CCG
Centro de Ciencias Genómicas

Entre Pares 2018

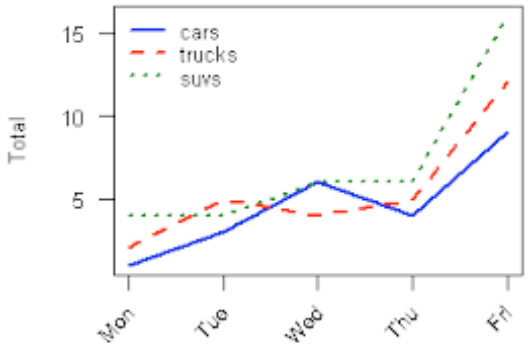
R para Bioinformática

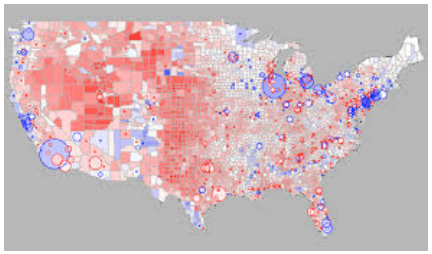
BUAP
10-09-18

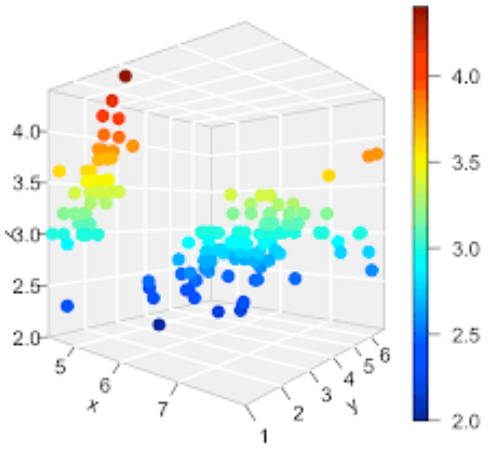
Dra. Daniela E. Ledezma Tejeida

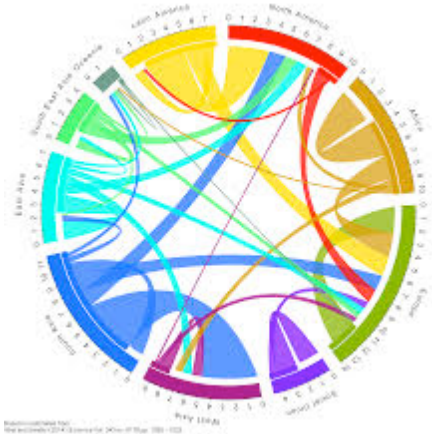
¿Qué es R?

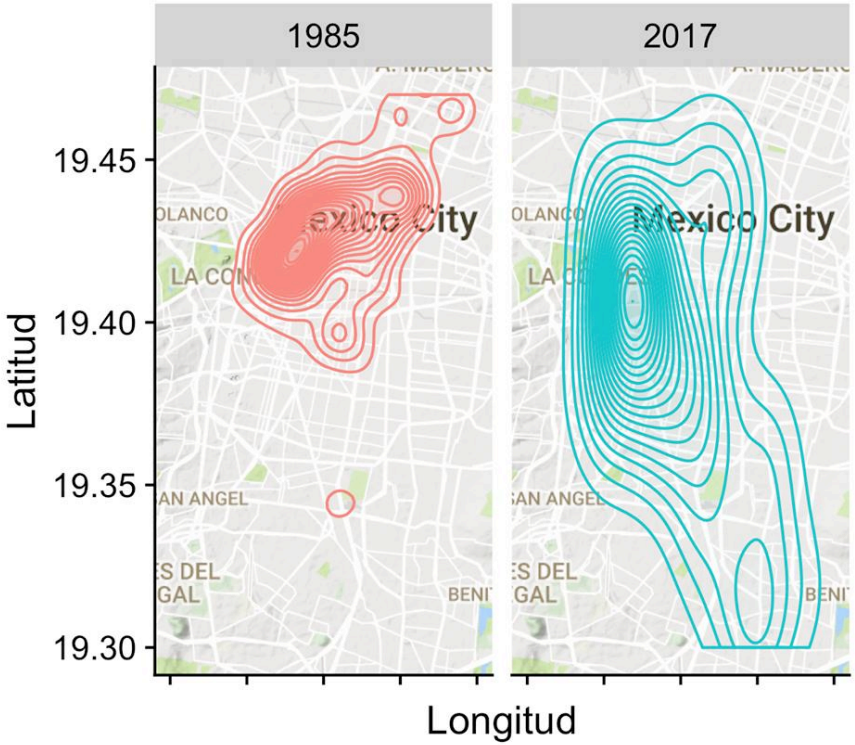
R es un entorno y un lenguaje de programación utilizado para manejo de datos, procesamiento estadístico y visualización











¿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.

