - Near Threatened

LC - Least Concern

DD - Data Deficient

Responder los siguientes puntos
1. ¿cuántas especies hay listadas en cada una de las categorías de amenaza?
2. ¿cuántas especies de cada género se encuentran en cada una de las categorías de amenaza?
3. ¿cuál es el género con mayor número de especies listadas?
4. Identificar las especies del género *Larus* (gaviotas) que se encuentren en las categorías NT (Near threatened) o VU (vulnerable).

0.77 Debugging, detectando errores

Correr las siguientes líneas de comando para generar datos de prueba. Examinar cada uno de los objetos

```
rm( list = ls() ) #eliminemos primero todos los objetos
```

```
x <- runif(100,10, 100)
y <- rnorm(100, 10 ,100)
z <- seq(0,200, 2)
```

```
g  <- sample(letters,100,T)

tabla <- data.frame(x,y,g)
matriz <- matrix(c(x,y,g), ncol=3)
lista <- list(tabla, matriz)
```

Para cada una de las siguientes líneas de código explicar porqué se obtiene un mensaje de error, warnning, o un resultado no esperado

1.

```
apply(lista[[1]], 2, mean)
```

```
apply(lista[[1]][ ,1:2], 2, mean) #funciona
```

2.

```
apply(lista[[1]][1:2], 2, mean)
```

3.

```
tapply(c(x,y), g, mean)
```

4.

```
colMeans(x)
```

R13 Condicionales

0.78 ifelse (RR)

Usaremos el objeto ***iris*** que viene instalado con el R. Asignar ***iris*** a un nuevo objeto al que llamaremos ***miX***.

```
miX <- iris
head(miX)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

Generaremos una nueva variable que llamaremos ***categoria***. A esta variable le asignaremos la categoría "Grande" para aquellos registros que cumplan la condición simultánea de que *Sepal.Length* sea mayor al valor de la mediana del vector *Sepal.Length* y *Sepal.Width* sea mayor al valor de la mediana del vector *Sepal.Width*, el resto de los registros serán categorizados como "Pequeños". Como primer paso se podría calcular el valor medio de esas variables mediante ***apply()***

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##         5.80         3.00         4.35         1.30
```

Si está bien resuelto la función table de miris$categoría debería dar

```
table(miX$categoria)
```

```
 Grande Pequeño
     25     125
```

0.79 ifelse (RRR)

Asignar a un objeto que llamaremos ***env*** la tabla contenida en el archivos DoubsEnv.csv, que se encuentra en la carpeta 4-Datos (la descripción de la base de datos se encuentra en el archivo datos_eer.html).

Generar una nueva variable a la que llamaremos ***cuadrante***. Cada registro tomará valores de cuadrantes (NO, NE, SO y SE) de acuerdo a su posición respecto de de los valores medios de latitud y longitud de ***env***. Por ejemplo, a aquellos registros cuyo valor de latitud y longitud sean mayores que el valor medio de las variables lon y lat respectivamente se les asignará el texto NE, como se muestra en la siguiente figura

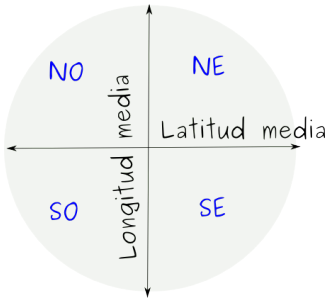


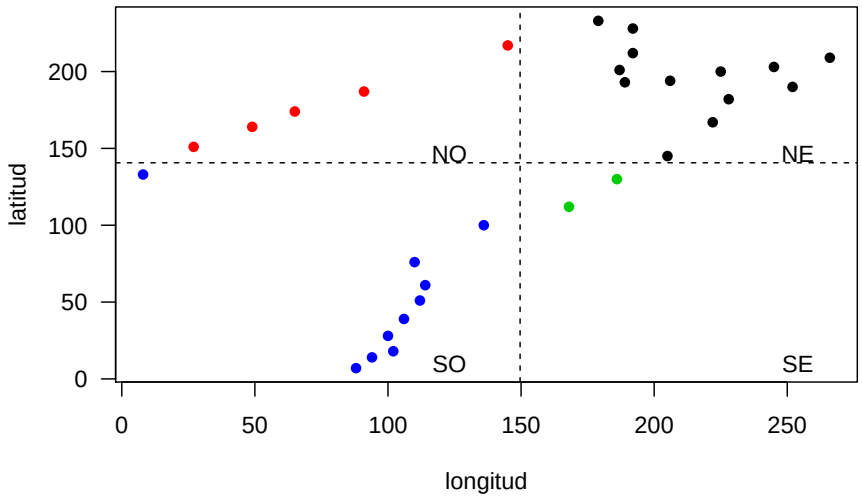
Figure 1: Cuadrantes definiddos por la posición media de longitud y latitud

Ayuda: se puede hacer un vector para las posiciones N – S y otro para las posiciones E – O. Luego mediante paste se pueden lograr todas las combinaciones posibles de cuadrantes

