

UNIVERSIDAD DE EL SALVADOR

FACULTAD MULTIDISCIPLINARIA DE OCCIDENTE



Universidad de El Salvador

Hacia la libertad por la cultura

PRACTICA 3 y 4 DE R

DOCENTE:

LIC. JAIME ISAAC PEÑA

PRESENTADO POR:

CRISTIAN ALBERTO ZALDAÑA ALVARADO

Índice

1. PRÁCTICA 3: Factores, listas y hojas de datos, operadores y funciones que operan sobre ellos.	3
1.1. FACTORES NOMINALES Y ORDINALES.	3
1.1.1. FACTORES NOMINALES.	3
1.1.2. FACTORES ORDINALES	4
1.2. CREACIÓN Y MANEJO DE LISTAS.	4
1.3. CREACIÓN Y MANEJO DE HOJAS DE DATOS (DATA FRAME).	7
1.3.1. ACCESO A LAS COMPONENTE O VARIABLES DE UNA HOJA DE DATOS. . .	11
1.3.2. TRABAJO CON HOJAS DE DATOS	12
2. PRÁCTICA 4: Importación y exportación de datos en R	12
2.1. USO DE LA FUNCIÓN READ.TABLE().	12
2.2. USO DE LA FUNCIÓN SCAN().	13
2.3. USO DE LA FUNCIÓN READ.CSV().	14
2.4. USO DEL PAQUETE RODBC.	15
2.5. IMPORTAR DATOS DE SPSS HACIA R.	16

1. PRÁCTICA 3: Factores, listas y hojas de datos, operadores y funciones que operan sobre ellos.

1.1. FACTORES NOMINALES Y ORDINALES.

Un factor es un vector utilizado para especificar una clasificación discreta de los elementos de otro vector de igual longitud. En R existen factores nominales y factores ordinales. Los factores son útiles a la hora de querer hacer contrastes o de calcular medidas de resúmenes para variables numéricas en distintos niveles de una segunda variable la cual es no numérica.

1.1.1. FACTORES NOMINALES.

- Ejemplo 1: Variables sexo (categórica) y edad en una muestra de 7 alumnos del curso.

Supongamos que se obtuvieron los siguientes datos: `sexo <- c("M", "F", "F", "M", "F", "F", "M");` `sexo`
`edad <- c(19, 20, 19, 22, 20, 21, 19);` `edad`

```
> sexo<-c("M", "F", "F", "M", "F", "F", "M")
> sexo
```

```
[1] "M" "F" "F" "M" "F" "F" "M"
```

```
> edad<-c(19,20,19,22,20,21,19)
> edad
```

```
[1] 19 20 19 22 20 21 19
```

Podemos construir un factor con los niveles o categorías de sexo

`FactorSexo = factor(sexo);` `FactorSexo`

```
> FactorSexo=factor(sexo)
> FactorSexo
```

```
[1] M F F M F F M
Levels: F M
```

Se pueden ver los niveles o categorías del factor con: `levels(FactorSexo)`

```
> levels(FactorSexo)
```

```
[1] "F" "M"
```

Crear una tabla que contenga la media muestral por categoría de sexo (nivel del factor): `mediaEdad <-`
`tapply(edad, FactorSexo, mean);` `mediaEdad`

```
> mediaEdad<-tapply(edad, FactorSexo, mean)
> mediaEdad
```

```
 F  M
20 20
```

Note que el primer argumento debe ser un vector, que es del cual se encontrarán las medidas de resumen; el segundo es el factor que se está considerando, mientras que en el tercero se especifica la medida de interés, solamente puede hacerse una medida a la vez.

