


Guía de Trabajo 1

1. Iniciando

Una vez instalado hay que hacer un doble click en el ícono de  (en Unix/Linux, se escribe R desde el símbolo de comandos (command prompt)). Cuando R se inicia, aparece la ventana del programa “Gui” (graphical user interface) con un mensaje de apertura, por ejemplo (para la última versión de octubre de 2016):

```
R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (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 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.
```

Debajo del mensaje de apertura en la Consola de R se encuentra el “prompt” que es el símbolo > (“mayor”).

La mayoría de las expresiones en R se escriben directamente a continuación del “prompt” en la Consola de R.

Se utiliza antes para escribir comentarios, todo lo que se escribe a continuación de este símbolo R lo interpreta como un comentario.

2. Lenguaje.

2.1 Operaciones básicas entre números reales.

En R pueden realizarse todas las operaciones básicas:

suma: +

resta: -

multiplicación: *

división: /

potenciación: ^

división entera: %/% o %%

Para ejecutarlas basta con escribir la expresión correcta y apretar <enter>. Al finalizar la expresión, y debajo de la misma aparecerá el resultado como se muestra

```
> 2+5
[1] 7
> 2-3
[1] -1
> 2*5
[1] 10
> 2/5
[1] 0.4
> 2^5
[1] 32
> 9^(1/2)
[1] 3
```

2.2 Algunas funciones.

R tiene incorporadas las funciones fundamentales, por ejemplo:

sqrt, sin, cos, exp, log, log10, log2, log(x, base), asin, acos, atan, abs

```
> sqrt(9)
[1] 3
> sin(pi/2)
[1] 1
> cos(pi)
[1] -1
> exp(1)
[1] 2.718282
> abs(2)
[1] 2
> abs(-3)
[1] 3
> round(1/3, 4)
[1] 0.3333
```

2.3 Sucesiones

Se pueden obtener sucesiones de números de diferentes formas:

- 1) El operador *dos puntos* : genera sucesiones de paso 1

```
> 1:5
[1] 1 2 3 4 5
```

#El [1] al comienzo es el índice del primer elemento de la fila.

```
>-1:2  
[1] -1  0  1  2
```

- 2) La función **seq** produce secuencias equi-espaciadas, con paso que se puede determinar. Por default usa paso 1.

```
> seq(1,5)  
[1] 1 2 3 4 5  
> seq(1,5,0.5)  
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

- 3) La función **rep** replica un valor dado la cantidad de veces que se le indique: **rep(valor, longitud)**

```
> rep(2,5)  
[1] 2 2 2 2 2
```

R trabaja con **OBJETOS**, por lo cual es conveniente asignar un nombre a cualquier elemento con el que queramos trabajar. Cada vez que lo invoquemos R ejecutará lo que le indiquemos al objeto nombrado. Para dar un nombre usamos el **operador de asignación**.

2.4 Operador de asignación "<-" : nos permite asignar nombre/ valor/formato a un objeto

```
> x<-seq(1,5)  
> x  
[1] 1 2 3 4 5
```

En este caso estamos indicando que a la sucesión `seq(1,5)` la llamamos `x`. (O bien, que a la variable `x` le estamos asignando la sucesión).

No se pueden poner espacios entre el "menor" y el "menos" en el **operador <- de asignación**

Se distinguen mayúsculas de minúsculas. Por ejemplo, si ahora le pedimos que nos muestre X

```
> X  
Error: objeto 'X' no encontrado
```

Genera el mensaje de error porque nosotros llamamos `x` (minúscula) al objeto y no asignamos `X` a nadie

Cada vez que creamos un objeto R lo identifica con alguno de los tipos de objeto con que trabaja: Vectores, Matrices, Data.Frames, Listas, Arreglos. Siempre hay que tener claro qué tipo de objeto se generó para saber cómo operar con él y sobre todo como R lo interpretará cuando le pidamos que ejecute una sentencia.

3. VECTORES

3.1 Para generar un vector usamos la función `c` que concatena objetos.