	Site	lon	lat	das	alt	pen	deb	pH	dur	pho	nit	amm	oxy	dbo	sec
1	1	88	7	0.3	934	48.0	0.84	7.9	45	0.01	0.20	0.00	12.2	2.7	upper
2	2	94	14	2.2	932	3.0	1.00	8.0	40	0.02	0.20	0.10	10.3	1.9	upper
3	3	102	18	10.2	914	3.7	1.80	8.3	52	0.05	0.22	0.05	10.5	3.5	upper
4	4	100	28	18.5	854	3.2	2.53	8.0	72	0.10	0.21	0.00	11.0	1.3	upper
5	5	106	39	21.5	849	2.3	2.64	8.1	84	0.38	0.52	0.20	8.0	6.2	upper
6	6	112	51	32.4	846	3.2	2.86	7.9	60	0.20	0.15	0.00	10.2	5.3	upper

0.80 Figura cuadrantes (RRR)

En base al resultado del punto anterior generar una figura como se muestra a continuación



Estructura

```
plot(env$lon, env$lat, xlab= "longitud", ylab= "latitud", yaxt="n", type= "n")
  points(COMPLETAR, pch= 16, col= "red")
  points(COMPLETAR, pch= 16, col= "black")
  points(COMPLETAR, pch= 16, col= "green")
  points(COMPLETAR, pch= 16, col= "blue")
  # eje x
  axis(side= 2, las= 2)
  # líneas medias de lon y lat
  abline(h= COMPLETAR, lty= 2)
  abline(v= COMPLETAR, lty= 2)
  # leyendas de cuadrantes
  par(usr= c(0,10,0,10))
  text("NE", x= 9.2, y= 6.1)
  text("NO", x= 4.5, y= 6.1)
  text("SE", x= 9.2, y= 0.5)
  text("SO", x= 4.5, y= 0.5)
```

0.81 if (RRR)

Supongamos que queremos ejecutar la siguiente operación:

```
        x= 1:10  ; x^2  # no hay ningún problema
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

Pero si hacemos

```
      x= letters[1:10]; x^2
```

Aparece un aviso de error `Error in x^2 : non-numeric argument to binary operator`

1. Qué tal si armamos nuestro propio aviso de error?

- Utilizar la función *if()* para que aparezca un aviso de error cuando la variable no sea numérica. Por ejemplo que diga "ESTA OPERACIÓN NO FUNCIONA PARA VARIABLES NO NUMÉRICAS"
- Un paso un poco más ambicioso es que además nos aparezca en consola un mensaje que nos indique de que clase es el objeto al que erroneamente intentamos aplicar la operación. **Ayuda**: combinar **paste()** y **class()**

0.82 if + plot (RRR)

Utilizar la función *if()* para generar una figura a partir de los valores de un único vector. Si el vector es de clase *integer* (números enteros) la figura será un histograma, mientras que si el vector es de clase *numeric* (números reales) se graficará un boxplot con los valores del vector. Generar los comandos y ponerlos a prueba con los vectores que se dan a continuación

```
x <- as.integer( sample(1:100, 100, replace=T) )
class (x)
```

```
## [1] "integer"
```

```
x <- seq(1, 10, 0.5)
class(x)
```

```
## [1] "numeric"
```

R14 Funciones

0.83 Funciones: elevar al cuadrado (R)

- Escribir una función que llamaremos **elev2** que calcule el cuadrado de una serie de números
- La función debe devolver en un data.frame con los números de ingreso y sus correspondientes valores al cuadrados

```
  x <-1:10
  elev2(x)
```

```
    x  x2
1   1   1
2   2   4
3   3   9
4   4  16
5   5  25
6   6  36
7   7  49
8   8  64
9   9  81
10 10 100
```

Estructura

```
elev2 <- function(completar) {
  y <- COMPLETAR ^2
  data.frame(x= x, x2 = y)
  }
```

0.84 Funciones: elevar al cuadrado 2 (RRR)

- Agregar a la estructura de la función anterior un "control", para que en los casos en que el vector de ingreso no sea numérico aparezca un cartel

que diga “ERROR: Los datos de ingreso no son numéricos”

```
  x <-c(1:10, NA, NA, NA)
  elev2(x)
```

```
    x  x2
1   1   1
2   2   4
3   3   9
4   4  16
5   5  25
6   6  36
7   7  49
8   8  64
9   9  81
10 10 100
11 NA  NA
12 NA  NA
13 NA  NA
```

```
  y <- letters
  elev2(y)
```

```
[1] "ERROR: Los datos de ingreso no son numéricos"
```

Estructura

```
elev2 <- function(COMPLETAR) {
      if(COMPLETAR) {
        COMPLETAR CON COMANDOS SI T
        } else {
        COMPLETAR CON COMANDOS SI F
       }
    }
```

0.85 Funciones: Promedio (RR)

Generar una función a la que llamaremos “promedio”, que calculará el valor promedio de un vector numérico como:

$$promedio = \frac{\sum x_i}{N}$$

Donde N es número de elementos del vector x (length(x)) Por el momento la función tendrá un único argumento x y no estará definida para vectores con NA’s

Por ejemplo:

```
y <- c(3, 5, 4, NA, NA, 8)
promedio(y)
```

```
## [1] NA
```

```
# En este caso el resultado es NA porque sum(x) = NA

z <- c(3, 5, 4, 4, 5, 8)
promedio(z)
```

```
## [1] 4.833333
```

0.86 Funciones: Promedio 2 (RRR)

Generar la misma función que en el punto anterior pero con un argumento extra que permita eliminar los valores NA para el cálculo del promedio