¿Qué tipo de objeto es la variable `mediaEdad`? `is.vector(mediaEdad); is.matrix(mediaEdad); is.list(mediaEdad);`
`is.table(mediaEdad); is.array(mediaEdad)`

```
> is.vector(mediaEdad)

[1] FALSE

> is.matrix(mediaEdad)

[1] FALSE

> is.list(mediaEdad)

[1] FALSE

> is.table(mediaEdad)

[1] FALSE

> is.array(mediaEdad)

[1] TRUE
```

1.1.2. FACTORES ORDINALES

Los niveles de los factores se almacenan en orden alfabético, o en el orden en que se especificaron en la función `factor()` si ello se hizo explícitamente.

A veces existe una ordenación natural en los niveles de un factor, orden que deseamos tener en cuenta en los análisis estadísticos. La función `ordered()` crea este tipo de factores y su uso es idéntico al de la función `factor()`. Los factores creados por la función `factor()` los denominaremos nominales o simplemente factores cuando no haya lugar a confusión, y los creados por la función `ordered()` los denominaremos ordinales. En la mayoría de los casos la única diferencia entre ambos tipos de factores consiste en que los ordinales se imprimen indicando el orden de los niveles. Sin embargo, los contrastes generados por los dos tipos de factores al ajustar Modelos lineales, son diferentes.

1.2. CREACIÓN Y MANEJO DE LISTAS.

Una lista es un objeto que contiene una colección ordenada de objetos de diferente tipo (vector, matriz, arreglo, función, o lista), conocidos como componentes. Se construye con la función `list()`, que tiene la forma general siguiente:

`Lista <- list(nombre1 = objeto1, nombre2 = objeto2, ..., nombren = objeton)`

Si omite los nombres, las componentes sólo estarán numeradas. Las componentes pueden accederse por su número o posición, ya que siempre están numeradas, o también pueden referirse por su nombre, si lo tienen.

- Ejemplo 1: Crear una Lista con cuatro componentes.

```
lista1<-list(padre="Pedro", madre="María", no.hijos=3, edad.hijos=c(4,7,9)) lista1

> lista1<-list(padre="Pedro", madre="María", no.hijos=3, edad.hijos=c(4,7,9))
> lista1

$padre
[1] "Pedro"

$madre
[1] "María"
```

```
$no.hijos  
[1] 3
```

```
$edad.hijos  
[1] 4 7 9
```

Revise algunos tipos como: `is.matrix(lista1)`; `is.vector(lista1$edad.hijos)`

- Ejemplo 2: Acceso a las componentes de una lista:

`lista1[1]` accede a la componente como una lista (con etiqueta y valor) `lista1["padre"]` el acceso es igual que con `lista1[1]` `lista1[[2]]` accede al valor o valores de la componente segunda pero no muestra el nombre de la componente. `lista1["madre"]` el acceso es igual que con `lista1[[1]]`

```
> lista1[1]
```

```
$padre  
[1] "Pedro"
```

```
> lista1["padre"]
```

```
$padre  
[1] "Pedro"
```

```
> lista1[[2]]
```

```
[1] "María"
```

```
> lista1["madre"]
```

```
$madre  
[1] "María"
```

- Ejemplo 3: Acceso a los elementos de la cuarta componente: `lista1[[4]][2]` (se indica el elemento a ingresar en el segundo corchete)

```
> lista1[[4]][2]
```

```
[1] 7
```

- Ejemplo 4: Acceso de las componentes de una lista por su nombre: `lista1$padre` similar a `lista1["padre"]`.

```
> lista1$padre
```

```
[1] "Pedro"
```

Forma general: **Nombre_de_lista\$nombre_de_componente**

Por ejemplo: `lista1$padre` equivale a `lista1[[1]]`; y `lista1$edad.hijos[2]` equivale a `lista1[[4]][2]`.