```
> x<-c(1,3,5,7)
> x
[1] 1 3 5 7
```

3.2 Operaciones con vectores

Los vectores se pueden sumar, restar, multiplicar por un escalar, con las operaciones habituales.

Multiplicación:

```
> x<-seq(1,5)
> x
[1] 1 2 3 4 5
> y<-seq(1,10,2)
> y
[1] 1 3 5 7 9
>x*y
[1] 1 6 15 28 45 # Multiplica componente a componente
> x%*%y
      [,1]
[1,] 95      # Realiza el producto escalar entre vectores.
```

Cuando los vectores tienen la misma longitud, R realiza las operaciones básicas "+", "-", "*", "^" componente a componente. Si queremos que multiplique los vectores en forma escalar debemos encerrar el * entre los signos %

¿Qué ocurre cuando los vectores no tienen la misma longitud?

```
> w<-c(1,2,3)
> x+w
[1] 2 4 6 5 7
Warning message:
In x + w :
  longitud de objeto mayor no es múltiplo de la longitud de uno menor
```

Hay que tener cuidado porque realiza la suma pero como le falta una componente repite la primera.

Los vectores pueden ser numéricos o también pueden ser lógicos o de caracteres:

```
x <- c(T,F,F,T) # un vector lógico contiene True o False en cada componente.
> x
[1] TRUE FALSE FALSE TRUE
```

Un vector de caracteres es el que resulta de unir cadenas de caracteres, que deben indicarse entre comillas :

```
x<-"Hoy es lunes" # vector de caracteres de longitud 1
y<-c("Juan", "Tomas", "Sol") # vector de caracteres de longitud 3
```

3.3 Algunas funciones sobre vectores

```
>x<-seq(1,5)
> length(x) # devuelve la dimension del vector
[1] 5
> dim(x) #da la dimensión de matrices y arreglos, no de vectores.
NULL
> is.vector(x) #preguntamos si el objeto que generamos es un vector
[1] TRUE
> rev(x) # invierte el orden de los elementos del vector
[1] 5 4 3 2 1
> sort(c(6,9,2,7,1)) # ordena los elementos en orden
ascendente
[1] 1 2 6 7 9
> sort(c("Juan", "Tomas", "Sol"))
[1] "Juan" "Sol" "Tomas"
> order((c("Juan", "Tomas", "Sol")))
[1] 2 3 1
# indica la componente del vector original que queda en cada lugar al
ordenar.
> x<-c("Juan", "Tomas", "Sol")
```

Los elementos de un vector se llaman con el nombre del vector y entre corchetes el número de componente que queremos llamar:

```
x[2]
[1] "Tomas"
```

Observar que:

```
> x[order(x)]
[1] "Juan" "Sol" "Tomas" # genera el mismo resultado que sort(x)
```

Observación: Los vectores admiten valores de un único modo. Si se intenta crear un vector con componentes con distinto modo el R fuerza los elementos a un modo común y no da error:

```
> x<-c(1,4,TRUE,"hola")
> x
[1] "1" "4" "TRUE" "hola" #R interpreta al vector como carácter
```

Con la función **data.class** podemos ver de qué tipo es un objeto que creamos:

```
data.class(x)
[1] "character"
```

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

R interpreta al vector como numérico transformando TRUE en 1

```
>data.class(x)
[1] "numeric"
```

```
> w<-c(1,4,FALSE)
> w
[1] 1 4 0
```

R interpreta al vector como numérico transformando TRUE en 0

4. MATRICES

4.1 Generando Matrices

a) A partir de un vector dado

```
> x<-1:12
> A<-matrix(x,nrow=3,ncol=4)
> A
      [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Las siguientes sentencias generan la misma matriz :

```
> A<-matrix(x,nrow=3)
> A<-matrix(x,ncol=4)
```

Alcanza con especificar el número de filas o columnas. Por defecto R arma la matriz con los valores del vector por columna, primero la primera columna, luego la segunda, etc . Para armar una matriz por fila hay que utilizar el argumento "byrow=T".

```
B<-matrix(x,nrow=3,byrow=T)
> B
      [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

b) La función *scan* permite entrar datos en forma interactiva en la ventana de comandos, separados por espacio o por <enter>. El ingreso datos se interrumpe con “dos veces enter”.

