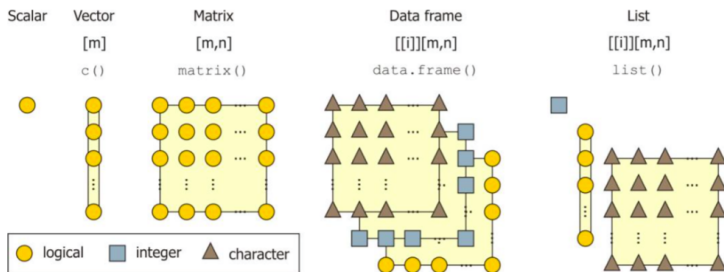


Manejo de datos en R (I)

Introducción a la Línea de Comandos para Análisis
Bioinformáticos

02 de Marzo, 2020

Breve repaso



A practical guide to the R package Luminescence (Dietze et al., 2013)

Breve repaso

Control Structures in R

If else condition

for loops

while loops

repeat statement

break statement

next statement

functions in R

Breve repaso



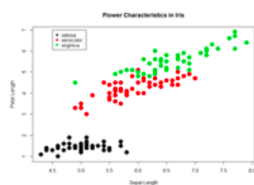
Breve repaso



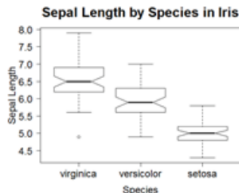
1. Basic Histogram



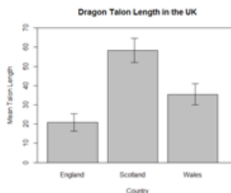
2. Line Graph with Regression



3. Scatterplot with Legend

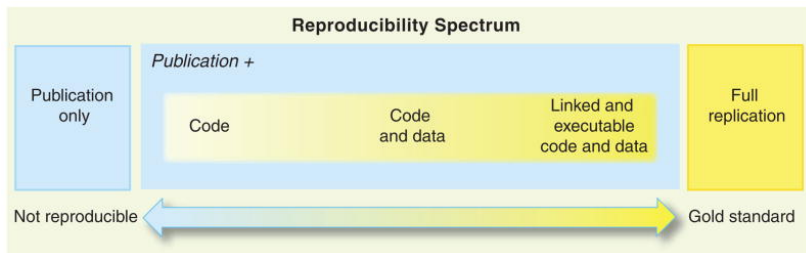


4. Boxplot with reordered/
formatted axes



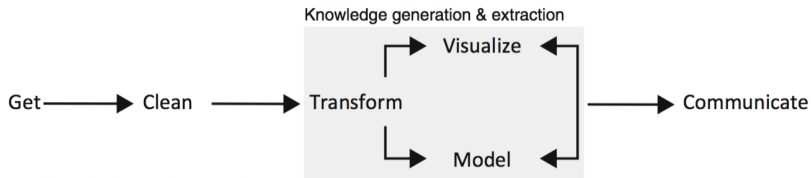
5. Boxplot with Error Bars

Análisis reproducible



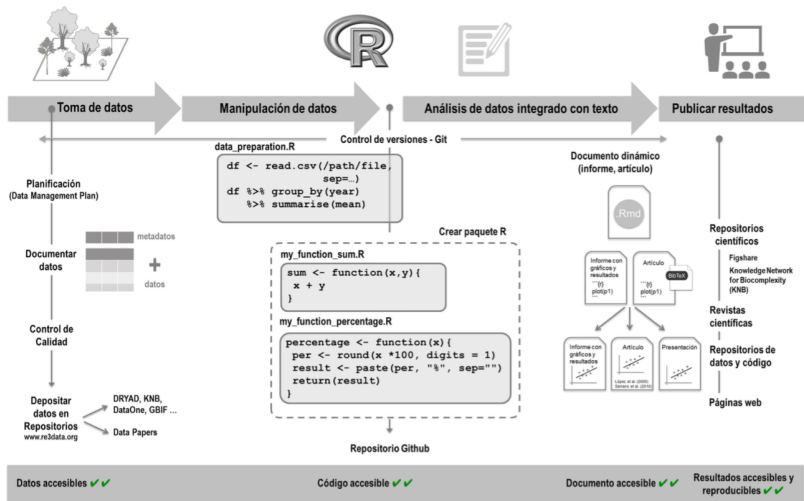
Reproducible Research in Computational Science (Peng, 2012)

Análisis reproducible



Data Wrangling with R (Boehmke, 2016)

Análisis reproducible en R



Ciencia reproducible: qué, por qué, cómo (Rodríguez-Sánchez et al., 2016)

Estructura de las clases

- Teórico/práctico.
- Práctico 11: repaso de loops y armado de funciones en R
- Práctico 12: manejo de datos con paquetes de la librería **tidyverse**.

