

Curso integral para el análisis de datos de Genómica y Transcriptómica

R

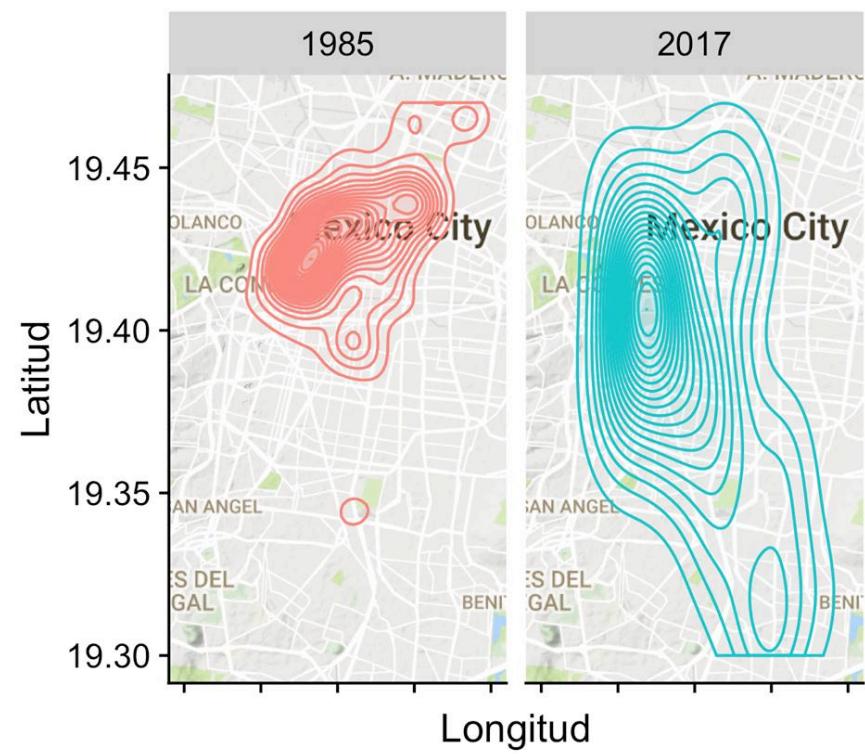
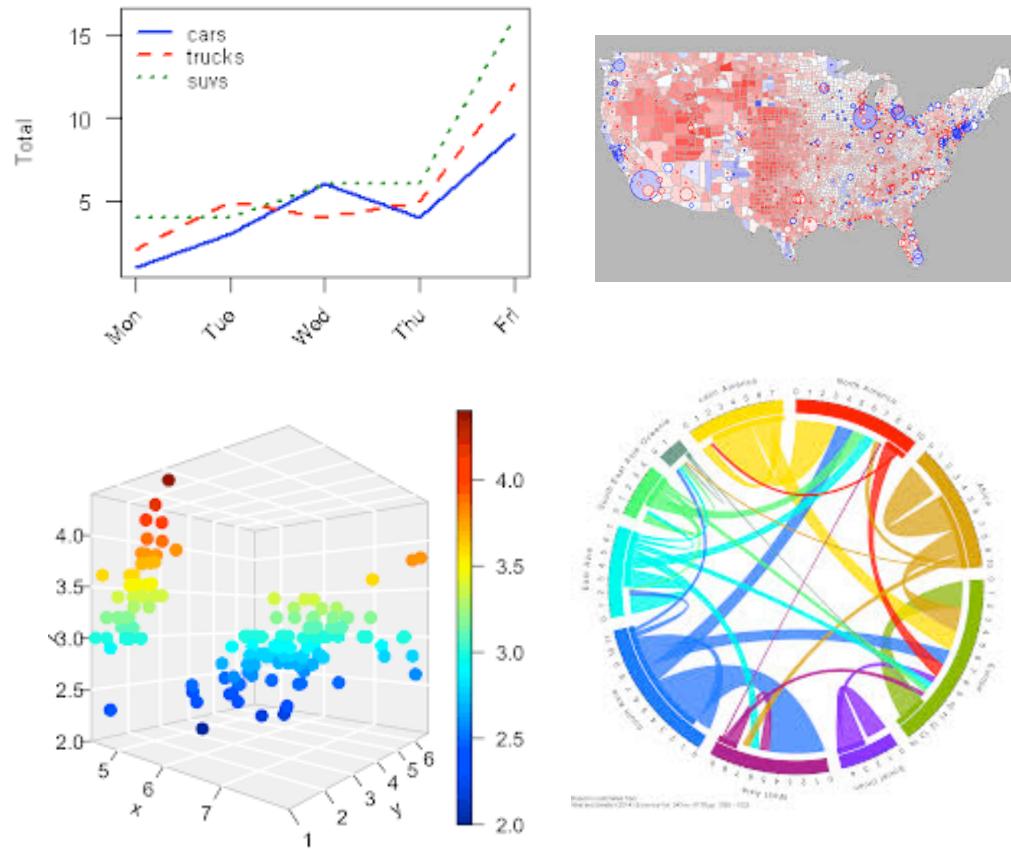
Sesión 4
18-01-18

Daniela E. Ledezma Tejeida



¿Qué es R?

R es un entorno y un lenguaje de programación utilizado para manejo de datos, procesamiento estadístico y visualización



¿Qué es R?



S – 1976 – John Chambers, Rick Becker y Allan Wilks
Lenguaje de programación para estadística
“to turn ideas into software, quickly and faithfully”

R – 1995/2000 – Ross Ihaka y Robert Gentleman
Implementación de S con ciertas modificaciones.
Ambos lenguajes son compatibles entre sí.



¿Por qué usar R?

- Facilidad para el manejo y almacenamiento de datos.
- Operadores para cálculo con vectores y matrices.
- Colección extensa e integrada de herramientas para el análisis estadístico de datos.
- Facilidades gráficas.
- Lenguaje de programación de propósito específico.
- Comunidad de desarrolladores y usuarios que constantemente cooperan para la mejora y desarrollo de herramientas nuevas que amplían las capacidades de R.

R en línea de comandos

```
[vjmenez@zazil:~$ R
```

```
R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"  
Copyright (C) 2013 The R Foundation for Statistical Computing  
Platform: x86_64-pc-linux-gnu (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.
```

```
Natural language support but running in an English locale
```

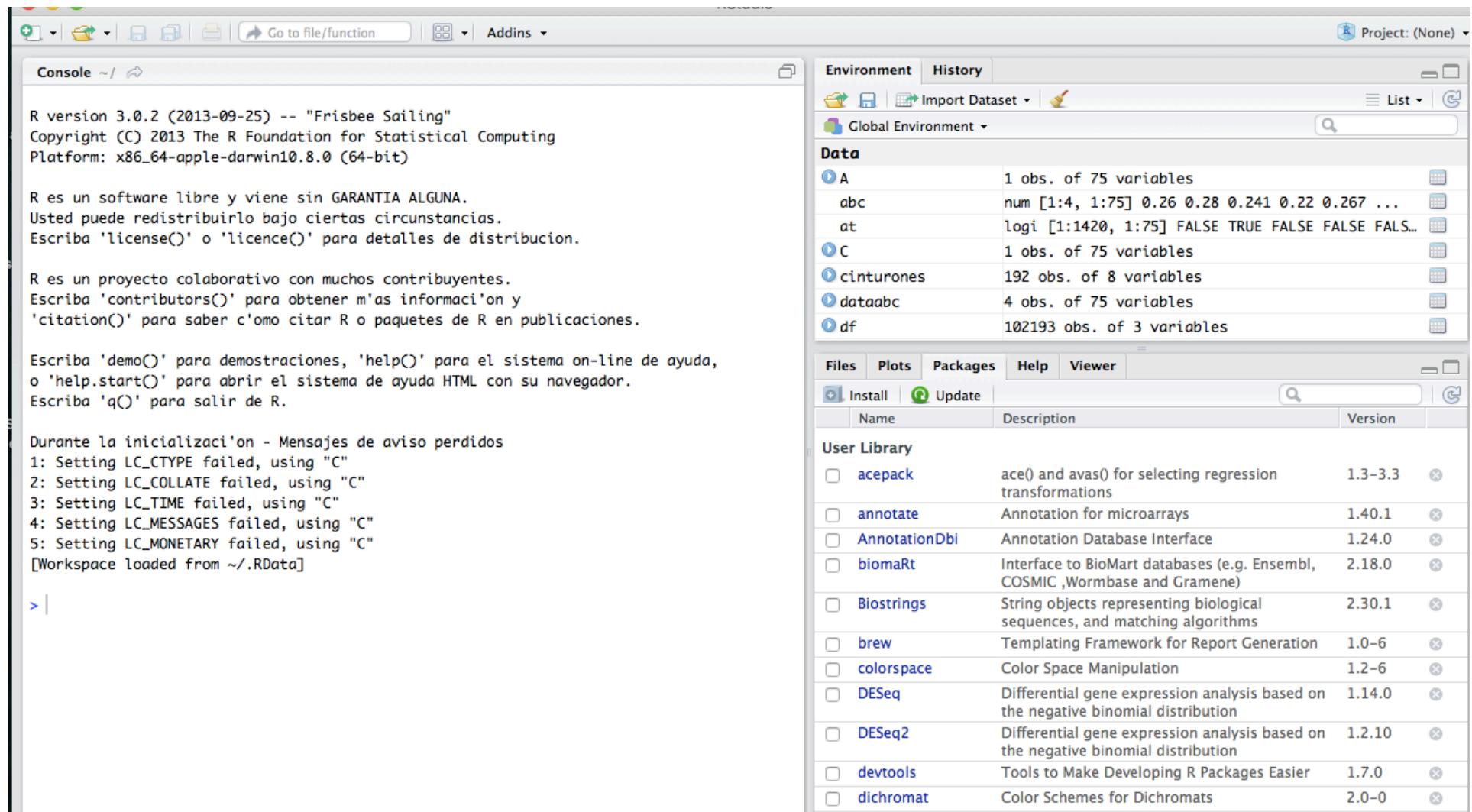
```
R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to cite R or R packages in publications.
```

```
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.
```

```
[Previously saved workspace restored]
```

```
> █
```

