Introducción a 

Tomàs Aluja
tomas.aluja@upc.es

Instalación

http://cran.es.r-project.org

- En el enlace podréis descargar R para Windows, Linux y Mac

base — Binaries for base distribution (managed by Duncan Murdoch). This is what you want to **install R for the first time**.

Una introducción a R más completa: http://cran.r-project.org/doc/contrib/R-intro-1.1.0-espanol.1.pdf

Tutorial de R: http://www.tutorialr.es/es/index.html

Primera sesión:

Entramos a R

```
# Esto es un comentario
# Generación de 40 observaciones con N(0,1)

x <- rnorm(40)
x <- sort(x) # Ordenamos x para los plots de después

x # Así vemos el contenido de x

# Estadísticos sumarios e histograma de x

summary(x)
hist(x)

# Generación de una variable de respuesta

y <- sin(x*pi)+0.1*rnorm(40)

# Estadísticos sumarios e histograma de x
```

```
summary(y) ; hist(y)

plot(x,y)

# Creación de un data.frame compuesto por los dos
# vectores

dd <- data.frame(x,y)

# Para ver los atributos de un data.frame

attributes(dd)

# Comprobamos que el data.frame no son factores
#(=variables categóricas)

sapply(dd, is.factor)

# Verificamos máximos y mínimos del data.frame

sapply(dd, range)

# Hacemos una regresión lineal entre y y x

reg1 <- lm(y ~ x, data=dd)

# Miramos que contiene el objeto reg1

attributes(reg1)

# Miramos, por ejemplo, los coeficientes

reg1$coefficients

# Vemos que ajuste hemos hecho

plot(x,y)

# y sin cerrar la ventana de plot...

lines(x, reg1$fitted.values)

# Ahora hacemos una regresión local

reg2 <- loess(y ~ x, data=dd)

# y miramos que hemos hecho

lines(x,reg2$fitted,col="blue")
```

```
# ¿Qué modelo se ajusta mejor a los datos?
```

En **R** casi todo son objetos:

vectors. Son las variables

```
> x <- c(1, 2, 3);  y <- c("a", "b", "c")
> z <- c(TRUE, TRUE, FALSE)
```

factors. Son les variables categóricas

```
> x <- factor(c(1, 2, 3))
```

matrices

```
> y <- matrix(x, nrow=1, ncol=3);  dim(y)
```

arrays. Son matrices de cualquier número de dimensiones

data.frame. Son tablas donde las variables son de cualquier tipo. Es el resultado natural de la lectura de un fichero de datos.

listas. Listas de otros objetos. Es muy general, puedes enlistar cualquier tipo de objeto

```
> ll <- list(x,y,z)
```

Para saber qué objetos tenemos

```
> ls()
```

Para saber si un objeto es un vector numérico, por ejemplo, hacemos

```
> is.numeric(x)
```

Cada objeto tiene una lista de atributos que lo componen. Para saber los atributos de un objeto hacemos:

```
> attributes(objecto)
```

R es "case sensitive", escribir en mayúsculas o minúsculas es importante.

Importar los datos

Lo mejor es leer un fichero de texto con los datos separados por espacios o tabuladores. Es recomendable poner en la primera fila los nombres de las variables.

```
> dd <- read.table("h:/dir/nom.txt", header=TRUE)
```

Atención a la inclinación de las barras del «path».

El objeto leído, dd, es un data.frame.

Header = TRUE indica que la primera fila contiene el nombre de las variables.

Si tenemos una primera columna indicando el nombre de las filas, esta columna no ha de llevar identificador de columna.

Los literales no deben contener ningún espacio en blanco.

También se pueden leer ficheros separados por comas o puntos y comas (CSV), así como importar ficheros de otros sistemas (SPSS, ...).

Hacer

```
> help(read.table)
```

Para ver todos los argumentos de la función y otras funciones relacionadas.

Manipulación de los datos

Primero vamos a mirar qué hemos leído

```
> dd
> summary(dd) # Nos da los mínimos y máximos para ver
si el fichero se ha leído correctamente.
> # Si R interpreta que el dataframe contiene
variables categóricas, lista una tabla de frecuencias
por modalidad
```

Para saber que ha interpretado R sobre el tipo de las variables de un dataframe, o para asegurarnos de que los tipus definidos son correctos (variables numéricas o categóricas), hacemos

```
> sapply(dd,class)
```

Operación con vectores

```
> nouv <- a+2*b+3*c+4*d
```

Estan disponibles los operadors aritmètics (+, -, *, /, ^, ...) y lógicos (==, !=, >, >=, <, <=, &, |) habituales.
También las funciones matemáticas usuales: sqrt, log, log10, exp, sin, ...