```
y <- c(3, 5, 4, NA, NA, 8)
promedio(y)
```

```
## [1] 5
```

0.87 Función obtener números pares o impares

Dado un vector numérico cualquiera, como el que se muestra a continuación:

```
x <- c(1,2,5,7,9,8,2,51)
```

Mediante el siguiente comando se puede obtener un vector de 1's y 0's para aquellos valores impares y pares del vector

```
x %% 2
```

```
## [1] 1 0 1 1 1 0 0 1
```

Generar una función, que llamaremos **parimpar** que permita obtener los valores pares o impares de un vector numérico. La función deberá tener dos argumentos, el primero, al que llamaremos **vec**, corresponderá al vector de entrada y el segundo al que llamaremos impar tomará valor por defecto TRUE y seleccionará a los valores impares, mientras que seleccionará a los pares en caso de ser FALSE

Pasos sugeridos * generar un vector lógico (idx1) con la condición vec %% 2 == 1 (impares) * generar un vector lógico (idx2) con la condición vec %% 2 == 0 (pares) * incluir un condicional que indexe por idx1 en caso de que el parámetro impar sea T y que indexe por idx2 en caso contrario

Probando la función

```
x <- c(1,2,5,7,9,8,2,51)

parimpar(x)
```

```
## [1]  1  2  9  2 NA 51  2 NA
```

```
parimpar(x, F)
```

```
## [1] 2 8 2
```

0.88 Funciones: No contar los NAs (RRR)

Las funciones de la familia ***apply*** no tienen argumentos para eliminar los NA's de los resultados cuando se usa ***length()*** Ejemplo

```
    x <- c( 1:20, rep(NA, 4) )
    x
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 NA NA NA NA
```

```
        y <- rep( c("A", "B"), each=12)
    y
```

```
 [1] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "B" "B" "B" "B" "B" "B" "B"
[20] "B" "B" "B" "B" "B"
```

```
        z <- data.frame(pob=y,x=x)
    z
```

```
   pob  x
1    A  1
2    A  2
3    A  3
4    A  4
5    A  5
6    A  6
7    A  7
8    A  8
9    A  9
10   A 10
11   A 11
12   A 12
13   B 13
14   B 14
15   B 15
16   B 16
17   B 17
```

```
18   B 18
19   B 19
20   B 20
21   B NA
22   B NA
23   B NA
24   B NA
```

```
        tapply(z$x, z$pob, mean, na.rm=T) #esto funciona
```

```
   A    B
 6.5 16.5
```

```
    tapply(z$x, z$pob, length) #esto funciona pero cuenta los NA's
```

```
 A  B
12 12
```

1. Generemos entonces una función que llamaremos ***sinNA*** para usar funciones de la familia ***apply*** y que no cuente los NA's

La salida de tapply() usando la función debería dar:

```
    tapply(z$x, z$pob, sinNA)
```

```
##  A  B
## 12  8
```

0.89 Función frecuencia relativa (RRR)

Muchas veces es necesario expresar valores de tablas en forma de frecuencia relativa. Es decir, pasar valores absolutos de observaciones a formato de frecuencia

$$\text{FR} = \frac{Ni}{N} * 100$$

Donde Ni es el valor de la celda i y N es la suma de todos los valores del vector

Los siguientes comandos generan datos simulados de 5 poblaciones de una especie hipotética de depredador que consume 5 tipos de presa.

```
presas <- paste("presa", 1:5)
pobl <- paste("pob", 1:5)
set.seed(300)
datos <- round( runif(25, 1, 100), 0)
tabla <- matrix(datos, ncol=5, dimnames=list(presas, pobl))
tabla
```

```
##         pob 1 pob 2 pob 3 pob 4 pob 5
## presa 1    92     2    94    99    99
## presa 2    77    76    33    89    50
```

```
## presa 3    81    50    80    67    47
## presa 4    74    47    59    64     7
## presa 5    69    90    66    51    10
```

1. El ejercicio consiste en estimar el índice de frecuencia relativa de las presas para cada ejemplar. Para ello, generaremos una función (**FR**) que calcule la frecuencia de ocurrencia en base a un vector de observaciones. Finalmente, la función será utilizada como argumento de la función de resumen en el ejercicio que sigue.

Aplicando la función recién creada a la primera fila de la tabla debería dar el siguiente resultado

```
prueba <- FR(tabla[1, ])
prueba
```

```
##      pob 1      pob 2      pob 3      pob 4      pob 5
## 23.8341969  0.5181347 24.3523316 25.6476684 25.6476684
```

```
sum(prueba)
```

```
## [1] 100
```

La tabla de salida debería verse como la siguiente (los valores obtenidos fueron redondeados (***round()***) a dos decimales)

```
salida <- round (apply(tabla, 1, FR), 2)
salida
```

```
##       presa 1 presa 2 presa 3 presa 4 presa 5
## pob 1   23.83   23.69   24.92   29.48   24.13
## pob 2    0.52   23.38   15.38   18.73   31.47
## pob 3   24.35   10.15   24.62   23.51   23.08
## pob 4   25.65   27.38   20.62   25.50   17.83
## pob 5   25.65   15.38   14.46    2.79    3.50
```

```
colSums(salida) ## la suma de las frecuencias realtivas por columna debe dar 1
```

```
## presa 1 presa 2 presa 3 presa 4 presa 5
##  100.00   99.98  100.00  100.01  100.01
```

–>

0.90 Debugging, detectando errores

Correr las siguientes líneas de comando para generar datos de prueba. Examinar cada uno de los objetos

```
rm( list = ls() ) #eliminemos primero todos los objetos
```

```
  x <- runif(100,10, 100)
    y <- rnorm(100, 10 ,100)
        z <- seq(0,200, 2)
        g  <- sample(letters,100,T)

        tabla <- data.frame(x,y,g)
        matriz <- matrix(c(x,y,g), ncol=3)
        lista <- list(tabla, matriz)
```

Para cada una de las siguientes líneas de código explicar por que se obtiene un mensaje de error, warnning, o un resultado no esperado

1.

