ANÁLISIS DE DATOS

Ricardo Aler Mur

R

ELEMENTOS BÁSICOS

HISTORIA DE R

HISTORIA DE R

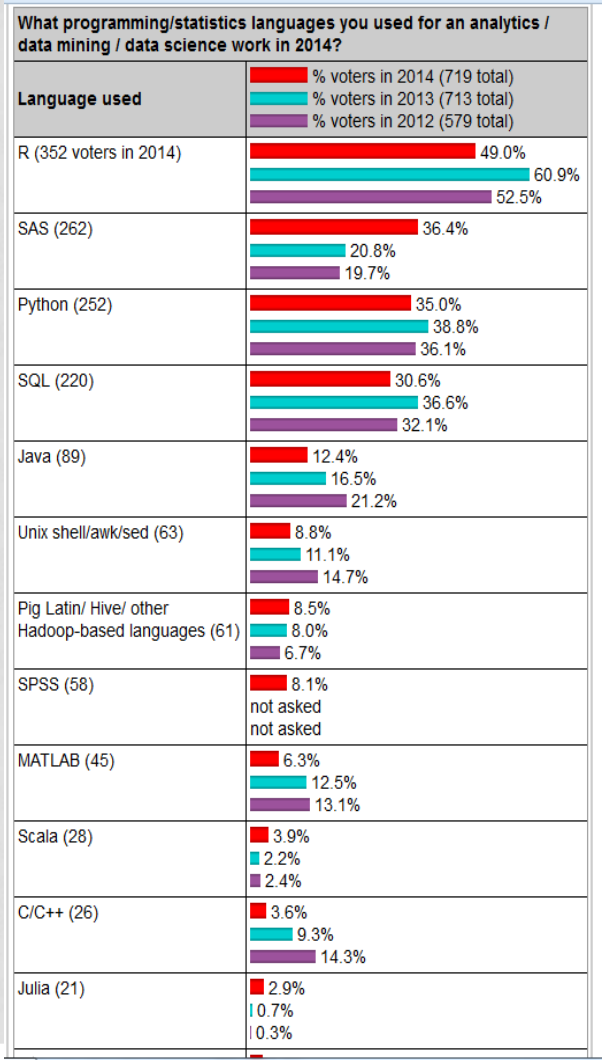
- R es un dialecto del lenguaje S
- S fue desarrollado por John Chambers en los laboratorios Bell en 1976. El objetivo era facilitar el análisis estadístico. Inicialmente usaba librerías en Fortran, después fueron reescritas a C.
- Característica de S: análisis de datos interactivo y también posibilidad de escribir scripts (programas)
- En la actualidad es propiedad de TIBCO (25 millones de dolares)

HISTORIA DE R

- R se crea en 1991en Nueva Zelanda por Ihaka y Gentleman, con el objetivo de tener similares posibilidades a S. Sintaxis similar aunque los detalles internos son distintos
- En el 2000 se crea R 1.0.0 con licencia GNU GPL (software libre)
- La versión 3.0.2 salió en Diciembre 2013. Desarrollo muy activo
- Ejecuta en cualquier plataforma (al parecer incluso en Playstation 3)

HISTORIA DE R

- Ventajas de R:
  - Es libre
  - Es bastante ligero (comparar el arranque de R con el arranque de Matlab)
  - Orientado al proceso y análisis de datos (gracias a la estructura data.frame). El acceso a matrices y data.frames es parecido al de Matlab (ej: m[1:10,1:2] accede a las diez primeras líneas de la matriz y a las dos primeras columnas)
  - Gráficos potentes
  - El más utilizado en análisis de datos (según encuestas)
  - Comunidad muy activa. 4000 paquetes desarrollados y disponibles en CRAN: http://cran.r-project.org/
  - Mucha documentación y libros sobre el lenguaje



What programming/statistics languages you used for an analytics / data mining / data science work in 2014?

Language used	% voters in 2014 (719 total)	% voters in 2013 (713 total)	% voters in 2012 (579 total)
R (352 voters in 2014)	49.0%	60.9%	52.5%
SAS (262)	36.4%	20.8%	19.7%
Python (252)	35.0%	38.8%	36.1%
SQL (220)	30.6%	36.6%	32.1%
Java (89)	12.4%	16.5%	21.2%
Unix shell/awk/sed (63)	8.8%	11.1%	14.7%
Pig Latin/ Hive/ other Hadoop-based languages (61)	8.5%	8.0%	6.7%
SPSS (58)	8.1%	not asked	not asked
MATLAB (45)	6.3%	12.5%	13.1%
Scala (28)	3.9%	2.2%	2.4%
C/C++ (26)	3.6%	9.3%	14.3%
Julia (21)	2.9%	0.7%	0.3%

Mathematical & Statistical Software
Computer Programming
Reference
Computer Programming Language & Tool
Software Development
+ See more

Kindle Store ›
Computer Programming
+ See more

Apps & Games ›
Game Apps

+ See All 32 Departments

Refine by

Eligible for Free Shipping
Free Shipping by Amazon

Book Format
Hardcover
Kindle Edition
Audible Audio Edition
Audio CD
Paperback

Avg. Customer Review
& Up
& Up
& Up
& Up

International Shipping
AmazonGlobal Eligible

Condition
New
Used

Related Searches: r in action, r programming, undefined.

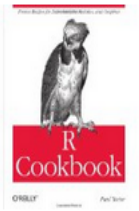
R for Everyone: Advanced Analytics and Graphics (Addison-Wesley Data & Analytics Series) by Jared Lander (Dec 29, 2013)

$44.99 **$25.30** Paperback Prime
Get it by Tuesday, Sep 16

$19.61 Kindle Edition
Auto-delivered wirelessly

More Buying Choices - Paperback
$24.31 new (48 offers)
$24.79 used (15 offers)

(29)
FREE Shipping on orders over $35
Trade-in eligible for an Amazon gift card
Books: See all 1,219,968 items



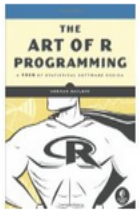
R Cookbook (O'Reilly Cookbooks) by Paul Teetor (Mar 22, 2011)

$39.99 **$28.22** Paperback Prime
Get it by Tuesday, Sep 16

from **$9.61** to rent Kindle Edition
$22.42 to buy
Auto-delivered wirelessly

More Buying Choices - Paperback
$20.24 new (99 offers)
$23.96 used (19 offers)

(46)
#1 Best Seller in Bioinformatics
FREE Shipping on orders over $35
Trade-in eligible for an Amazon gift card
Books: See all 1,219,968 items



The Art of R Programming: A Tour of Statistical Software Design by Norman Matloff (Oct 15, 2011)

$39.95 **$25.09** Paperback Prime
Get it by Tuesday, Sep 16

$23.06 Kindle Edition
Auto-delivered wirelessly

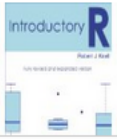
More Buying Choices - Paperback
$16.70 new (62 offers)
$16.46 used (40 offers)

(76)
FREE Shipping on orders over $35
Trade-in eligible for an Amazon gift card
Books: See all 1,219,968 items

Learn R in a Day by Steven Murray (Oct 30, 2013)

$4.89 Kindle Purchase
Auto-delivered wirelessly

(17)
Prime members read for free Join Amazon Prime
Books: See all 1,219,968 items



