**R. Curso Roger Peng**

**Tipos de datos, classes**

* Carácter
* Numeric
* Integer
* Complex
* Logical

**Tipo de objetos**:

* Números(operador L para enteros, Inf para infinito, NaN para not a number), vectores o matrices (contienen datos de la misma clase),
* listas, puede contener objetos o datos de distinta clase
* frames, contiene listas (es el que se utiliza normalmente como base de datos))

**Atributos de los objetos**

* Nombre, dim, names
* Dimensiones
* Class
* Lenght
* Otros como metadata

**Expresiones y Operadores**

* **<-**  x <- 1 operador de asignación
* **#** para comentarios
* **X <- 1:20** # los dos puntos crean una secuencia
* **c() # para crear vectores separados de comas c(“a”, “b”, “d”)**
* **x <- vector(“numeric”, lenght=5) # vector** para crear vectores nulo especificando clase y longitud
* **as.numeric (x)**
* **as.character(x)**
* **as.logical(x)**
* **class(x)**
* **dim**(número de filas, número de columnas)
* x<- **matrix**(1:6, nrow=2, ncol=3), crea rellenando la primera columna y luego se pasa a la siguiente así hasta que termina
* Pegar filas o columnas para crear matrices: cbind(x,y), rbid(x,y)
* X<- **list**(1, “a”, true)
* Factores, x<-**factor**(c(“yes”, “yes”, “no”)), levels=c(“yes”, “no”), table(x), los factores se crean por orden alfabético
* Valores perdidos NA o NaN, **is.na()** **is.nan()**. Na: no available; NaN: not a number
* Frames: como listas x<- **data.frame**(nombre1= c(1:3), nombre2= c(1,t,u))
* **nrow**(x), devuelve el número de filas del frame, **ncol**(x) develve el número de columnas
* Puede llamarse a una única variable de un frame por su nombre x$nombre o x[[i]], siendo i el orden.
* Puedo dar nombre a las columnas de un vector con **names(x)**<- c(“nombre1”, “nombre2”, “nombre3”),
* para dar nombre a las filas y las columnas **dimnames**(x)=list(c(1:n), c(1:k))
* **str(x)** para saber la estructura del objeto x

**Selección de un subconjunto de elementos**

* [ always returns an object of the same class as the original; can be used to select more than one element (there is one exception)
  + **Vectores**:
    - x[3], devuelve el tercer elemento
    - x[x >1], devuelve los elementos mayores que 1
  + **Matrices**
    - x[1, 2], devuelve los elementos de una matriz [i, j] i=fila, j=columna
    - x[, 2] devuelve los elementos de la segunda columna
    - x[3, ] devuelve los elementos de la tercera fila
  + **Listas**: ejemplo x <- list(foo = 1:4, bar = 0.6, baz=”helo”)
    - x[1], x$foo, x[“foo”], selecciona los primeros elementos de la lista
    - x[c(1,3)], para seleccionar el elemento 1 y 3 de una lista (ojo sólo un corchete, porque si ponemos dos corchetes x[[c(1,3)]] seleccionaría el *tercer* elemento de la *primera* variable, en nuestro caso un 3.
* [[ is used to extract elements of **a list or a data frame**; it can only be used to extract a single element and the class of the returned object will not necessarily be a list or data frame
  + **Listas**: ejemplo x <- list(foo = 1:4, bar = 0.6, baz=”helo”)
    - x[[1]], en este caso como no he especificado mostrará toda la primera variable es como x[1]
    - x[[c(1,3)]], seleccionaría el *tercer* elemento de la *primera* variable
* $ is used to extract elements of a list or data frame by name; semantics are similar to hat of [[.
* **Subset** para seleccionar un subconjunto de datos:

newdata <- subset(mydata, sex=="m" & age > 25, select=weight:income)

**ELIMINAR DATOS PERDIDOS**

* x <- c(1, 2, NA, 4, NA, 5)
* bad <- is.na(x) #esto crea un índice de malos que luego se puede utilizar
* y<- x[!bad], en este caso y sólo tiene cuatro valores : 1 2 4 5

Parecido se podría hacer directamente con **complete.cases(x)**, sólo selecciona los no NA.

Tiene la ventaja de que puede utilizarse para dos o más variables:

* x <- c(1, 2, NA, 4, NA, 5)
* y <- c("a", "b", NA, "d", NA, "f")
* good <- complete.cases(x, y)
* good

[1] TRUE TRUE FALSE TRUE FALSE TRUE

* x[good]

[1] 1 2 4 5

* y[good]

[1] "a" "b" "d" "f"

Para un frame: ejemplo

* good <- complete.cases(airquality)
* airqualitysinna <- airquality[good, ]

Para una serie temporal

ygood <- y[complete.cases(y)]

**Operaciones con matrices y vectores**

* x %\*% y para **multiplicar matrices** (ojo si se utiliza \* o / a secas se multiplica elemento a elemento)

**Lectura de datos**

* data <- read.table("foo.txt")

argumentos:

* + file, the name of a file, or a connection
  + header, logical indicating if the file has a header line
  + sep, a string indicating how the columns are separated
  + colClasses, a character vector indicating the class of each column in the dataset
  + nrows, the number of rows in the dataset
  + comment.char, a character string indicating the comment character
  + skip, the number of lines to skip from the beginning
  + stringsAsFactors, should character variables be coded as factors?
* data <- read.csv("foo.txt") (para texto separado por comas)

write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",

eol = "\n", na = "NA", dec = ".", row.names = TRUE,

col.names = TRUE, qmethod = c("escape", "double"),

fileEncoding = "")

write.csv(...)

write.csv2(...)