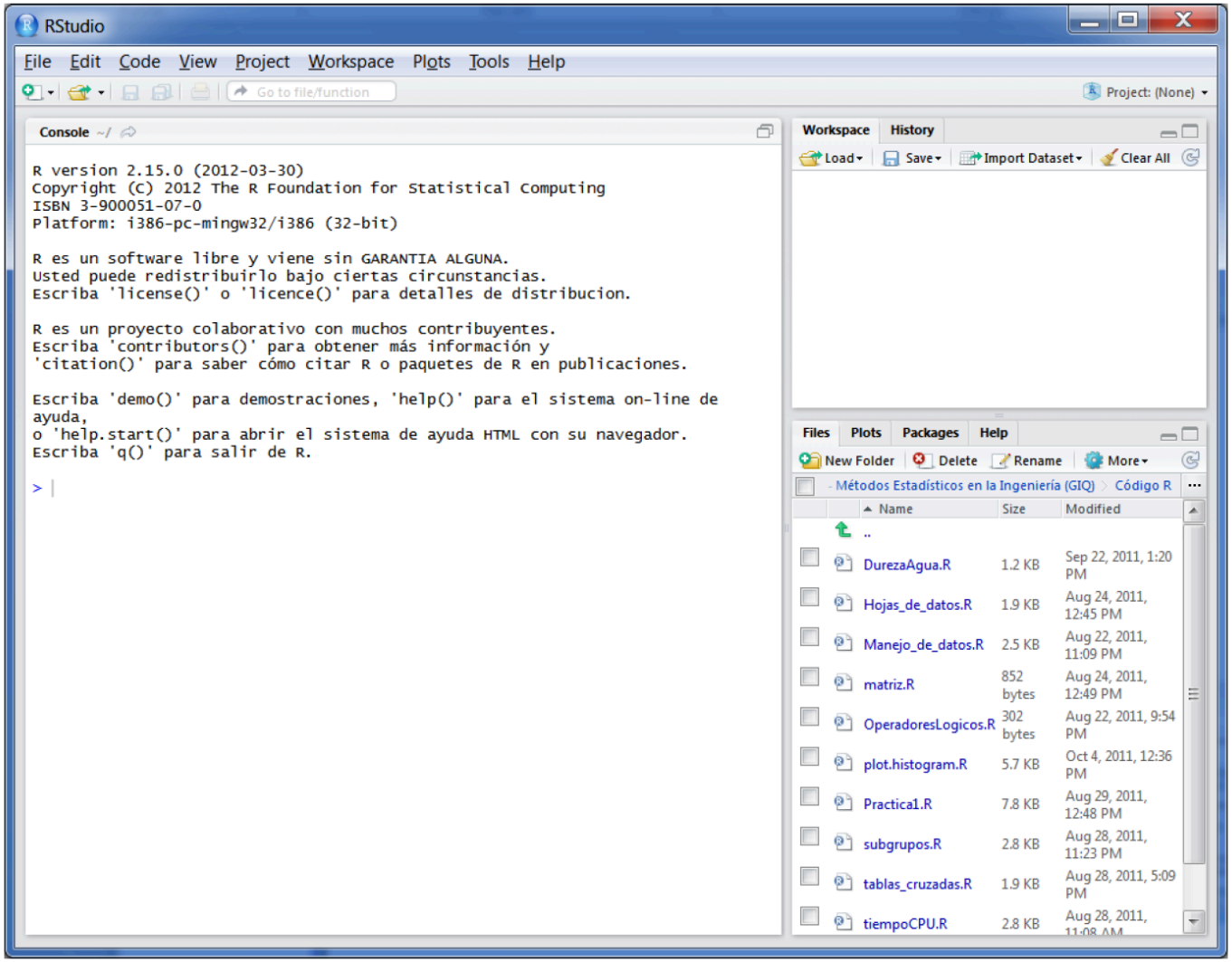
- Lenguaje de programación de propósito específico.