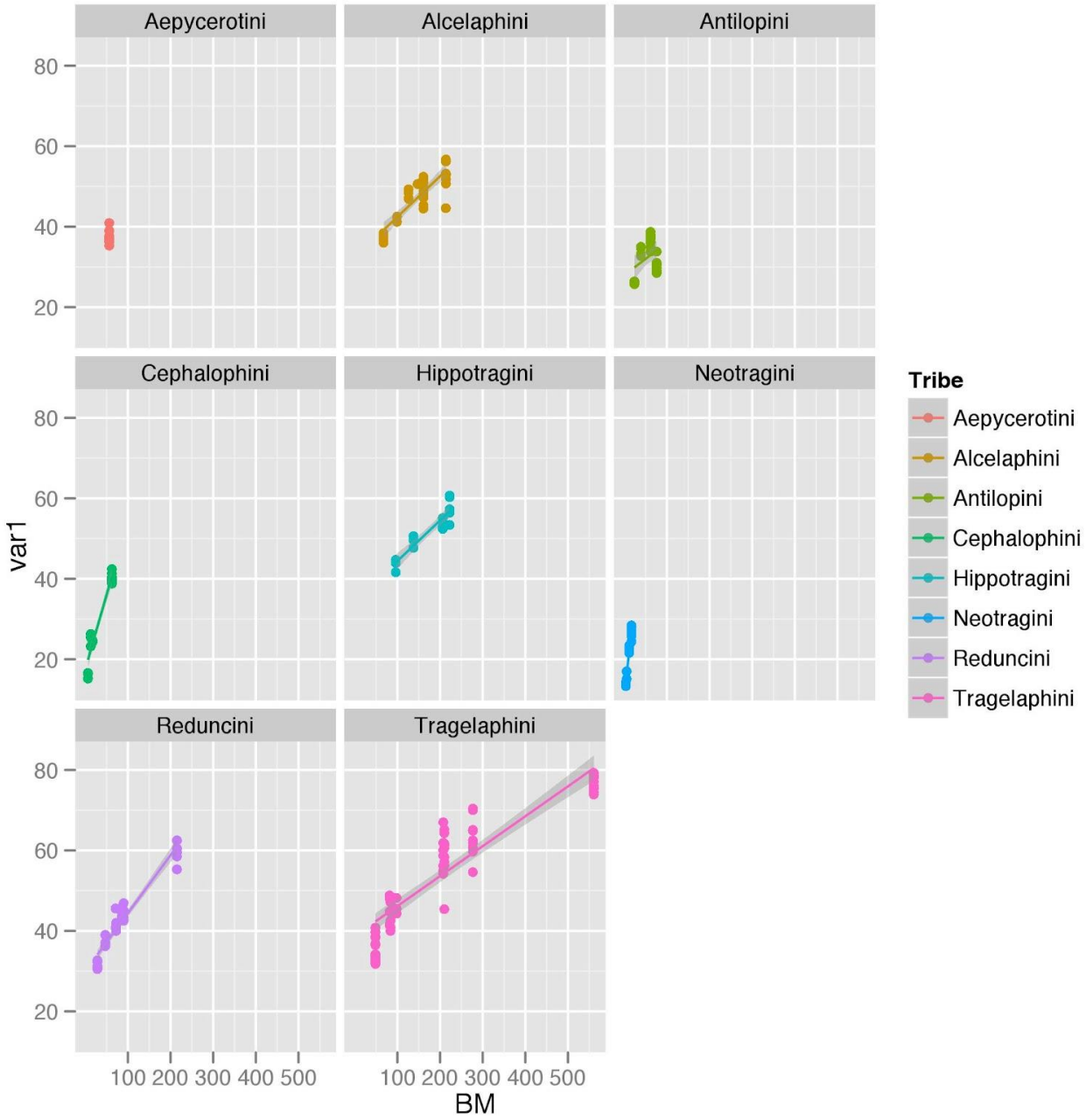
Introductory R: A Beginner's Guide to Data Visualisation, Statistical Analysis and Programming in R by Robert Knell (Mar 13, 2013)

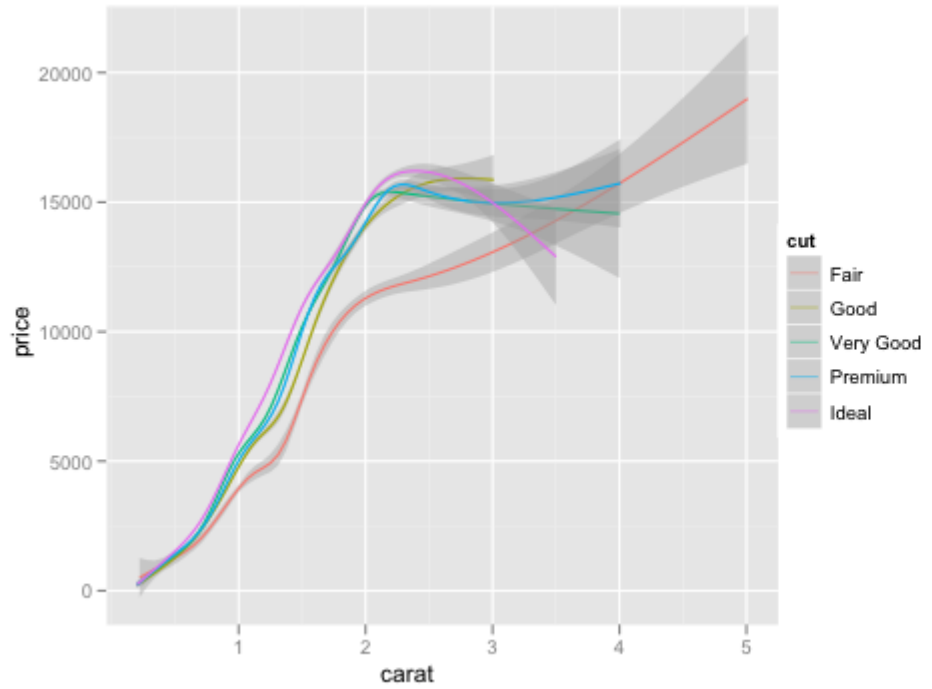
$7.12 Kindle Edition
Auto-delivered wirelessly

