Introdución a R: dia 4

```
#unha vez que se ten o codigo listo, e de cara a sacar un documento limpo podense inicializar uns
#parámetros, que me eviten a aparicióndе avisos, a non ser que sexan erros
knitr::opts_chunk$set(warning=FALSE)
knitr::opts_chunk$set(message=FALSE)
#que significa knitr::algo_mais?
#ese par de "dous puntos" indica que queremos usar un comando dun paquete
#que non está subido á memoria con library. Só nos interesa puntualmente,
#polo que só usamos ese comando con referencia ao nome do paquete
```

Colócase ao comezo as librerías que se van usar nesta sesión. É o que se fai habitualmente cando se quere o código ben organizado.

```
#cargar librerias
library(ggplot2)
library(dplyr)
library(tibble)
library(tidyr)
library(forcats)
library(psych)
library(corrgram)
library(pastecs)
library(psych)
library(DescTools)
```

Máis estructuras de datos

Matrices

Unha *matriz* en R pode dicirse que é *un vector con dimensión*, ou sexa *con filas e columnas*

De feito pode crearse como un vector e despois asignarlle a estrutura:

```
ola=c(2,3,3,2,1,4)#creo un vector
dim(ola)#observo que non ten asignada dimensión
```

```
## NULL
```

```
ola=matrix(ola,nrow = 3)#convirtoo nunha matriz con 3 liñas, polo tanto terá 2 columnas
dim(ola)#agora si ten dimensión
```

```
## [1] 3 2
```

```
ola#aquí está a matriz
```

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

Pódese operar con ela introducindo os valores de filas e columnas, como se facía cos data.frame:

```
# Elemento 2,2
ola[2,2]
```

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

```r
# Fila 2
ola[2,]
```

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

```r
#columna 1
ola[,1]
```

```
## [1] 2 3 3
```

```r
#neste caso non funciona se non se pon a coma
ola[1]#extrae o primeiro elemento, como se fose un vector, e non a primeira columna
```

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

```r
#o mesmo para dobres corchetes
ola[[1]]
```

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

```r
#Unha matriz máis grande para filtros máis complexos
set.seed(2021)
matriz=matrix(rnorm(50),nrow = 10)
matriz[1:4,] # filas de primeira a cuarta
```

```
##            [,1]       [,2]       [,3]      [,4]       [,5]
## [1,] -0.1224600 -1.0822049 -0.9424433 -1.963825  0.1163887
## [2,]  0.5524566 -0.2728252 -0.1856850 -1.447944  1.7602137
## [3,]  0.3486495  0.1819954 -1.1011246  1.019443 -0.3451165
## [4,]  0.3596322  1.5085418  1.2081153 -1.421417  2.1200002
```

```r
matriz[1:4,3] # 3ª columna das filas de primeira a cuarta
```

```
## [1] -0.9424433 -0.1856850 -1.1011246  1.2081153
```

```r
colMeans(matriz)# media por columnas
```

```
## [1]  0.3034808  0.4847642 -0.3343189 -0.8324701  0.4579256
```

```r
rowMeans(matriz)
```

```
##  [1] -0.79890890  0.08124316  0.02076944  0.75497447  0.04773513 -1.20685895
##  [7]  0.12383882 -0.29746057  0.32530046  1.10813013
```

```r
matriz[,colMeans(matriz)>0]
```

```
##              [,1]       [,2]        [,3]
##  [1,] -0.12245998 -1.0822049  0.11638871
##  [2,]  0.55245663 -0.2728252  1.76021373
##  [3,]  0.34864950  0.1819954 -0.34511646
##  [4,]  0.35963224  1.5085418  2.12000016
##  [5,]  0.89805369  1.6044701 -0.03437749
##  [6,] -1.92256952 -1.8414756 -0.79215405
##  [7,]  0.26174436  1.6233102  1.47551521
##  [8,]  0.91556637  0.1313890 -0.72555721
##  [9,]  0.01377194  1.4811225  0.31237904
## [10,]  1.72996316  1.5133183  0.69196411
```

```r
matriz[rowMeans(matriz)>0,colMeans(matriz)>0]
```

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

```
## [1,] 0.55245663 -0.2728252  1.76021373
## [2,] 0.34864950  0.1819954 -0.34511646
## [3,] 0.35963224  1.5085418  2.12000016
## [4,] 0.89805369  1.6044701 -0.03437749
## [5,] 0.26174436  1.6233102  1.47551521
## [6,] 0.01377194  1.4811225  0.31237904
## [7,] 1.72996316  1.5133183  0.69196411
```

Listas

Unha *lista* pode entenderse como un vector que pode conter elementos de tipos diferentes.

Por exemplo:

```
lista=list(8,3,"w","ola",data.frame(X=c(8,2,1),Y=c("a","b","c")))
lista
```

```
## [[1]]
## [1] 8
##
## [[2]]
## [1] 3
##
## [[3]]
## [1] "w"
##
## [[4]]
## [1] "ola"
##
## [[5]]
##   X Y
## 1 8 a
## 2 2 b
## 3 1 c
```

Como se extrae un elemento dunha lista?