```
> M<-matrix(scan(),ncol=3)
1: 1 2 3
4: 4 5 6
7: 7 8 9
10:
Read 9 items
> M
      [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

c) Como en a) pero creando el vector en la misma sentencia:

```
> M<- matrix(c(2,3,1,2,4,5,2,4,3),ncol=3)
> M
      [,1] [,2] [,3]
[1,]    2    2    2
[2,]    3    4    4
[3,]    1    5    3
```

d) Con las funciones *cbind* *rbind*

```
> col1<-c(1,2,3)
> col2<-c(0,1,1)
> A<-cbind(col1,col2)
> A
      col1 col2
[1,]    1    0
[2,]    2    1
[3,]    3    1
```

“Pegamos” los vectores en columnas

```
> B<-rbind(col1,col2)
> B
      [,1] [,2] [,3]
col1    1    2    3
col2    0    1    1
```

“Pegamos” los vectores en filas

4.2. Atributos de una matriz

Las matrices tienen los siguientes atributos: cantidad de filas, cantidad de columnas, dimensión, longitud y modo.

Se accede a los valores de los atributos mediante las funciones: **nrow()**, **ncol()**, **dim()**, **length()** y **mode()** respectivamente.

OBSERVACIÓN MUY IMPORTANTE: las matrices, igual que los vectores, deben contener elementos de un *mismo modo*.

4.3 Operaciones algebraicas.

Las operaciones algebraicas (+, -, *, /, ^, etc) aplicadas a matrices se realizan **componente a componente**, igual que para los vectores.

Nuevamente hay que tener cuidado, porque si las matrices no tienen igual dimensión, la de menor dimensión es expandida.

```
> A<-matrix(1:10,ncol=5)
> A
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10

> A+c(0,1)
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    3    5    7    9   11
```

Si se quiere multiplicar en forma matricial se debe usar %*%:

```
> A<-matrix(1:12,ncol=4)
> A
      [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

> B<-matrix(seq(1,24,3),ncol=2)
> B
      [,1] [,2]
[1,]    1   13
[2,]    4   16
[3,]    7   19
[4,]   10   22

> A%*%B
      [,1] [,2]
[1,]   166  430
[2,]   188  500
[3,]   210  570
```


Traspuesta

```
> M
      [,1] [,2] [,3]
[1,]     2     2     2
[2,]     3     4     4
[3,]     1     5     3
> t(M)
      [,1] [,2] [,3]
[1,]     2     3     1
[2,]     2     4     5
[3,]     2     4     3
```

Determinante

```
> det(M)
[1] -4
```

Inversa

```
> solve(M)
      [,1] [,2] [,3]
[1,]  2.00 -1   0.0
[2,]  1.25 -1   0.5
[3,] -2.75  2  -0.5
```

Podemos obtener más información sobre matrices, utilizando el help: **help(matrix)** o bien **?matrix**

4.4 Acceso a los elementos de las matrices.

Como para los vectores, los corchetes también permiten el acceso a los elementos de una matriz.

Una coma dentro de los corchetes separa los índices de filas y columnas.

```
> A[1,3] # devuelve el elemento de la fila 1 y columna 3

> A[1,]  # devuelve la primer fila de la matriz

> A[,4]  # devuelve la cuarta columna de la matriz.

> A[7]   # devuelve el septimo elemento de la matriz
recorriéndola por columnas.
```

4.5. Operaciones por filas o columnas.

La función **apply** permite realizar un cálculo por fila ó por columna.

```
> A<-matrix(1:20,ncol=5)
> A
      [,1] [,2] [,3] [,4] [,5]
[1,]     1     5     9    13    17
[2,]     2     6    10    14    18
[3,]     3     7    11    15    19
[4,]     4     8    12    16    20
```

```
> apply(A,2,sum) #Sumamos las columnas de la matriz A
[1] 10 26 42 58 74

> apply(A,1,sum) #Sumamos las filas de la matriz A
[1] 45 50 55 60
```

Si A es una matriz de $n \times p$ entonces

```
> apply(A,2,sum)
```

Se obtiene un vector de longitud p (cantidad de columnas) con las sumas de las columnas. El argumento 2 en "**apply(A,2,sum)**" indica que el cálculo debe realizarse en la segunda dimensión.

```
> apply(A,1,sum)
```

Se obtiene un vector de longitud n (cantidad de filas) con las sumas de las filas.