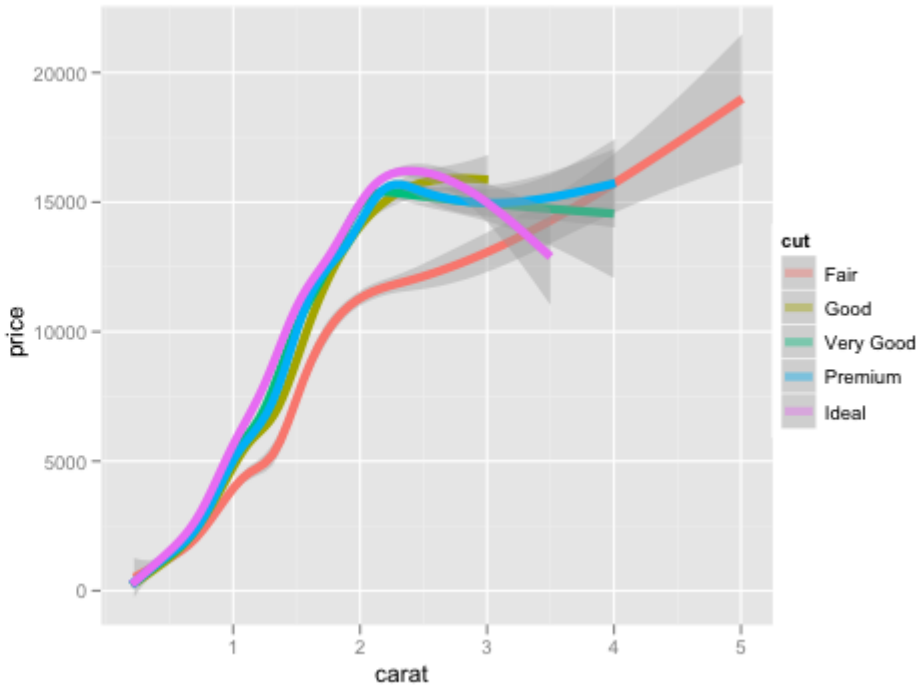
(11)
Books: See all 1,219,968 items

HISTORIA DE R

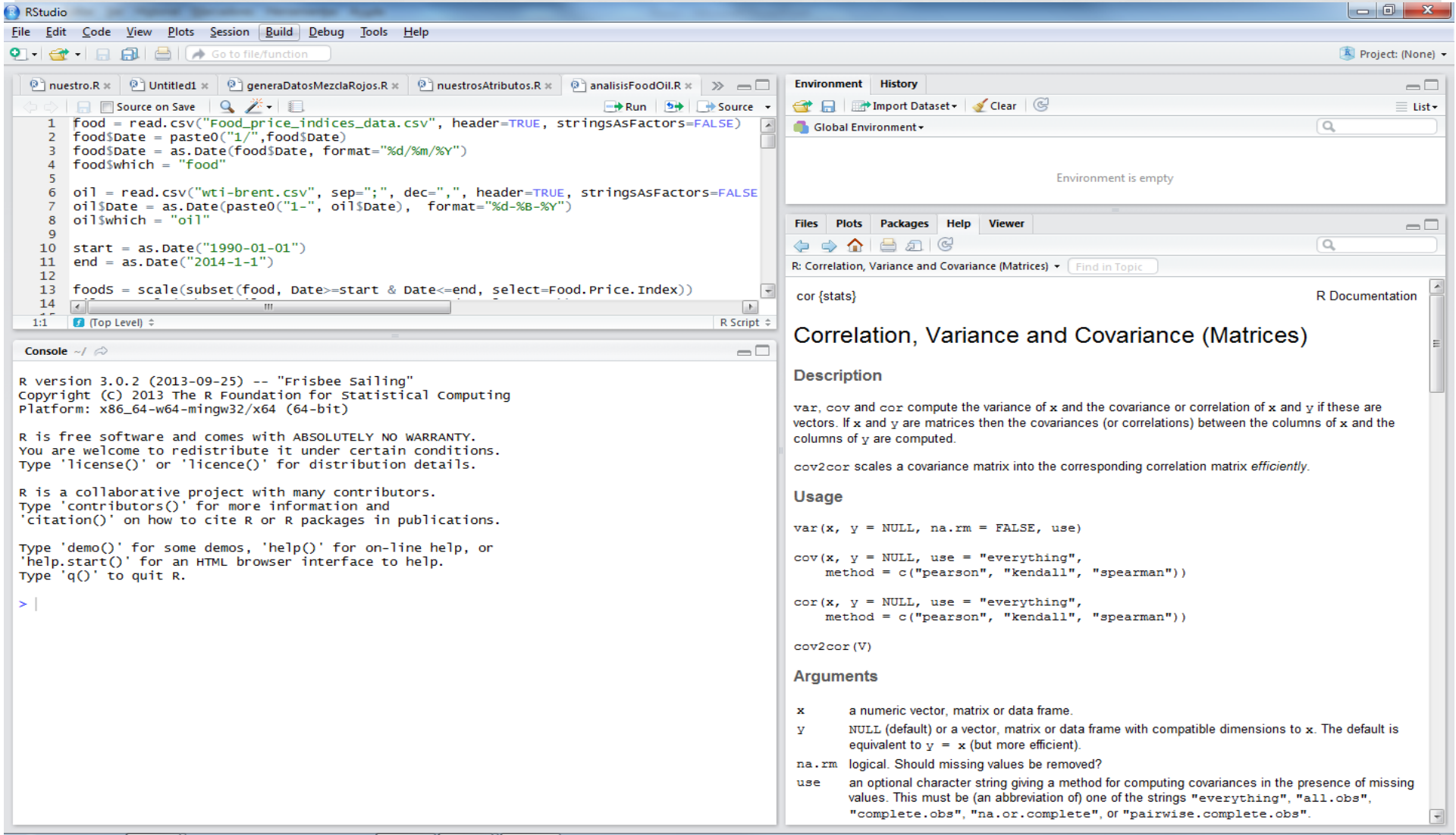
- Desventajas de R:
  - Ya tiene 40 años
  - El procesamiento es con datos en memoria (no tan bueno para conjuntos de datos masivos, aunque dispone de alternativas: comunicación con mySQL, paquetes para que los datos residan parcialmente en disco, H2O, …)
  - Poco soporte para gráficos en 3D, gráficos dinámicos y gráficos interactivos, aunque en el último año han aparecido maneras de generar gráficos interactivos en javascript desde R, con Shiny, rCharts, …
    - http://shiny.rstudio.com/gallery/
    - Ej: http://shiny.rstudio.com/gallery/nvd3-line-chart-output.html


Aepycerotini
Alcelaphini
Antilopini
Cephalophini
Hippotragini
Neotragini
Reduncini
Tragelaphini
var1
BM
80
60
40
20
100 200 300 400 500
Tribe
Aepycerotini
Alcelaphini
Antilopini
Cephalophini
Hippotragini
Neotragini
Reduncini
Tragelaphini


20000
15000
10000
5000
0
price
1
2
3
4
5
carat
cut
Fair
Good
Very Good
Premium
Ideal


20000
15000
10000
5000
0
price
1
2
3
4
5
carat
cut
Fair
Good
Very Good
Premium
Ideal

RSTUDIO


RStudio
File Edit Code View Plots Session Build Debug Tools Help
Project: (None)
nuestro.R
Untitled1
generaDatosMezclaRojos.R
nuestrosAtributos.R
analisisFoodOil.R
Source on Save
Run
Source
1 food = read.csv("Food_price_indices_data.csv", header=TRUE, stringsAsFactors=FALSE)
2 food$Date = paste0("1/",food$Date)
3 food$Date = as.Date(food$Date, format="%d/%m/%Y")
4 food$which = "food"
5
6 oil = read.csv("wti-brent.csv", sep=";", dec=",", header=TRUE, stringsAsFactors=FALSE
7 oil$Date = as.Date(paste0("1-", oil$Date), format="%d-%B-%Y")
8 oil$which = "oil"
9
10 start = as.Date("1990-01-01")
11 end = as.Date("2014-1-1")
12
13 foodS = scale(subset(food, Date>=start & Date<=end, select=Food.Price.Index))
14
1:1 (Top Level) R Script
Console ~/
R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Environment History
Import Dataset Clear List
Global Environment
Environment is empty
Files Plots Packages Help Viewer
R: Correlation, Variance and Covariance (Matrices)
Find in Topic
cor {stats} R Documentation
Correlation, Variance and Covariance (Matrices)
Description
var, cov and cor compute the variance of x and the covariance or correlation of x and y if these are vectors. If x and y are matrices then the covariances (or correlations) between the columns of x and the columns of y are computed.
cov2cor scales a covariance matrix into the corresponding correlation matrix efficiently.
Usage
var(x, y = NULL, na.rm = FALSE, use)
cov(x, y = NULL, use = "everything",
method = c("pearson", "kendall", "spearman"))
cor(x, y = NULL, use = "everything",
method = c("pearson", "kendall", "spearman"))
cov2cor(V)
Arguments
x a numeric vector, matrix or data frame.
y NULL (default) or a vector, matrix or data frame with compatible dimensions to x. The default is equivalent to y = x (but more efficient).
na.rm logical. Should missing values be removed?
use an optional character string giving a method for computing covariances in the presence of missing values. This must be (an abbreviation of) one of the strings "everything", "all.obs", "complete.obs", "na.or.complete", or "pairwise.complete.obs".

OBJETOS EN R

OBJETOS EN R

- Escalares: son realmente vectores de un elemento
- Vectores: todos los elementos del mismo tipo
  - Factores
- Matrices: todos los elementos del mismo tipo
- Listas: permite combinar elementos de tipos distintos
- Data frames: son matrices con elementos de distintos tipos

- **Nota**: para este tutorial se ha seguido
- http://www.dcc.fc.up.pt/~ltorgo/DataMiningWithR/code.html

VARIABLES

- Asignaciones a variables:
  - x <- 945 es lo mismo que x = 945
  - Ojo! R es case-sensitive

```
x <- 945

x
```

```
## [1] 945
```

LISTAR Y BORRAR VARIABLES

- ls(): listar objetos, rm(): borrar objetos
- rm(list=ls()): borrar todos los objetos

```
ls()
```

```
## [1] "i"        "metadata" "w"        "x"        "y"        "z"
```

```
rm(y)
rm(z,w,i)
```

TIPOS DE DATOS ESCALARES ("ATOMIC")

- Character:

  x = "cadena de caracteres"

  x = 'cadena de caracteres'

  > paste("variable=",3,"\n")

  [1] "variable= 3 \n"

  > cat(paste("variable=",3,"\n"))

  variable= 3

- Logical:
  - x = TRUE, x = T
  - x = FALSE, x = F

- Numeric:
  - x = 3
  - x = NA
    - is.na(x) == TRUE
    - (valor por omisión, missing value)
  - x = 1/0 == Inf
  - X=Inf/Inf == NaN
    - is.nan(x) == TRUE
- Complex:
  - x = 3 + 4i
  - is.complex(x) == TRUE

OBJETOS EN R

- Escalares
- **Vectores**
  - Factores
- Matrices
- Data frames
- Listas

VECTORES

```
v <- c(4,7,23.5,76.2,80)
v
```

```
## [1]  4.0  7.0 23.5 76.2 80.0
```

```
length(v)
```

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

```
mode(v)
```

```
## [1] "numeric"
```

```
v[2]
```

```
## [1] "7"
```

```
v[1] <- 'hello'
v
```

```
## [1] "hello" "7"     "23.5"  "76.2"  "80"    "rrt"
```

NA es "sin valor" o valor faltante (missing value, not acknowledged)

```
u <- c(4,6,NA,2)
u
```

```
## [1]  4  6 NA  2
```

- Todos los elementos deben ser del mismo tipo:

```
v <- c(4,7,23.5,76.2,80,"rrt")
v
```

```
## [1] "4"    "7"    "23.5" "76.2" "80"   "rrt"
```

VECTORES

- Vector vacío:

```
> x <- vector()
```

- Redimensionamiento dinámico de vectores:

```
> x[3] <- 45
> x

[1] NA NA 45
```

VECTORES

- Concatenación de vectores:

```
> x = c(1,2,3)
> y = c(4,5)
> c(x,y)
[1] 1 2 3 4 5
```

VECTORIZACIÓN

- Normalmente, una función aplicada a un vector, es aplicada a cada uno de los elementos

```
v <- c(4,7,23.5,76.2,80)
x <- sqrt(v)
x
```

```
## [1] 2.000 2.646 4.848 8.729 8.944
```

```
v1 <- c(4,6,87)
v2 <- c(34,32.4,12)
v1+v2
```

```
## [1] 38.0 38.4 99.0
```

VECTORIZATION

- Nuestras propias funciones también están vectorizadas

```
> mif = function(x) {x^2}
> x
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> mif(x)
     [,1] [,2] [,3]
[1,]    1    4    9
[2,]   16   25   36
```