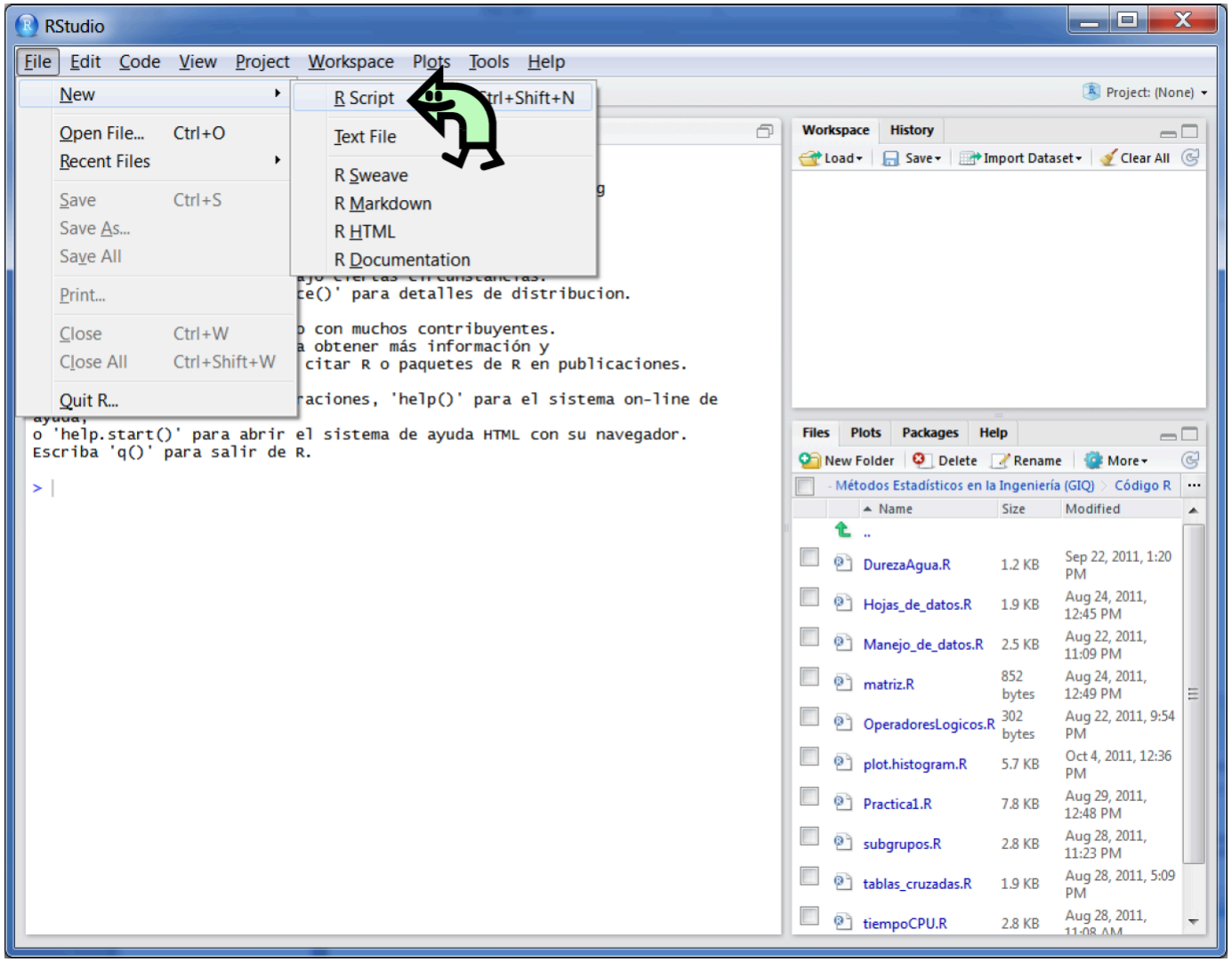
- Comunidad de desarrolladores y usuarios que constantemente cooperan para la mejora y desarrollo de herramientas nuevas que amplían las capacidades de R.

R en línea de comandos

```
vjimenez@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]

>
```