Manejo de datos

- ***Data wrangling***: es el proceso mediante el cual modificamos datos iniciales con el fin de analizarlos.

Manejo de datos

- ***Data wrangling***: es el proceso mediante el cual modificamos datos iniciales con el fin de analizarlos.
- Incluye la edición, el filtrado, la obtención de nuevos y valores y más.

Manejo de datos

- ***Data wrangling***: es el proceso mediante el cual modificamos datos iniciales con el fin de analizarlos.
- Incluye la edición, el filtrado, la obtención de nuevos y valores y más.
- *"In our experience, the tasks of exploratory data mining and data cleaning constitute 80% of the effort that determines 80% of the value of the ultimate data mining results. (...)". Dasu & Johnson. Exploratory Data Mining and Data Cleaning (2003).*

Manejo de datos

Esto generalmente incluye

- Práctico 11
 - accionar sobre los datos para transformarlos: funciones
 - realizar acciones repetitivas: loops
- Práctico 12
 - filtrado y edición de datos
 - visualización de los datos

Funciones en R

Qué es una función?

- Un conjunto de **operaciones** definidas que toman **argumentos** para dar un resultado

Funciones: una parte central de R

- Es un lenguaje de programación en base a funciones: casi todo lo que hacemos las utiliza.
 - Otros lenguajes operan de forma diferente.
- R tiene funciones que vienen incorporadas por defecto
- Utilizando librerías obtenemos nuevas funciones (como las de **seqinr**, por ejemplo)
- Nosotros podemos hacer nuestras propias funciones

Componentes de una función

- **cuerpo:** el código dentro de la función
- **formales:** la lista de argumentos que controlan cómo se llama a la función
- **ambiente:** el “mapa” de la locación de las variables de la función

Componentes de una función

```
library(seqinr)
```

```
body(seqinr::GC)
```

```
## {
```

```
##   if (length(seq) == 1 && is.na(seq))
```

```
##       return(NA)
```

```
##   if (nchar(seq[1]) > 1)
```

```
##       stop("sequence is not a vector of chars")
```

```
...}
```

Componentes de una función

```
library(seqinr)

formals(seqinr::GC)
```

```
## $seq
##
##
## $forceToLower
## [1] TRUE
##
## $exact
## [1] FALSE
##
## $NA.GC
...

```

Componentes de una función

```
library(seqinr)
```

```
environment(seqinr::GC)
```

```
## <environment: namespace:seqinr>
```

Funciones particulares

- Podemos, además, distinguir tipos especiales de funciones:
 - **Funciones primitivas:** llaman directamente a C
 - No tienen cuerpo ni formales.
 - **Funciones de alto rango:** operan sobre funciones
 - Tienen cuerpo y formales, pero constituyen un caso interesante en sí

Funciones primitivas

```
sum
```

```
## function (... , na.rm = FALSE) .Primitive("sum")
```

```
body(sum)
```

```
## NULL
```

```
formals(sum)
```

```
## NULL
```

```
environment(sum)
```

```
## NULL
```

Funciones primitivas

```
`[`
```

```
## .Primitive("[")
```

```
`for`
```

```
## .Primitive("for")
```

Definición de funciones

```
mi_funcion = function(argumento_1, argumento_2, ...){  
  # en este bloque suceden operaciones con argumento_1  
  ...  
  # en este bloque suceden operaciones con argumento_2  
  ...  
  # se devuelve algo como resultado de aplicar  
  # la funcion a los argumentos  
  return(una_variable_nueva)  
}
```


Definición de funciones

```
eleva_y_resta = function(x,y){  
  resultado = x^2 - y^2  
  return(resultado)  
}
```

Definición de funciones

```
eleva_y_resta = function(x,y){  
  resultado = x^2 - y^2  
  return(resultado)  
}
```

```
eleva_y_resta(2,3)
```

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

Definición de funciones

```
eleva_y_resta = function(x,y){  
  resultado = x^2 + y^2  
  return(resultado)  
}
```

```
eleva_y_resta(y = 2, x =3)
```

```
## [1] 13
```