Rstudio



The screenshot shows the RStudio interface with the following panes:

- Console** pane (left):


```
R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin10.8.0 (64-bit)

R es un software libre y viene sin GARANTIA ALGUNA.
Usted puede redistribuirlo bajo ciertas circunstancias.
Escriba 'license()' o 'licence()' para detalles de distribucion.

R es un proyecto colaborativo con muchos contribuyentes.
Escriba 'contributors()' para obtener m'as informaci'on y
'citation()' para saber c'omo citar R o paquetes de R en publicaciones.

Escriba 'demo()' para demostraciones, 'help()' para el sistema on-line de ayuda,
o 'help.start()' para abrir el sistema de ayuda HTML con su navegador.
Escriba 'q()' para salir de R.

Durante la inicializaci'on - Mensajes de aviso perdidos
1: Setting LC_CTYPE failed, using "C"
2: Setting LC_COLLATE failed, using "C"
3: Setting LC_TIME failed, using "C"
4: Setting LC_MESSAGES failed, using "C"
5: Setting LC_MONETARY failed, using "C"
[Workspace loaded from ~/.RData]
```
- Environment** pane (top right):
 - Global Environment: Shows objects A, at, C, cinturones, dataabc, and df.
- Packages** pane (bottom right):
 - User Library: Shows installed packages: acepack, annotate, AnnotationDbi, biomaRt, Biostrings, brew, colorspace, DESeq, DESeq2, devtools, and dichromat.

Variables y constantes

Las variables son localidades de memoria a las que les asignamos un nombre para almacenar un valor. El valor que contienen puede cambiar en el transcurso del programa:

A

Una variable se asigna con el operador “`<-`”. E.g.

3

`A <- 3`

Las constantes no cambian en el transcurso del programa. Pueden ser numéricas o de caracteres. E.g.

`pi = 3.141593`

Variables y constantes

Los nombres de una variable:

- No deben comenzar con un número.
- No deben contener espacios en blanco.
- No deben contener símbolos excepto “_” o “.” (guión bajo o punto).
- Pueden tener mayúsculas y minúsculas.

E.g.

Variable	
variable_32	suma.total
	x_1

Práctica 1

¿Cuáles de estas definiciones son correctas?

- 1) Abracadabra <- 8
- 2) 8 <- "Hola"
- 3) LogNatural2 <- 25
- 4) 2log <- 10

Si te queda mas tiempo:

`log2 <- 3` ¿Por qué es o no correcta esta definición?

Tipos de asignación

Para asignar un valor a una variable se utiliza
“<-” o “=”

```
> x <- 3
> Y = 15
```

R distingue entre mayúsculas y minúsculas

```
> A <- 3
> a <- 2
```

Cada asignación es un comando individual para R. Los comandos pueden estar en líneas separadas o en la misma línea, separados por “;”

```
> a <- 6;   A <- A + 11
```

También puede escribirse un sólo comando en varias líneas

```
> X <-
+     pi
```

```
> b <- "Esto es
+     otro ejemplo"
```

El “+” indica que el comando no ha sido completado

Tipos de datos

Cada dato guardado en R tiene un tipo. Puede ser:

- Carácter. E.g. “palabras”
- Numérico. Puede ser real o decimal. E.g. 5, 5.3
- Entero. E.g. 2L (la “L” le indica a R que se trata de un entero)
- Lógico. Falso o verdadero.
- Complejo. Números con partes reales e imaginarias. E.g. 1+4i

Para conocer el tipo de datos que contiene un objeto se utiliza:

typeof()

Datos tipo carácter

Los datos tipo carácter deben ser delimitados con comillas simples o dobles.

```
> geneName='CRP'  
> Name="C-reactive protein"  
> typeof(geneName)  
[1] "character"  
> typeof(Name)  
[1] "character"
```

Práctica 2

¿Por qué las siguientes asignaciones causan error?

```
2A <- 76
Name=AraC
Fold change = 2.7
Suma <- A + Cultivo
```

Funciones

Una de las propiedades de R es que permite llevar a cabo funciones predeterminadas.

E.g.

```
> sqrt(25)
```

¿Qué crees que haga la función “sqrt()”?

Para usar las funciones debes conocer su sintaxis (comando exacto, argumentos de la función, etc.). Para interpretar el resultado ayuda saber qué hace la función.

¿Cómo saber más sobre una función?

```
help([función]) ó ?[función]
```

E.g.

```
> help(sqrt)  
> ?sqrt
```

Ayuda en línea

Para consultar la ayuda de R en HTML se usa la función:

```
> help.start()
```



Manuals

[An Introduction to R](#)

[Writing R Extensions](#)

[R Data Import/Export](#)

[The R Language Definition](#)

[R Installation and Administration](#)

[R Internals](#)

Reference

[Packages](#)

[Search Engine & Keywords](#)

Miscellaneous Material

[About R](#)

[Authors](#)

[Resources](#)

[License](#)

[Frequently Asked Questions](#)

[Thanks](#)

[NEWS](#)

[User Manuals](#)

[Technical papers](#)

Ayuda en línea

Para buscar ayuda relacionada a un término:

```
> help.search("clustering")
```

Search Results 

The search string was "clustering"

Vignettes:

[graph::clusterGraph](#) clusterGraph and distGraph [PDF](#) [source](#) [R code](#)

Help pages:

Hmisc::varclus	Variable Clustering
graph::clusteringCoefficient	Clustering coefficient of a graph
vegan::reorder.hclust	Reorder a Hierarchical Clustering Tree
MotifV::motifDistances	Clustering PWMs Computation
cluster::agnes	Agglomerative Nesting (Hierarchical Clustering)
cluster::bannerplot	Plot Banner (of Hierarchical Clustering)
cluster::clara	Clustering Large Applications
cluster::clara.object	Clustering Large Applications (CLARA) Object
cluster::diana	Dlvisive ANAlysis Clustering
cluster::fanny	Fuzzy Analysis Clustering
cluster::mona	MONothetic Analysis Clustering of Binary Variables
cluster::plot.agnes	Plots of an Agglomerative Hierarchical Clustering

Otros comandos para pedir ayuda

Buscar ejemplos de cómo se utiliza una función:

```
> example([función])
```

```
> example(seq)

seq> seq(0, 1, length.out = 11)
[1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

seq> seq(stats::rnorm(20)) # effectively 'along'
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

seq> seq(1, 9, by = 2)      # matches 'end'
[1] 1 3 5 7 9

seq> seq(1, 9, by = pi)    # stays below 'end'
[1] 1.000000 4.141593 7.283185

seq> seq(1, 6, by = 3)
[1] 1 4
```

Otros comandos para pedir ayuda

Información sobre los argumentos que recibe una función:

```
> args([función])
```

```
> args(read.table)
function (file, header = FALSE, sep = "", quote = "\"\"", dec = ".",
  numerals = c("allow.loss", "warn.loss", "no.loss"), row.names,
  col.names, as.is = !stringsAsFactors, na.strings = "NA",
  colClasses = NA, nrows = -1, skip = 0, check.names = TRUE,
  fill = !blank.lines.skip, strip.white = FALSE, blank.lines.skip = TRUE,
  comment.char = "#", allowEscapes = FALSE, flush = FALSE,
  stringsAsFactors = default.stringsAsFactors(), fileEncoding = "",
  encoding = "unknown", text, skipNul = FALSE)
```

Cuando no recuerdas la sintaxis específica de una función:

```
> apropos([texto])
```

```
> apropos("Seq")
[1] "seq"          "seq_along"    ... "seq_len"      "seq.Date"     "seq.default"
[6] "seq.int"      "seq.POSIXt"   "sequence"
```

Práctica 3

- 1) Utiliza la función **help** para saber qué hace la función **c()**.
- 2) Busca qué hace la función **paste**.

Funciones Numéricas

Función	Comando
Raíz cuadrada	<code>sqrt()</code>
Máximo	<code>max()</code>
Media	<code>mean()</code>
Desviación Estándar	<code>sd()</code>
Logaritmo	<code>log()</code>

Práctica 4

Calcula utilizando el comando correcto:

- 1) Raíz cuadrada de 9.
- 2) Raíz cuadrada de -2
- 3) Máximo de `c(10,1,5,62,3,75,8,75)`
- 4) Mínimo de `c(10,1,5,62,3,75,8,75)`

Operadores Relacionales

Se utilizan para indicar relaciones entre dos valores (variables o constantes). Producen un resultado de falso o verdadero.

Operador	Significado	Ejemplo	Resultado de ejemplo
<code>==</code>	Igual	<code>a==b</code>	<code>5 == 3 # False</code>
<code><</code>	Menor que	<code>a < b</code>	<code>5 < 3 # False</code>
<code><=</code>	Menor o igual que	<code>a<=b</code>	<code>5 <= 5 # True</code>
<code>></code>	Mayor	<code>a>b</code>	<code>5 > 3 # True</code>
<code>>=</code>	Mayor o Igual que	<code>a>=b</code>	<code>5 <= 5 # True</code>
<code>!=</code>	Distinto	<code>a!=b</code>	<code>5 != 3 # True</code>

Por ejemplo:

```
> 3 < 2
[1] FALSE

> 5 == 5
[1] TRUE
```

```
> "Hola" != "hola"
[1] TRUE
```

Estructuras de datos

Una de las ventajas de R es que permite guardar no sólo variables con un tipo de dato, sino estructuras de datos complejas. Las más utilizadas son:

- Vectores
- Listas
- Matrices
- *Data frames*
- *Factors*

Vectores

Un vector en R es, esencialmente, una lista ordenada de datos del mismo tipo.