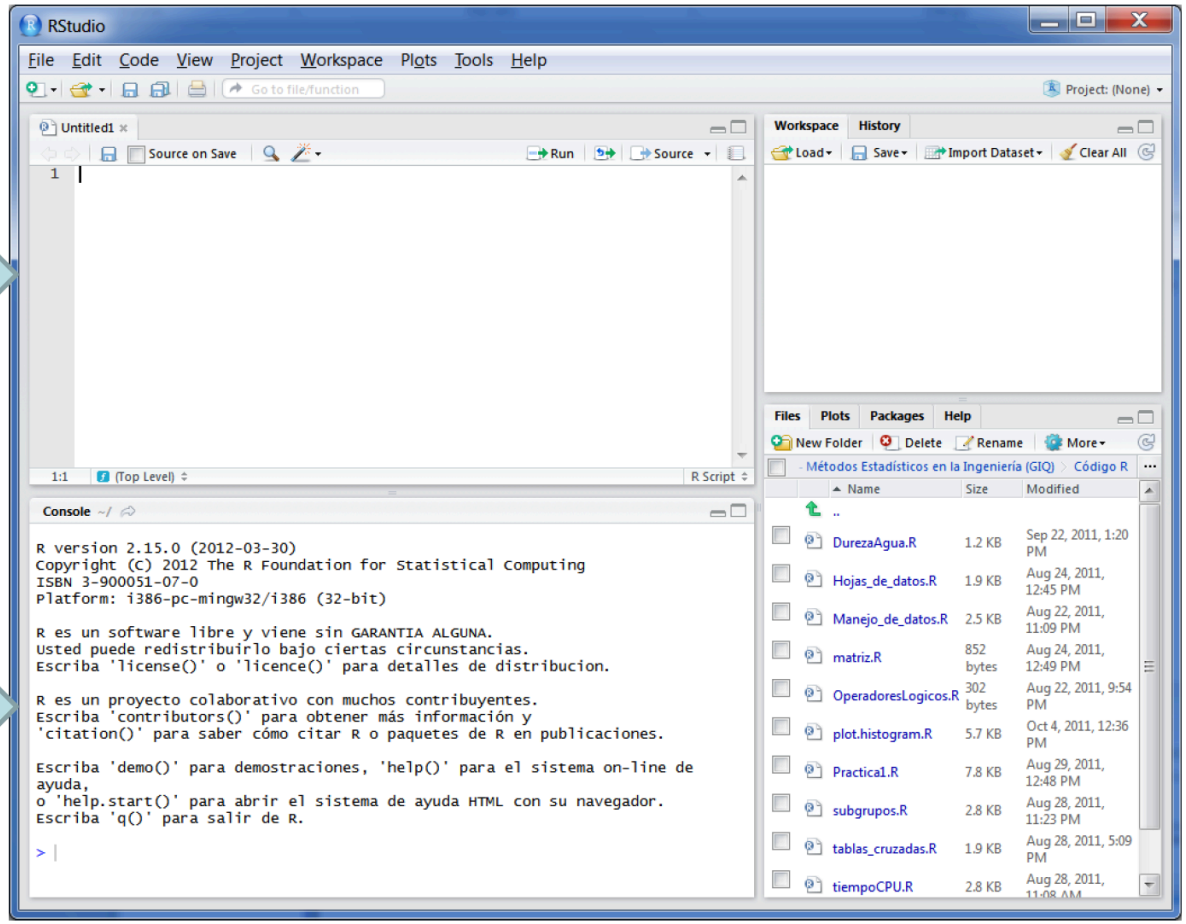
Rstudio


RStudio
File Edit Code View Project Workspace Plots Tools Help
Go to file/function
Project: (None)
Console ~/
R version 2.15.0 (2012-03-30)
Copyright (C) 2012 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: i386-pc-mingw32/i386 (32-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 más información y
'citation()' para saber cómo 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.
>
Workspace History
Load Save Import Dataset Clear All
Files Plots Packages Help
New Folder Delete Rename More
Métodos Estadísticos en la Ingeniería (GIQ) Código R
Name Size Modified
DurezaAgua.R 1.2 KB Sep 22, 2011, 1:20 PM
Hojas_de_datos.R 1.9 KB Aug 24, 2011, 12:45 PM
Manejo_de_datos.R 2.5 KB Aug 22, 2011, 11:09 PM
matriz.R 852 bytes Aug 24, 2011, 12:49 PM
OperadoresLogicos.R 302 bytes Aug 22, 2011, 9:54 PM
plot.histogram.R 5.7 KB Oct 4, 2011, 12:36 PM
Practica1.R 7.8 KB Aug 29, 2011, 12:48 PM
subgrupos.R 2.8 KB Aug 28, 2011, 11:23 PM
tablas_cruzadas.R 1.9 KB Aug 28, 2011, 5:09 PM
tiempoCPU.R 2.8 KB Aug 28, 2011,


RStudio
File Edit Code View Project Workspace Plots Tools Help
New
Open File... Ctrl+O
Recent Files
Save Ctrl+S
Save As...
Save All
Print...
Close Ctrl+W
Close All Ctrl+Shift+W
Quit R...
R Script
Text File
R Sweave
R Markdown
R HTML
R Documentation
Project: (None)
Workspace History
Load Save Import Dataset Clear All
Files Plots Packages Help
New Folder Delete Rename More
Métodos Estadísticos en la Ingeniería (GIQ) Código R
Name Size Modified
DurezaAgua.R 1.2 KB Sep 22, 2011, 1:20 PM
Hojas_de_datos.R 1.9 KB Aug 24, 2011, 12:45 PM
Manejo_de_datos.R 2.5 KB Aug 22, 2011, 11:09 PM
matriz.R 852 bytes Aug 24, 2011, 12:49 PM
OperadoresLogicos.R 302 bytes Aug 22, 2011, 9:54 PM
plot.histogram.R 5.7 KB Oct 4, 2011, 12:36 PM
Practica1.R 7.8 KB Aug 29, 2011, 12:48 PM
subgrupos.R 2.8 KB Aug 28, 2011, 11:23 PM
tablas_cruzadas.R 1.9 KB Aug 28, 2011, 5:09 PM
tiempoCPU.R 2.8 KB Aug 28, 2011,
o 'help.start()' para abrir el sistema de ayuda HTML con su navegador.
Escriba 'q()' para salir de R.
>
ce()' para detalles de distribucion.
o con muchos contribuyentes.
a obtener más información y
citar R o paquetes de R en publicaciones.
raciones, 'help()' para el sistema on-line de