```
mm <- ifelse(x >= 23 & y =< 5, "target", NA)
```

2.

```
log(y)
```

3.

```
dim.names(tabla)
```

4.

```
if(is.numeric(x)) hist(x) print("la variable no es numerica")
```

4.

```
if(is.numeric(g)){
    hist(g)
        } else
        print("la variable no es numerica")
  }
```

5.

```
if(x>5) "grande" else "chico"
```

R15 Loops

0.91 Loops: paso a paso

En este ejemplo vamos a analizar la conveniencia de utilizar loops cuando realizamos tareas repetida.

Supongamos que tenemos los siguientes datos

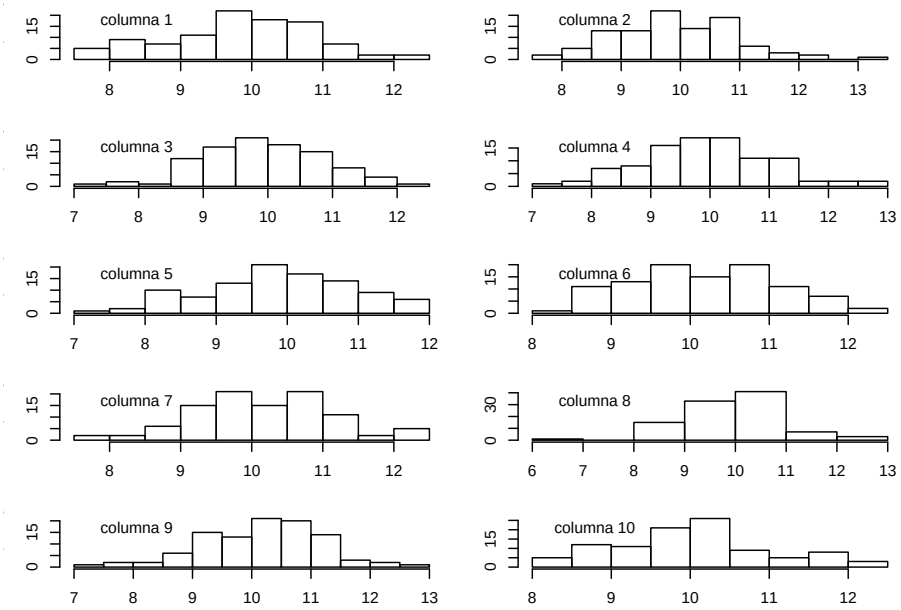
```
x <- matrix(rnorm(1000, 10, 1), ncol= 10) # una matríz de diez columnas, con números aleatorios
```

Ahora queremos generar una figura compuesta con histogramas de los valores de cada una de las columnas de x, indicando en cada figura cuál es la columna que se está graficando. Para lo cual generamos una ventana grafica y damos los comandos de diez histogramas, como se muestra a continuación.

```
nombres <- paste("columna", 1:10) # rótulo para indicar cuál es la columna que se está graficando
```

```
if (Sys.info()['sysname'] == "Windows") windows() else x11() ## abre una ventana gráfica usando l

par(mfrow = c(5,2), mar= c(3, 3, 1, 1))
#figura 1
hist(x[ ,1], main= "")
par(usr= c(0,10,0,10)) #redefinimos los ejes para poner el rótulo en la misma posición en todas l
text(x= 2, y= 8, nombres[1]) #rótulo indicando la columna que se está graficando

#figura 2
hist(x[ ,2], main= "")
par(usr= c(0,10,0,10))
text(x= 2, y= 8, nombres[2])

#figura 3
hist(x[ ,3], main= "")
par(usr= c(0,10,0,10))
text(x= 2, y= 8, nombres[3])

#figura 4
```

```
hist(x[ ,4], main= "")
par(usr= c(0,10,0,10))
text(x= 2, y= 8, nombres[4])

#figura 5
hist(x[ ,5], main= "")
par(usr= c(0,10,0,10))
text(x= 2, y= 8, nombres[5])

#figura 6
hist(x[ ,6], main= "")
par(usr= c(0,10,0,10))
text(x= 2, y= 8, nombres[6])

#figura 7
hist(x[ ,7], main= "")
par(usr= c(0,10,0,10))
text(x= 2, y= 8, nombres[7])

#figura 8
hist(x[ ,8], main= "")
par(usr= c(0,10,0,10))
text(x= 2, y= 8, nombres[8])

#figura 9
hist(x[ ,9], main= "")
par(usr= c(0,10,0,10))
text(x= 2, y= 8, nombres[9])