Generación de secuencias

```
> a <- 1:20;  b <- -10:10; c <- c(1:20, seq(from=22, +
to=40, by=2)
> a <- rep(4,3); b <- gl(4,3, label=c("b1","b2", +
"b3", "b4")
```

Seleción de elementos de un vector

```
> x <- 1:100
> y <- x[1:10]
> z <- x[c(2,21:30)]   # seleccionamos elementos
disjuntos
> x[100]
> z <- list(x,y)  # Una lista de vectores
> z[[2]]; y  # el segundo elemento de la lista es un
vector
>
> y <- x[x>3]  # selecciona los elementos de x que
cumplen la condición.
> #No confundir con:
> z <- x>3   # genera un vector lógico indicando qué
elementos cumplen la condición
```

Los valores faltantes se especifican con NA (not available). El resultado de operaciones aritméticas no pemitidas da NaN (not a number) o Infinity.

Operaciones con valores perdidos

```
> x <- c(1,2,3,NA)
> in.na(x); which(is.na(x))
> x2 <- x[!is.na(x)]  # quitamos los NA
> x[is.na(x)] <- 0    # los substituimos por 0
>
> x3 <- (5, 3, NA, 7)
> mean(x3)                        # no funciona
```

```
> mean(x3, na.rm=T)              # ahora sí
>
> X <- matrix(c(1,2,3,NA,5,6,NA,7), nrow=4)
> X2 <- na.omit(X); cor(X); cor(X2)
```

Estadísticos sobre los vectores

```
> mean(x); min(x);  max(x); var(x); sd(x); ...
```

Recodificación de los vectores

```
> z <- cut(x, 5)  #  recodificacióm en 5 intervalos de
igual medida
>
> z <- cut(x, breaks=c(0,100, 130, 200, 999) #
recodificación de x en 4 intervalos definidos por los
cortes anteriores.
>
> z <- cut(x, quantile(x, c(0, 1/3, 2/3, 1)))
> # recodificación en 3 intervalos equidistribuidos.
>
> z <- split(x, a)
> # partición del vector x en función del factor a
```

Ordenación de vectores

```
> sort(x)
> rank(x)
> order(x)

> sort(x1, partial=x2) # ordenación de x1 según un
vector de índices x2
```

Ordenación de dataframes por una columna

```
> dd_sorted <- dd[order(dd$x),]
```

Matrices y vectores

Definición

```
> A <- 1:24; dim(A) <- c(3,4,2); A
> B <- matrix(1:24, nrow=6); B
> C <- cbind(x, y)
>
> D <- B[2:4, 3:4]
```

Operaciones con matrices

```
> A + B
> A – B
> A * B      # Atención: es el producto término a
               término
> A %*% B    # Es el producto matricial
> A %o% B    # Producto exterior (todos contra todos)
> t(A)       # Transpuesta de la matriz
> solve(A)   # Inversa de la matriz
> eigen(A)   # Valores y vectores propios
> svd(A)     # Descomposiciób en valores singulares
> diag(A)    # Vector con la diagonal de A
> diag(x)    # Matriz diagonal, con diagonal = x
```

Operaciones con vectores

```
> z <- x * x    # Es un vector formado por el producto
término a término de x
> k <- x %*% x   # = t(x) %*% x. Es el producto
escalar de x con sí mismo (norma al cuadrado).
> fq <- t(x) %*% A %*% x.  Es la forma cuadrática de A
asociada a x.
```

Operaciones por filas o por columnas.

```
> row.mean <- apply(dd, 1, mean)  # Media por filas
> col.sum <- apply(dd, 2, sum)    # Suma por columnas

> str.mean <- tapply(x, fac, mean)  # Media del vector
x según las modalidades del factor fac

> dvector <- c(dd) # Apila la matriz dd como un vector
```

Creación de tablas

```
> taula <- table(fac1, fac2)  # Cruce de dos factores
```

La tabla creada es una matriz que podemos manipular y, eventualmente, exportar.

Gráficos

El gráfico básico es:

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

Podemos mejorar este gráfico:

```
> plot(x,y, main=”título chulo”, sub=”subtítulo”,
xlab=”eje x”, ylab=”eje y”)
# o como una línea:
> plot(x,y, main=”título chulo”, sub=”subtítulo”,
xlab=”eje x”, ylab=”eje y”, type=”l”) # Por defecto
dibuja un gráfico de puntos, pero con type = “l” los
une con líneas
```

También podemos hacer plots con etiquetas:

```
> num <- rep(1:length(x))   # Numeración secuencial
> etiq <- paste(“ind”,num)  # Creación de etiquetas
> plot(x,y, type=”n”)  # y sin cerrar la ventana
> text(x,y, labels=etiq)
```

Según el contexto **R** es inteligente:

```
> plot(sin, -pi, 2*pi)
```

Otros gráficos son:

```
> hist(x)
> barplot(fac)
> boxplot(edat~ fac) # boxplots para cada nivel de fac
> qqplot(edat, zscores)
> persp(x=c(1:30), y=c(1:30), z) # plot de una
superficie de respuesta z, z ha de ser una matriz de
dimensiones 30 por 30
> ...
```

Funciones

```
> nombre <- function(arg1, arg2, ... ){expresión}
```

{expresión} indica una formula o conjunto de formulas (instrucciones de **R**). La función puede retornar un valor numérico, un vector, un gráfico o un mensaje.