Por *default* los vectores son de tipo lógico:

```
> un_vector <- vector(length = 3)
> typeof(un_vector)
[1] "logical"
> un_vector
[1] FALSE FALSE FALSE
```

Sin embargo, es posible especificar el tipo de dato:

```
> otro_vector <- vector(mode='character', length=5)
> typeof(otro_vector)
[1] "character"
> otro_vector
[1] "" "" "" "" "
```

Vectores

Otra manera de generar vectores es con la función **c()** que proviene de *combine*.

```
> primos <- c(1,3,5,7,11,13,17,19)
> primos
[1] 1 3 5 7 11 13 17 19
> vocales <- c("a","e","i","o","u")
> vocales
[1] "a" "e" "i" "o" "u"
> logicos <- c(FALSE,TRUE,FALSE,TRUE,TRUE)
> logicos
[1] FALSE  TRUE FALSE  TRUE  TRUE
```

Clases de vectores

El vector es el tipo de dato básico de R, todas las estructuras de datos, desde una variable simple hasta la más compleja pueden tratarse como vectores estructurados de formas distintas.

Vimos que **typeof()** regresa el tipo de datos que contiene un vector, sin embargo, los vectores como objeto individual pueden pertenecer a distintas clases:

- numéricos
- carácter
- lógicos
- complejos

Los tipos de dato y las clases de vectores parecen muy similares, sin embargo, en estructuras de datos más complejas la diferencia será más informativa.

Tamaño de vectores

Para conocer la longitud de un vector se usa la función **length()**

```
> primos <- c(2,3,5,7,11,13)
> length(primos)
[1] 6
> palabras <- c("gato", "vaso", "noche", "malteada")
> length(palabras)
[1] 4
```

Práctica 5

- 1) Crea un vector numérico, uno de caracteres y uno lógico.
- 2) Investiga la clase de cada vector.
- 3) Investiga el tipo de dato de cada vector.
- 4) Investiga la longitud de cada vector.

Clases de vectores

Los vectores pueden ser coercionados a cambiar de clase usando la función **as.[nueva clase]()**. Por ejemplo:

```
> x<-c(0,1,2,3,4,5,6)
> class(x)
[1] "integer"
> as.numeric(x)
[1] 0 1 2 3 4 5 6
> as.logical(x)
[1] FALSE  TRUE   TRUE   TRUE   TRUE   TRUE   TRUE
> as.character(x)
[1] "0"  "1"  "2"  "3"  "4"  "5"  "6"
```

Clases de vectores

No todos los cambios son posibles. Cambios no válidos introducen NA's al vector:

```
> x<-c("a","b","c")
> as.numeric(x)
[1] NA NA NA
Warning message:
NAs introduced by coercion
> as.logical(x)
[1] NA NA NA
> as.complex(x)
[1] NA NA NA
Warning message:
NAs introduced by coercion
```

Clases de vectores

La regla de coerción es:

logical -> integer -> numeric -> complex -> character

```
> x<-c(FALSE,TRUE,TRUE,TRUE)
> y<-as.integer(x)
> y
[1] 0 1 1 1
> z<-as.numeric(y)
> z
[1] 0 1 1 1
> c<-as.complex(z)
> c
[1] 0+0i 1+0i 1+0i 1+0i
> car<-as.character(c)
> car
[1] "0+0i" "1+0i" "1+0i" "1+0i"
```

Práctica 6

- 1) Coerciona los vectores de la práctica 5 (numérico, caracteres y lógico) para que todos sean de la misma clase.
- 2) Verifica tu respuesta usando **class()**
- 3) ¿Qué clase de vector se creará con la siguiente asignación?

```
N <- c(5,"cinco")
```

La regla de coerción es:

logical -> integer -> numeric -> complex -> character

Operaciones con vectores

Los vectores numéricos pueden participar en operaciones

Generemos un vector numérico que contenga tamaños de
genomas

```
genomeSize<-c(3000000000,3000000000,135600000,97000000,12100000)
```

Al aplicar un operador a un vector, todos los elementos del
vector son sujetos al operador.

```
> genomeSize*100
[1] 3.000e+11 3.000e+11 1.356e+10 9.700e+09 1.210e+09
> genomeSize/3
[1] 1000000000 1000000000      45200000      32333333      4033333
```

Operaciones con vectores

También es posible aplicar funciones a los vectores

```
> mean(genomeSize)
[1] 1248940000
> max(genomeSize)
[1] 3e+09
> min(genomeSize)
[1] 12100000
> sum(genomeSize)
[1] 6244700000
```

Nombres de vectores

Muchas veces es importante recordar atributos de los valores de un vector. En nuestro ejemplo anterior, sería útil saber a qué genoma pertenece cada tamaño.

```
genomeSize<-c(3000000000,3000000000,135600000,97000000,12100000)
```

R permite asignar nombres a los elementos de un vector con la función **names()**

Es necesario definir previamente un vector con los nombres que se quieren asignar

```
organism<-c("HomoSapiens","Mouse","Fruit Fly","Roundworm","Yeast")
```

Nombres de vectores

Posteriormente se asigna el vector de nombres al los nombres del vector original

```
> names(genomeSize)<-organism
```

```
> genomeSize
HomoSapiens      Mouse     Fruit Fly    Roundworm      Yeast
 3.000e+09      3.000e+09   1.356e+08   9.700e+07   1.210e+07
```

Detalles que considerar:

- El vector de nombres debe tener la misma longitud que el vector original
- **names()** hace que la asignación sea sobre le atributo nombres y no sobre los datos del vector

Operaciones con elementos de vectores

Además de operaciones sobre todos los elementos del vector pueden editarse elementos individuales utilizando el índice del vector con []

```
> primos <- c(2,3,5,7,11,13)
> primos
[1] 2 3 5 7 11 13 + Send!
```

Cada elemento del vector tiene una posición definida, esto puede observarse al imprimir el vector. El [1] al principio de la línea indica que es la primera posición.

Para acceder a un elemento específico es necesario saber su posición en el vector e indicarla usando []

```
> primos[3]
[1] 5
```

Operaciones con elementos de vectores

Una vez que se especificó el elemento, se pueden efectuar operaciones sobre él

```
> primos[3]*4  
[1] 20
```

Sin embargo, definir la operación no afecta al vector original

```
> primos[3]*4  
[1] 20  
> primos  
[1] 2 3 5 7 11 13
```

Para editar el vector original es necesario hacer una asignación

```
> primos  
[1] 2 3 5 7 11 13  
> primos[3] <- primos[3]*4  
> primos  
[1] 2 3 20 7 11 13
```

Operaciones con elementos de vectores

De la misma forma se puede editar sólo un nombre del vector

```
> names(genomeSize)[1] <- "Human"
```

También es posible extraer secciones de elementos usando ":"

```
> genomeSize[3:5]
Fruit Fly Roundworm      Yeast
135600000  97000000  12100000
> genomeSize[1,5]
Error in genomeSize[1,5] : incorrect number of
dimensions
> genomeSize[c(1,5)]
      Human      Yeast
3.00e+09 1.21e+07
> genomeSize[-c(2:4)]
      Human      Yeast
3.00e+09 1.21e+07
```

Agregar elementos a un vector

¿Cómo agregar otro genoma al vector?

Utilizando la función **append()**, especificando el vector y el valor a agregar

```
> genomeSize <- append(genomeSize,175000)
> genomeSize
Human      Mouse   Fruit Fly Roundworm       Yeast
 3.000e+09  3.000e+09  1.356e+08  9.700e+07  1.210e+07
1.750e+05
```

¿Cómo agregar el nombre al nuevo elemento?

```
> genomeSize
Human      Mouse   Fruit Fly Roundworm       Yeast
 3.000e+09  3.000e+09  1.356e+08  9.700e+07  1.210e+07  1.750e+05
> names(genomeSize)[6] <- "Frog"
> genomeSize
Human      Mouse   Fruit Fly Roundworm       Yeast       Frog
 3.000e+09  3.000e+09  1.356e+08  9.700e+07  1.210e+07  1.750e+05
```

Práctica 7

- 1) Genera un vector con las edades de los miembros de tu familia
- 2) Agrega los “roles” de cada miembro usando la función names (madre, padre, hermano, etc)
- 3) Agrega las edades de dos amigos. Especifica su rol de “amigos”
- 4) Calcula el promedio, edad mínima y edad máxima del vector

Listas

En ocasiones es necesario crear un vector con distintos tipos de datos. Sabemos que los vectores automáticamente coercionan su datos para que sean del mismo tipo.

```
> x<-c("hola",5,6+2i,TRUE)
> class(x)
[1] "character"
> x
[1] "hola"    "5"      "6+2i"   "TRUE"
> x<-c(6+2i,TRUE,5)
> class(x)
[1] "complex"
> x
[1] 6+2i 1+0i 5+0i
```

Listas

Para resolver este problema existen las listas, para R, **lists**.

Las listas esencialmente son vectores que permiten distintos tipos de datos.

```
> x<- list("hola", FALSE, 5, 3 + 6i)
> x
[[1]]
[1] "hola"

[[2]]
[1] FALSE

[[3]]
[1] 5

[[4]]
[1] 3+6i
```

Listas

Internamente, las listas están conformadas por vectores de distintas clases y pueden manipularse como tales.

```
> x<- list(c("hola","adios"),c(TRUE,FALSE), 5, 3+ 6i)
> x
[[1]] ← Índice de la lista
[1] "hola"  "adios"
[[2]] ↑ Índice del vector
[1] TRUE FALSE

[[3]]
[1] 5

[[4]]
[1] 3+6i
```

Listas

Corroboremos las clases de los vectores

Los índices se indican en dobles corchetes porque hacen referencia al índice de la lista, no del vector.

```
> class(x)
[1] "list"
> class(x[[1]])
[1] "character"
> class(x[[2]])
[1] "logical"
> class(x[[3]])
[1] "numeric"
> class(x[[4]])
[1] "complex"
```

Para acceder a elementos individuales de la lista se utilizan ambos índices:

```
> x<- list(c("hola","adios"),c(TRUE, FALSE), 5, 3+ 6i)
> (x[[1]])[1]
[1] "hola"
> (x[[2]])[2]
[1] FALSE
```

Matrices

Una matriz es un vector de 2 dimensiones. Como los vectores, sólo permiten un tipo de dato.