#figura 10
hist(x[ ,10], main= "")
par(usr= c(0,10,0,10))
text(x= 2, y= 8, nombres[10])
```



Muy bien, para hacerlo escribimos los comandos para hacer un histograma, luego lo copiamos 9 veces y reemplazamos en la indexación de cada caso por la columna y el nombre de la columna correspondiente. NO es tanto trabajo después de todo. Sin embargo, ahora vemos que la leyendas de los ejes no son las que nos gustaría tener, y el color no nos convence. Tendríamos entonces que editar uno por uno los comandos para modificar cada histograma. Esto además de ser tedioso aumenta las chances de que cometamos errores.

Una opción un poco más saludable es generar una sola vez los comandos e ir cambiando los valores de indexación (donde dice REEMPLAZAR en el código que sigue). Cada vez que lo corramos se irán completando los paneles de nuestra figura múltiple hasta que completemos los 10 histogramas

```
if (Sys.info()['sysname'] == "Windows") windows() else x11()

par(mfrow = c(5,2), mar= c(3, 3, 1, 1))
#figura
hist(x[ ,REEMPLAZAR], main= "")
par(usr= c(0,10,0,10))
text(x= 2, y= 8, nombres[REEMPLAZAR]) #rótulo indicando la columna que se está graficando
```

Para optimizar un poco más podríamos hacer la indexación usando un índice, que podríamos llamar i. En cada caso podríamos entonces dar valores a i de uno a 10, según corresponda en cada caso. La ventaja es que tendríamos que escribir una sola vez el valor de i en cada figura y no necesitamos "tocar" el código, de hecho podemos dar valores de i en consola

```
if (Sys.info()['sysname'] == "Windows") windows() else x11()
par(mfrow = c(5,2), mar= c(3, 3, 1, 1))

i = 5 # VAMOS CAMBIANDO LOS VALORES DE _i_ de 1 a 10, uno por vez y volvemos a correr

#figura
hist(x[ ,i], main= "")
par(usr= c(0,10,0,10))
text(x= 2, y= 8, nombres[i])

## FINAL
```

Muy bien, avanzamos bastante, ahora podríamos cambiar cualquier argumento del plot y simplemente deberíamos repetir estos pasos otra vez 10 veces. Pero mejor todavía, podríamos dar los comandos adecuados para que las repeticiones las haga R!!!! Aquí es donde vienen los loop en nuestra ayuda. En las siguientes líneas de comando reemplazar los tres puntos por lo que corresponda para que el loop haga la tarea por nosotros

```
if (Sys.info()['sysname'] == "Windows") windows() else x11()

par(mfrow = c(5,2), mar= c(3, 3, 1, 1))

for(i in ...){

#figura
hist(x[ ,...], main= "")
par(usr= c(0,10,0,10))
text(x= 2, y= 8, nombres[...])

} ## FINAL
```

0.92 loop muy sencillo (R)

Asignar el vector letters a un objeto que llamaremos ***miX***

```
miX <- letters
miX
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
```

a) Generar un loop que itere sobre el vector ***miX*** y que en cada iteración muestre en pantalla la palabra “letra:” y la letra que corresponda, como se muestra a continuación

Estructura

```
for(i in miX)  print( paste("letra:", COMPLETAR, sep= " ") )
```

b) Usaremos el mismo objeto ***miX***, pero ahora se espera que el vector de iteración sea numérico de 1:length(***miX***)

Estructura