Para extraer elementos hai que pensar nelas como unha mistura de vector e data.frame.

```
#coma un vector
lista[4]# unha lista menor co 4º elemento da lista
```

```
## [[1]]
## [1] "ola"
```

```
lista[c(1,5)] # unha lista menor cos elementos 1 e 5 da lista
```

```
## [[1]]
## [1] 8
##
## [[2]]
##   X Y
## 1 8 a
## 2 2 b
## 3 1 c
```

```
# coma un data.frame
lista[[4]] # O que hai dentro do 4º elemento da lista
```

```
## [1] "ola"
```

```
lista[[5]] # o que hai dentro do 5º elemento da lista
```

```
##   X Y
## 1 8 a
## 2 2 b
## 3 1 c
```

As listas tamén poden ter nomes e acceder aos seus elementos por nome

```
names(lista)=c("a","b","c","d","e")
lista$e
```

```
##   X Y
## 1 8 a
## 2 2 b
## 3 1 c
```

```
lista$a
```

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

En ocasións pode ser necesario simplificar a estructura de lista e obter outras estructuras máis manexables. Por exemplo, o comando "*unlist()*", transforma a lista nun vector

```
unlist(lista)
```

```
##     a     b     c     d  e.X1  e.X2  e.X3  e.Y1  e.Y2  e.Y3
##   "8"   "3"   "w" "ola"   "8"   "2"   "1"   "a"   "b"   "c"
```

```
# En alguns casos, se hai compatibilidade, pode transformarse a lista nun data.frame
lista2=list(A=c(2,1,3),B=c("a","b","f"),C=c(2,4,1))
lista2
```

```
## $A
## [1] 2 1 3
##
## $B
## [1] "a" "b" "f"
##
## $C
## [1] 2 4 1
```

```
str(lista2)
```

```
## List of 3
##  $ A: num [1:3] 2 1 3
##  $ B: chr [1:3] "a" "b" "f"
##  $ C: num [1:3] 2 4 1
```

```
df2=as.data.frame(lista2)
str(df2)
```

```
## 'data.frame':    3 obs. of  3 variables:
##  $ A: num  2 1 3
##  $ B: chr  "a" "b" "f"
##  $ C: num  2 4 1
```

Mesturar ficheiros

2 vectores

Os vectores son estruturas simples, que poden ser unidas cun comando de asignación: ***c()***

```
### vectores
# Creo dous vectores para o exemplo
v1=c(2,1,2,3)
v2=c(2,2,3,4,5,4)
#pégoos usando c()
vv=c(v1,v2)
vv#isto é o que sae
```

```
##  [1] 2 1 2 3 2 2 3 4 5 4
```

Matrices

Como se fusionan dúas matrices (ou dous data.frames)?

Para fusionalas hai que ter en conta o seu nº de filas e de columnas, e o sentido no que se fusionan.

```
#preparar exemplos
mat1=matrix(1:40,nrow = 10)#crear unha matriz 10x4
dim(mat1)
```

```
## [1] 10  4
```

```
mat2f=matrix(1:20,nrow = 10)#crear unha matriz 10x2
mat2c=matrix(1:20,ncol=4)#crear unha matriz 5x4
```

Pódense fusionar pegando a derradeira fila dunha coa primeira fila da outra, sempre e cando teñan o mesmo nº de columnas: comando ***cbind()***

Ou fusionar pegando a derradeira columna dunha coa primeira columna da outra, se teñen o mesmo nº de filas: comando ***rbind()***

```
### data.frames ou matrices
cbind(mat1,mat2f)#mesmo nº filas, o nº de columnas é suma de nº colmnas
```

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

```
rbind(mat1,mat2c)#mesmo nº columnas, o nº de filas é suma do nº filas
```

```
##       [,1] [,2] [,3] [,4]
##  [1,]    1   11   21   31
##  [2,]    2   12   22   32
##  [3,]    3   13   23   33
##  [4,]    4   14   24   34
##  [5,]    5   15   25   35
##  [6,]    6   16   26   36
##  [7,]    7   17   27   37
```

```
##  [8,]    8   18   28   38
##  [9,]    9   19   29   39
## [10,]   10   20   30   40
## [11,]    1    6   11   16
## [12,]    2    7   12   17
## [13,]    3    8   13   18
## [14,]    4    9   14   19
## [15,]    5   10   15   20
```

Misturar grupos de datos diferentes

Para usar nesta sesión tiña preparados datos procedentes do **ficheiro de membros** da EPF de 2016, pero non aparecía a comunidade autónoma de pertenza, algo que si aparecía no ficheiro de fogares. Ademais o que me interesa é *engadir unha columna nova* que indica a CCAA de cada membro, polo que se quero usar *__cbind()__ deberían ter o mesmo nº de filas, algo que non ocorre (membros2a ten 48 709; membros2b so ten 22011, o nº de fogares, posto que en cada fogar pode haber máis de 1 membro)

A filosofía desta fusión vai ser a mesma que se usa manexando bases de datos: indicar unha columna índice, que nos sirva de referencia á hora de fusionar.

O índice vai ser o *nº de fogar* que aparece nos dous ficheiros, así se un *membro* está no fogar 316, váiselle asignar a CCAA que ten o fogar CCAA na outra base de datos, e o mesmo para os outros membros do fogar

```
datos0=read.csv2("membros2016.csv")
datos1=read.csv2("fogares2016.csv")
#con R básico: merge()
head(merge(datos0,datos1,by = "NUMERO"))#saco só os seus primeiros
```

```
##   NUMERO EDAD SEXO ECIVILLEGAL UNION ESTUDIOS SITUACT OCU IMPEXACP CCAA
## 1      1   66    1           2     1        2       1   1       -9    7
## 2      1   64    6           2     1        2       1   1       -9    7
## 3      2   35    1           2     1        7       3   2       -9   12
## 4      2   35    6           2     1        4       1   1       -9   12
## 5      3   64    1           2     1        3       1   1       -9   13
## 6      3   60    6           2     1        4       1   1       -9   13
```

```
#con dplyr: hai varios comandos "join"
datos=inner_join(datos0,datos1,"NUMERO")
```

Un método aplicado: Regresión

Sen querer entrar na explicación de métodos estatísticos, que deberían levar un certo tempo, vaise explicar aquí como facer unha regresión, en parte por que é unha técnica común, dado que é o que se ve en Econometría (ou se verá). Por outra parte permite explicar un exemplo de como R crea estructuras especificas para os resultados dos métodos que calcula, e como se poden manexar.