Con la función **apply()** se pueden calcular varianzas, medianas, sumas, productos, ..etc. Casi cualquier función que calcule un valor resumen de un vector:

```
> apply(A,2,mean) # medias por columnas
> apply(A,1,var) # varianzas por filas
> sqrt(apply(A,2,var)) # desvíos estándar por columna
```

4.6. Nombres de las filas y columnas de una matriz.

Una forma de agregar nombres a las filas y columnas de una matriz A es con **rownames()** y **colnames()**

```
> notas <- c(10, 9, 10, 9,8, 10)
> notas <- matrix(notas, 3, 2, byrow = T)
> notas
      [,1] [,2]
[1,]   10    9
[2,]   10    9
[3,]    8   10

> rownames(notas)<-c("Juan", "Ana","Pedro")

> notas
      [,1] [,2]
Juan    10    9
Ana     10    9
Pedro    8   10

> colnames(notas)<-c("Parcial", "Final")

> notas
      Parcial Final
Juan       10     9
Ana        10     9
Pedro        8    10
```

Una matriz tiene un atributo llamado **dimnames()**,

Si A es una matriz entonces `dimnames(A)` es una lista con dos elementos. El primer elemento es un vector de cadenas de caracteres de longitud `nrow(A)` que contiene los nombres de las filas. El segundo elemento de `dimnames(A)` es un vector conteniendo los nombres de las columnas, es un vector de cadenas de caracteres de longitud `ncol(A)`

```
> dimnames(notas)
[[1]]
[1] "Juan" "Ana" "Pedro"

[[2]]
[1] "Parcial" "Final"
```

Cuando las filas o las columnas tienen nombres, es posible referirse a filas o columnas específicas por su nombre:

```
> notas["Juan",]
Parcial Final
    10     9

> notas[, "Final"]
Juan Ana Pedro
    9    9    10
```

4.7 Otras funciones de interés:

```
> x<- 1:12
> x
[1] 1 2 3 4 5 6 7 8 9 10 11 12

> dim(x)
NULL # da NULL porque x es un vector

> x<-as.matrix(x) #transformamos a x en una matriz
> dim(x) # dim nos devuelve las dimensiones de la matriz
[1] 12 1

> is.matrix(x) # preguntamos si es una matriz
[1] TRUE

> data.class(x) # preguntamos qué tipo de objeto es
[1] "matrix"
```

5. DATA FRAMES

Es un arreglo rectangular con las filas representando unidades muestrales y las columnas representando variables. Es similar a una matriz pero es más general ya que, a diferencia de las matrices, **las columnas pueden tener diferentes modos de almacenamiento**. Pero tienen la restricción de que **todas las columnas deben tener la misma longitud**.

5.1 Generando data frames

```
> x1<-c("Juan", "Sol", "Tomás")
> x2<-c(10, 9, 8)
```

```
> x3<-c(9,9,7)
> x4<-c(TRUE,FALSE,TRUE)
> Notas<-data.frame(x1,x2,x3,x4)
> Notas
  x1  x2  x3  x4
1 Juan 10  9 TRUE
2 Sol  9  9 FALSE
3 Tomás 8  7 TRUE
```

Al igual que las matrices se puede poner nombre a las columnas usando colnames:

```
> colnames(Notas)<-c("Nombre","Primer parcial","Segundo
Parcial","Tutoría")
> Notas
  Nombre Primer parcial Segundo Parcial Tutoría
1   Juan              10              9     TRUE
2    Sol              9              9    FALSE
3  Tomás              8              7     TRUE
```

Observación: Si en alguna variable falta algún valor, debe completarse con NA (Not Available) en R indica valor faltante para que todos los vectores que forman las columnas tengan igual longitud.