- Ejemplo 5: Utilizar el nombre de la componente como índice:
`lista1[["nombre"]]` se puede ver que equivale a `lista1$nombre`
También es útil la forma: `x <- "nombre"; lista1[x]`

```
> lista1[["nombre"]]
```

```
NULL
```

- Ejemplo 6: Creación de una sublista de una lista existente:
`subLista <- lista1[4]; subLista`

```
> sublista<-lista1[4]
> sublista

$edad.hijos
[1] 4 7 9
```

- Ejemplo 7: Ampliación de una lista: por ejemplo, la lista `lista1` tiene 4 componentes y se le puede agregar una quinta componente con:
`lista1[5] <- list(sexo.hijos=c("F", "M", "F")); lista1`

```
> lista1[5] <- list(sexo.hijos=c("F", "M", "F"))
> lista1

$padre
[1] "Pedro"

$madre
[1] "María"

$no.hijos
[1] 3

$edad.hijos
[1] 4 7 9

[[5]]
[1] "F" "M" "F"
```

Observe que no aparece el nombre del objeto agregado, pero usted puede modificar la estructura de la lista `lista1` con: `lista1 <- edit(lista1)`

Nota: Se puede aplicar la función `data.entry()` para modificar la estructura de una lista.

- Ejemplo 8: Funciones que devuelven una lista.

Las funciones y expresiones de R devuelven un objeto como resultado, por tanto, si deben devolver varios objetos, previsiblemente de diferentes tipos, la forma usual es una lista con nombres. Por ejemplo, la función `eigen()` que calcula los autovalores y autovectores de una matriz simétrica.

Ejecute las siguientes instrucciones:

```
S <- matrix(c(3, -sqrt(2), -sqrt(2), 2), nrow=2, ncol=2); S
autovS <- eigen(S); autovS

> S <- matrix(c(3, -sqrt(2), -sqrt(2), 2), nrow=2, ncol=2)
> S

      [,1]      [,2]
[1,] 3.000000 -1.414214
[2,] -1.414214 2.000000

> autovS <- eigen(S)
> autovS
```

```
eigen() decomposition
$values
[1] 4 1

$vectors
      [,1]      [,2]
[1,] -0.8164966 -0.5773503
[2,]  0.5773503 -0.8164966
```

Observe que la función `eigen()` retorna una lista de dos componentes, donde la componente `autovvalues` es el vector de autovalores de `S` y la componente `autovvectors` es la matriz de los correspondientes auto-vectores. Si quisiéramos almacenar sólo los autovalores de `S`, podemos hacer lo siguiente:

```
evals <- eigen(S)$values; evals
```

```
> evals<-eigen(S)$values
> evals
```

```
[1] 4 1
```

- Ejemplo 9: Crear una matriz dando nombres a las filas y columnas

```
Notas <- matrix(c(2, 5, 7, 6, 8, 2, 4, 9, 10), ncol=3,
dimnames=list(c("Matemática", "Álgebra", "Geometría"),
c("Juan", "José", "René"))); Notas
```

Los nombres se dan primero para filas y luego para columnas.

```
> Notas <- matrix(c(2, 5, 7, 6, 8, 2, 4, 9, 10), ncol=3,
+ dimnames=list(c("Matemática", "Álgebra", "Geometría"),
+ c("Juan", "José", "René")))
> Notas
```

	Juan	José	René
Matemática	2	6	4
Álgebra	5	8	9
Geometría	7	2	10

1.3. CREACIÓN Y MANEJO DE HOJAS DE DATOS (DATA FRAME).

Una hoja de datos (data frame) es una lista que pertenece a la clase "data.frame". Un data.frame puede considerarse como una matriz de datos. Hay restricciones en las listas que pueden pertenecer a esta clase, en particular:

- Los componentes deben ser vectores (numéricos, cadenas de caracteres, o lógicos), factores, matrices numéricas, listas u otras hojas de datos.
- Las matrices, listas, y hojas de datos contribuyen a la nueva hoja de datos con tantas variables como columnas, elementos o variables posean, respectivamente.
- Los vectores numéricos y los factores se incluyen sin modificar, los vectores no numéricos se fuerzan a factores cuyos niveles son los únicos valores que aparecen en el vector.
- Los vectores que constituyen la hoja de datos deben tener todos la misma longitud, y las matrices deben tener el mismo tamaño de filas.