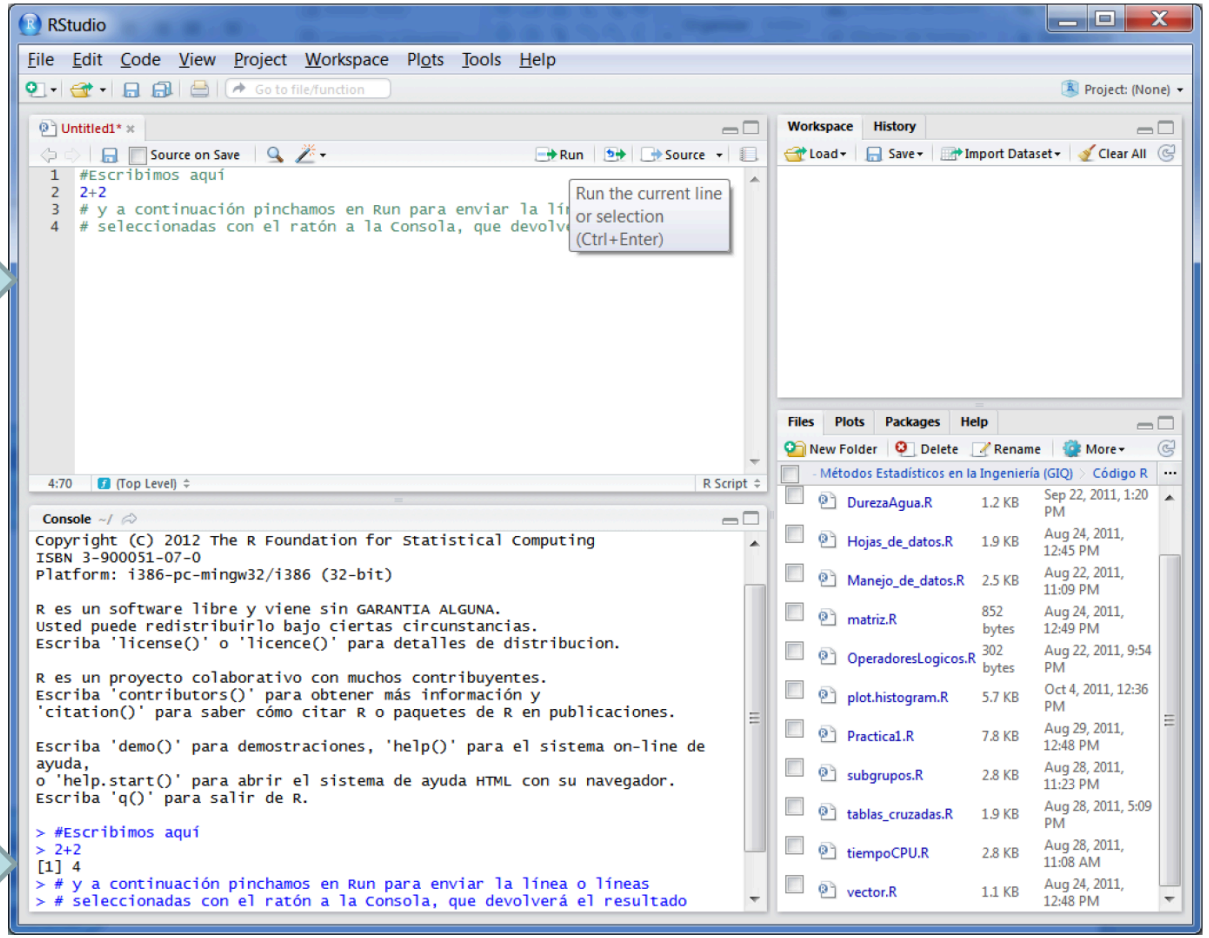
Editor

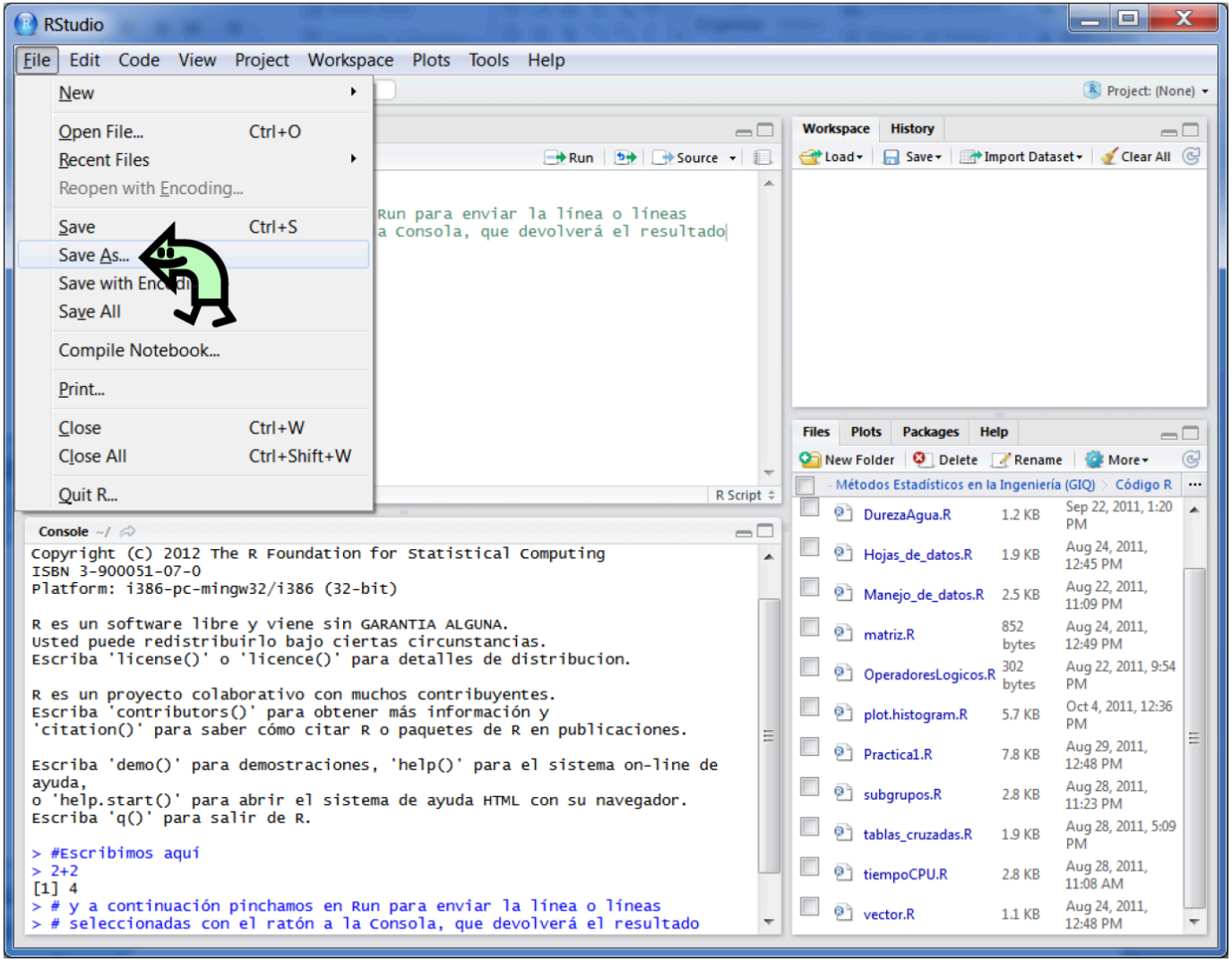
Consola



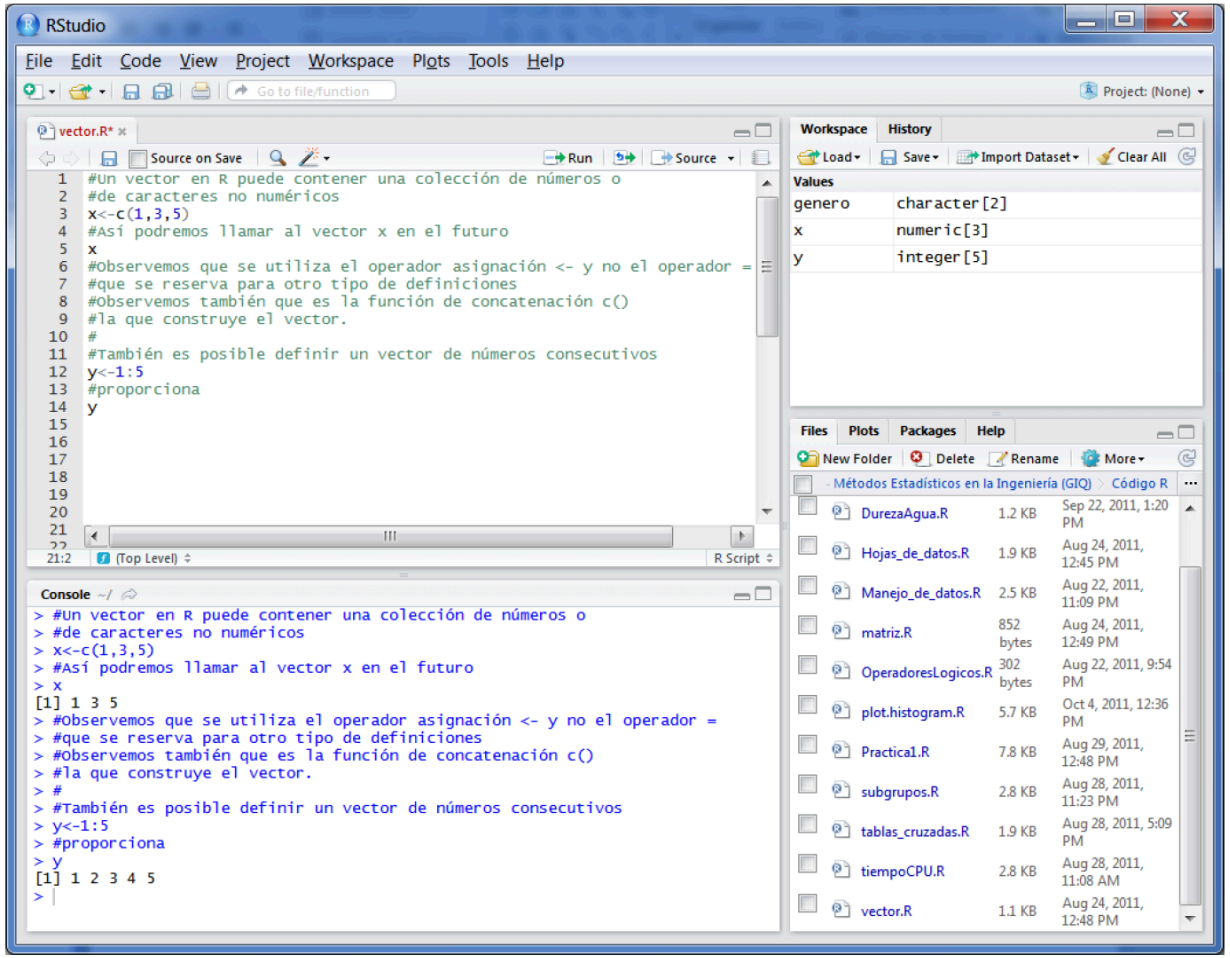
Escribimos en el Editor

El resultado se vuelca en la Consola



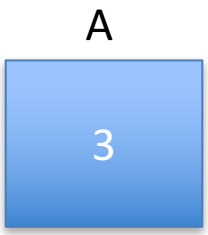

RStudio
File Edit Code View Project Workspace Plots Tools Help
New
Open File... Ctrl+O
Recent Files
Reopen with Encoding...
Save Ctrl+S
Save As...
Save with Encoding...
Save All
Compile Notebook...
Print...
Close Ctrl+W
Close All Ctrl+Shift+W
Quit R...
Project: (None)
Run
Source
Run para enviar la línea o líneas
a Consola, que devolverá el resultado
R Script
Workspace History
Load Save Import Dataset Clear All
Files Plots Packages Help