Ejemplo: intentamos armar un data frame con los vectores:

```
x<-c(1,1,1,1)
y<-c(2,2,2)
> z<-data.frame(x,y)
Error in data.frame(x, y) :
  arguments imply differing number of rows: 4, 3
```

Nos avisa que no puede armar el data frame porque las columnas tienen distinta longitud. Agregamos a y una componente NA:

```
y<-c(2,2,2,NA)
> z<-data.frame(x,y)
> z
  x  y
1 1  2
2 1  2
3 1  2
4 1 NA
```

5.2 Trabajando con data frames

El manejo de data frames tiene muchas cosas en común con las matrices:

- tienen los atributos "dim", "nrow", "ncol"
- se puede invocar a sus elementos utilizando corchetes: `z[1,2]`, `z[,1]`, `z[2,]`

Para los data frames se puede también acceder a las a través de sus nombres: es igual pedir

Notas[,1] que pedir **Notas\$Nombre**

```
> Notas[,1]
[1] Juan Sol Tomás
Levels: Juan Sol Tomás
```

```
> Notas$Nombre
[1] Juan Sol Tomás
Levels: Juan Sol Tomás
```

Observación:

```
> Notas$Primer Parcial
Error: unexpected symbol en "Notas$Primer Parcial"
```

No acepta espacios en los nombres (si para nombrarlos, no para leerlos)

```
> Notas$Primer
[1] 10 9 8
```

Se puede usar el comando `apply` a sus columnas:

```
> apply(Notas[,2:3],2, mean)
Primer parcial Segundo Parcial
9.000000 8.333333
```

Pero si alguna no es numérica:

- Las lógicas las transforma a numéricas asignando 0 a False y 1 a True
- Si alguna es tipo carácter no va a realizar el `apply` sobre ninguna. En este caso conviene aplicarlo a las columnas numéricas por separado.

Incluimos la columna 4, "Tutoría", que es lógica:

```
> apply(Notas[,2:4],2,mean)
Primer parcial Segundo Parcial Tutoría
9.000000 8.333333 0.666667
```

Incluimos la columna 1, "Nombre", que es carácter:

```
> apply(Notas,2,mean)
Nombre Primer parcial Segundo Parcial Tutoría
NA NA NA NA
```

Mensajes de aviso perdidos

```
1: In mean.default(newX[, i], ...) :
argument is not numeric or logical: returning NA
2: In mean.default(newX[, i], ...) :
argument is not numeric or logical: returning NA
3: In mean.default(newX[, i], ...) :
argument is not numeric or logical: returning NA
4: In mean.default(newX[, i], ...) :
argument is not numeric or logical: returning NA
```

Podemos transformar una matriz en un data frame:

```
> A<-matrix(1:10,2,5)
> A
      [,1] [,2] [,3] [,4] [,5]
[1,]     1     3     5     7     9
[2,]     2     4     6     8    10
> data.class(A)
[1] "matrix"
> Adf<-data.frame(A)
> data.class(Adf)
[1] "data.frame"
```

Otra forma:

```
> B<-matrix(1:15,ncol=3)
> data.class(B)
[1] "matrix"
> Bdf<-as.data.frame(B)
> data.class(Bdf)
[1] "data.frame"
```

5.3 Cálculos sobre subgrupos de datos

Los data.frames nos permiten calcular medidas como el apply pero por categorías indicadas en una columna. Por ejemplo, agreguemos a Notas la variable Sexo

```
> sexo<-c("M", "F", "M")
> Notas2<-data.frame(x1,x2,x3,sexo)

> colnames(Notas2)<-c("Nombre", "Primer parcial", "Segundo
Parcial", "Género")

> Notas2
  Nombre Primer parcial Segundo Parcial Género
1   Juan             10             9      M
2    Sol              9             9      F
3  Tomás              8             7      M
```

Calculemos el promedio en cada uno de los parciales por género

```
> tapply(Notas2[,2],Notas2[,4],mean)
F M
9 9
```