> Usado como referencia: FEIR 40: Modelos de Regresión (Universidade de Murcia)

- Datos para o exemplo:

```
set.seed(43210)
X1=rnorm(100,2,0.5)# xerar 100 numeros aleatorios dunha N(2;0.5)
X2=runif(100,-1,1)# xerar 100 numeros aleatorios dunha Uniforme(-1;1)
X3=rchisq(100,9)# xerar 100 numeros aleatorios dunha chi con 9 graos de liberdade
Y=-3+2*X1-3*X2+0.5*X3+rnorm(100)# Xerar unha regresión a paartir das anteriores X
auxiliar=data.frame(Y,X1,X2,X3)
head(auxiliar)
```

```
##          Y       X1         X2        X3
## 1 8.149117 1.784413 -0.6593172 10.007490
## 2 5.955153 1.812070 -0.2727398  6.291288
## 3 4.221932 2.191241  0.9689510 12.865858
## 4 4.446917 1.963556 -0.2213267  5.778286
## 5 8.025017 1.256601 -0.9267843 10.070912
## 6 7.089026 2.118430  0.2469485 13.517199
```

```
summary(auxiliar)
```

```
##        Y                X1              X2                 X3        
##  Min.   :-0.2769   Min.   :0.725   Min.   :-0.99853   Min.   : 0.9128  
##  1st Qu.: 3.0866   1st Qu.:1.664   1st Qu.:-0.57118   1st Qu.: 5.7522  
##  Median : 4.9643   Median :1.922   Median :-0.07454   Median : 8.3401  
##  Mean   : 5.3115   Mean   :1.943   Mean   :-0.04636   Mean   : 8.7179  
##  3rd Qu.: 7.1945   3rd Qu.:2.217   3rd Qu.: 0.48923   3rd Qu.:10.7770  
##  Max.   :14.8709   Max.   :3.243   Max.   : 0.97073   Max.   :26.4227
```

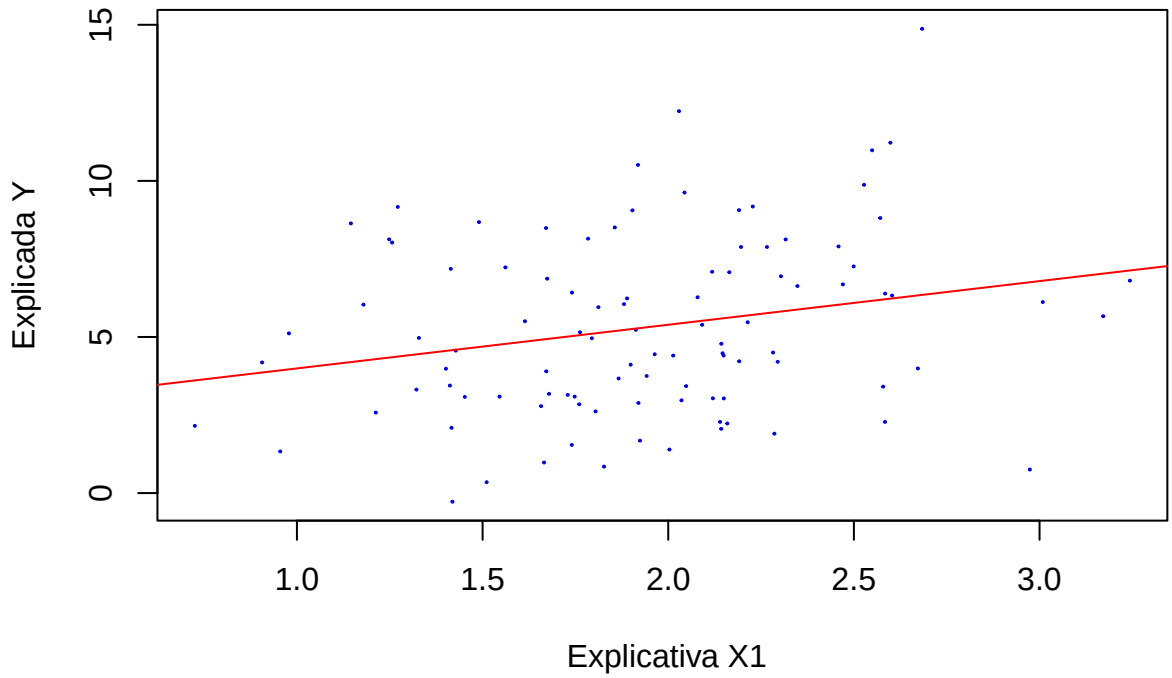
```
apply(auxiliar,2,sd)
```

```
##         Y        X1        X2        X3 
## 2.8873270 0.4939858 0.5949776 4.2304280
```