New Folder Delete Rename More
Métodos Estadísticos en la Ingeniería (GIQ) Código R
DurezaAgua.R 1.2 KB Sep 22, 2011, 1:20 PM
Hojas_de_datos.R 1.9 KB Aug 24, 2011, 12:45 PM
Manejo_de_datos.R 2.5 KB Aug 22, 2011, 11:09 PM
matriz.R 852 bytes Aug 24, 2011, 12:49 PM
OperadoresLogicos.R 302 bytes Aug 22, 2011, 9:54 PM
plot.histogram.R 5.7 KB Oct 4, 2011, 12:36 PM
Practica1.R 7.8 KB Aug 29, 2011, 12:48 PM
subgrupos.R 2.8 KB Aug 28, 2011, 11:23 PM
tablas_cruzadas.R 1.9 KB Aug 28, 2011, 5:09 PM
tiempoCPU.R 2.8 KB Aug 28, 2011, 11:08 AM
vector.R 1.1 KB Aug 24, 2011, 12:48 PM
Console ~/
Copyright (C) 2012 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: i386-pc-mingw32/i386 (32-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 más información y
'citation()' para saber cómo 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.
> #Escribimos aquí
> 2+2
[1] 4
> # y a continuación pinchamos en Run para enviar la línea o líneas
> # seleccionadas con el ratón a la Consola, que devolverá el resultado

El Script permite escribir, ejecutar y modificar nuestras líneas de código con comodidad y guardarlas para el futuro.



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:



Una variable se asigna con el operador "<-". E.g.

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

suma.total

variable_32

x_1

Práctica 1

¿Cuáles de estas definiciones son correctas?