La estructura general es: **tapply(x,y,función)**

Donde:

x: columna del data frame a quien se quiere calcular la función

y: columna del data frame que indica las categorías en las cuales separar a x.

función: la función a aplicar a x en cada categoría de y.

x debe ser un vector, no acepta data.frame. Por ej, si queremos las notas del primer y segundo parcial debemos usar dos tapply, **no se puede en uno solo**:

```
> tapply(Notas2[,2:3],Notas2[,4],mean)
Error in tapply(Notas2[, 2:3], Notas2[, 4], mean) :
  arguments must have same length
```

Observación: Cuando leemos un archivo (por ejemplo de texto o un archivo Excel), R lo guarda como un data frame.

6. LISTAS

Una lista es una colección de objetos pegados de manera que pueden ser indicados por un único nombre. Pueden ser de distinto tipo y de distinta longitud

6.1 Generando una lista

Puede utilizarse la función **list()** como en este ejemplo:

```
> x <- matrix(1:4, 4, 7)
> y <- "Hoy es jueves"
> z <- c(T, F, T, NA)
> lista <- list(x, y, z)

> lista
[[1]]
      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    1    1    1    1    1    1
[2,]    2    2    2    2    2    2    2
[3,]    3    3    3    3    3    3    3
[4,]    4    4    4    4    4    4    4

[[2]]
[1] "Hoy es jueves"

[[3]]
[1]  T  F  T NA
```

“lista” tiene tres elementos. El primero es una matriz, el segundo es una cadena de caracteres, y el tercero es un vector lógico. Se puede acceder a los elementos individuales de la lista utilizando doble corchetes:

```
> lista[[2]]
[1] " Hoy es jueves"
```

6.2 Trabajando con Listas

- Se puede asignar nombres con **names()**:

```
> names(lista)<- c("Matriz", "Leyenda","Vector Logico")
```

```
> lista
$Matriz
  [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]   1   1   1   1   1   1   1
[2,]   2   2   2   2   2   2   2
[3,]   3   3   3   3   3   3   3
[4,]   4   4   4   4   4   4   4

$Leyenda
[1] "Hoy es jueves"

$`Vector Logico`
[1] TRUE FALSE TRUE NA
```

- Para referirse a un elemento de la lista, además del uso de los corchetes, tenemos dos formas más si hay nombres asignados:

```
> lista$Leyenda
[1] "Hoy es jueves "
> lista[["Leyenda"]]
[1] "Hoy es jueves "
```

- Como los vectores, las listas tienen asociada *longitud* y no dimensión. Los nombres de la lista constituyen un vector de caracteres de igual longitud que la lista

```
> length(lista)
[1] 3

> names(lista)
[1] "Matriz"          "Leyenda"          "Vector Logico"
```

- Podemos agregar o eliminar elementos a una lista

```
> lista2<-lista[-1]# Eliminamos el primer elemento

lista2
$Leyenda
[1] "Hoy es jueves"

$`Vector Logico`
[1] T F T NA

> lista[[4]]<-c(1,2,3) # Agregamos una nueva componente a nuestra
lista de 3 elementos
> lista
$Matriz
  [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]   1   1   1   1   1   1   1
[2,]   2   2   2   2   2   2   2
[3,]   3   3   3   3   3   3   3
[4,]   4   4   4   4   4   4   4

$Leyenda
[1] "Hoy es jueves"
```



```
$`Vector Logico`
[1] TRUE FALSE TRUE NA

[[4]]
[1] 1 2 3      #ahora lista tiene 4 elementos
```

6.3 Una lista particular: dimnames

Las matrices y data frames tienen un atributo llamado **dimnames()**.