Las hojas de datos pueden interpretarse, en muchos sentidos, como matrices cuyas columnas pueden tener diferentes modos y atributos. Pueden imprimirse en forma matricial y se pueden extraer sus filas o columnas mediante la indexación de matrices. En una hoja de datos cada columna corresponde a una variable y cada fila a un elemento del conjunto de observaciones.

- Ejemplo 1: Creación de un data frame teniendo como columnas tres vectores:

En primer lugar generamos los tres vectores

El primer vector tendrá 20 elementos que se obtienen con reemplazamiento de una muestra aleatoria de valores lógicos.

```
log <- sample(c(TRUE, FALSE), size = 20, replace = T); log  
Note que puede usar T en lugar de TRUE y F en lugar de FALSE.
```

```
> log <- sample(c(TRUE, FALSE), size = 20, replace = T)  
> log  
  
[1] FALSE FALSE FALSE TRUE FALSE TRUE TRUE FALSE FALSE TRUE TRUE TRUE  
[13] TRUE FALSE TRUE TRUE TRUE TRUE FALSE FALSE
```

El segundo vector tendrá 20 elementos de valores complejos cuya parte real proviene de una distribución Normal estándar y cuya parte imaginaria lo hace de una distribución Uniforme(0,1)

```
comp <- rnorm(20) + runif(20) (1i); comp  
  
> comp <- rnorm(20) + runif(20) * (1i)  
> comp  
  
[1] 0.0159406+0.5796206i -1.4896932+0.9902378i 1.3365527+0.2466562i  
[4] -1.4402613+0.1816044i -1.1307199+0.3260516i -0.8543347+0.0357109i  
[7] 1.2268921+0.3086174i 1.0396053+0.5465419i -1.1211581+0.9845849i  
[10] -0.6239029+0.0976011i 0.1953949+0.7864661i 0.0916286+0.9375789i  
[13] -1.3554043+0.7940257i 0.6301781+0.9154478i 0.1484154+0.0767450i  
[16] 0.5493649+0.2095209i -0.1927275+0.6246449i 0.0500429+0.7485399i  
[19] -1.8508197+0.5515432i 0.0990660+0.1797361i
```

El tercer vector tendrá 20 elementos de una distribución Normal estándar

```
num <- rnorm(20, mean=0, sd=1); num  
  
> num <- rnorm(20, mean=0, sd=1)  
> num  
  
[1] -0.74897306 -1.14028686 -0.52080427 -0.52822623 0.91437715 -0.56976626  
[7] 0.38839638 -0.50643029 -0.21894639 0.09917573 -0.50894513 -1.08559979  
[13] 2.04918579 1.04916197 2.39842686 0.50029071 0.84728444 0.03318790  
[19] 1.18665706 -0.42913789
```

Crear un data frame compuesto por los tres vectores anteriores

```
df1 <- data.frame(log, comp, num); df1  
  
> df1<-data.frame(log, comp, num)  
> df1
```

	log	comp	num
1	FALSE	0.0159406+0.5796206i	-0.74897306
2	FALSE	-1.4896932+0.9902378i	-1.14028686
3	FALSE	1.3365527+0.2466562i	-0.52080427
4	TRUE	-1.4402613+0.1816044i	-0.52822623
5	FALSE	-1.1307199+0.3260516i	0.91437715
6	TRUE	-0.8543347+0.0357109i	-0.56976626
7	TRUE	1.2268921+0.3086174i	0.38839638
8	FALSE	1.0396053+0.5465419i	-0.50643029
9	FALSE	-1.1211581+0.9845849i	-0.21894639


```
10 TRUE -0.6239029+0.0976011i 0.09917573
11 TRUE 0.1953949+0.7864661i -0.50894513
12 TRUE 0.0916286+0.9375789i -1.08559979
13 TRUE -1.3554043+0.7940257i 2.04918579
14 FALSE 0.6301781+0.9154478i 1.04916197
15 TRUE 0.1484154+0.0767450i 2.39842686
16 TRUE 0.5493649+0.2095209i 0.50029071
17 TRUE -0.1927275+0.6246449i 0.84728444
18 TRUE 0.0500429+0.7485399i 0.03318790
19 FALSE -1.8508197+0.5515432i 1.18665706
20 FALSE 0.0990660+0.1797361i -0.42913789
```