1) Abracadabra <- 8

2) 8 <- ”Hola”

3) LogNatural2 <- 25

4) 2log <- 10

pi <- 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:

Datos tipo caracter

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
```

Práctica 3

1) Utiliza la función **help** para saber qué hace la funcion **c().**

2) Busca qué hace la función **paste**.

Funciones Numéricas

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

Práctica 4

Calcula utilizando el comando correcto:

1) Raíz cuadrada de 9.

3) Máximo de [10,1,5,62,3,75,8,75]

4) Mínimo de [10,1,5,62,3,75,8,75]

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

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
==	Igual	a==b	5 == 3 # False
<	Menor que	a < b	5 < 3 # False
<=	Menor o igual que	a<=b	5 <= 5 # True
>	Mayor	a>b	5 > 3 # True
>=	Mayor o Igual que	a>=b	5 <= 5 # True
!=	Distinto	a!=b	5 != 3 # True

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
- caracter
- 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.

1) Investiga el tipo de dato de cada vector.

2) 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 a 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
```

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]]
[1] "hola"  "adios"

[[2]]
[1]  TRUE FALSE

[[3]]
[1] 5

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

Índice de la lista

Índice del vector

Listas

Corroboremos las clases de los vectores

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

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

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?

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
```

¡Cuida 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
```

Factores

Un factor es una estructura de datos que sólo permite ciertos valores previamente definidos.

Por ejemplo, el factor “género” puede sólo admitir los valores “femenino” o “masculino”.

Los valores permitidos en un factor se llaman ***levels***

Para declarar un factor se utiliza el comando **factor()**

```
> tissue<-factor(c("liver","kidney","liver","kidney","liver"))
> tissue
[1] liver  kidney liver  kidney liver
Levels: kidney liver
```

Factores

Los niveles de un factor pueden obtenerse directamente utilizando **levels()**

```
> levels(tissue)
[1] "kidney" "liver"
```

¿Por qué los factores son útiles para R?

Pista:

```
> class(tissue)
[1] "factor”
> typeof(tissue)
[1] "integer"
```

R convierte los niveles de un factor en valores numéricos, haciendo más eficiente su manejo

Factores

Creemos un factor

```
> survey <- c("M", "F", "F", "M", "M”)
> factor_survey <- factor(survey)
> levels(factor_survey) <-
c("Female","Male”)
> factor_survey
[1] Male Female Female Male   Male
Levels: Female Male
```

Podemos observar la diferencia entre vectores y factores utilizando **summary()**

```
> summary(survey)
   Length     Class      Mode
        5 character character
> summary(factor_survey)
Female   Male
     2      3
```

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]:
```

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)
```

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, ...)
```

Práctica 10

Guarda en una variable de R el archivo de la red de regulación transcripcional usado en la parte de UNIX

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"
```

Paquetes - dplyr

R permite instalar paquetes que han creado otras personas de la comunidad

```
> install.packages("dplyr")
```

dplyr es un paquete que permite manejar *data frames* de forma más fácil

Sólo es necesario instalarlo una vez, pero debes indicarle a R que lo vas a usar cada que inicies una nueva sesión

```
> library("dplyr")
```

Es el equivalente de tener un programa instalado en tu computadora que abres cada que lo necesitas

dplyr

Los paquetes contienen funciones nuevas – entre las más útiles de dplyr se encuentran:

select()
filter()
mutate()
group_by()
summarize()

select()

Permite seleccionar columnas de un *data frame*. Recibe como argumentos el *data frame* y las columnas a seleccionar

```
> select(trn,TF,Gene)
```

dplyr

filter()
Permite seleccionar columnas de un *data frame*. Recibe como argumentos el *data frame* y la condición para filtrar

```
> filter(trn,Effect=="+")
```

Combinar **filter()** y **select()**
Se logra a través de un *pipe*, donde se toma el resultado de una instrucción para aplicarle una segunda instrucción

```
> trn %>%
+ filter(Effect == “+”) %>%
+ select(TF)
```

¿Cómo harías para guardar el resultado en una variable nueva?

dplyr

mutate()

Permite hacer cálculos y crear una nueva columna con el resultado

```
> mutate(trn,new=paste(TF,Gene,sep="-"))
```

group_by()

Organiza el *data frame* en grupos de acuerdo a una columna

```
> group_by(trn,TF)
```

Es común que se quiera aplicar una función a cada grupo. Esto se hace con **summarize()**

```
> trn %>%
  group_by(Effect) %>%
  summarize(n())
```

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 11

Utiliza dplyr para contestar dos preguntas de la práctica de UNIX:

¿Cuántos reguladores transcripcionales se conocen hasta el momento (columna 1)?

¿Total de genes a quienes se les conoce su regulación (columna 2)?

Queremos obtener el total de genes regulados (columna 2) por cada regulador (columna 1)

Queremos saber el total de reguladores por cada gene. Es decir, cuántos TFs están regulando a cada gene.

¿Preguntas?

Contacto

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