```
n <- length(miX)
for(i in 1:n)  print( paste("letra:", miX[COMPLETAR], sep= " ") )
```

c) Modificar el loop del punto *b)* para que también se imprima en pantalla la iteración que corresponda, como se muestra a continuación

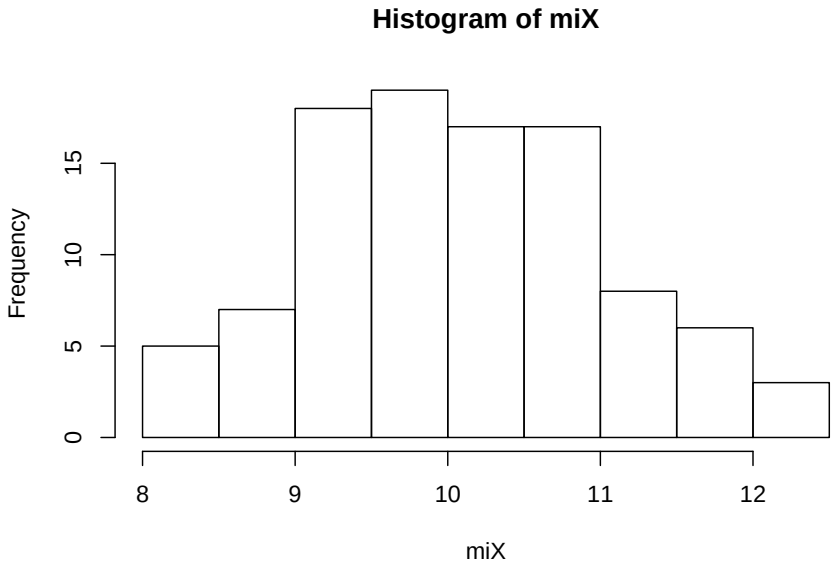
```
## [1] "letra: a iteración: 1"
## [1] "letra: b iteración: 2"
## [1] "letra: c iteración: 3"
## [1] "letra: d iteración: 4"
## [1] "letra: e iteración: 5"
## [1] "letra: f iteración: 6"
## [1] "letra: g iteración: 7"
## [1] "letra: h iteración: 8"
## [1] "letra: i iteración: 9"
## [1] "letra: j iteración: 10"
## [1] "letra: k iteración: 11"
## [1] "letra: l iteración: 12"
## [1] "letra: m iteración: 13"
## [1] "letra: n iteración: 14"
## [1] "letra: o iteración: 15"
## [1] "letra: p iteración: 16"
## [1] "letra: q iteración: 17"
## [1] "letra: r iteración: 18"
## [1] "letra: s iteración: 19"
## [1] "letra: t iteración: 20"
## [1] "letra: u iteración: 21"
## [1] "letra: v iteración: 22"
## [1] "letra: w iteración: 23"
## [1] "letra: x iteración: 24"
## [1] "letra: y iteración: 25"
## [1] "letra: z iteración: 26"
```

0.93 Figura múltiple (RR)

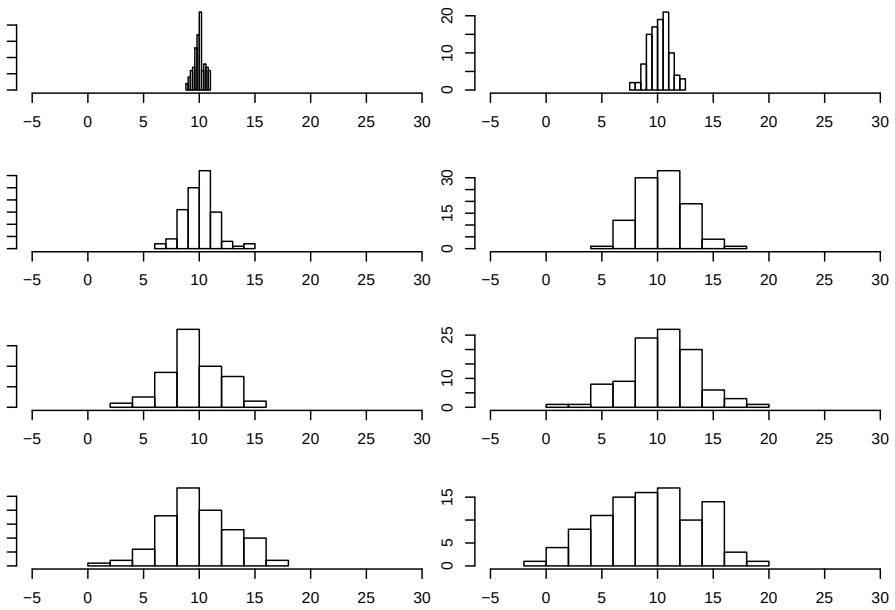
Queremos evaluar visualmente cambios en la dispersión de un conjunto de datos aleatorios, con distribución normal, cuando modificamos el valor de sigma.

Sabemos que la siguiente función nos va a dar 100 valores aletorios con estructura de error normal, con media= 10 y desvío estandar = 1

```
miX <- rnorm(100, 10, 1)
hist(miX)
```



El objetivo es generar mediante un loop una figura de paneles múltiples (4 filas y dos columnas), donde en cada panel se graficará un histograma con distribuciones normales de 100 valores, con media= 10. Cada figura tendrá un valor de sd definido a priori de acuerdo al vector miSD, como se muestra a continuación

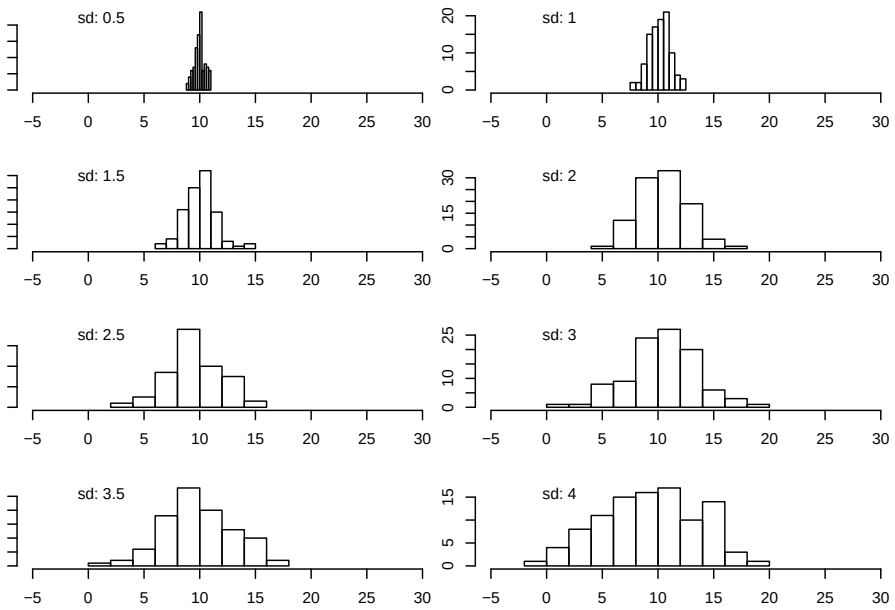


Estructura