Si A es una matriz o un data frame entonces **dimnames(A)** es una *lista con dos elementos*. El primer elemento es un vector de cadenas de caracteres de longitud `nrow(A)` que *contiene los nombres de las filas*. El segundo elemento de `dimnames(A)` es un vector *conteniendo los nombres de las columnas*, es un vector de cadenas de caracteres de longitud `ncol(A)`

Podemos agregar nombres a las filas y columnas de una matriz (no de un data frame) creando esa lista y asignándosela a `dimnames(A)`.

```
> notas <- c(10, 9, 10, 9, 8, 10)
> notas <- matrix(notas, 3, 2, byrow = T)
> notas
      [,1] [,2]
[1,]   10   9
[2,]   10   9
[3,]    8  10

> dimnames(notas) <- list(c("Juan", "Ana", "Pedro"), c("Parcial",
"Final"))

> notas
      Partial Final
Juan       10     9
Ana        10     9
Pedro       8    10
Si se quiere dar nombres unicamente a las columnas ó a las filas
utilice NULL:

> dimnames(notas) <- list(NULL, c("Parcial", "Final"))
> notas
      Partial Final
[1,]       10     9
[2,]       10     9
[3,]        8    10

> dimnames(notas) <- list(c("Juan", "Ana", "Pedro"), NULL)
> notas
      [,1] [,2]
Juan    10   9
Ana     10   9
Pedro   8   10
```

7. Cómo guardar los objetos si salimos de la sesión

7.1 El espacio de trabajo – Workspace

Todas las variables u “objetos” creados en R están guardados en lo que se llama el espacio de trabajo *workspace*. Para ver que variables están en el espacio de trabajo puede usarse la función **ls()**(esta función no necesita argumentos entre los paréntesis).

```
> ls() # fíjese cuáles son los objetos que ha creado.
```

7.2 Guardar el Área de Trabajo

El programa pregunta si quiere guardar la imagen de su espacio de trabajo (workspace image). Si se clickea en “yes” todos los objetos (los nuevos creados en la sesión actual y en las anteriores) serán guardados y estarán disponibles en la siguiente sesión.

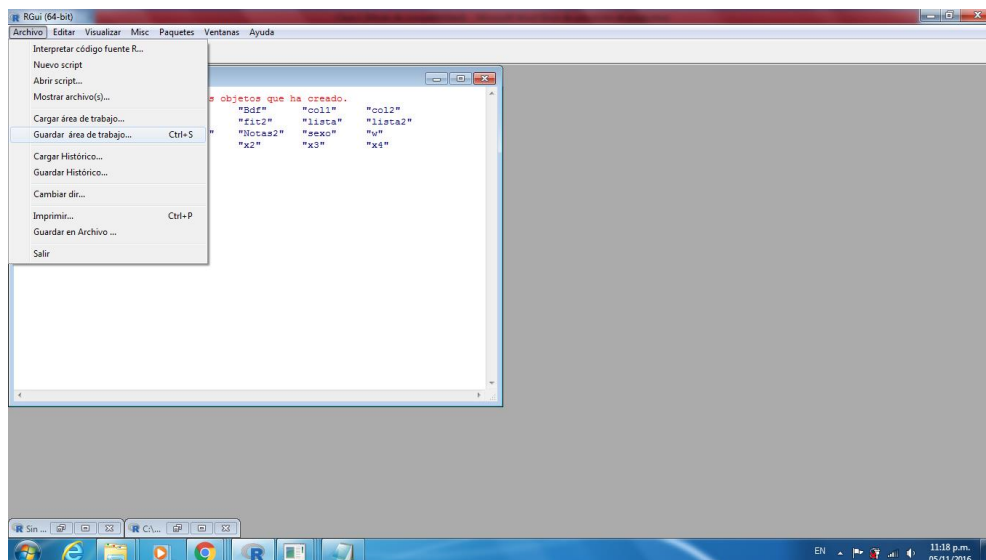
Si clickea en “no”, se perderán todos los objetos nuevos y el espacio de trabajo será restaurado al último momento en que la imagen fuera guardada. Es recomendable salvar el trabajo.