Por ejemplo, si queremos crear una función que nos de las probabilidades teóricas de una tabla de contingencia, bajo la hipótesis de independencia, podemos hacer:

```
> probt <- function(taula){
          rm <- apply(taula,1,sum)
          cm <- apply(taula,2,sum)
          gm <- sum(taula)
          prt <- rm %*% t(cm)/gm
     }
>
> pp <- probt(tt) # Donde tt es una tabla de
contingencia
```

Si queremos crear una función dando más de un resultado, por ejemplo, si queremos hacer una función que centre y reduzca una matriz ponderada, dar como salida la matriz estandarizada, los vectores de medias y desviaciones tipos, haremos lo siguiente:

```
# Función de centrar y reducir ponderada
# pes   vector de pesos
# Xraw  matriz de datos original

> stan = function(Xraw,pes){
> X = as.matrix(Xraw)
> n = dim(X)[1]
> pes.rel = pes/sum(pes)

> xmean = pes.rel %*% X
> xsdev = sqrt(apply(pes.rel %*% (X - rep(1,n) %*%
xmean)^2, 2, sum))
> Xstan = (X - rep(1,n) %*% xmean) / rep(1,n) %*%
t(xsdev)

> list (stan = Xstan, mean = xmean, sdev = xsdev) }

# Entonces podemos llamar a la función stan y ver los
atributos de lo que retorna
> Xstn = stan(Xdat, poids)
> attributes(Xstn)
```

Programación en R

R dispone de todas las instrucciónes para el control de una ejecución:

```
> for (i in 1:5) print (1:i)
>
> xg <- split(x, fac)   # Separación de dos vectores
> yg <- split(y, fac)   # Plot conjunto de la

> for (i in 1:length(yg)) {
         plot(xg[[i]], yg[[i]]);
         abline(lsfit(xg[[i]], yg[[i]]))
       }
>
> for (i in 1:length(x) {
       if (x[i] > 99) print(iden[i])}
>
> if (condicio) expr_1 else expr_2
> while (condicio) expr
> repeat expr
> break
> next
```

Funciones probabilísticas:

Disponemos de un set amplísimo de distribuciones de probabilidad, en particular las más corrientes: Normal, Uniforme, t de student, F de Fisher, χ^2, etc. En **R** se indican respectivamente por: norm, unif, t, f, chisq

El prefijo indica cuál es la función llamada: `rxxx, pxxx, qxxx y dxxx.`

```
> rnorm(10)  # Indica random, es decir, generación
     aleatoria de 10 obs. de una distribución N(0,1).
>
> pnorm(1.96)  # Obtiene el valor de la función de
                    distribución = P(x≤1.96)
>
> qnorm(0.5)   # Es la función inversa de la función
                    de probabilidad qnorm(pnorm(x))=x
>
> dt(0)  # Da el valor de la función de densidad de la
distribución t en el punto 0.
```

Pruebas estadísticas clásicas:

Podéis realizar las pruebas de hipótesis más clásicas de la siguiente manera:

```
> # prueba de Chi-cuadrado
> chisq.test(taula)
> chisq.test(fac1,fac2, p=probteo, simulate.p.value=T)
>
> # t de Student para una muestra
> t.test(edat, mu=45, alternative=”two.sided”)
>
> # t de Student para dos muestras (emparejadas o no)
> t.test(pres1,pres2, paired=T,alternative==”greater”)
>
> # Prueba de la igualdad de varianzas
> var.test(pres1, pres2, ratio=1, conf.level=0.90)
>
> # F de Fisher para la igualdad de k medias
> oneway.test(edat~mar2, dd, var.equal=T)
...
```

Una pizca de análisis multivariante

Vamos a hacer un análisis de componentes principales normalizado, seguido del screeplot de los valores propios y del biplot en Rp. Después hacemos una clasificación sobre las 2 primeras componentes principales. Por último cortamos el árbol en 5 clases y listamos el contenido de cada una:

```
> pcadd <- princomp(dataframe, cor=T)
> screeplot(pcadd)
> biplot(pcadd$scores, pcadd$loadings)
>
> d2d <- dist(pcad2$scores[,1:2])
> d2clu <- hclust(d2d, method = "ward.D2")
> plot(d2clu)
>
> d2cut <- cut(d2clu, k=5)
> split(1:length(d2clu), d2cut)
```

Exportar datos

```
> write.table(nom.dataframe, file="output.txt", +
col.names=TRUE)
```

Cargar más paquetes

La carga de paquetes es muy fácil a partir del item «package» de la barra principal (R GUI). Podemos cargar los paquetes disponibles localmente o acceder a los paquetes disponibles en CRAN.

Para instalar paquetes:

```
> install.packages(c("MASS", "pack2"))  # Preguntará
de qué mirror queréis decargar. Como «mirror»
cualquiera vale.
> library(MASS) # Para cargar el paquete MASS
```

Para cada paquete podemos ver la lista de funciones que contienen accediendo a HTML help, clicando sobre el paquete que interese (R GUI).

Trabajando con programas R

Una forma cómoda de trabajar con R es a partir de programas. Los programas son simplemente ficheros de texto, con extensión xxxx.r, conteniendo las instrucciones de R que queremos ejecutar debidamente comentadas.