REGLA DEL RECICLADO

- Si por ejemplo se suman dos vectores de distinto tamaño, el mas pequeño se replica hasta que tiene el mismo tamaño que el grande (esto es cierto incluso con valores individuales)

```
v1 <- c(4,6,8,24)
v2 <- c(10,2)
v1+v2
```

```
## [1] 14  8 18 26
```

```
v1 <- c(4,6,8,24)
2*v1
```

```
## [1]  8 12 16 48
```

```
v1 <- c(4,6,8,24)
v2 <- c(10,2,4)
v1+v2
```

```
## Warning: longitud de objeto mayor no es múltiplo de la longitud de uno
## menor
```

```
## [1] 14  8 12 34
```

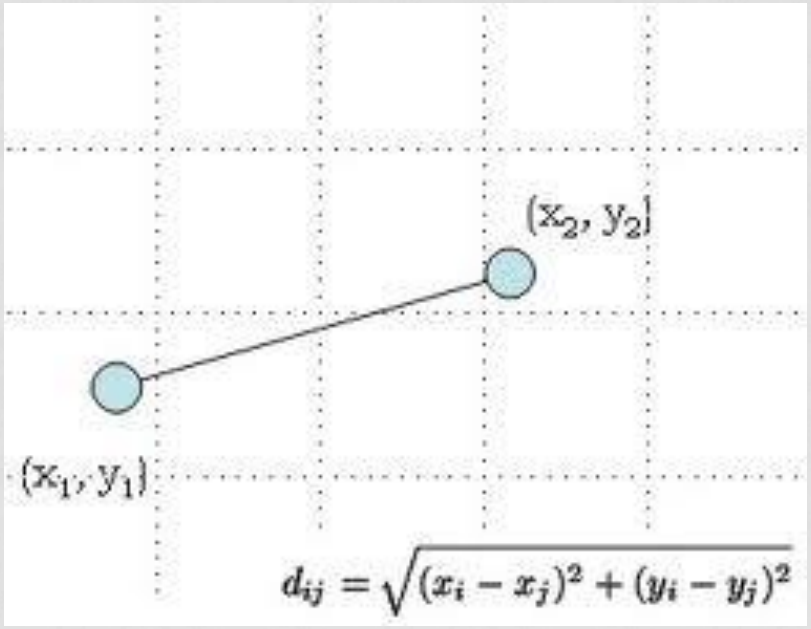
ARITMÉTICA DE VECTORES

- +,-,*,/,^: Se aplican componente a componente. Si los dos vectores tienen tamaños distintos, se aplica la regla del reciclado:

```
> x = c(1,2,3,4)
> x+1
[1] 2 3 4 5
> x^c(1,2)
[1]  1  4  3 16
> x+c(10,11,12,13)
[1] 11 13 15 17
```

- Producto escalar: x %*% y
  - (x1,x2,x3) . (y1,y2,y3) == x1*y1+x2*y2+x3*y3
- %/%: división entera, %%: módulo (resto)

EJEMPLO: CÁLCULO DE LA DISTANCIA EUCLIDEA



$$E(x,y) = \sqrt{\sum_{i=0}^{n}(x_i - y_i)^2}$$

EJEMPLO: CÁLCULO DE LA DISTANCIA EUCLIDEA

$$E(x,y) = \sqrt{\sum_{i=0}^{n}(x_i - y_i)^2}$$

```
> x = c(1,2,3,4)
> y = c(6,7,8,9)
> x-y
[1] -5 -5 -5 -5
> (x-y)*(x-y)
[1] 25 25 25 25
> x = c(1.1,1.5,3.7)
> y = c(7.9,8.0,1.3)
> x-y
[1] -6.8 -6.5  2.4
> (x-y)*(x-y)
[1] 46.24 42.25  5.76
> sum((x-y)*(x-y))
[1] 94.25
> sqrt(sum((x-y)*(x-y)))
[1] 9.708244
```

```
> deuclidea1 = function(x,y)
{return(sqrt(sum((x-y)*(x-y))))}
> deuclidea1(x,y)
[1] 9.708244
```

```
> deuclidea2 = function(x,y)
{return(sqrt((x-y) %*% (x-y)))}
> deuclidea2(x,y)
         [,1]
[1,] 9.708244
```

OBJETOS EN R

- Escalares
- Vectores
  - **Factores**
- Matrices
- Data frames
- Listas

FACTORES

- Se utilizan en análisis de datos como una representación eficiente de valores discretos (categóricos)

```
g <- c('f','m','m','m','f','m','f','m','f','f')
g
```

```
##  [1] "f" "m" "m" "m" "f" "m" "f" "m" "f" "f"
```

```
g <- factor(g)
g
```

```
##  [1] f m m m f m f m f f
## Levels: f m
```

FACTORES

```
other.g <- factor(c('m','m','m','m','m'),levels=c('f','m'))
other.g
```

```
## [1] m m m m m
## Levels: f m
```

GENERAR VECTORES MEDIANTE SECUENCIAS

- Crear un vector con enteros de 1 a 1000

```
x <- 1:1000
```

- Cuidado con la precedencia de los operadores:

```
10:15-1
```

```
## [1]  9 10 11 12 13 14
```

```
10:(15-1)
```

```
## [1] 10 11 12 13 14
```

GENERAR VECTORES MEDIANTE SECUENCIAS

- Secuencias invertidas y secuencias de números reales:

```
5:0
```

```
## [1] 5 4 3 2 1 0
```

```
seq(-4,1,0.5)
```

```
##  [1] -4.0 -3.5 -3.0 -2.5 -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0
```

```
seq(length=10,from=-2,by=.2)
```

```
##  [1] -2.0 -1.8 -1.6 -1.4 -1.2 -1.0 -0.8 -0.6 -0.4 -0.2
```

GENERAR VECTORES MEDIANTE REPETICIONES

```
rep(5,10)
```

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

```
rep('hi',3)
```

```
## [1] "hi" "hi" "hi"
```

```
rep(1:2,3)
```

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

```
rep(1:2,each=3)
```

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

SECUENCIAS ALEATORIAS

- 10 valores de una gaussiana(0,1)

```
rnorm(10)
```

```
##  [1] -0.8171  0.8899 -0.1813  0.2822  1.6907 -0.4557  1.4784  0.3828
##  [9]  0.2951  0.3880
```

- Gaussiana con media 10 y desviación 3

```
rnorm(4,mean=10,sd=3)
```

```
## [1] 14.15  9.17  9.28 11.16
```

- 5 valores de t-student con 10 grados de libertad

```
rt(5,df=10)
```

```
## [1]  0.06662 -2.04782  0.53320 -0.11451 -0.37334
```

ACCESO A VECTORES (SUB-SETTING)

- Acceso simple: x[3]
- Acceso mediante índices. Tres tipos:
  - Índices lógicos (booleanos)
  - Índices por valor
  - Índices por nombres

ACCESO A VECTORES MEDIANTE ÍNDICES LÓGICOS

```
x[x>0]
```

```
## [1]  4 45 90
```

```
x[x <= -2 | x > 5]
```

```
## [1] -3 45 90 -5
```

```
x[x > 40 & x < 100]
```

```
## [1] 45 90
```

==, !=, >, <, >=, <=, &, | , !, is.na(), is.nan()

ACCESO A VECTORES MEDIANTE ÍNDICES BOOLEANOS

- Convertir todos los valores NA a cero:

```
> x = c(1,2,NA,3,NA,4)
> x
[1]  1  2 NA  3 NA  4
> x[is.na(x)] = 0
> x
[1] 1 2 0 3 0 4
```

- Cambiar el signo a los valores negativos (igual que x = abs(x)):

```
> x = c(-1, 2, -3, 4)
> x[x<0] = -x[x<0]
> x
[1] 1 2 3 4
```

ACCESO A VECTORES MEDIANTE ÍNDICES DE VALORES

```
> x <- c(0, -3, 4, -1, 45, 90, -5)
```

```
x[c(4,6)]
```

```
## [1] -1 90
```

```
x[1:3]
```

```
## [1]  0 -3  4
```

```
y <- c(1,4)
x[y]
```

```
## [1]  0 -1
```

Se puede usar el "-" para excluir valores:

```
x[-1]
```

```
## [1] -3  4 -1 45 90 -5
```

```
x[-c(4,6)]
```

```
## [1]  0 -3  4 45 -5
```

```
x[-(1:3)]
```

```
## [1] -1 45 90 -5
```

ACCESO A VECTORES POR NOMBRE

- Las posiciones de un vector pueden tener nombre

```
pH <- c(4.5,7,7.3,8.2,6.3)
names(pH) <- c('area1','area2','mud','dam','middle')
pH
```

```
##  area1  area2    mud    dam middle
##    4.5    7.0    7.3    8.2    6.3
```

```
pH <- c(area1=4.5,area2=7,mud=7.3,dam=8.2,middle=6.3)
```