Para construirlas basta con cambiar las dimensiones de un vector usando **dim()**

```
> m <- c(1:6)
> class(m)
[1] "integer"
> dim(m) <- c(2,3)
> class(m)
[1] "matrix"
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

En R las dimensiones siempre se expresan como **[fila, columna]**

Observa lo que sucede si utilizas
> dim(m) <- c(3,2)
para crear la matriz

Matrices

Otra forma de construir una matriz es con la función **matrix()**

```
> m <- matrix(1:6,nrow=2,ncol=3)
> class(m)
[1] "matrix"
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Para referirse a un elemento de la matriz se utiliza [] indicando la posición del elemento, ahora en dos dimensiones.

Siempre en formato [fila, columna]

```
> m[2,3]
[1] 6
> m[1,2]
[1] 3
```

Matrices

Las matrices permiten (en general) las mismas funciones que los vectores.

Se pueden nombrar sus filas y columnas usando **rownames()** y **colnames()**

```
> rownames(m) <- c("mouse1","mouse2")
> colnames(m) <- c("week1","week2")
> m
      week1 week2 week3
mouse1     1     3     5
mouse2     2     4     6
```

Demos un nombre más informativo a nuestra matriz

```
mice_weight <- m
```

Matrices

También se pueden utilizar las funciones **mean()**, **max()**, **min()**, **sum()** sobre todos los elementos de la matriz

```
> mean(mice_weight)
[1] 3.5
> max(mice_weight)
[1] 6
> min(mice_weight)
[1] 1
> sum(mice_weight)
[1] 21
```

¿Cómo obtener el promedio de un ratón o de una semana (filas o renglones individuales)?

```
> mean(mice_weight[1,])
[1] 3
> mean(mice_weight[,1])
[1] 1.5
```

Matrices

Se puede acceder a varios elementos de la matriz usando en el índice:

c() – para elementos específicos

: - para elementos consecutivos

-c() – para esconder elementos de la matriz

```
> mice_weight [1,2:3]
week2 week3
      3      5
> mice_weight [,c(1,3)]
      week1 week3
mouse1      1      5
mouse2      2      6
> mice_weight[, -c(1,3)]
mouse1 mouse2
      3      4
```

Al usar **-c()** los elementos no desaparecen del vector, sólo no son mostrados

¿Cómo harías para obtener una matriz sin ellos?

Práctica 8

- 1) Crea una matriz con las edades de 3 compañeros. El nombre de las filas debe ser el nombre de los compañeros, las columnas tendrán la edad que tenían en 1985, 2000 y 2018.
- 2) ¿Cuál es el promedio de edades en el año 2000?
- 3) ¿Cuál es la suma de las edades en el año 2017?
- 4) ¿Qué comando utilizarías para obtener tu edad en el año 2000?

Matrices

Para agregar nuevas filas a la matriz se utiliza la función **rbind()**

```
> mouse3 <- c(3,5,7)
> mice_weight <- rbind(mice_weight,mouse3)
> mice_weight
      week1 week2 week3
mouse1      5     3     5
mouse2      2     4     6
mouse3      3     5     7
```

Para agregar nuevas columnas se utiliza **cbind()**

```
> week4 <- c(6,8,9)
> mice_weight <- cbind(mice_weight,week4)
> mice_weight
      week1 week2 week3 week4
mouse1      5     3     5     6
mouse2      2     4     6     8
mouse3      3     5     7     9
```

¡Cuidá las dimensiones de los vectores que agregues!

Matrices

Para un análisis rápido de la matriz existe **summary()**

```
> mice_weight
      week1 week2 week3 week4
mouse1     5     3     5     6
mouse2     2     4     6     8
mouse3     3     5     7     9
> summary(mice_weight)
      week1          week2          week3          week4
Min.   :2.000   Min.   :3.0   Min.   :5.0   Min.   :6.000
1st Qu.:2.500 1st Qu.:3.5   1st Qu.:5.5   1st Qu.:7.000
Median :3.000  Median :4.0   Median :6.0   Median :8.000
Mean   :3.333  Mean   :4.0   Mean   :6.0   Mean   :7.667
3rd Qu.:4.000 3rd Qu.:4.5   3rd Qu.:6.5   3rd Qu.:8.500
Max.   :5.000  Max.   :5.0   Max.   :7.0   Max.   :9.000
```

Data frames

Todas las estructuras de datos vistas hasta ahora son muy útiles, sin embargo, en el día a día los datos más utilizados se encuentran en forma de tablas, con filas y columna, y distintos tipos de datos.

Para cubrir esta necesidad existen los ***data frames***

Internamente, los data frames son listas donde cada elemento de la lista contiene un vector, produciendo dos dimensiones.

Permiten filas y columnas donde cada columna puede tener un tipo de dato distinto

Data frames

Para crear un data frame se utiliza la función **data.frame()**

```
> comparativeGenomeSize<-data.frame(  
+ organism=c("Human","Mouse","Fruit Fly","Roundworm","Yeast"),  
+ genomeSizeBP=c(3000000000,3000000000,135600000,97000000,12100000),  
+ estGeneCount=c(30000,30000,13061,19099,6034)  
+ )  
> comparativeGenomeSize  
  organism genomeSizeBP estGeneCount  
1   Human     3.000e+09      30000  
2   Mouse     3.000e+09      30000  
3 Fruit Fly    1.356e+08     13061  
4 Roundworm    9.700e+07     19099  
5   Yeast     1.210e+07      6034
```

Data frames

Existen dos formas de acceder a elementos específicos del data frame.

- 1) Al igual que en las matrices, indicando las coordenadas del elemento con el formato **[fila, columna]**

```
> comparativeGenomeSize
  organism genomeSizeBP estGeneCount
  1      Human    3.000e+09     30000
  2     Mouse    3.000e+09     30000
  3 Fruit Fly   1.356e+08    13061
  4 Roundworm  9.700e+07    19099
  5      Yeast   1.210e+07     6034
> comparativeGenomeSize[1,2]
[1] 3e+09
```

Data frames

Existen dos formas de acceder a elementos específicos del data frame.

2) Utilizando el símbolo “\$” para indicar la columna que nos interesa.

```
> comparativeGenomeSize$organism  
[1] Human      Mouse      Fruit Fly Roundworm Yeast  
Levels: Fruit Fly Human Mouse Roundworm Yeast
```

comparativeGenomeSize\$organism regresa un vector con los elementos de la columna **organism** por lo que se puede acceder a un elemento específico utilizando su índice.

```
> comparativeGenomeSize$organism[2]  
[1] Mouse  
Levels: Fruit Fly Human Mouse Roundworm Yeast
```

Práctica 9

Compara el resultado de los siguientes comandos:

- 1) comparativeGenomeSize\$organism
- 2) comparativeGenomeSize[“organism”]
- 3) comparativeGenomeSize[,1]

Práctica 10

- 1) Crea un data frame cuyas filas sean los nombres de los integrantes de tu familia. Incluye las columnas de edad, género, gusto por la pizza con piña y película favorita.
- 2) Crea un nuevo data frame donde los miembros estén ordenados según su edad.
- 3) Crea un tercer data frame que sólo incluya a los integrantes mayores de 18 años.
- 4) Crea un cuarto data frame que sólo incluya a los integrantes que les gusta la pizza con piña y elimina esa columna.

Filtrar datos en data frames

Otras formas de filtrar datos en un data frame son las funciones **which()** y **subset()**

```
> which(comparativeGenomeSize$estGeneCount > 10000)
[1] 1 2 3 4
```

¿Cómo obtendrías sólo estas columnas del data frame?

```
> which(comparativeGenomeSize$estGeneCount > 10000)
[1] 1 2 3 4
> low_gncount <- which(comparativeGenomeSize$estGeneCount > 10000)
> comparativeGenomeSize[low_gncount,]
  organism genomeSizeBP estGeneCount
 1 Human    3.000e+09      30000
 2 Mouse    3.000e+09      30000
 3 Fruit Fly 1.356e+08     13061
 4 Roundworm 9.700e+07     19099
```

Filtrar datos en data frames

Otras formas de filtrar datos en un data frame son las funciones **which()** y **subset()**

```
> subset(comparativeGenomeSize, estGeneCount > 10000)
  organism genomeSizeBP estGeneCount
  1 Human    3.000e+09      30000
  2 Mouse     3.000e+09      30000
  3 Fruit Fly 1.356e+08      13061
  4 Roundworm 9.700e+07      19099
```

Las condiciones pueden combinarse

```
> subset(comparativeGenomeSize, estGeneCount > 10000 & genomeSizeBP
< 3.000e+09)
  organism genomeSizeBP estGeneCount
  3 Fruit Fly    135600000      13061
  4 Roundworm    97000000      19099
```

Detalles importantes de mi sesión de R

Como una terminal en UNIX, tu sesión de R está posicionada en un directorio de tu computadora.

En la terminal es común que la sesión esté posicionada en el directorio donde mandaste llamar a R.

¿Cómo saber en qué directorio se encuentra la sesión actual de R?

```
> getwd()
```

```
> getwd()  
[1] "/Users/Daniela"
```

¿Cómo cambiar de directorio?

```
> setwd([ruta del nuevo directorio])
```

```
> getwd()
```

```
[1] "/Users/Daniela"
```

```
> setwd("/Users/Daniela/Desktop/Temporal/")
```

```
> getwd()
```

```
[1] "/Users/Daniela/Desktop/Temporal"
```

Detalles importantes de mi sesión de R

Para ver los archivos que se encuentran en el directorio:

```
> dir()
```

```
> dir()  
[1] "cleanAdapter.R"  
[2] "correlation.R"  
[3] "datAnalysis.r"
```

Para terminar una sesión:

```
> q()  
> quit()
```

Al salir, el sistema preguntará:

```
>Save workspace image? [y/n/c]:
```

Workspace Image

Guardar el *workspace image* significa guardar la historia de comandos que has utilizado en tu sesión de R, así como las variables que has creado y los paquetes que han sido cargados.