Crear un vector de nombres de los tres vectores anteriores

```
nombres <- c("logico", "complejo", "numerico")
```

```
> nombres<-c("logico", "complejo", "numerico")
```

Define los nombres de las columnas del data frame asignándoles el vector nombres

```
names(df1) <- nombres; df1
```

```
> names(df1)<-nombres
```

```
> df1
```

	logico	complejo	numerico
1	FALSE	0.0159406+0.5796206i	-0.74897306
2	FALSE	-1.4896932+0.9902378i	-1.14028686
3	FALSE	1.3365527+0.2466562i	-0.52080427
4	TRUE	-1.4402613+0.1816044i	-0.52822623
5	FALSE	-1.1307199+0.3260516i	0.91437715
6	TRUE	-0.8543347+0.0357109i	-0.56976626
7	TRUE	1.2268921+0.3086174i	0.38839638
8	FALSE	1.0396053+0.5465419i	-0.50643029
9	FALSE	-1.1211581+0.9845849i	-0.21894639
10	TRUE	-0.6239029+0.0976011i	0.09917573
11	TRUE	0.1953949+0.7864661i	-0.50894513
12	TRUE	0.0916286+0.9375789i	-1.08559979
13	TRUE	-1.3554043+0.7940257i	2.04918579
14	FALSE	0.6301781+0.9154478i	1.04916197
15	TRUE	0.1484154+0.0767450i	2.39842686
16	TRUE	0.5493649+0.2095209i	0.50029071
17	TRUE	-0.1927275+0.6246449i	0.84728444
18	TRUE	0.0500429+0.7485399i	0.03318790
19	FALSE	-1.8508197+0.5515432i	1.18665706
20	FALSE	0.0990660+0.1797361i	-0.42913789

Define los nombres de las filas del data frame asignándoles un vector de 20 elementos correspondientes a las 20 primeras letras del abecedario

```
row.names(df1) <- letters[1:20]; df1
```

```
> row.names(df1)<-letters[1:20]
```

```
> df1
```

	logico	complejo	numerico
a	FALSE	0.0159406+0.5796206i	-0.74897306
b	FALSE	-1.4896932+0.9902378i	-1.14028686

```
c FALSE 1.3365527+0.2466562i -0.52080427
d TRUE -1.4402613+0.1816044i -0.52822623
e FALSE -1.1307199+0.3260516i 0.91437715
f TRUE -0.8543347+0.0357109i -0.56976626
g TRUE 1.2268921+0.3086174i 0.38839638
h FALSE 1.0396053+0.5465419i -0.50643029
i FALSE -1.1211581+0.9845849i -0.21894639
j TRUE -0.6239029+0.0976011i 0.09917573
k TRUE 0.1953949+0.7864661i -0.50894513
l TRUE 0.0916286+0.9375789i -1.08559979
m TRUE -1.3554043+0.7940257i 2.04918579
n FALSE 0.6301781+0.9154478i 1.04916197
o TRUE 0.1484154+0.0767450i 2.39842686
p TRUE 0.5493649+0.2095209i 0.50029071
q TRUE -0.1927275+0.6246449i 0.84728444
r TRUE 0.0500429+0.7485399i 0.03318790
s FALSE -1.8508197+0.5515432i 1.18665706
t FALSE 0.0990660+0.1797361i -0.42913789
```

Ejemplo 2: Vamos a crear la siguiente hoja de datos que tiene 4 variables o columnas:

Edad Estatura Peso Sexo

```
1 18 150 65 F
2 21 160 68 M
3 45 180 65 M
4 54 205 69 M
```

```
> edad <- c(18, 21, 45, 54)
> edad
[1] 18 21 45 54

> datos <- matrix(c(150, 160, 180, 205, 65, 68, 65, 69), ncol=2, dimnames=list(c(),
+ c("Estatura", "Peso")))
> datos
      Estatura Peso
[1,]      150   65
[2,]      160   68
[3,]      180   65
[4,]      205   69

> sexo <- c("F", "M", "M", "M")
> sexo
[1] "F" "M" "M" "M"

> hoja1 <- data.frame(Edad=edad, datos, Sexo=sexo)
> hoja1
  Edad Estatura Peso Sexo
1   18      150   65    F
2   21      160   68    M
3   45      180   65    M
4   54      205   69    M
```

Para editar o agregar datos, o componentes utilice: `fix(hoja1)`

Nota: Puede forzar que una lista, cuyos componentes cumplan las restricciones para ser una hoja de datos, realmente lo sea, mediante la función `as.data.frame()`

1.3.1. ACCESO A LAS COMPONENTE O VARIABLES DE UNA HOJA DE DATOS.

Normalmente para acceder a la componente o variable Edad de la hoja de datos se utilizará la expresión `hoja1$Edad`, pero existe una forma más sencilla, consiste en conectar la hoja de datos para que se pueda hacer referencia a sus componentes directamente por su nombre.

Conexión de listas o hojas de datos.

La función `search()` busca y presenta qué hojas de datos, listas o bibliotecas han sido conectadas o desconectadas. Teclee `search()`

```
> search()

[1] ".GlobalEnv"      "package:stats"    "package:graphics"
[4] "package:grDevices" "package:utils"    "package:datasets"
[7] "package:methods" "Autoloads"        "package:base"
```

La función `attach()` es la función que permite conectar en la trayectoria de búsqueda no sólo directorios, listas y hojas de datos, sino también otros tipos de objetos. Teclee `attach(hoja1)` y luego `search()`

```
> attach(hoja1)
> search()

[1] ".GlobalEnv"      "hoja1"            "package:stats"
[4] "package:graphics" "package:grDevices" "package:utils"
[7] "package:datasets" "package:methods"   "Autoloads"
[10] "package:base"
```

Luego puede acceder a las componentes por su nombre:

```
Edad
hoja1$Peso <- Peso+1; hoja1
```

```
> Edad

[1] 18 21 45 54
```

```
> hoja1$Peso<-Peso+1
> hoja1
```

	Edad	Estatura	Peso	Sexo
1	18	150	66	F
2	21	160	69	M
3	45	180	66	M
4	54	205	70	M

Posteriormente podrá desconectar el objeto utilizando la función `detach()`, utilizando como argumento el número de posición o, preferiblemente, su nombre. Teclee `detach(hoja1)` y compruebe que la hoja de datos ha sido eliminada de la trayectoria de búsqueda con `search()`.

```
> detach(hoja1)
> search()

[1] ".GlobalEnv"      "package:stats"    "package:graphics"
[4] "package:grDevices" "package:utils"    "package:datasets"
[7] "package:methods" "Autoloads"        "package:base"
```

Pruebe si puede acceder a una componente sólo con su nombre, por ejemplo, Teclee `Edad`

```
> # Edad, no se puede acceder
```

1.3.2. TRABAJO CON HOJAS DE DATOS

- Una metodología de trabajo para tratar diferentes problemas utilizando el mismo directorio de trabajo es la siguiente:
- Reúna todas las variables de un mismo problema en una hoja de datos y dé le un nombre apropiado e informativo;
- Para analizar un problema, conecte, mediante `attach()`, la hoja de datos correspondiente (en la posición 2) y utilice el directorio de trabajo (en la posición 1) para los cálculos y variables temporales;
- Antes de terminar un análisis, añada las variables que deba conservar a la hoja de datos utilizando la forma `$` para la asignación y desconecte la hoja de datos mediante `detach()`;
- Para finalizar, elimine del directorio de trabajo las variables que no desee conservar, para mantenerlo lo más limpio posible.