Gráficos previos: Diagrama de dispersión

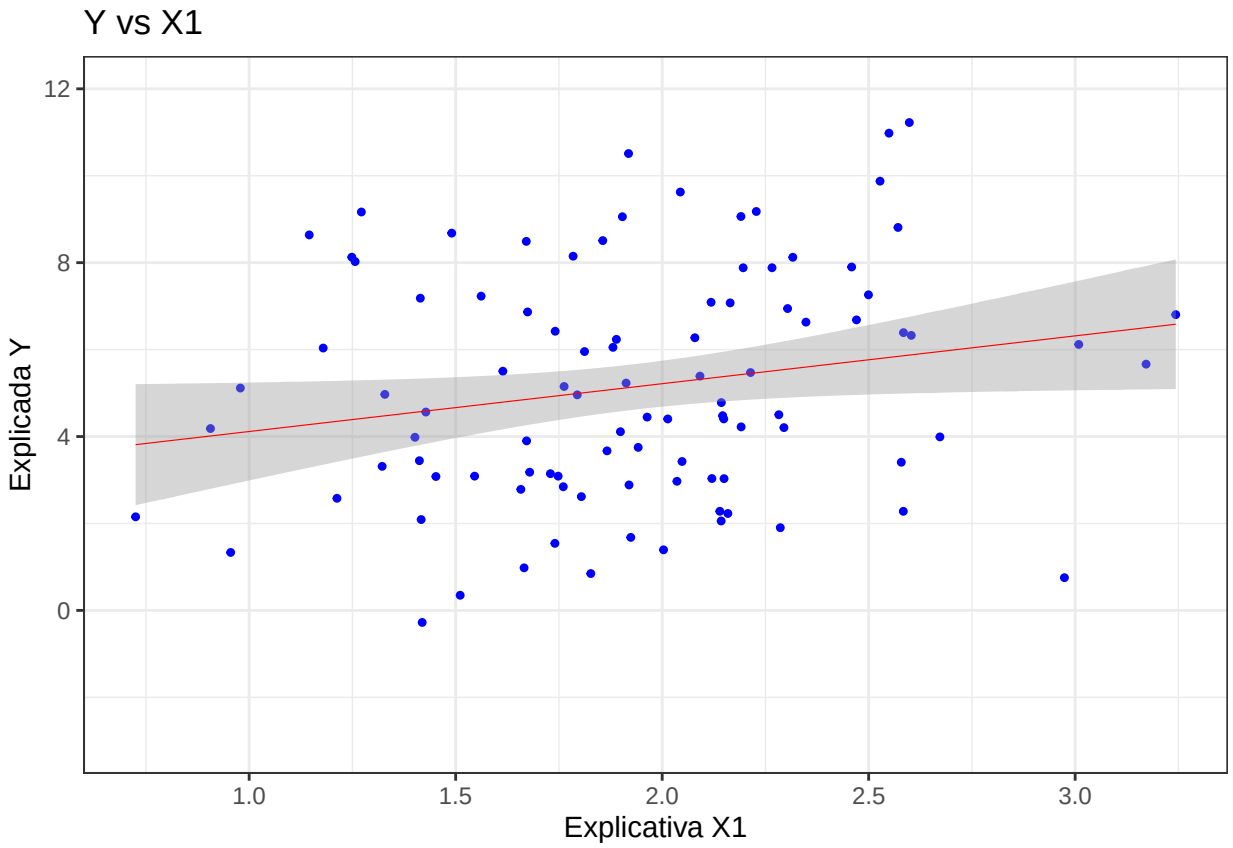
Para observar posible relación entre 2 variables:

```
plot(auxiliar$X1,auxiliar$Y,xlab=" Explicativa X1",ylab="Explicada Y",pch=20,col="blue",cex=0.2)
#cex regula o tamaño dos puntos representados

# Pódese engadir unha recta de regresión, co comando *abline()* indicando como parámetro un comando que
abline(lm(auxiliar$Y~auxiliar$X1),col="red")
```




```
#Con ggplot2: http://www.cookbook-r.com/Graphs/Scatterplots_(ggplot2)/
ggplot(auxiliar, aes(x=X1, y=Y)) +
```

```
    geom_point(shape=20,col="blue") + ylim(-3,12)    +
    geom_smooth(method=lm,col="red",cex=0.2)+ labs(title = "Y vs X1",x="Explicativa X1",y="Explicada Y")
    theme_bw()
```