La idea es que, al regresar a esa sesión, puedes continuar trabajando como si nada, en lugar de tener que declarar de nuevo variables y cargar paquetes.

¿Cómo funciona?

Al responder sí a guardar el *workspace* se crean dos archivos:

.Rdata – Guarda todos los objetos que haya definido el usuario. Se encuentra en un formato binario (no puede ser leído por humanos).

.Rhistory – Guarda el historial de comandos utilizados durante la sesión. Se encuentra en formato texto.

Ambos archivos se crearán en el directorio en que se encuentra la sesión actual de R. ¡Sólo se podrán recuperar si se conoce ese directorio!

Práctica 11

Utilizando una terminal, localiza los archivos .RData y .Rhistory de tu sesión actual.

- 1) ¿Qué significan los puntos en el nombre del archivo?
- 2) Visualiza su contenido.

Crea una carpeta llamada “R” y dos subcarpetas llamadas “sesión 1” y “sesión 2” desde tu línea de comandos.

- 1) Inicia una sesión de R en cada carpeta, ejecuta un par de comandos en cada una y guárdalas.
- 2) Reinicia tu sesión de R desde cada carpeta. ¿Puedes identificar las dos sesiones?
- 3) Vuelve a la sesión en “sesión 2”, declara una nueva variable y guarda tu sesión. Al reabrirla, ¿qué notas? ¿dónde quedó la versión anterior?

Guardar datos

Para evitar el problema de recordar qué sesión se encuentra en cada carpeta, es posible guardar los datos generados en una sesión para poder cargarlos en cualquier otro momento.

¿Cómo?

> save.image()	> savehistory(.Rhistory)
> load()	> loadhistory(.Rhistory)

```
> save.image(file = "archivo.Rdata")
> load(file = "archivo.Rdata")
```

Si sólo quieren guardarse uno o varios objetos específicos se usa el comando **save()**

```
> save(objeto, file = "archivo.Rdata")
```

```
> save(objeto1,objeto2, file = "archivo.Rdata")
```

Para cargarlos posteriormente, también se utiliza **load()**.

Lectura de archivos

Para la lectura y escritura de archivos es importante saber en qué directorio se encuentra nuestra sesión actual de R y qué archivos se encuentran ahí

```
> getwd()
[1] "/Users/Daniela"
> dir()
[1] "AIC-prefs"                  "cursor.RData"
[3] "CytoscapeConfiguration"    "Desktop"
[5] "Documents"                  "Downloads"
[7] "Dropbox (UNAM-CCG)"        "GU_groups"
[9] "igv"                         "Library"
[11] "Movies"                      "Music"
[13] "objeto.RData"                "pathway-tools"
[15] "Pictures"                    "ptools-local"
[17] "Public"
```

Lectura de archivos

Los datos más comunes se encuentran en forma de tablas.

Los archivos que contienen tablas usualmente se denominan CSV (*comma separated value*), aunque muchas veces los valores están separados por tabuladores.

Para leerlos se utiliza la función **read.table()**

```
> gene_expression <- read.table(file="gene_expression_data.txt",
header=TRUE, row.names=1)
> gene_expression
           time.40 time.50      time.60 time.70
YAL001C   -0.070000000   -0.23 -0.100000000    0.03
YAL014C    0.215000000    0.09  0.025000002   -0.04
YAL016W    0.150000000    0.15  0.220000000    0.29
YAL020C   -0.350000000   -0.28 -0.215000000   -0.15
```

Lectura de archivos

Automáticamente R convierte la información del archivo en un data frame

```
> class(gene_expression)
[1] "data.frame"
```

read.table() permite varios argumentos que vale la pena mencionar:

Usage:

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".", row.names, col.names,
           as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = default.stringsAsFactors(),
           fileEncoding = "", encoding = "unknown", text)
```

Práctica 12

Abre el archivo phenotypes.csv en un editor de textos.

- 1) Observa su formato y decide los argumentos óptimos para leer este archivo.
- 2) Modifica los parámetros por default que sean necesarios para obtener un data frame con nombres de columnas y nombres de filas

Lectura de archivos

Algunas cosas que podemos preguntar a nuestro data frame:

Ver las primeras seis columnas con **head()**

```
> head(gene_expression)
      time.40 time.50 time.60 time.70
YAL001C -0.070   -0.23  -0.100    0.03
YAL014C  0.215    0.09   0.025   -0.04
YAL016W  0.150    0.15   0.220    0.29
YAL020C -0.350   -0.28  -0.215   -0.15
YAL022C -0.415   -0.59  -0.580   -0.57
YAL036C  0.540    0.33   0.215    0.10
```

Ver los nombres de las columnas con **colnames()**

```
> colnames(gene_expression)
[1] "time.40" "time.50" "time.60" "time.70"
```

Lectura de archivos

Existen variantes de **read.table()** cuyos parámetros por *default* son más útiles para leer ciertos tipos de archivos:

```
read.csv(file, header = TRUE, sep = ",", quote = "\"\"", dec = ".",
fill = TRUE, ...)
```

```
read.csv2(file, header = TRUE, sep = ";", quote = "\"\"", 
dec = ",",
fill = TRUE, ...)
```

```
read.delim(file, header = TRUE, sep = "\t", quote = "\"\"", 
dec = ".",
fill = TRUE, ...)
```

```
read.delim2(file, header = TRUE, sep = "\t", quote = "\"\"", 
dec = ",",
fill = TRUE, ...)
```

Escritura de archivos

Los data frames y las matrices pueden escribirse en un archivo utilizando la función **write.table()** y sus variantes **write.csv()** y **write.csv2()**

```
write.table                  package:utils          R Documentation
Data Output
Description:
  'write.table' prints its required argument 'x' (after converting
  it to a data frame if it is not one nor a matrix) to a file or
  connection.

Usage:
  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 = "")
```

Práctica 13

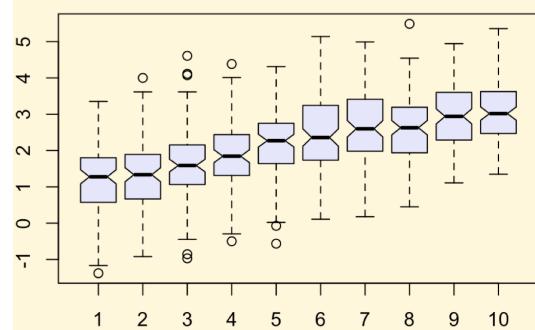
- 1) Lee el archivo molecular_weight.txt y conviértelo en una matriz de 5x4.
- 2) Agrega nombres de columnas y filas.
- 3) Agrega una nueva columna y una nueva fila.
- 4) Exporta tu matriz a un archivo llamado “molecular_weight_matrix.csv” con las columnas separadas por comas.

Gráficas en R

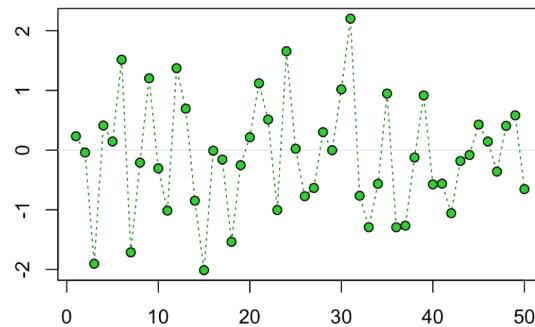
La instalación básica de R cuenta con funciones para graficar datos. Para explorar el tipo de gráficas que se pueden generar utilizamos:

```
> demo(graphics)
```

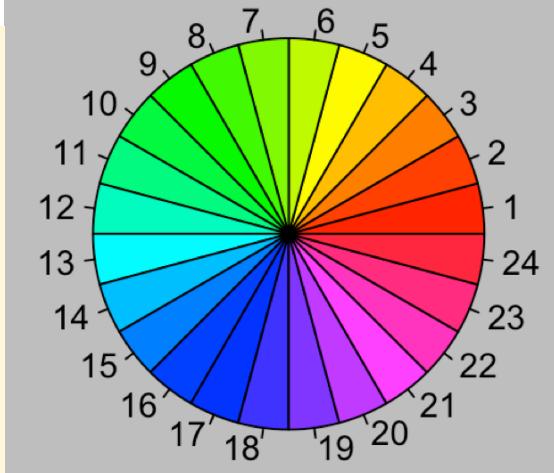
Notched Boxplots



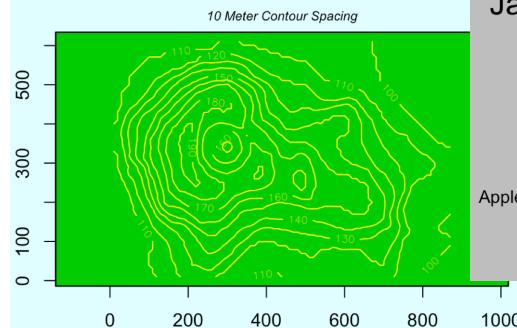
Simple Use of Color In a Plot



A Sample Color Wheel



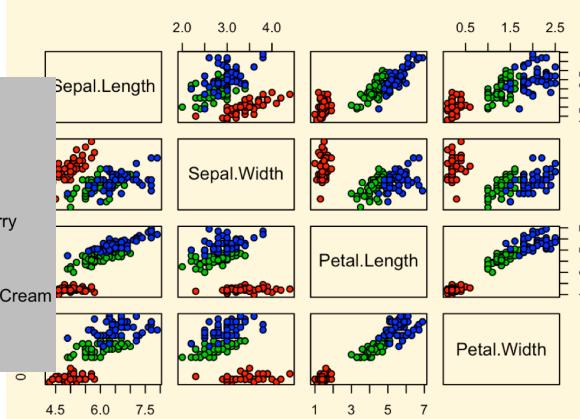
A Topographic Map of Maunga Whau



January Pie Sales

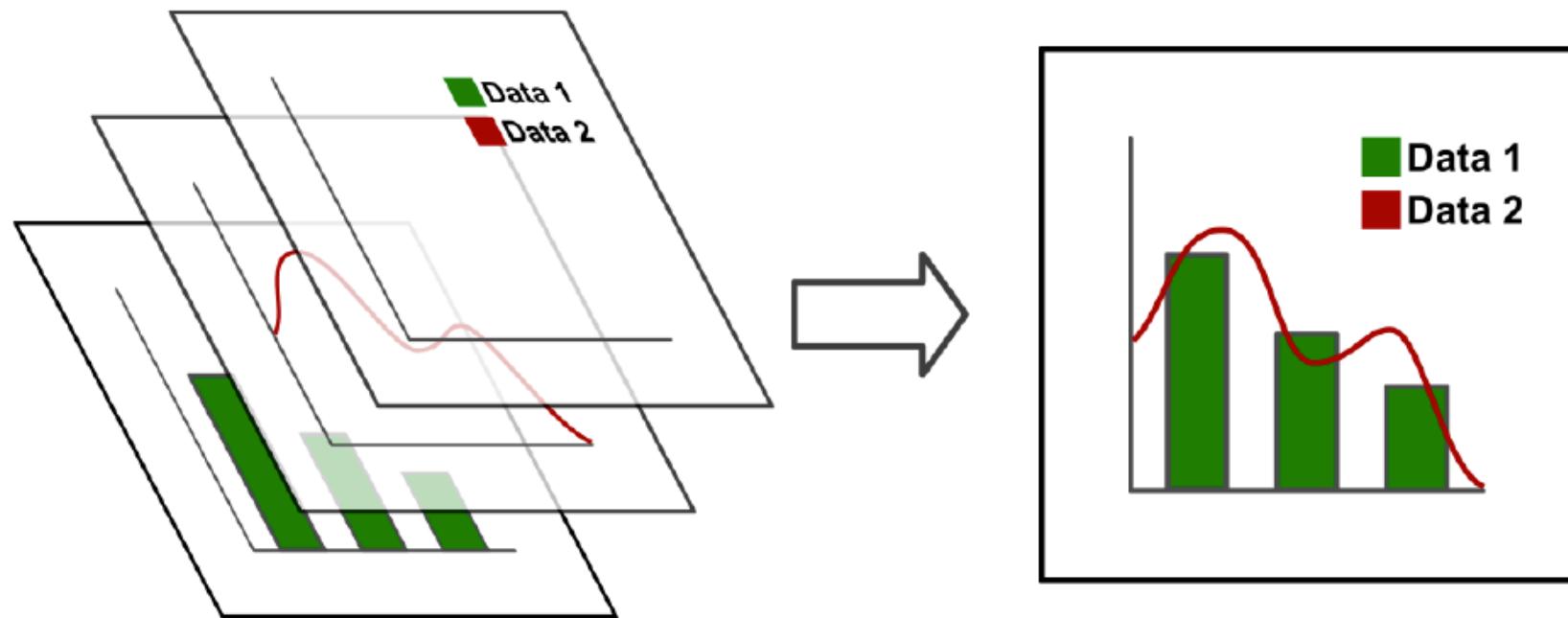


Edgar Anderson's Iris Data



Gráficas en R

Las gráficas en R siguen un “modelo de pintor”, es decir, no se dibujan en una sola operación, sino que se construyen con varias capas que son colocadas una encima de la otra



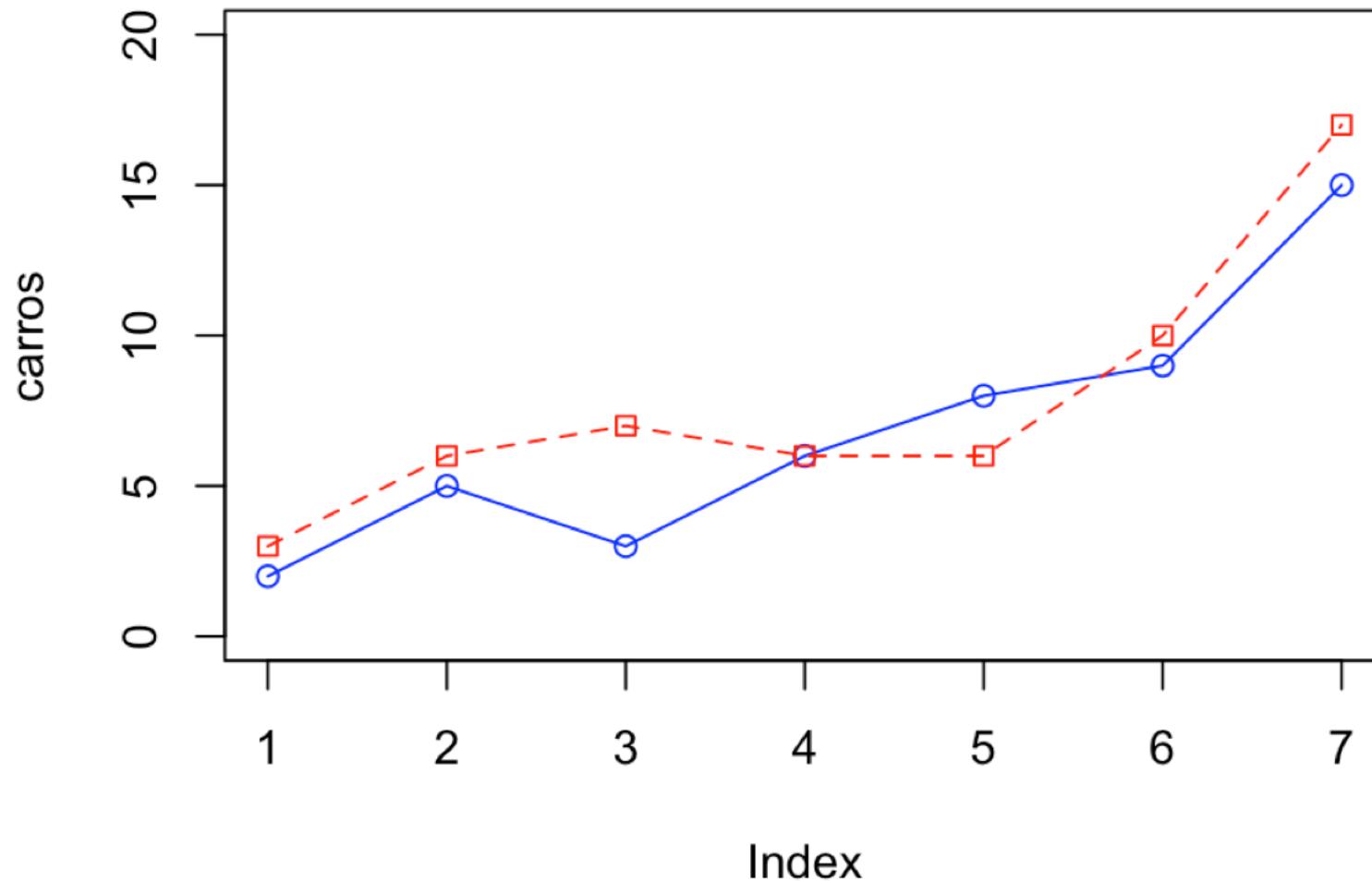
Gráficas en R

Consideremos dos vectores de datos:

```
> carros<-c(2,5,3,6,8,9,15)
> camionetas<-c(3,6,7,6,6,10,17)
> plot(carros, type="o", col="blue",
ylim=c(0,20))
> lines(camionetas, type="o", pch=22,
lty=2, col="red")
> title(main="Vehiculos recibidos",
col.main="red", font.main=4)
```

Gráficas en R

Vehiculos recibidos



Gráficas en R

Ahora por partes... utilizamos tres funciones distintas:

```
> plot(carros, type="o", col="blue", ylim=c(0,20))
```

carros – variable a graficar, en este caso es un vector pero puede ser cualquier estructura numérica en 1D o 2D

type – indica el tipo de gráfica:

“p” – puntos

“l” – líneas

“b” – puntos y líneas

“c” – como “b” sin dibujar los puntos

“o” – líneas y puntos superpuestos

“h” – histograma-like

“s” – escaleras

Gráficas en R

Ahora por partes... utilizamos tres funciones distintas:

```
> plot(carros, type="o", col="blue", ylim=c(0,20))
```

col – color de las líneas, más info: Rcolor.pdf

ylim – Rango del eje Y