```
set.seed(333) # definimos la semilla para que todos obtengamos los mismos números pseudoaleatorio

miSD <- seq(from= 0.5, to= 4, by= 0.5) # valores de sd que queremos evaluar

par(mfrow= c(4, 2), mar= c(3, 1, 1, 1) ) #definimos los paneles y márgenes de las figuras

  for (i in miSD){
    miX <- rnorm( 100, 10, sd= COMPLETAR )
    hist(miX, main= "", ylab="", xlim= c(-5, 30))
    }
```

0.94 Figura múltiple con leyenda (RRR)

Dado que usamos par(mfrow) en el ejercicio anterior sabemos que los valores de sd se fueron completando por filas. Pero el lector, reviewer, etc. no tiene acceso al script. ¿Que tal si en cada figura agregamos una leyenda con el valor de sd utilizado?. El resultado debería verse como en la siguiente figura



Estructura

Recordemos que con par(usr= c(0, 10, 0, 10)) reescaleamos nuestra figura para independizarnos de los valores de los ejes x y y (ver R10.Gráficos2)

```
## RTA
set.seed(333)
miSD <- seq(from= 0.5, to= 4, by=0.5)
par(mfrow= c(4,2), mar= c(3, 1, 1, 1) ) #definimos los paneles y márgenes de las figur

  for (i in miSD){
    miX <- rnorm( 100, 10, sd= COMPLETAR )
    hist(miX, main= "", ylab="", xlim= c(-5, 30))
    par( usr= c(0, 10, 0, 10) ) ## este comando redefine la escala de los ejes
    text(x= 2, y= 9, paste( COMPLETAR ) ) ## este comando agrega la leyenda en x y y d
    }
```

0.95 Efecto de la varianza (RRR)

En un modelo lineal determinístico el valor esperado de una variable dependiente (y) en función de una variable independiente (x) está dado por:

$$y_i = a + b * x_i$$

donde a y b son la ordenada al origen y la pendiente respectivamente

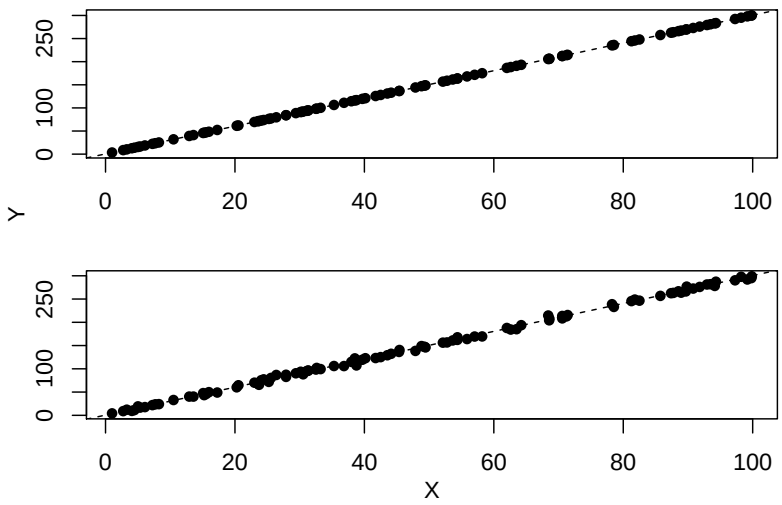
En un modelo estocástico se agrega un componente aleatorio. Por ejemplo, en una relación lineal puede incorporarse como un error aditivo

$$y_i = a + b * x + e_i$$

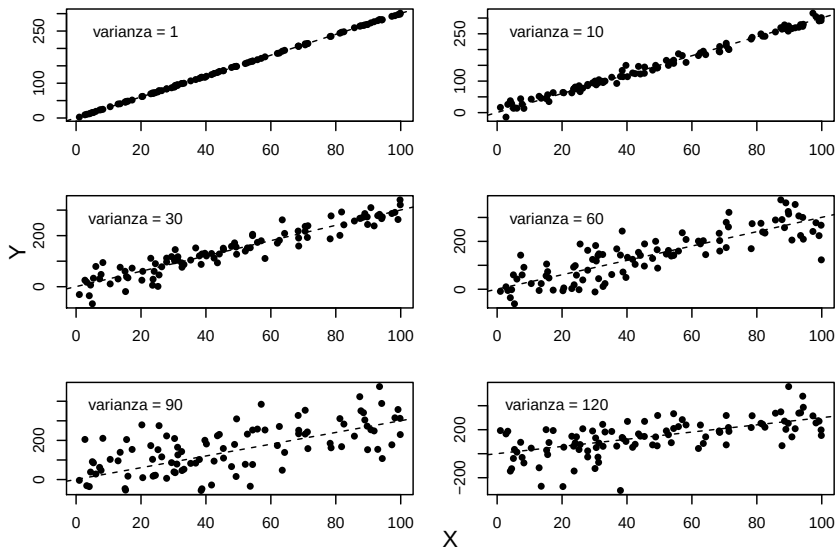
donde e_i es un error aleatorio con media 0 y varianza = sigma^2

Por ejemplo, el siguiente código genera una figura de dos paneles, donde se muestra el modelo determinístico (panel superior) y un modelo estocástico con error aditivo normal, con media = 0 y sd = 3