- Acceso por nombre:

```
pH['mud']
```

```
## mud
## 7.3
```

```
pH[c('area1','dam')]
```

```
## area1   dam
##   4.5   8.2
```

ACCESO AL VECTOR COMPLETO

- Por ejemplo, para borrar todos los elementos de un vector x:
  - x[] = 0 es lo mismo que x[1:length(x)] = 0 (se aplica la "regla de reciclado"). Asigna 0 a todas las posiciones del vector
  - pero es distinto de x = 0, el cual convierte x en un único valor

```
> x
[1] 1 2 3 4
> x[]=0
> x
[1] 0 0 0 0
> x = 0
> x
[1] 0
```

ESTRUCTURAS DE CONTROL

- ESTRUCTURAS DE CONTROL:
  - for
  - while
  - repeat

ESTRUCTURAS DE CONTROL

El bucle for recorre vectores (de cualquier tipo):

```
> for (i in c(1,2,3)) { print(i)}
[1] 1
[1] 2
[1] 3

> for (i in c("uno","dos","tres"))
{print(i)}
[1] "uno"
[1] "dos"
[1] "tres"
```

```
> i=1
> while(i<4){print(i); i=i+1}
[1] 1
[1] 2
[1] 3
```

```
> i=1
> while(TRUE){print(i); i=i+1;
if(i>=4) break}
[1] 1
[1] 2
[1] 3
```

HACIENDO LAS COSAS A LA MANERA DE R

- Contar cuantos valores son impares en un vector x

PEOR

```
for (i in 1:length(x)) {
   if (x[i] %% 2 == 1) k <- k+1
}
```

MEJOR

```
for (n in x)  {
   if (n %% 2 == 1) k <- k+1
}
```

CON VECTORIZACIÓN: sum(x %% 2) o sum(x %% 2 == 1)

ESTRUCTURAS DE CONTROL

IF:

```
> if(i>3) {print(i)}
> if(i>2) {print(i)} else {print(2*i)}
```

IF es una función!

```
> x = if(i>3) i else 2*i
> x
[1] 4
```

ESTRUCTURAS DE CONTROL: FUNCIONES

- Las funciones son objetos
- Pueden llevar parámetros con nombre con valor por omisión
- Una función devuelve lo último que se ejecuta (se puede utilizar return, pero no es necesario)

```
> f = function(x, y=FALSE) {if(y) x else x*x}
> f
function(x, y=FALSE) {if(y) x else x*x}
> f(2)
[1] 4
> f(2,TRUE)
[1] 2
> f(2,y=TRUE)
[1] 2
```

FICHEROS CON PROGRAMAS (SCRIPTS)

- Archivo / Nuevo script
- Archivo / Abrir script
- CTRL-R: ejecutar una línea del script
- Es necesario usar source("miscript.R") cada vez que modifiquemos el script
  - (esto se hará automáticamente cuando usemos Rstudio)

ALGUNOS EJERCICIOS

EXPERIMENTO VECTORIZACIÓN

- Como medir el tiempo: t0 = proc.time()
- **Ejercicio**: hacer un script que genere dos vectores:
  - x=seq(1,10^6)
  - y=x*5.2

  - Y los multiplique componente a componente de dos maneras:
    - Con un bucle
    - x*y
    - Medir tiempos y comparar

HACIENDO LAS COSAS AL MODO R

- Sea un vector x = seq(1,10^5)
- Queremos calcular otro vector y de tal manera que
  - y[i] = x[i]-x[i+1] para todo i de 1 a length(x)-1
- Hacedlo de dos maneras distintas (con bucle y vectorizado) y pensad si se os ocurre alguna variante mas. Medid tiempos.

- Regla: evitar usar bucles en la medida de lo posible

EJERCICIOS

- Crear un vector de 10 elementos así:
  - x=sample(1:100, 10)
- Ejercicios:
  - Poner a cero los valores pares.
  - Poner a cero las **posiciones** pares. Hacedlo de **tres** maneras distintas
  - Escribir una función avg_gt con dos argumentos: x y gt (x es un vector y gt es un real). La función computa la media de los valores de x mas grandes que gt.

OBJETOS EN R

- Escalares
- Vectores
  - Factores
- **Matrices**
- Data frames
- Listas

MATRICES

- Son como los vectores, pero en **DOS** dimensiones
- Se pueden crear a partir de un vector y cambiando la dimensión. Notar que los valores se extienden por columnas:

```
m <- c(45,23,66,77,33,44,56,12,78,23)
m
```

```
##  [1] 45 23 66 77 33 44 56 12 78 23
```

```
dim(m) <- c(2,5)
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   45   66   33   56   78
## [2,]   23   77   44   12   23
```

- De manera equivalente:

```
m <- matrix(c(45,23,66,77,33,44,56,12,78,23),2,5)
```

MATRICES

- Por filas:

```
m <- matrix(c(45,23,66,77,33,44,56,12,78,23),2,5,byrow=T)
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   45   23   66   77   33
## [2,]   44   56   12   78   23
```

CONSTRUYENDO MATRICES

POR FILAS:

```
> rbind(c(1,2,3),c(4,5,6))
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

POR COLUMNAS:

```
> cbind(c(1,2,3),c(4,5,6))
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

CONSTRUYENDO MATRICES

```
> x
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> y
     [,1] [,2] [,3] [,4] [,5]
[1,]   10   11   12   13   14
[2,]   15   16   17   18   19
> cbind(x,y)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    2   10   11   12   13   14
[2,]    3    4   15   16   17   18   19
```

ACCESOS A MATRICES: POR VALORES

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   45   23   66   77   33
## [2,]   44   56   12   78   23
```

```
m[2,3]
```

```
## [1] 12
```

```
m[-2,1]
```

```
## [1] 45
```

```
m[1,-c(3,5)]
```

```
## [1] 45 23 77
```

```
m[1,]
```

```
## [1] 45 23 66 77 33
```

```
m[,4]
```

```
## [1] 77 78
```

ACCESOS A MATRICES

- Manteniendo el formato columna:

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   45   23   66   77   33
## [2,]   44   56   12   78   23
```

```
m[1,]
```

```
## [1] 45 23 66 77 33
```

```
m[,4]
```

```
## [1] 77 78
```

```
m[1,,drop=F]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   45   23   66   77   33
```

```
m[,4,drop=F]
```

```
##      [,1]
## [1,]   77
## [2,]   78
```

ACCESOS A MATRICES: MEDIANTE UNA LISTA DE POSICIONES

- Mediante una lista de posiciones

```
>  x = rbind(c(1,2,3),c(4,5,6))
> x
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> i = rbind(c(1,1),c(2,2),c(2,3))
> i
     [,1] [,2]
[1,]    1    1
[2,]    2    2
[3,]    2    3
> x[i]
[1] 1 5 6
```

ACCESOS A MATRICES: MEDIANTE ÍNDICES BOOLEANOS

```
>  x = rbind(c(1,2,3),c(4,5,6))
> x
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> x>2
      [,1]  [,2] [,3]
[1,] FALSE FALSE TRUE
[2,]  TRUE  TRUE TRUE
> x[x>2]
[1] 4 5 3 6
```

DANDO NOMBRES A FILAS Y COLUMNAS Y ACCESO POR NOMBRE

```
results <- matrix(c(10,30,40,50,43,56,21,30),2,4,byrow=T)
colnames(results) <- c('1qrt','2qrt','3qrt','4qrt')
rownames(results) <- c('store1','store2')
results
```

```
##        1qrt 2qrt 3qrt 4qrt
## store1   10   30   40   50
## store2   43   56   21   30
```

```
results['store1',]
```

```
## 1qrt 2qrt 3qrt 4qrt
##   10   30   40   50
```

```
results['store2',c('1qrt','4qrt')]
```

```
## 1qrt 4qrt
##   43   30
```

ARITMÉTICA DE MATRICES

- %*%: producto matricial entre dos matrices
- +,-,*,/,^: Se aplican componente a componente.
- Regla del reciclado:
  - Si x e y son matrices, tienen que tener el mismo número de componentes
  - Si x es matriz e y es escalar, entonces se aplica la regla del reciclado
  - Si x es matriz e y es vector, se considera que ambos son columnas y se aplica reciclado
  - t(): transpone la matriz. Si se aplica a un vector se considera que este es **columna**
- Se puede utilizar log(), sqrt(), …, sobre matrices

```
> x
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> x+1
     [,1] [,2] [,3]
[1,]    2    3    4
[2,]    5    6    7
> x^2
     [,1] [,2] [,3]
[1,]    1    4    9
[2,]   16   25   36
```

```
> t(c(1,2,3))
     [,1] [,2] [,3]
[1,]    1    2    3
> t(t(c(1,2,3)))
     [,1]
[1,]    1
[2,]    2
[3,]    3
```