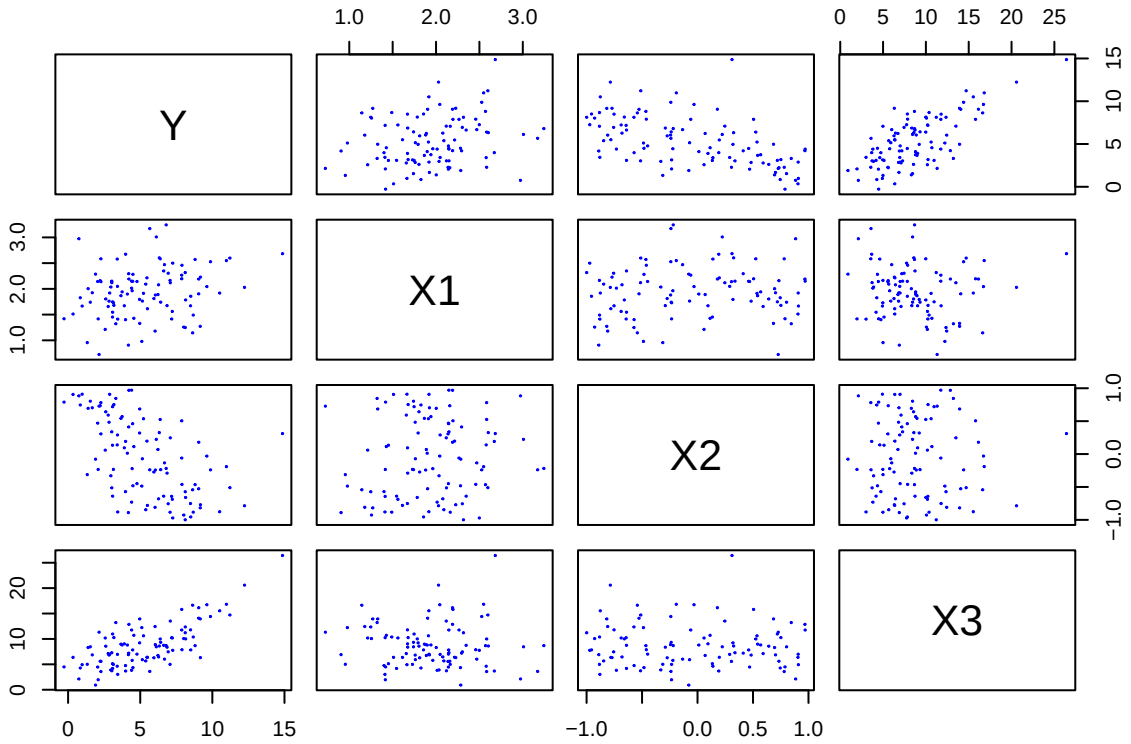


Matriz de dispersión

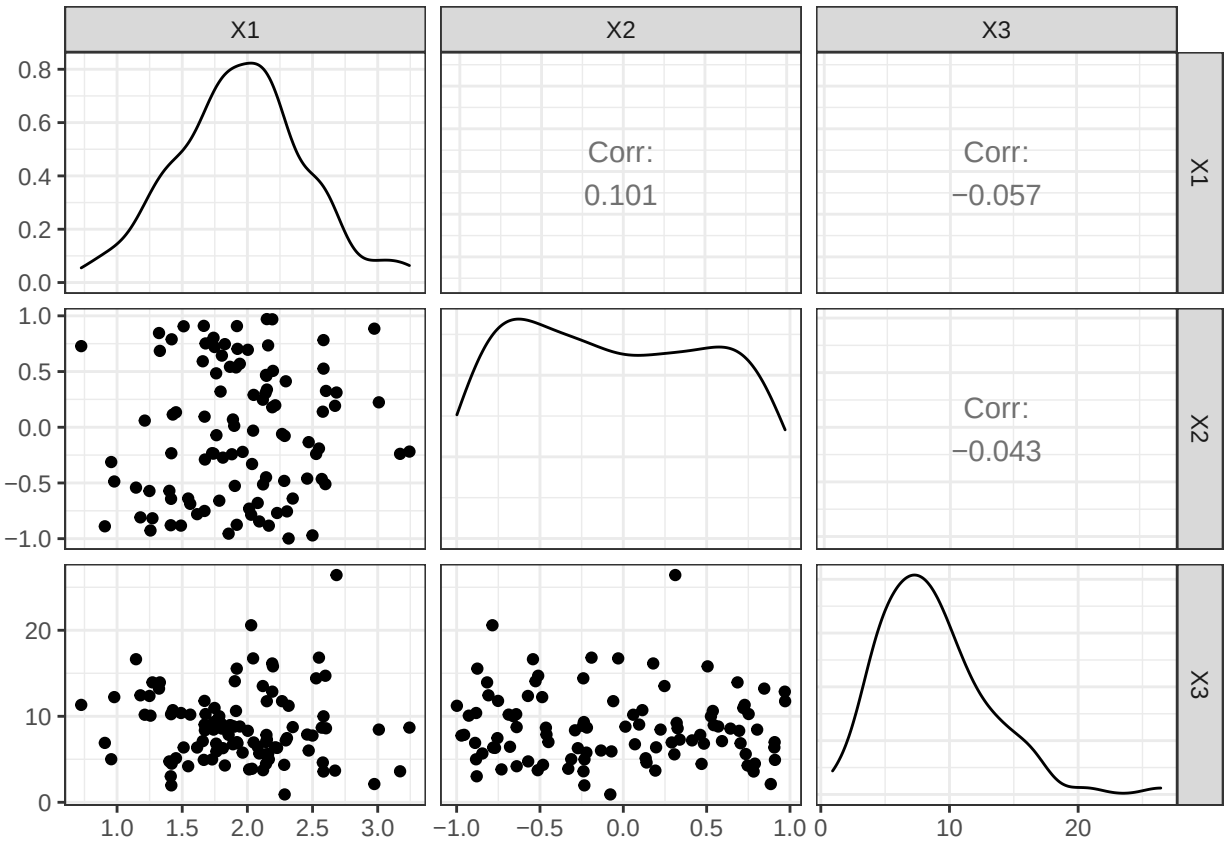
Para observar posible relación entre máis de 2 variables:

plot(data.frame, main="Scatterplot Example", pch=19, col="blue")

```
plot(auxiliar,pch=20,col="blue",cex=0.2)
```




```
#cex regula o tamaño dos puntos representados


#Con ggplot2 úsase un paquete diferente GGally
library(GGally)
ggpairs(auxiliar,columns=c(2,3,4)) +
    theme_bw()    #Hai que seguir explorando
```




```
#coeficiente de correlación simple
cor(auxiliar$X1,auxiliar$Y)
```

Correlación linear simple

```
## [1] 0.2392174
```

```
cor(auxiliar[c(2:4)])
```

```
##             X1          X2          X3
## X1  1.00000000  0.10054464 -0.05671145
## X2  0.10054464  1.00000000 -0.04337028
## X3 -0.05671145 -0.04337028  1.00000000
```