```
  x <- runif(100, 0, 100) # variable independiente
  ydet <- 0.5 + 3 * x # variable dependiente (modelo determinístico)
  yest <- 0.5 + 3 * x + rnorm(100, 0, 3) # variable dependiente (modelo estocástico)

par(mfrow= c(2,1), mar= c(2, 2, 2, 2), oma= c(3, 3, 1, 1) )

    plot(x, ydet, pch=16)
    abline(a= 0.5, b= 3, lty=2)

    plot(x, yest, pch=16)
    abline(a= 0.5, b= 3, lty=2)

    mtext(side= 2, "Y", outer= T)
    mtext(side= 1, "X", outer= T)
```



El ejercicio consiste en generar un loop para evaluar el efecto de sd en la dispersión de los datos alrededor del valor medio de y. El objetivo es generar una figura múltiple como la que se muestra a continuación



Estructura

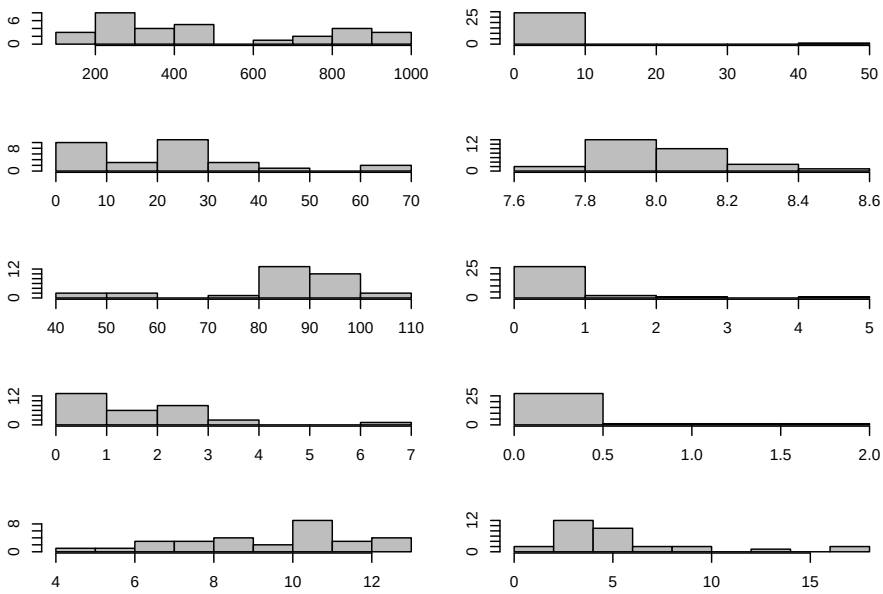
```
var = c(1, 10, 30, 60, 90, 120)

par(mfrow= c(3,2), mar= c(2, 2, 2, 2), oma= c(3, 3, 1, 1) )

for(i in COMPLETAR) {

  y <- 0.5 + 3 * x + rnorm(100, 0, COMPLETAR)
  plot(x, y, pch= 16)
  abline(a= 0.5, b= 3, lty=2)
    par( usr = c(0, 10, 0,10) )
    text( paste(COMPLETAR), x= 2, y= 8)
}
mtext(side= 2, "Y", outer= T)
mtext(side= 1, "X", outer= T)
```

0.96 Figuras múltiples (RRRR)

Asignar a un objeto que llamaremos ***env*** la tabla contenida en el archivos DoubsEnv.csv, que se encuentra en la carpeta 4-Datos (la descripción de la base de datos se encuentra en el archivo datos_eer.html).

Hemos visto que mediante el siguiente código podemos generar una figura múltiple de las variables ambientales de env

```
par(mfrow= c(5, 2), mar= c(3, 2, 2, 2))
apply(env[ , 5:14], 2, hist, main= "", col= "grey")
```



Sin embargo nos gustaría poder personalizar más la figura. Por ejemplo, agregando el nombre de las variables (en castellano y sin abreviar) y opcionalmente una letra que identifique cada gráfico dentro de la figura general. Recurriremos al uso de un loop para generarlo

Lineamientos
* Hacer un subset del objeto env con las columnas 5:14
* Generar un vector con los nombres de las variables (en castellano), como se muestra más abajo
* Dar las opciones del par() para la figura múltiple
* Definir las condiciones iniciales del loop (for i in ...)
* Seleccionar la columna de la variable de la iteración correspondiente
* Realizar el histograma
* Agregar el texto
* Fin del loop
* Leyenda "Frecuencia" en el eje y, común a todos los gráficos

```
variables <- c("Altura", "Pendiente", "Descarga", "pH", "Dureza", "Fosfato", "Nitrato"

subEnv <-  ### subset de env con todas las filas y las columnas 5:14
par( mfrow= c(5, 2), mar= c(4, 2, 1, 1), oma= c(2, 3, 1, 1) )
   for(i in ....){
     tg <- # un objeto con la columna i de subEnv
     hist(tg, .......) # entre los argumentos debería estar el nombre de la variable i
     par( usr(.....) )
     text( ...... ) # entre los argumentos debería estar la letra (a, b c, d, etc.) co
```

```
    }
mtext(.....) # leyenda del eje Y "Frecuencia"
```

0.97 Multiples archivos gráficos (RRRR)

Re diseñar el loop del ejercicio anterior, pero en vez de generar una figura múltiple se deberá guardar el histograma de cada una de las variables como archivo de imagen (jpg). El nombre de la variable deberá estar compuesto por el siguiente código: hist_variable$i$.jpg. Donde variable$i$ es el nombre de la variable correspondiente a la iteración i

0.98 Helping Bart (RRR)

La siguiente figura muestra la típica penitencia aplicada a Bart Simpson