EJERCICIO MATRICES

1. Crear una matriz de 10x10 y poner a cero aquellas coordenadas (i,j) donde i es par y j es impar
2. Crear una matriz de 10x10 y poner a cero el rectángulo 3 a 5 (en la coordenada x) y de 5 a 8 (en la coordenada y)

OBJETOS EN R

- Escalares
- Vectores
  - Factores
- Matrices
  - **Arrays**
- Data frames
- Listas

ARRAYS

- Son matrices en mas de dos dimensiones

```
a <- array(1:24,dim=c(4,3,2))
a
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]   13   17   21
## [2,]   14   18   22
## [3,]   15   19   23
## [4,]   16   20   24
```

```
a[1,3,2]
```

```
## [1] 21
```

```
a[1,,2]
```

```
## [1] 13 17 21
```

```
a[4,3,]
```

```
## [1] 12 24
```

```
a[c(2,3),,-2]
```

```
##      [,1] [,2] [,3]
## [1,]    2    6   10
## [2,]    3    7   11
```

OBJETOS EN R

- Escalares
- Vectores
  - Factores
- Matrices
- **Listas**
- Data frames

LISTAS

- Pueden contener distintos tipos de datos (números, cadenas, …)
- Los campos pueden tener nombre
- Creación de una lista

```
my.lst <- list(stud.id=34453,
               stud.name="John",
               stud.marks=c(14.3,12,15,19))


my.lst
```

```
## $stud.id
## [1] 34453
##
## $stud.name
## [1] "John"
##
## $stud.marks
## [1] 14.3 12.0 15.0 19.0
```

LISTAS

- Acceso a un componente concreto de una lista (doble corchete)

```
> my.lst[[1]]
[1] 34453
> my.lst[[3]]
[1] 14.3 12.0 15.0 19.0
```

- Si se usa simple corchete, se accede a una **sublista**

```
> my.lst[2]
$stud.name
[1] "John"

> my.lst[2:3]
$stud.name
[1] "John"

$stud.marks
[1] 14.3 12.0 15.0 19.0
```

LISTAS

- Acceso a un componente de una lista por nombre:

```
my.lst$stud.id

## [1] 34453
```

- Acceder a la lista de nombres de una lista:

```
names(my.lst)

## [1] "stud.id"    "stud.name"  "stud.marks"
```

- Dando nombres a los componentes de una lista:

```
names(my.lst) <- c('id','name','marks')
```

- Paso de lista a vector: unlist()

OBJETOS EN R

- Escalares
- Vectores
  - Factores
- Matrices
- Listas
- **Data frames**

DATA FRAMES

- Es la estructura adecuada para almacenar tablas de datos, porque permiten combinar en una matriz distintos tipos de datos (números, cadenas, …)
- Ejemplo de creación de data frame (se hace por columnas):

```
my.dataset <- data.frame(site=c('A','B','A','A','B'),
 season=c('Winter','Summer','Summer','Spring','Fall'),
 pH = c(7.4,6.3,8.6,7.2,8.9))
```

DATA FRAMES

- Acceso por índice:

```
my.dataset[3,2]
```

```
## [1] Summer
## Levels: Fall Spring Summer Winter
```

- Acceso por nombre de columna:

```
my.dataset$pH
```

```
## [1] 7.4 6.3 8.6 7.2 8.9
```

DATA FRAMES. ACCESO POR SUBSETTING

```
my.dataset[my.dataset$pH > 7,]
```

```
##   site season  pH
## 1    A Winter 7.4
## 3    A Summer 8.6
## 4    A Spring 7.2
## 5    B   Fall 8.9
```

```
my.dataset[my.dataset$site == 'A','pH']
```

```
## [1] 7.4 8.6 7.2
```

```
my.dataset[my.dataset$season == 'Summer',c('site','pH')]
```

```
##   site  pH
## 2    B 6.3
## 3    A 8.6
```

LEER DATA FRAMES DESDE FICHERO

```
100   a1   b1
200   a2   b2
300   a3   b3
400   a4   b4
```

```
Col1,Col2,Col3
100,a1,b1
200,a2,b2
300,a3,b3
```

```
> mydata = read.table("mydata.txt")

> mydata
  V1 V2 V3
1 100 a1 b1
2 200 a2 b2
3 300 a3 b3
4 400 a4 b4
```

```
> mydata = read.csv("mydata.csv")
> mydata
  Col1 Col2 Col3
1  100   a1   b1
2  200   a2   b2
3  300   a3   b3
```

ESCRIBIR DATA.FRAMES EN FICHERO

- write.table(df, "mydata.txt")
- write.csv(df, "mydata.csv")

FUNCIONES AVANZADAS

FUNCIONES DE ALTO NIVEL

- Provienen de la programación funcional, donde una característica es que las funciones son objetos y se pueden asignar a variables o pasar como argumentos
- Aplican una función a cada elemento de un vector o lista. Devuelven un vector o lista. Evitan bucles
- lapply, sapply, apply, tapply, mapply
  - split

FUNCIONES DE ALTO NIVEL: LAPPLY

- lapply, aplica una función a una lista y devuelve una lista

```
> x = list(a=1:2, b=1:3, c=1:4, d=1:5)
> x
$a
[1] 1 2

$b
[1] 1 2 3

$c
[1] 1 2 3 4

$d
[1] 1 2 3 4 5

> lapply(x,mean)
$a
[1] 1.5

$b
[1] 2

$c
[1] 2.5

$d
[1] 3

>
```

FUNCIONES DE ALTO NIVEL: SAPPLY

- sapply: como lapply, pero intenta convertir el resultado a un vector (si todos los elementos de la lista a la salida tienen longitud 1) o a una matriz (si todos los elementos de la lista de salida tienen la misma longitud y tipo)

```
> x = list(a=1:2, b=1:3, c=1:4, d=1:5)
> x
$a
[1] 1 2

$b
[1] 1 2 3

$c
[1] 1 2 3 4

$d
[1] 1 2 3 4 5

> lapply(x,mean)
$a
[1] 1.5

$b
[1] 2

$c
[1] 2.5

$d
[1] 3

> sapply(x,mean)
  a   b   c   d
1.5 2.0 2.5 3.0
> |
```

FUNCIONES DE ALTO NIVEL

- lapply con función sin nombre (anónima)

```
> x <- list(a = matrix(1:4, 2, 2), b = matrix(1:6, 3, 2))
> x
$a
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$b
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

```
> lapply(x, function(elt) elt[,1])
$a
[1] 1 2

$b
[1] 1 2 3
```

FUNCIONES DE ALTO NIVEL: APPLY

- apply: aplica una función a cada fila (margin=1) o a cada columna (margin=2) de una matriz
- Ej: calcula la media de cada columna, o la media de cada fila

```
> x <- matrix(rnorm(200), 20, 10)
> apply(x, 2, mean)
 [1]  0.04868268  0.35743615 -0.09104379
 [4] -0.05381370 -0.16552070 -0.18192493
 [7]  0.10285727  0.36519270  0.14898850
[10]  0.26767260

> apply(x, 1, sum)
 [1] -1.94843314  2.60601195  1.51772391
 [4] -2.80386816  3.73728682 -1.69371360
 [7]  0.02359932  3.91874808 -2.39902859
[10]  0.48685925 -1.77576824 -3.34016277
[13]  4.04101009  0.46515429  1.83687755
[16]  4.36744690  2.21993789  2.60983764
[19] -1.48607630  3.58709251
```

FUNCIONES DE ALTO NIVEL: APPLY

- rowSums = apply(x,1,sum)
- rowMeans = apply(x,1,mean)
- colSums = apply(x,2,sum)
- colMeans = apply(x,2,mean)

FUNCIONES DE ALTO NIVEL: TAPPLY

- tapply: descompone el primer argumento en grupos, y le aplica la función a cada grupo.
  - En el ejemplo, los grupos van en la segunda columna de la matriz. La media de valores que pertenecen al grupo a es 5, la media de los valores que pertenecen al grupo b es 6

```
> valores = 1:10
> grupos = rep(c("a","b"), 5)
> valores
 [1]  1  2  3  4  5  6  7  8  9 10
> grupos
 [1] "a" "b" "a" "b" "a" "b" "a" "b" "a" "b"
> tapply(valores, grupos, mean)
a b
5 6
>
```

```
> tapply(valores, grupos, mean)
a b
5 6
> split(valores, grupos)
$a
[1] 1 3 5 7 9

$b
[1]  2  4  6  8 10

> sapply(split(valores,grupos), mean)
a b
5 6
>
```

SPLIT CON UN DATA.FRAME

```
> library(datasets)
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
```

SPLIT CON UN DATA.FRAME