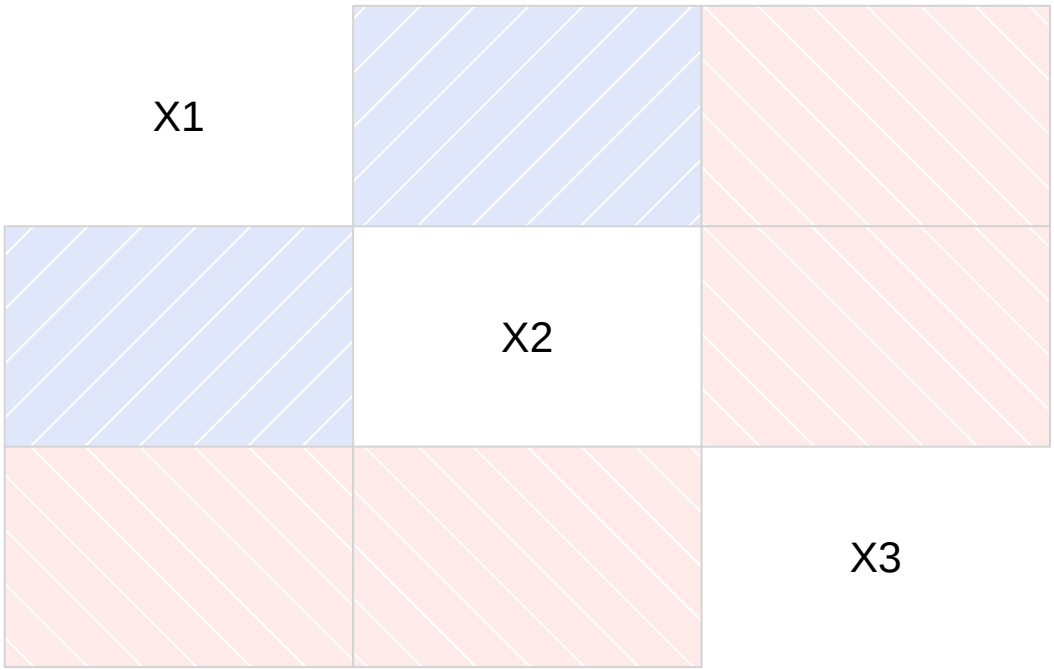
```
#contraste
cor.test(auxiliar$X1,auxiliar$Y)
```

```
##
##  Pearson's product-moment correlation
##
## data:  auxiliar$X1 and auxiliar$Y
## t = 2.4389, df = 98, p-value = 0.01653
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.04490947 0.41608512
## sample estimates:
##       cor
## 0.2392174
```

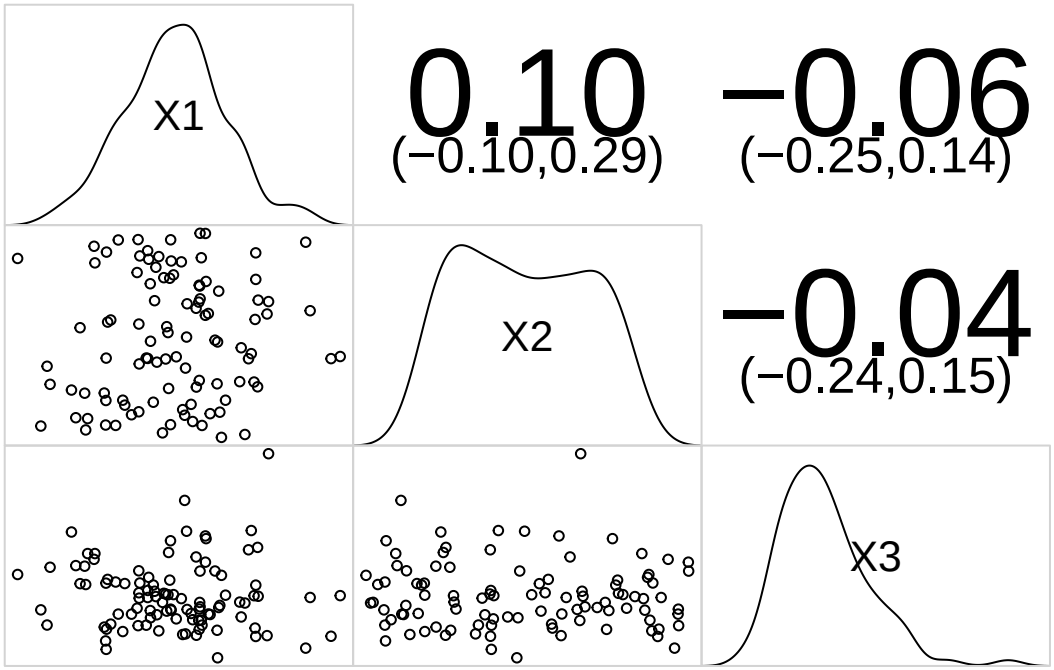
```r
#library psych para matriz de contrastes

corr.test(auxiliar[c(2:4)], use = "complete", method = "pearson")#library("psych")
```

```
## Call:corr.test(x = auxiliar[c(2:4)], use = "complete", method = "pearson")
## Correlation matrix
##       X1    X2    X3
## X1  1.00  0.10 -0.06
## X2  0.10  1.00 -0.04
## X3 -0.06 -0.04  1.00
## Sample Size
## [1] 100
## Probability values (Entries above the diagonal are adjusted for multiple tests.)
##      X1   X2 X3
## X1 0.00 0.96  1
## X2 0.32 0.00  1
## X3 0.58 0.67  0
##
##  To see confidence intervals of the correlations, print with the short=FALSE option
```

```r
corrgram(auxiliar[c(2:4)])#library(corrgram)
```




```r
corrgram(auxiliar[c(2:4)], lower.panel=panel.pts, upper.panel=panel.conf,
         diag.panel=panel.density)
```




```
library("ppcor")
pcor.test(auxiliar$X1,auxiliar$Y,auxiliar[c(3,4)])
```