De este modo podrá analizar diferentes problemas utilizando el mismo directorio, aunque todos ellos compartan variables denominadas `x`, `y` o `z`, por ejemplo.

2. PRÁCTICA 4: Importación y exportación de datos en R

Generalmente los datos suelen leerse desde archivos externos y no teclearse desde la consola. Las capacidades de lectura de archivos de R son sencillas y sus requisitos son bastante estrictos, por lo que hay que tenerlas muy en cuenta, de lo contrario los resultados en la lectura no serán los esperados.

2.1. USO DE LA FUNCIÓN `READ.TABLE()`.

Ejemplo: Guardar (escribir) determinados datos en un archivo de texto (ASCII) y luego recuperar (leer) dicho archivo desde R.

1. Cambiar el directorio de trabajo a su directorio de trabajo, en el cual ha almacenado sus prácticas, desde el menú File.
2. Abrir el R Editor para crear un nuevo script desde el menú File.
3. En la ventana del R Editor, teclee los datos tal como se muestra:

Observaciones:

- La primera línea del archivo debe contener el nombre de cada objeto o variable.
 - En cada una de las siguientes líneas, el primer elemento es la etiqueta de la fila, y a continuación deben aparecer los valores de cada variable.
 - Si el archivo tiene un elemento menos en la primera línea que en las restantes, obligatoriamente será el diseño anterior el que se utilice.
 - A menudo no se dispone de etiquetas de filas. En ese caso, también es posible la lectura y el programa añadirá unas etiquetas predeterminadas.
 - La última línea debe finalizar con ENTER para que R reconozca el fin del archivo.
4. Oprimir con el puntero del ratón el icono que representa un disquete (Save script as) y guarde el archivo con el nombre "datos01.txt". También puede darle el nombre de "datos01.dat" (otro formato soportado por la función `read.table()`), e incluso puede leer datos directamente desde una página de internet, solamente proporcionando la dirección URL completa.

También puede realizar estos pasos utilizando un editor de texto como NotePad o WordPad.

5. Recuperar los objetos o datos guardados en el archivo "datos01.txt"

```
Entrada1 <- read.table("datos01.txt", header=T);Entrada1
Entrada2 <- read.table("datos01.dat", header=T);Entrada2
No existe diferencia entre ambos archivos a la hora de leerlos
```

```
> Entrada1 <- read.table("datos01.txt", header=T)
> Entrada1
```

	Edad	Estatura	Peso	Sexo
1	26	1.65	146	F
2	21	1.73	158	M
3	21	1.81	167	M
4	20	1.70	152	F

```
> Entrada2 <- read.table("datos01.dat", header=T)
> Entrada2
```

	Edad	Estatura	Peso	Sexo
1	26	1.65	146	F
2	21	1.73	158	M
3	21	1.81	167	M
4	20	1.70	152	F

2.2. USO DE LA FUNCIÓN SCAN().

La función `scan()` es más flexible que `read.table()` y permite realizar lecturas más complejas, como puede consultar en la ayuda: `help(scan)`

- Ejemplo 1: Leer sólo las dos primeros objetos o columnas del archivo "datos01.txt"

```
Edat1 <- scan("datos01.txt", list(X1=0, X2=0), skip = 1, flush = TRUE, quiet = TRUE);Edat1
Edat2<- scan("datos01.dat", list(X1=0, X2=0), skip = 1, flush = TRUE, quiet = TRUE);Edat2
```

Observe que en `list(X1=0, X2=0)` se les da el nombre a las dos primeras columnas o variables (puede darle el nombre que crea más conveniente) y se indica que son variables numéricas; sin embargo, del archivo únicamente se leen las dos primeras columnas, si se quisiera leer las columnas primera y tercera, nos veríamos obligados a leer las tres primeras.