```
> s <- split(airquality, airquality$Month)
> lapply(s, function(x) colMeans(x[, c("Ozone", "Solar.R", "Wind")]))
$`5`
   Ozone  Solar.R     Wind
      NA       NA 11.62258
$`6`
    Ozone   Solar.R      Wind
       NA 190.16667  10.26667
$`7`
     Ozone    Solar.R       Wind
        NA 216.483871   8.941935
```

SPLIT CON UN DATA.FRAME

```
> sapply(s, function(x) colMeans(x[, c("Ozone", "Solar.R", "Wind")]))
               5         6          7        8        9
Ozone         NA        NA         NA       NA       NA
Solar.R       NA 190.16667 216.483871       NA 167.4333
Wind    11.62258  10.26667   8.941935 8.793548  10.1800

> sapply(s, function(x) colMeans(x[, c("Ozone", "Solar.R", "Wind")],
                                 na.rm = TRUE))
                5           6           7           8           9
Ozone    23.61538    29.44444    59.115385   59.961538   31.44828
Solar.R 181.29630   190.16667   216.483871  171.857143  167.43333
Wind     11.62258    10.26667     8.941935    8.793548   10.18000
```

FUNCIONES DE ALTO NIVEL: MAPPLY

- Es como lapply, pero para funciones con 2 o mas argumentos

```
> mapply(rep, 1:4, 4:1)
[[1]]
[1] 1 1 1 1

[[2]]
[1] 2 2 2

[[3]]
[1] 3 3

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

EJERCICIOS FUNCIONES ALTO NIVEL

1. Crear una lista con 5 vectores numéricos. Escribir una función que ordene cada vector (nota: usar la función sort()).
2. Escribir una función que compute el valor mínimo de cada columna de una matriz de cualquier tamaño (cualquier número de columnas). Probadla con una matriz aleatoria de 10x5

PAQUETE PLYR: SPLIT/APPLY/COMBINE

- install.packages("plyr") #Sólo hay que hacerlo la primera vez
- library(plyr)
- ddply es el equivalente a tapply, pero para data.frames
- Sea la llamada a la función ddply
  - ddply(dataframe, c("var1", "var2), mifuncion)
- Lo que hace esta función es:
  1. Split: recorrer el dataframe para todas las posibles combinaciones de los valores de las variables "var1" y "var2". Eso divide (split) el dataframe en varios subdataframes.
  2. Apply: Para cada subdataframe se aplica "mifuncion", la cual también devuelve un dataframe.
  3. Combine: El resultado final es la concatenación de todos los dataframes.

DDPLY (ES EL TAPPLY PARA DATA.FRAMES)

```
> ddply(airquality, "Month", function(x) {colMeans(x[,c("Ozone", "Solar.R", "Wind")], na.rm=TRUE)})
  Month    Ozone  Solar.R     Wind
1     5 23.61538 181.2963 11.622581
2     6 29.44444 190.1667 10.266667
3     7 59.11538 216.4839  8.941935
4     8 59.96154 171.8571  8.793548
5     9 31.44828 167.4333 10.180000
>
```

DDPLY (ES EL TAPPLY PARA DATA.FRAMES)

- Ahora supongamos que queremos calcular esas mismas medias para cada mes, pero diferenciando la primera mitad del mes (dias 1-15) de la segunda quincena (días después del 15)
- Primero añadimos una columna al data.frame indicando la quincena (1 o 2), así:

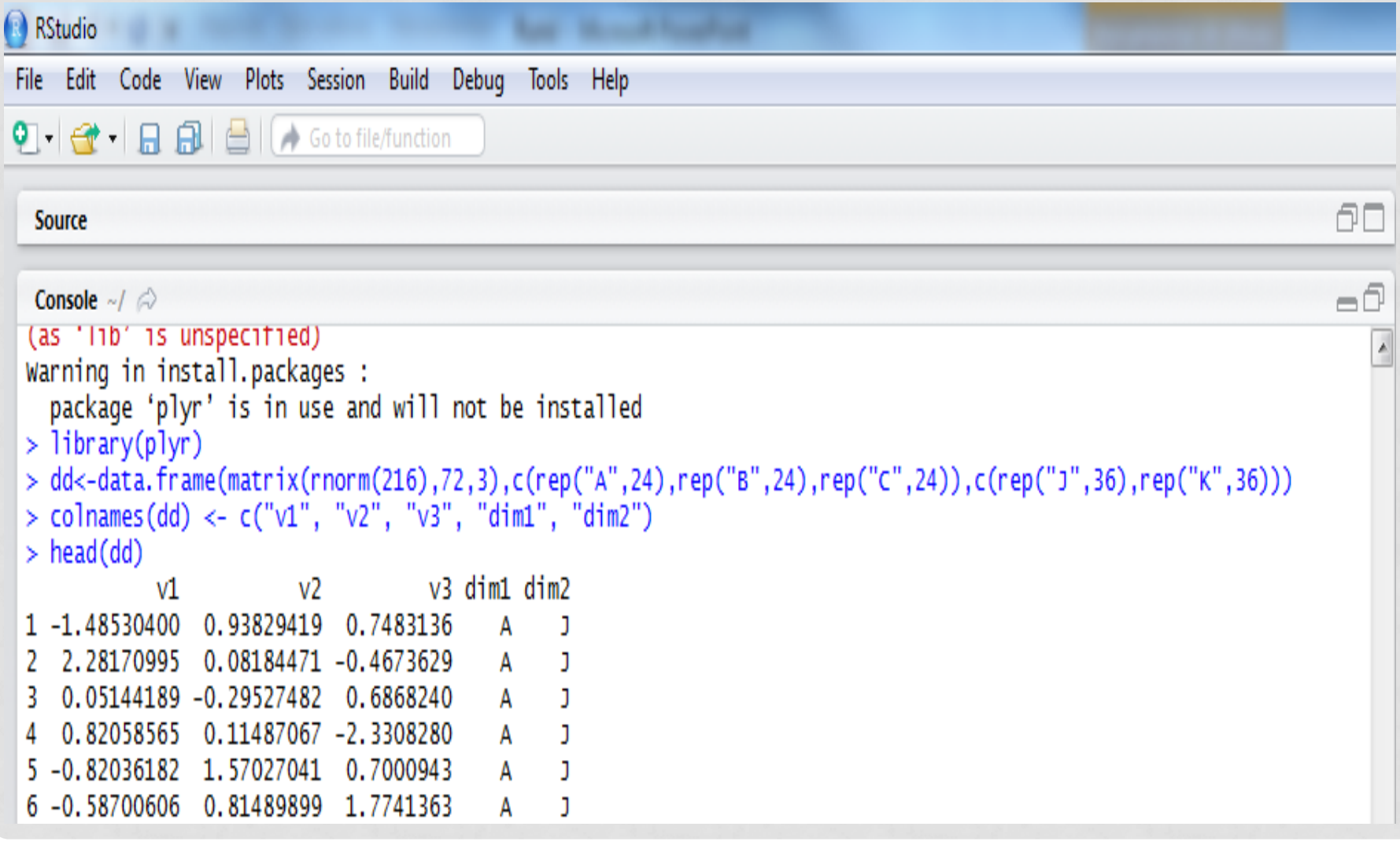
```
> airquality$quincena = ifelse(airquality$Day<=15, 1,2)
>
> head(airquality)
  Ozone Solar.R Wind Temp Month Day quincena
1    41     190  7.4   67     5   1         1
2    36     118  8.0   72     5   2         1
3    12     149 12.6   74     5   3         1
4    18     313 11.5   62     5   4         1
5    NA      NA 14.3   56     5   5         1
6    28      NA 14.9   66     5   6         1
>
```

DDPLY (ES EL TAPPLY PARA DATA.FRAMES)

- Después usamos ddply con "Month" y "quincena"

```
> airquality$quincena = ifelse(airquality$Day<=15, 1,2)
>
> head(airquality)
  Ozone Solar.R Wind Temp Month Day quincena
1    41     190  7.4   67     5   1        1
2    36     118  8.0   72     5   2        1
3    12     149 12.6   74     5   3        1
4    18     313 11.5   62     5   4        1
5    NA      NA 14.3   56     5   5        1
6    28      NA 14.9   66     5   6        1
> ddply(airquality, c("Month", "quincena"), function(x) {colMeans(x[,c("Ozone", "Solar.R", "Wind")], na.rm=TRUE)})
   Month quincena    Ozone  Solar.R      Wind
1      5        1 19.30769 188.8333 11.313333
2      5        2 27.92308 175.2667 11.912500
3      6        1 40.50000 254.0000 10.786667
4      6        2 20.60000 126.3333  9.746667
5      7        1 60.00000 216.6667  9.360000
6      7        2 58.35714 216.3125  8.550000
7      8        1 58.41667 160.0000  8.906667
8      8        2 61.28571 180.7500  8.687500
9      9        1 41.33333 183.5333  9.546667
10     9        2 20.85714 151.3333 10.813333
>
```