Correlacion parcial

```
##    estimate      p.value statistic   n gp  Method
## 1 0.7416081 2.460111e-18  10.83167 100  2 pearson
```

Calculo da regresión A regresión obtense co comando *lm()*

lm(variableY ~ variable X1+variableX2+variableX3+variableX4)

Lembrade que dentro de lm vai unha fórmula, polo cal a variable Y vai antes cá variable X

```
lm(Y~X1+X2+X3)
```

```
##
## Call:
## lm(formula = Y ~ X1 + X2 + X3)
##
## Coefficients:
## (Intercept)           X1           X2           X3
##     -2.8656       1.9666      -2.7481       0.4851
```

Só con lm apenas proporciona os valores dos coeficientes, pero podemos usar o comando *summary para sacar uns resultados máis completos*

```
summary(lm(Y~X1+X2+X3))
```

```
##
## Call:
## lm(formula = Y ~ X1 + X2 + X3)
##
## Residuals:
```

```
##     Min      1Q  Median      3Q     Max
## -1.8019 -0.5766  0.0452  0.5693  2.4794
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.86563    0.41645  -6.881 6.08e-10 ***
## X1           1.96662    0.18156  10.832  < 2e-16 ***
## X2          -2.74812    0.15064 -18.243  < 2e-16 ***
## X3           0.48511    0.02111  22.976  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8866 on 96 degrees of freedom
## Multiple R-squared:  0.9086, Adjusted R-squared:  0.9057
## F-statistic:   318 on 3 and 96 DF,  p-value: < 2.2e-16
```

O resultado do comando pódese gardar, como case calquera outra cousa en R. Así podemos empregar eses resultados para outras cousas.

```
regresion=lm(Y~X1+X2+X3)
summary(regresion)
```

```
##
## Call:
## lm(formula = Y ~ X1 + X2 + X3)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.8019 -0.5766  0.0452  0.5693  2.4794
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.86563    0.41645  -6.881 6.08e-10 ***
## X1           1.96662    0.18156  10.832  < 2e-16 ***
## X2          -2.74812    0.15064 -18.243  < 2e-16 ***
## X3           0.48511    0.02111  22.976  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8866 on 96 degrees of freedom
## Multiple R-squared:  0.9086, Adjusted R-squared:  0.9057
## F-statistic:   318 on 3 and 96 DF,  p-value: < 2.2e-16
```

Os resultados gárdanse nun obxecto de clase *lm* que é unha estructura especifica para os resultados do comando ***lm()***, ainda que tamén podemos consideralo como unha **lista**, e acceder aos seus elementos coas mesmas ferramentas que se accede a unha lista.

```
#como ver o que ten ese obxecto: mirando os nomes dos seus elementos, algúns sonaranvos só con velos
names(regresion)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

```
#O summary deste comando produce un obxecto diferente.
# Observade tamén, neste caso, como trata a variable cyl se a consideramos un factor
(regresion2=summary(lm(Y~.,data=auxiliar)))
```

```
##
## Call:
## lm(formula = Y ~ ., data = auxiliar)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.8019 -0.5766  0.0452  0.5693  2.4794
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.86563    0.41645  -6.881 6.08e-10 ***
## X1           1.96662    0.18156  10.832  < 2e-16 ***
## X2          -2.74812    0.15064 -18.243  < 2e-16 ***
## X3           0.48511    0.02111  22.976  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8866 on 96 degrees of freedom
## Multiple R-squared:  0.9086, Adjusted R-squared:  0.9057
## F-statistic:   318 on 3 and 96 DF,  p-value: < 2.2e-16
```

```
names(regresion2)
```

```
##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

Como acceder aos elementos destes obxectos "lista"? da mesma maneira que aos elementos dun data.frame

```
regresion$coefficients#os coeficientes
```

```
## (Intercept)          X1          X2          X3
##  -2.8656304   1.9666172  -2.7481208   0.4851094
```

```
coefficients(regresion)#que tamén teñen comando propio
```

```
## (Intercept)          X1          X2          X3
##  -2.8656304   1.9666172  -2.7481208   0.4851094
```

```
regresion2$adj.r.squared#R2 axustado
```

```
## [1] 0.9057021
```

Comandos *apply*

Os comandos ***apply()*** son unha colección de comandos que permiten aplicar unha función a varios elementos á vez. Hai varios diferentes, e distínguense entre eles polo tipo de estructuras ás que son aplicadas.