Definición de funciones

```
eleva_y_resta = function(x,y){  
  resultado = x^2 - y^2  
  return(resultado)  
}
```

```
eleva_y_resta(y = 2)
```

```
## Error in eleva_y_resta(y = 2): argument "x" is missing,
```

Definición de funciones

```
eleva_y_resta = function(x = 1, y = 1){  
  resultado = x^2 - y^2  
  return(resultado)  
}
```

```
eleva_y_resta(y = 2)
```

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

Definición de funciones

```
# ojo con el alcance de las variables!
x = 2
eleva_yresta = function(x, y){
  resultado = x^2 - y^2
  return(resultado)
}

eleva_yresta(y = 2)

## [1] 0
```

Programación defensiva

- la función **stop()** permite salir de la función si algo esperable no sucede

```
eleva_y_resta = function(x = 1, y =1){  
  
  if(!is.numeric(x) | !is.numeric(y)){  
    stop('Alguno de los argumentos dados no es un numero.')  
  }  
  
  resultado = x^2 - y^2  
  return(resultado)  
}  
  
eleva_y_resta(x = 2, y = "2")
```

```
## Error in eleva_y_resta(x = 2, y = "2"): Alguno de los argumen
```

Salida de funciones

```
# ojo con el alcance de las variables!
x = 2
eleva_resta_y_suma = function(x, y){
  resta = x^2 - y^2
  suma = x^2 + y^2
  return(c(resta, suma))
}

eleva_resta_y_suma(x = 2, y = 2)

## [1] 0 8
```


Salida de funciones

```
# ojo con el alcance de las variables!  
x = 2  
eleva_resta_y_suma = function(x, y){  
  resta = x^2 - y^2  
  suma = x^2 + y^2  
  return(c(list(resta, suma)))  
}  
  
eleva_resta_y_suma(x = 2, y = 2)
```

```
## [[1]]  
## [1] 0  
##  
## [[2]]  
## [1] 8
```

Funciones de alto rango (*high-order functions*)

- Son funciones que toman a otras funciones como argumentos y devuelven una función o valor.
- Funciones como `apply()`, `sapply()`, `lapply()`, `mapply()`...

sapply

```
# definimos un vector
numeros = c(1,2,3,4)

# aplicamos una funcion anonima sobre este vector
numeros_cuadrado = sapply(X = numeros, FUN = function(x){x^2})

numeros_cuadrado

## [1] 1 4 9 16
```

lapply

```
# definimos un vector
numeros = c(1,2,3,4)

# aplicamos una funcion anonima sobre este vector
numeros_cuadrado = lapply(X = numeros, FUN = function(x){x^2})

numeros_cuadrado

## [[1]]
## [1] 1
##
## [[2]]
## [1] 4
...

```

mapply

```
# creando una matriz de 4x4 con mapply  
matriz = mapply(rep, 1:4, 4)
```

```
matriz
```

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

Loops en R

For loop (abstracto)

```
# for loop

un_vector = ...
for (i in ____ ) {
  ...
  ... un_vector[i] ....
  ...
}
```

For loop (ejemplo)

```
numeros = c(3,40,15,6)
numeros_cuadrado = c()

for (i in 1:length(numeros)) {
  numeros_cuadrado[i] = numeros[i]^2
}
```


For loop (usando seq_along())

```
numeros = c(3,40,15,6)
numeros_cuadrado = c()

for (i in seq_along(numeros)) {
  numeros_cuadrado[i] = numeros[i]^2
}
```

Operador next

```
numeros = c(3,40,15,6)
numeros_cuadrado = c()

for (i in seq_along(numeros)) {

  if(!numeros[i] %% 3 == 0){
    next
  }
  else {
    numeros_cuadrado[i] = numeros[i]^2
  }
}
numeros_cuadrado
```

```
## [1]    9  NA 225   36
```

Operador break

```
numeros = c(3,40,15,6)
numeros_cuadrado = c()

for (i in seq_along(numeros)) {

  if(!numeros[i] %% 3 == 0){
    break
  }

  numeros_cuadrado[i] = numeros[i]^2
}
numeros_cuadrado
```

```
## [1] 9
```

While loop

- Sigue la misma lógica que en Bash
- Se utiliza la función `while()` en vez de `for()`
- Se puede realizar combinando un `for` loop con un operador `break`