**ESTRUCTURAS DE PROGRAMA**

* if, else: testing a condition
* for: execute a loop a \_xed number of times
* while: execute a loop while a condition is true
* repeat: execute an in\_nite loop
* break: break the execution of a loop
* next: skip an interation of a loop
* return: exit a function

If (<condición>) {

## se hace algo

} else {

## en caso contrario hacer esto otro

}

If (<condición1>) {

## se hace algo

} else if (<condición 2>){

## en caso de no condición 1 pero sí condición 2 contrario hacer esto otro

} else {

## en caso de que no se cumplas ni condición 1 ni condición2 hacer esto

}

for(i in 1:10) {

## hace algo

}

count <- 0

while(count < 10) {

## hace algo

count <- count + 1

}

repeat {

x1 <- computeEstimate()

if(abs(x1 - x0) < tol) {

break

} else {

x0 <- x1

}

}

for(i in 1:100) {

if(i <= 20) {

## Skip the first 20 iterations

next

}

## Do something here

}

**Funciones**

f <- **function**(<arguments>) {

## Do something interesting

}

**Simulación**

* rnorm: generate random Normal variates with a given mean and standard

deviation

rnorm(n, mean = 0, sd = 1)

set.seed(1)

* dnorm: evaluate the Normal probability density (with a given mean/SD) at a point (or vector of points)

dnorm(x, mean = 0, sd = 1, log = FALSE)

* pnorm: evaluate the cumulative distribution function for a Normal distribution

pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE) ## Pr(x <= q)

* qnorm: q for quantile function

qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)

* d for density
* r for random number generation
* p for cumulative distribution
* q for quantile function

**ejemplos:**

* + rpois: generate random Poisson variates with a given rate rpois(10,2)
  + x <- **rbinom**(100, 1, 0.5)

**GRAFICOS**

* plot: make a scatterplot, or other type of plot depending on the class of the object being plotted
* hist (x)
* ?par, para un listado de posibles parámetros
* lines: add lines to a plot, given a vector x values and a corresponding vector of y values (or a 2-column matrix); this function just connects the dots
* points: add points to a plot
* text: add text labels to a plot using specified x, y coordinates
* title: add annotations to x, y axis labels, title, subtitle, outer margin
* mtext: add arbitrary text to the margins (inner or outer) of the plot
* axis: adding axis ticks/labels

**Procedimiento de gráficos:**

* Abrir un device (eje9mplo xindows(), x11(), o a un archivo pdf(nombrearchivo), png(nombrearchivo
* Dibujar lo que fuere
* Cerrar el device

*?Devices*

dev.copy: copy a plot from one device to another

dev.copy2pdf: copy a plot to a Portable Document Format (PDF) file

dev.list: show the list of open graphics devices

dev.next: switch control to the next graphics device on the device list

dev.set: set control to a specific graphics device

dev.off: close the current graphics device

**Crear serie temporal**

y<-as.ts(datos)

y<-as.zoo(datos)

**Regresión lineal**

El comando básico es lm (linear models).

El primer argumento de este comando es una fórmula y ~ x en la que se especifica cuál es la variable respuesta o dependiente (*y*) y cuál es la variable regresora o independiente (*x*).

El segundo argumento, llamado data especifica cuál es el fichero en el que se encuentran las variables.

El resultado lo guardamos en un objeto llamado regresion. Este objeto es una lista que contiene toda la información relevante sobre el análisis.

Mediante el comando summary obtenemos un resumen de los principales resultados

regresion <- lm(y ~ x, data = nombreframe)

summary(regresion)

coefficients(reg)

confint(regresion)

anova(regresion)

residuos <- rstandard(regresion)

valores.ajustados <- fitted(regresion)

predict(regresion, nuevos.x)

nuevas.edades <- data.frame(edad = seq(20, 60))

# Grafico de dispersion y recta

plot(grasas$edad, grasas$grasas, xlab = "Edad", ylab = "Grasas")

abline(regresion)

# Intervalos de confianza de la respuesta media: ic es una matriz con tres

# columnas: la primera es la prediccion, las otras dos son los extremos

# del intervalo

ic <- predict(regresion, nuevas.edades, interval = "confidence")

lines(nuevas.edades$edad, ic[, 2], lty = 2)

lines(nuevas.edades$edad, ic[, 3], lty = 2)

# Intervalos de prediccion

ic <- predict(regresion, nuevas.edades, interval = "prediction")

lines(nuevas.edades$edad, ic[, 2], lty = 2, col = "red")

lines(nuevas.edades$edad, ic[, 3], lty = 2, col = "red")

**Paquetes interesantes:**

**Package ‘tseries’: BDS, adf, ARMA, GARCH**

**Package ‘fNonlinear’: BDS, información mutua, falsos vecinos**

**Package ‘fgarch’: Modelos Garch**

**garchFit: Fits the parameters of a GARCH process**

**residuals: Extracts residuals from a ﬁtted ’fGARCH’ object,**

**predict: Forecasts from an object of class ’fGARCH’**