- *apply(matriz,numero,función)*: aplícase a matrices
- *lapply(list,función)*: aplícase aos elementos dunha lista (ou as columnas dun data.frame) e saca os resultados nunha lista
- *sapply(list,función)*: aplícase aos elementos dunha lista (ou as columnas dun data.frame) e saca os resultados da maneira máis simple que pode (un vector ou unha data.frame se hai esa posibilidade), é un equivalente simple de ***lapply()***

apply() O comando ***apply()*** hai que indicarlle unha matriz, un numero (1 para que traballe sobre as filas, 2 sobre as columnas) e a función que queremos aplicar.

```
#uso o obxecto auxiliar, que é un data.frame
# apply está pensado para matrices de R, pero tamén funciona con data.frames

#media aritmética por columnas
apply(auxiliar,2,mean)
```

```
##           Y          X1          X2          X3
##  5.31151334  1.94273106 -0.04636185  8.71788404
```

```
#media aritmética por filas
apply(auxiliar,1,mean)
```

```
##   [1]  4.820426  3.446443  5.061995  2.991858  4.606437  5.742901  4.574016
##   [8]  3.180479  4.155058  7.092719  3.568309  2.121148  2.663847  3.133599
##  [15]  6.469217  4.549285  3.175820 11.072318  4.917478  5.890750  2.283226
##  [22]  4.313902  4.443651  5.291646  4.819413  3.100648  2.447806  4.465438
##  [29]  4.312719  1.683489  3.527739  5.299377  6.643708  3.968219  3.077546
##  [36]  3.762673  4.450695  3.676461  5.231317  4.675112  3.050817  4.239510
##  [43]  2.776547  3.201020  3.507924  2.976846  2.793001  1.924779  4.143296
##  [50]  2.392026  6.773976  2.407197  3.759859  4.270319  3.369937  2.688759
##  [57]  1.310192  7.006102  7.539395  2.574981  3.338340  2.046981  4.136568
##  [64]  2.377678  4.898083  3.988683  2.093033  3.735267  2.143985  3.730921
##  [71]  6.127836  2.635064  3.704487  3.531540  4.710821  3.407988  3.768701
##  [78]  5.165556  4.459004  3.189872  8.516057  2.285109  6.597683  3.732953
##  [85]  4.874123  2.306089  4.076218  1.255916  2.683758  1.606537  4.574634
##  [92]  4.628504  5.461985  3.036078  6.894148  1.750034  1.746070  3.176439
##  [99]  4.204544  4.131510
```

```
#desviación típica por filas
apply(auxiliar,1,sd)
```

```
##   [1]  5.073814  3.208855  5.372778  2.662288  5.272100  5.932792  5.012569
##   [8]  2.826076  4.124139  7.649970  2.874571  1.903473  2.328832  3.566140
##  [15]  7.864398  5.037577  3.350843 12.058449  5.459315  6.877244  1.644314
##  [22]  4.356012  4.152391  6.021557  4.836511  3.514097  2.153024  3.706323
##  [29]  4.613018  1.058547  2.895930  5.832859  6.711182  4.313613  3.079622
##  [36]  3.701432  3.593394  3.901023  6.105734  5.795522  2.448110  4.391748
##  [43]  3.462296  2.944279  4.566043  2.736680  2.765215  1.643092  4.101106
##  [50]  2.440749  7.592578  2.208278  3.187264  4.217855  3.229642  1.896475
##  [57]  1.070109  7.144031  7.802049  2.192139  3.460119  2.094464  4.148497
##  [64]  2.308960  4.609639  3.921916  1.858391  3.274390  1.816378  3.194416
##  [71]  6.680774  1.723244  3.784478  3.459248  5.898964  3.533537  3.606385
##  [78]  5.523383  5.697257  2.825727  9.799994  2.767862  6.902412  5.106167
##  [85]  4.190654  1.157760  3.944048  1.060880  2.070044  2.048770  4.485361
##  [92]  3.941015  5.361152  2.859553  7.244947  1.959588  2.285870  2.668712
##  [99]  4.723346  4.660469
```

```
#liala un pouco, introducir na matriz unha fila que só teña NA's, para que non funcione ben mean ou sd
auxiliarNA=rbind(auxiliar,rep(NA,3))
```

Aparte dos comandos xa definidos en **R**, podemos usar as nosas propias funcións, que se son simples, xa poden definirse dentro do comando, e non saen ao exterior del:

```
apply(auxiliarNA,2,function(x) mean(x,na.rm = T))
```

```
##           Y          X1          X2          X3
##  5.31151334  1.94273106 -0.04636185  8.71788404
```

Se queremos algo un pouco máis complexo, podemos crear unha función e usala dentro de apply. Pero verémolo na parte despois de ver como crear funcións.

```
# #funcion gorda
# cousas=function(X){
#   mda=mean(X,na.rm = T)
#   d.t.=sd(X,na.rm = T)
#   c(mda,d.t.)
# }
# apply(auxiliar,2,cousas)
# apply(auxiliar,1,cousas)
```

lapply() e sapply() ***lapply()*** só necesita unha lista, ou coma neste caso, un data.frame, no que aplicará a función sobre as suas columnas:

```
lapply(auxiliar,mean)
```

```
## $Y
## [1] 5.311513
##
## $X1
## [1] 1.942731
##
## $X2
## [1] -0.04636185
##
## $X3
## [1] 8.717884
```

```
# lapply(auxiliar,cousas)
```

O inconveniente é que este comando produce unha lista como saída, o que pode ser complicado de manexar.

Pode obterse unha saida máis manexable con ***sapply()***:

```
sapply(auxiliar,mean)
```

```
##           Y          X1          X2          X3
##  5.31151334  1.94273106 -0.04636185  8.71788404
```

```
# sapply(auxiliar,cousas)
```

```
#Como son estas saidas
str(sapply(auxiliar,mean))
```

```
##  Named num [1:4] 5.3115 1.9427 -0.0464 8.7179
##  - attr(*, "names")= chr [1:4] "Y" "X1" "X2" "X3"
```

```
str(lapply(auxiliar,mean))
```

```
## List of 4
##  $ Y : num 5.31
##  $ X1: num 1.94
##  $ X2: num -0.0464
##  $ X3: num 8.72
```