```
> lines(camionetas, type="o", pch=22, lty=2, col="red")
```

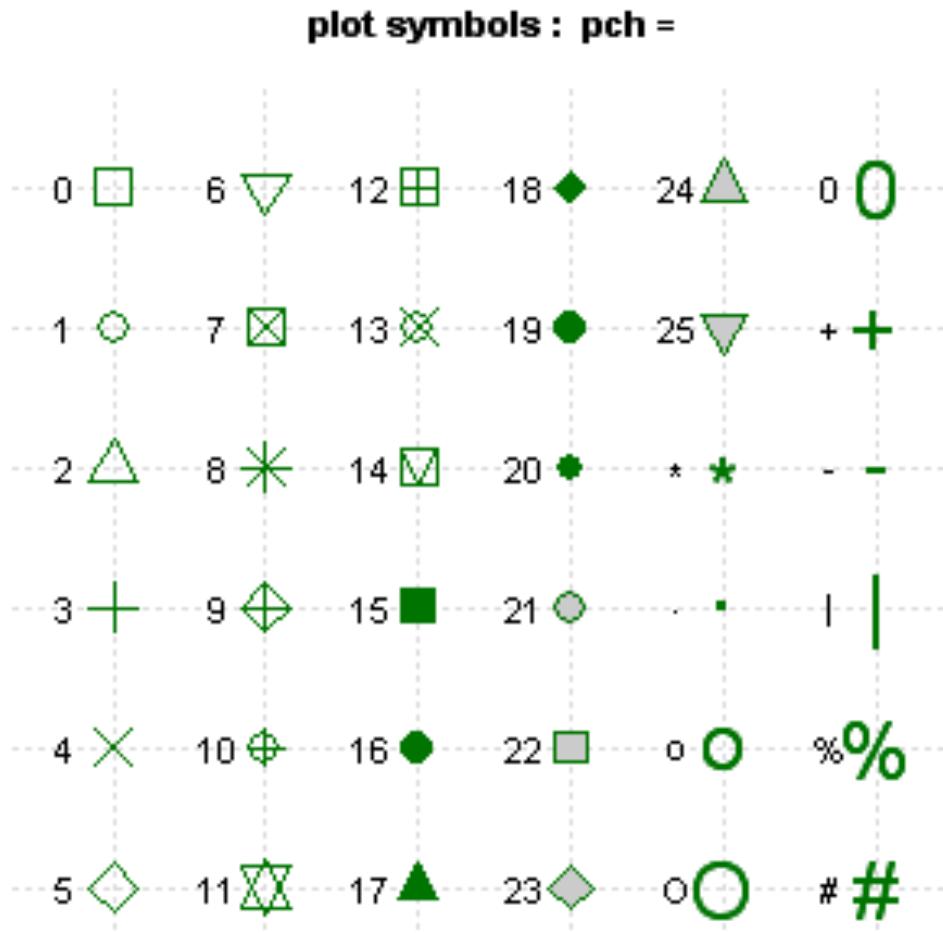
camionetas – variable a añadir en la gráfica anterior

type – tipo de gráfica para los puntos que se agregarán, los valores son iguales que para **plot()**

Gráficas en R

```
> lines(camionetas, type="o", pch=22, lty=2, col="red")
```

pch – tipo de puntos



Gráficas en R

```
> lines(camionetas, type="o", pch=22, lty=2, col="red")
```

lty – tipo de líneas



col – color, al igual que en **plot()**

Gráficas en R

```
> title(main="Vehiculos recibidos", col.main="red",
font.main=4)
```

main – titulo principal de la gráfica

col.main – color del titulo principal

font.main – tipo de fuente del titulo principal

Feature	Argument
Title	<code>font.main</code>
Subtitle	<code>font.sub</code>
Axis	<code>font.axis</code>
Labels	<code>font.lab</code>

Value	Font Type
1	Plain
2	Bold
3	Italic
4	Bold Italic
5	Symbol

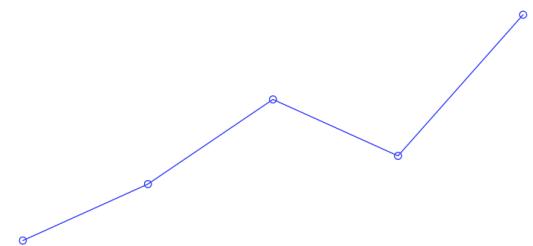
Gráficas en R

Los datos a graficar también pueden provenir de un archivo

```
> getwd()
[1] "/Dropbox/Camila/R/datasets"
> autos<-read.table("autos.dat",header=TRUE,sep="\t")
> autos
  compactos pick-ups SUVs
1          1         2     4
2          3         5     4
3          6         4     6
4          4         5     6
5          9        12    16
```

Creemos nuestra gráfica paso a paso...

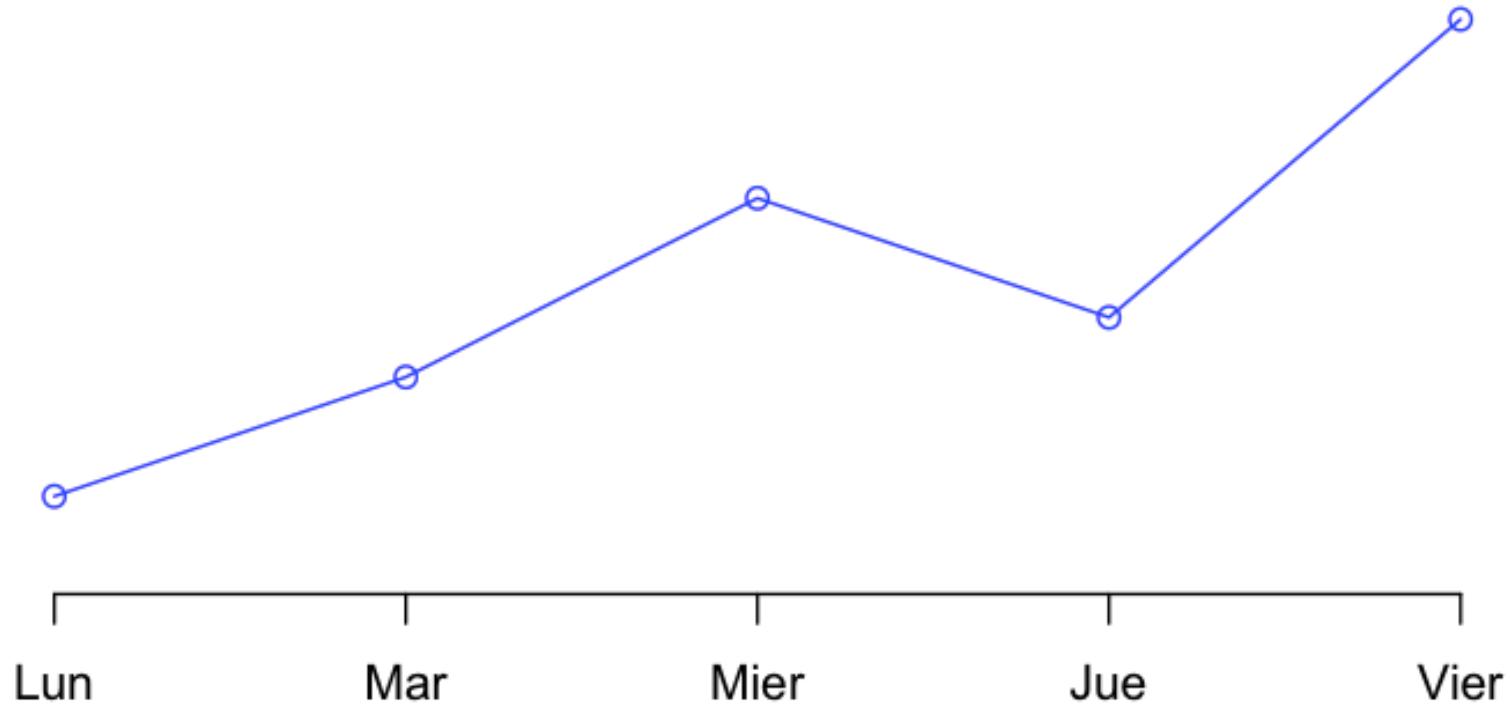
```
> max_y <- max(autos)
> colores <- c("blue", "red", "forestgreen")
> plot(
+   autos$compactos, type="o",
+   col=colores[1], ylim=c(0,max_y),
+   axes=FALSE, ann=FALSE)
```



Gráficas en R

Crear las etiquetas del eje X

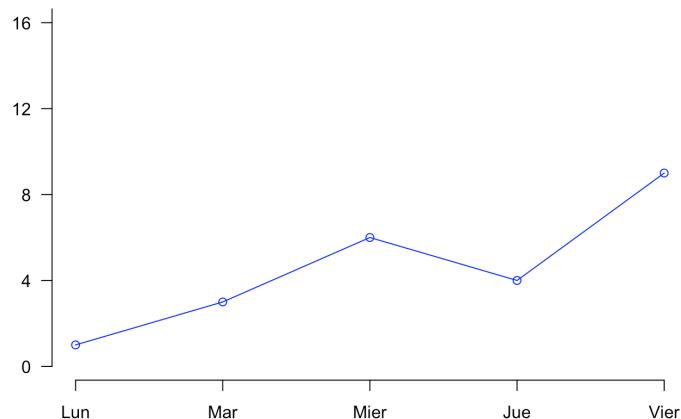
```
axis(1, at=1:5, lab=c("Lun", "Mar", "Mier", "Jue", "Vier"))
```



Gráficas en R

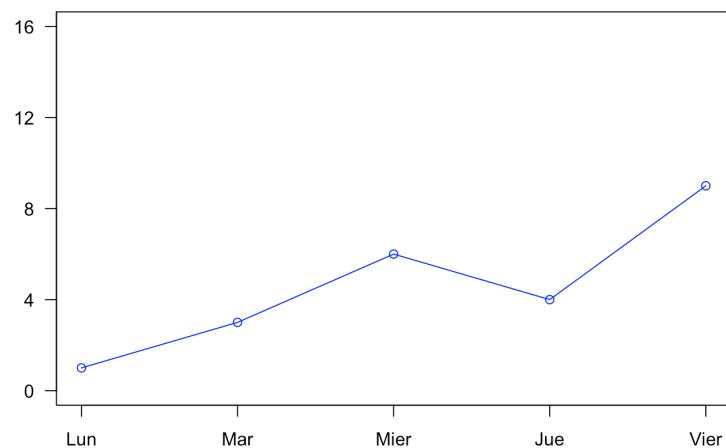
Eje Y con etiquetas horizontales, marcas cada 4 valores:

```
> valores <- 4*(0:max_y)
> axis(2, las=1, at=valores)
```



Agregar un marco a la gráfica

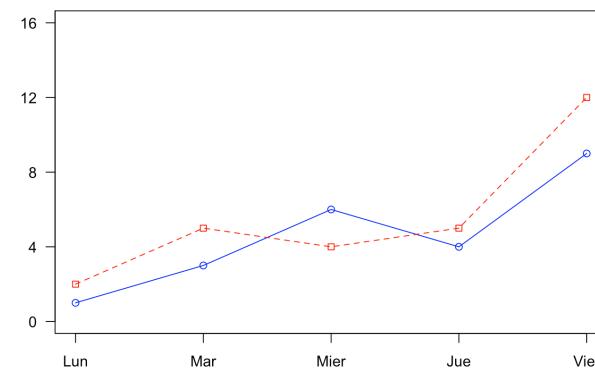
```
box()
```



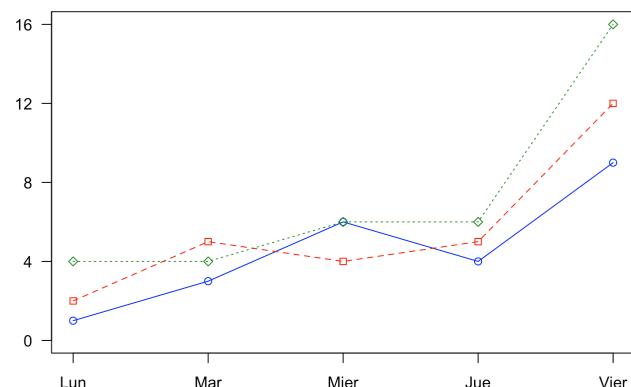
Gráficas en R

Agregar los datos de la columna pick-ups con una línea discontinua roja y cuadrados

```
> lines(autos$pick.ups, type="o",
+ pch=22, lty=2, col=colores[2])
```



Agregar los datos de la columna SUVs con una línea de puntos y diamantes



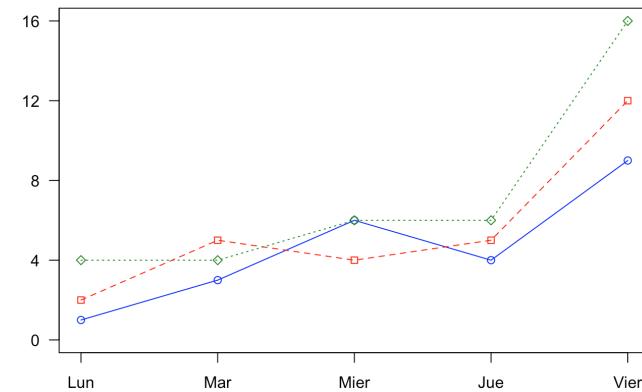
```
> lines(autos$SUVs, type="o",
+ pch=23, lty=3, col=colores[3])
```

Gráficas en R

Crear un título principal con letras en **ítälica**, negrita, color rojo

```
title(main="Lavado de Autos",  
+ col.main="red", font.main=4)
```

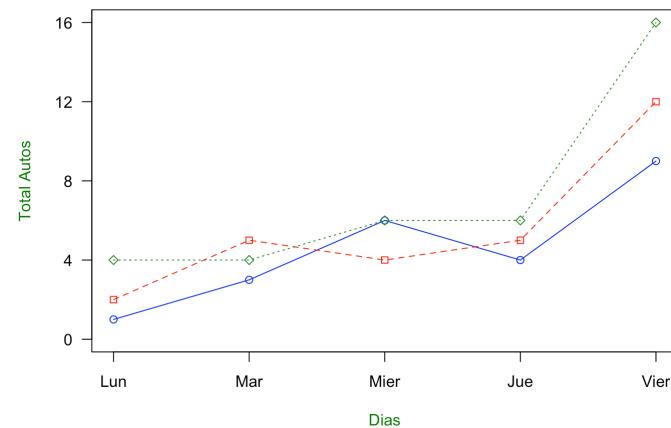
Lavado de Autos



Etiquetar los ejes X y Y con texto verde

```
> title(xlab= "Dias",  
+ col.lab=rgb(0,0.5,0))  
> title(ylab= "Total Autos",  
+ col.lab=rgb(0,0.5,0))
```

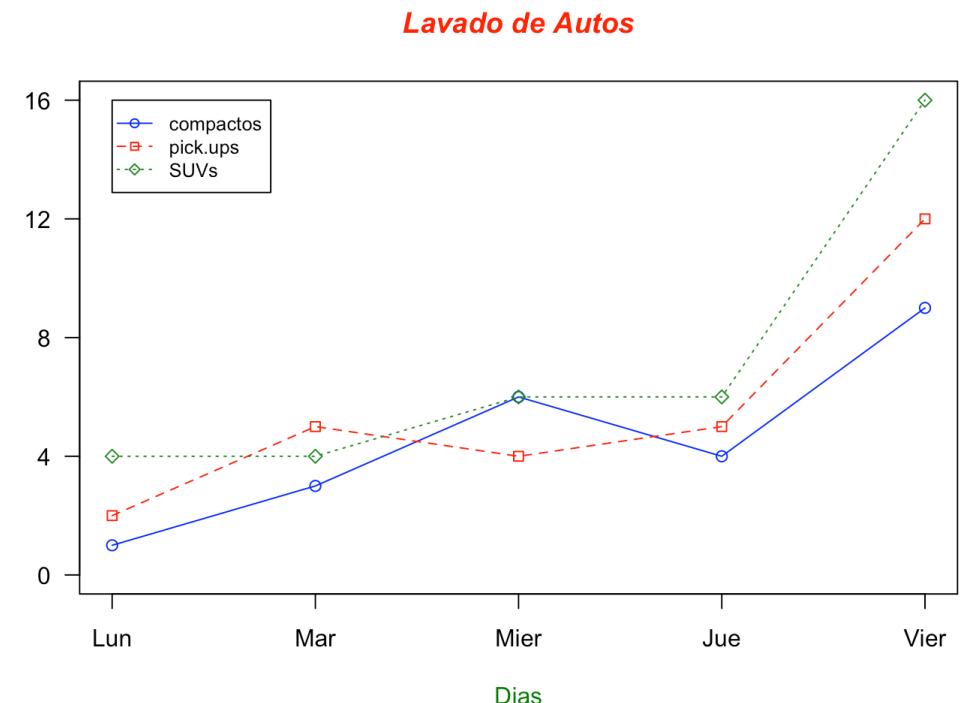
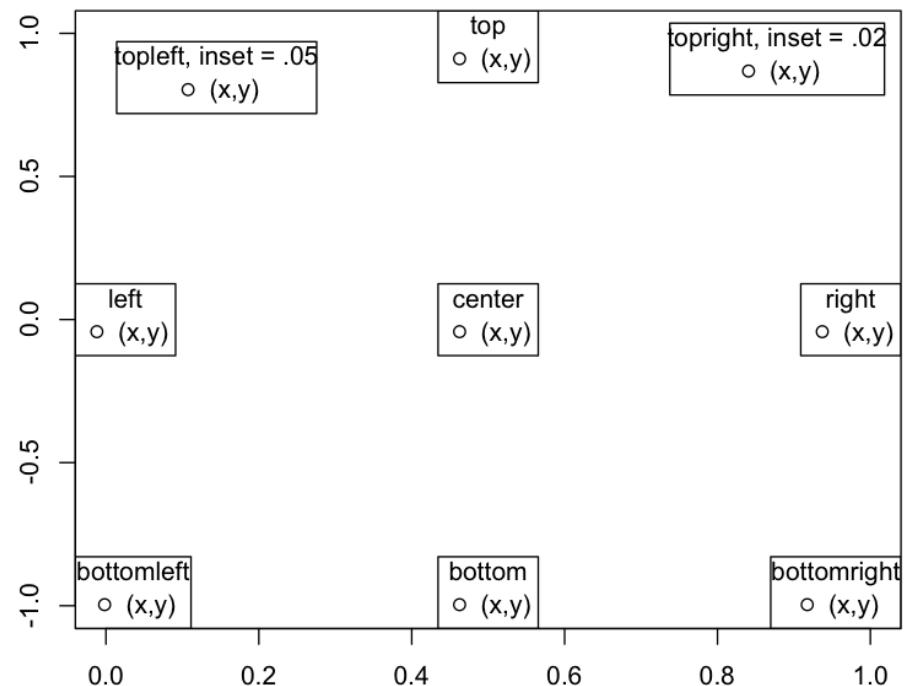
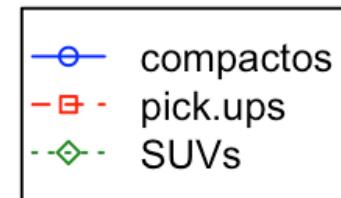
Lavado de Autos



Gráficas en R

Crear una leyenda que indique a qué color/línea/punto corresponde cada columna de autos

```
> legend(topleft, max_y, names(autos),
+ cex=0.8, col=colores, pch=21:23, lty=1:3)
```



Gráficas en R

¿Cómo exportar nuestra gráfica?

R permite enviar cada capa dibujada a un archivo png, jpeg, pdf
bmp o tiff

```
> png(filename="Dropbox/R/datasets/autos.png", height=295,  
width=300, bg="white")  
  
> plot...  
> axis...  
> title...  
> legend...  
  
> dev.off()
```

dev.off() le indica a R que la gráfica está lista para ser impresa
y automáticamente se genera el archivo png

Gráficas en R

Para exportar en otros tipos de archivos existen las siguientes funciones:

```
> bmp(filename, width=480, height=480, ...)  
> jpeg(filename, width=480, height=480, ...)  
> png(filename, width=480, height=480, ...)  
> tiff(filename, width=480, height=480, ...)  
> pdf(filename, width=480, height=480, ...)
```

Práctica 14

Personaliza tu gráfica de autos cambiando los siguientes parámetros:

- 1) Color de las líneas para cada columna
- 2) Tipo de punto para cada columna
- 3) Aspecto del título principal
- 4) Color de los nombres de los ejes

Lee la ayuda de la función **par()** y cambia un parámetro extra de la gráfica.

Exporta tu gráfica como un archivo jpeg.

¡RECUERDA! La función para exportar la gráfica debe ser la primera

¿Preguntas?

Contacto

Daniela E. Ledezma Tejeida
dledezma@lcg.unam.mx



Except where otherwise noted, this work is licensed under

<http://creativecommons.org/licenses/by-nc/3.0/>