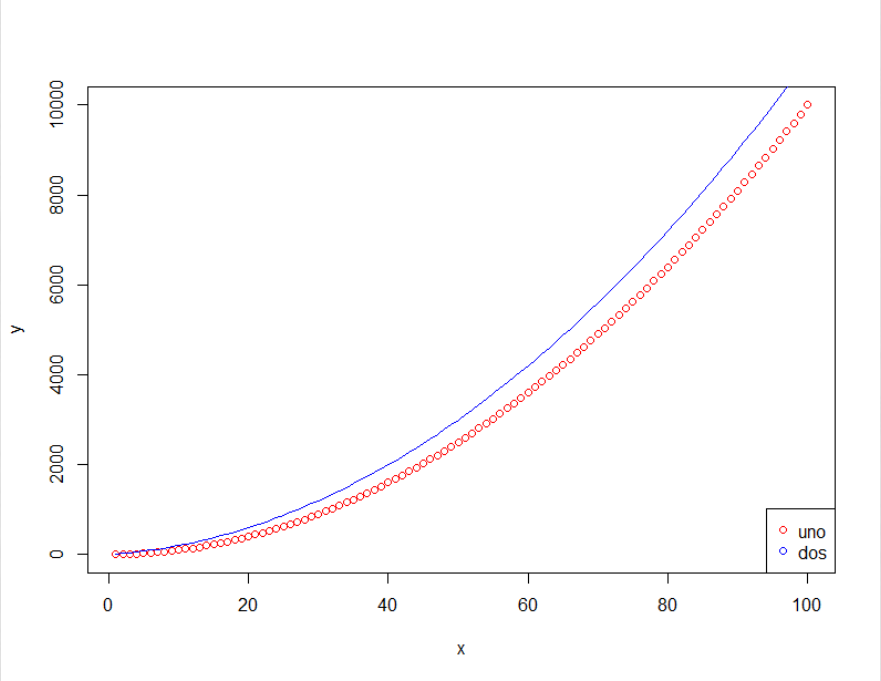
PAQUETE PLYR: SPLIT/APPLY/COMBINE



```
> dd<-data.frame(matrix(rnorm(216),72,3),c(rep("A",24),rep("B",24),rep("C",24)),c(rep("J",36),rep("K",36)))
> colnames(dd) <- c("v1", "v2", "v3", "dim1", "dim2")
> head(dd)
         v1         v2          v3 dim1 dim2
1 -1.8727390 -0.5774326 -0.86802106    A    J
2 -0.3591283  0.5634015  0.04566798    A    J
3  0.2425982 -1.9184006 -0.41654336    A    J
4 -1.6853468  0.3175484  0.34023245    A    J
5  1.0686203 -0.3925017  0.06415597    A    J
6  1.3402615  0.6850056  0.11385861    A    J
> ddply(dd, c("dim1","dim2"), function(df) {mean(df$v1)})
  dim1 dim2          V1
1    A    J -0.198491433
2    B    J  0.260122492
3    B    K  0.104016858
4    C    K  0.005305193
> ddply(dd, c("dim1","dim2"), function(df) {c(mediav1=mean(df$v1))})
  dim1 dim2      mediav1
1    A    J -0.198491433
2    B    J  0.260122492
3    B    K  0.104016858
4    C    K  0.005305193
> ddply(dd, c("dim1","dim2"), function(df)c(mv1=mean(df$v1),mv2=mean(df$v2),mv3=mean(df$v3),
+                                           sdv1=sd(df$v1),sdv2=sd(df$v2),sdv3=sd(df$v3)))
  dim1 dim2          mv1          mv2         mv3      sdv1      sdv2      sdv3
1    A    J -0.198491433 -0.002131354  0.04491923 1.1931246 0.8396458 0.8714301
2    B    J  0.260122492  0.072079229  0.07014289 0.8383656 0.8546995 0.9620618
3    B    K  0.104016858 -0.190979234  0.02137120 0.9861965 0.9528182 1.1086170
4    C    K  0.005305193  0.199573469 -0.13454428 1.0433404 0.8596553 1.1198714
> 
```

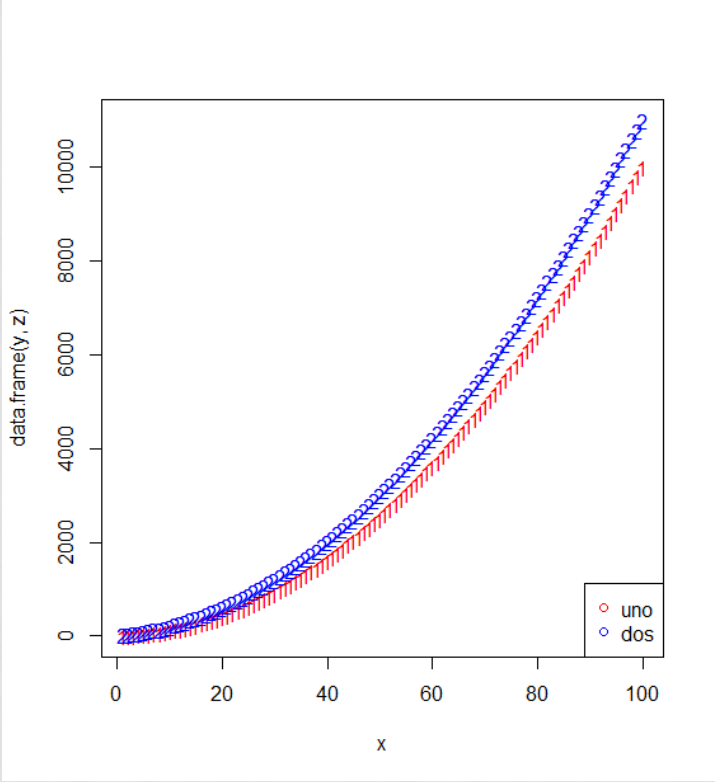
PLOTS BÁSICOS: CON BASE

- x=1:100
- y=x^2
- plot(x,y, col="red")
- z = x^2+10*x
- points(x,z, col="blue", type="l")
- legend("bottomright", c("uno", "dos"), col=c("red", "blue"), pch=1)



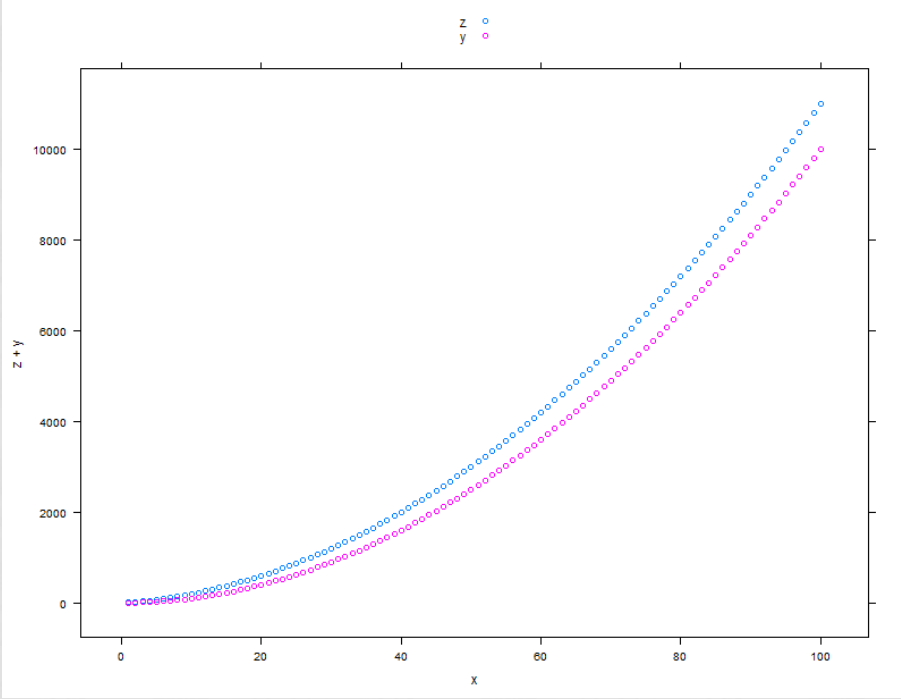
PLOTS BÁSICOS: CON BASE

- matplot(x,data.frame(y,z), col=c("red","blue"))
- legend("bottomright", c("uno", "dos"), col=c("red", "blue"), pch=1)
-



PLOTS BÁSICOS: CON LATTICE

- install.packages("lattice")
- library(lattice)
- xyplot(z+y~x, data.frame(x,y,z), auto.key=TRUE)



PLOTS BÁSICOS: CON GGPLOT2

- install.packages("ggplot2")
- library(ggplot2)
- misdatos = rbind(data.frame(x,valor=y,serie="una"), data.frame(x, valor=z, serie="dos"))
- qplot(x,valor,data=misdatos,colour=serie)