Más conveniente aún es *guardar el espacio de trabajo en una carpeta específica*, por ejemplo en C:\ Curso de R. Para ello en el **menú principal** de R, seleccionar

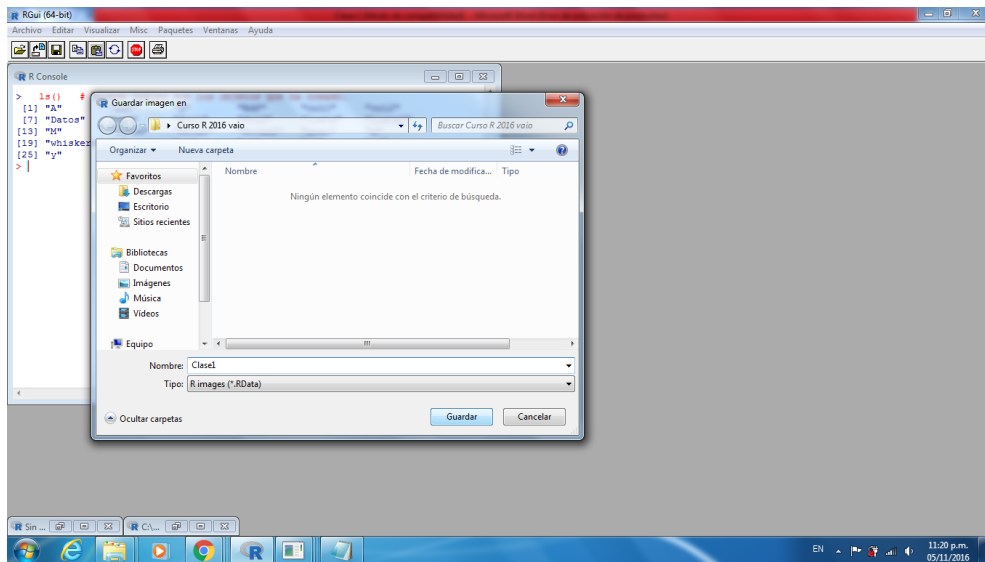
Archivo -> Guardar área de Trabajo

O

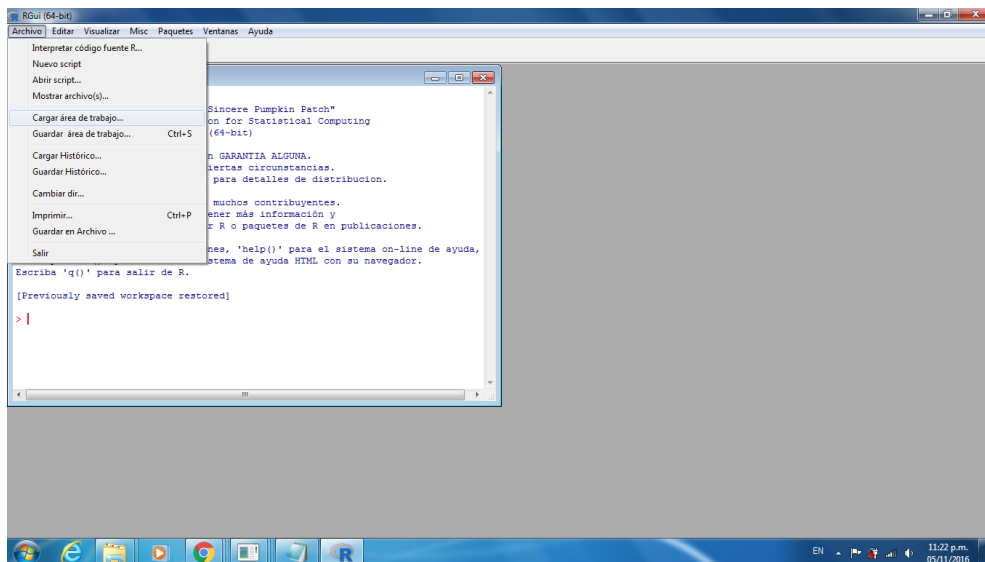
File -> Save Workspace



Se abre un navegador y se puede elegir la carpeta donde guardar el espacio de trabajo.



Cuando iniciamos una nueva sesión y queremos recuperar los objetos creados anteriormente debemos Cargar el área de Trabajo guardada:



Si lo guardamos con la extensión R.Data, guarda la sesión con el icono de R, esto permite abrir el programa desde esa carpeta y con el espacio de trabajo allí guardado.

