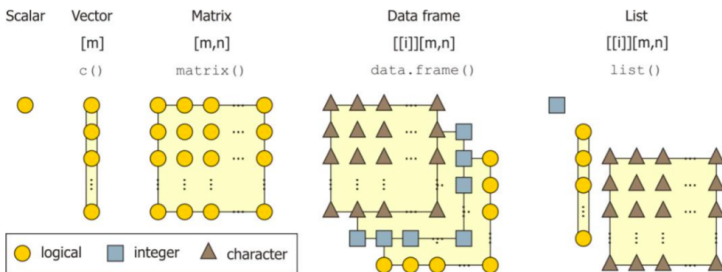


Manejo de datos en R (I)

Introducción a la Línea de Comandos para Análisis
Bioinformáticos

02 de Marzo, 2020

Breve repaso

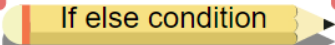


A practical guide to the R package Luminescence (Dietze et al., 2013)

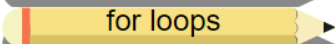
Breve repaso



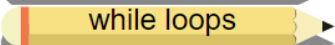
Control Structures in R



If else condition



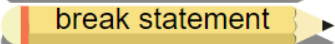
for loops



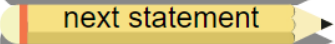
while loops



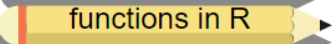
repeat statement



break statement



next statement



functions in R

Breve repaso



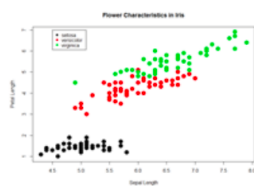
Breve repaso



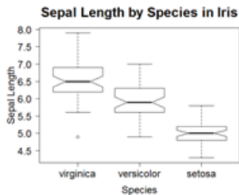
1. Basic Histogram



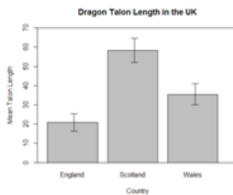
2. Line Graph with Regression



3. Scatterplot with Legend

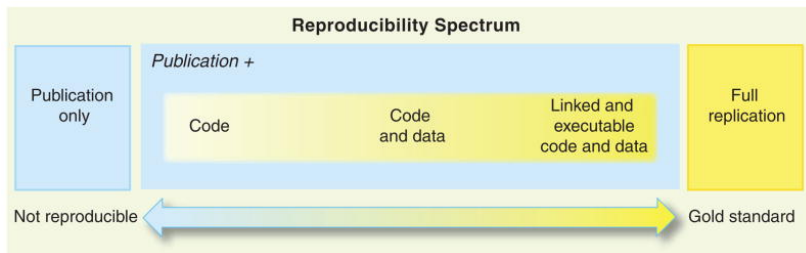


4. Boxplot with reordered/
formatted axes



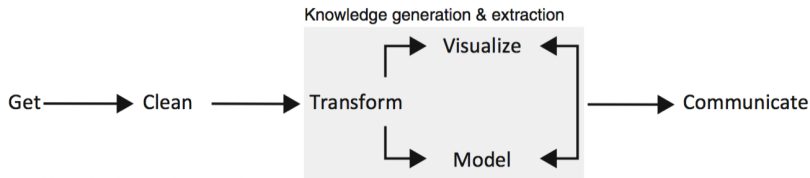
5. Boxplot with Error Bars

Análisis reproducible



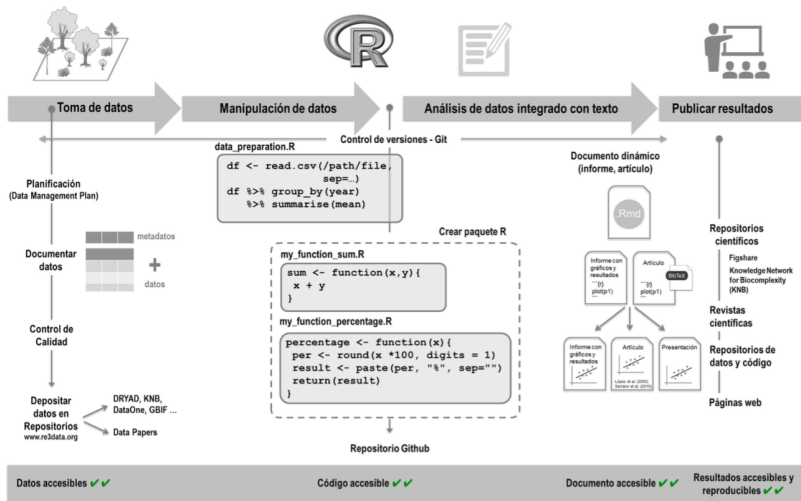
Reproducible Research in Computational Science (Peng, 2012)

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

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

Pero... qué es una función?

- una definición de función

componentes de la función clásica

- componentes de una función en R
 - **cuerpo:**
 - **formales:**
 - **ambiente:**

componentes de la funcion clasica

```
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")  
##   if (forceToLower)  
##     seq <- tolower(seq)  
##   nc <- sum(seq == "c")  
##   ng <- sum(seq == "g")  
##   na <- sum(seq == "a")  
##   nt <- sum(seq == "t")  
##   if (oldGC) {  
##     if (length(seq) == 1 && is.na(seq))  
##       return(NA)
```

componentes de la funcion clasica

```
library(seqinr)

formals(seqinr::GC)
```

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

componentes de la función clásica

```
library(seqinr)
```

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

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

R tiene varios tipos de funciones

- 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")
```

definiendo funciones en R

```
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 función a los argumentos  
  return(una_variable_nueva)  
}
```


definiendo funciones en R

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

definiendo funciones en R

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

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

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

definiendo funciones en R

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

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

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

definiendo funciones en R

```
eleva_y_suma = 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,
```

definiendo funciones en R

```
eleva_y_suma = function(x = 1, y = 1){  
  resultado = x^2 + y^2  
  return(resultado)  
}
```

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

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

definiendo funciones en R

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

eleva_y_resta(y = 2)

## [1] 0
```

Un ejemplo un poco más refinado

se arma una función arbitraria que evalúa si un número es

```
evalua_par = function(numero){  
  # si la variable numero no es de clase numeric devuelvo  
  if(!is.numeric(numero)){  
    return('Ingrese un numero')  
  }  
  
  else {  
    # evaluo si numero es par, considerando el resto de di  
    resto = numero %% 2  
  
    if(resto == 0){  
      return(TRUE)  
    }  
  }  
}
```

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

- Higher-order functions are functions that take other functions as arguments and return either another function, or a value.

funciones que trabajan como Map(), como apply(), lapply(), mapply(), sapply(), vapply() y tapply().

en esta seccion esta bueno mostrar eso de usar
una funcion explicita o de definirla al boleo

Familia de funcion `apply()`

The `apply()` family pertains to the R base package and is populated with functions to manipulate slices of data from matrices, arrays, lists and dataframes in a repetitive way. These functions allow crossing the data in a number of ways and avoid explicit use of loop constructs. They act on an input list, matrix or array and apply a named function with one or several optional arguments.

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){x2})
```

```
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
```

```
numeros_cuadrado
```

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

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

```
##
```

```
## [[2]]
```

```
## [1] 4
```

```
##
```

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

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

```
##
```

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

loops

```
# for loop

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

loops

```
# for loop

numeros = c(1,40,2,6)
numeros_cuadrado = c()

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


loops

```
# for loop

numeros = c(1,40,2,6)
numeros_cuadrado = c()

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