Note que si escribimos `list(0, 0)`, indica que se leerán las dos primeras columnas del archivos y que los datos leídos son numéricos (asigna nombres por defecto). Para indicar que los datos que se leen son cadenas se utiliza "" en lugar de 0.

```
> Edat1 <- scan("datos01.txt", list(X1=0, X2=0), skip = 1, flush = TRUE, quiet = TRUE)
> Edat1
```

```
$X1
[1] 26 21 21 20
```

```
$X2
[1] 1.65 1.73 1.81 1.70
```

```
> Edat2<- scan("datos01.dat", list(X1=0, X2=0), skip = 1, flush = TRUE, quiet = TRUE)
> Edat2
```

```
$X1  
[1] 26 21 21 20
```

```
$X2  
[1] 1.65 1.73 1.81 1.70
```

- Ejemplo 2: Crear un archivo con la función `cat()` y luego recuperarlo
`cat("TITULO Línea extra", "2 3 5 7", "11 13 17", file="datos02.txt", sep="\n")`

El archivo lo recuperamos con la función `scan()`:
`pp <- scan("datos02.txt", skip = 1, quiet= TRUE)`

```
> cat("TITULO Línea extra", "2 3 5 7", "11 13 17", file="datos02.txt", sep="\n")  
> pp <- scan("datos02.txt", skip = 1, quiet= TRUE)  
> pp  
  
[1] 2 3 5 7 11 13 17
```

La función `scan` es muy útil cuando en el archivo de datos a importar cada línea representa un único caso. En caso contrario (cada cierta cantidad de columnas representa un caso) es mucho más fácil y recomendable utilizar la función `read.table`.

2.3. USO DE LA FUNCIÓN `READ.CSV()`.

Leer un conjunto de datos de Microsoft Excel pero los datos no están almacenados en el formato conocido de Excel “.xls”, sino más bien un formato menos conocido como “.csv”.

1. Ingresar al Microsoft Excel y crear la hoja de datos siguiente:

Observe que debe guardar la hoja Excel en su directorio de trabajo y que el archivo debe ser de tipo: CSV(delimitado por comas)

2. Regresar al entorno de R y recuperar el archivo "HojaE1.csv".

```
> hojaR <- read.csv("HojaE1.csv", sep = ";", strip.white = TRUE)  
> hojaR
```

	Producto	Cantidad.S1	Cantidad.S2	Cantidad.S3	Cantidad.S4
1	Desayunos	132	125	142	120
2	Almuerzos	120	125	122	114
3	Cenas	115	105	130	108
4	Tazas de caf\xe9	200	180	210	140
5	Gaseosas	75	90	62	80

Note que R ha reemplazado “-” en los encabezados de las columnas por “.”; en general reemplazará cualquier carácter.

Puede investigar el tipo de objeto que es `hojaR` con:

```
> is.matrix(hojaR)  
  
[1] FALSE  
  
> is.list(hojaR)  
  
[1] TRUE
```

```
> is.data.frame(hojaR)
```

```
[1] TRUE
```

Acceda a la componente Producto de hojaR con:

```
> hojaR$Producto
```

```
[1] "Desayunos"      "Almuerzos"      "Cenas"          "Tazas de caf\xe9"
[5] "Gaseosas"
```

Observe que R toma esta columna (variable de caracteres) como un Factor Nominal, verifíquelo tecleando:

```
> is.vector(hojaR$Producto)
```

```
[1] TRUE
```

```
> is.factor(hojaR$Producto)
```

```
[1] FALSE
```

¿Qué tipo de objeto es la columna Cantidad.S1?

```
> is.vector(hojaR$Cantidad.S1)
```

```
[1] TRUE
```

```
> is.factor(hojaR$Cantidas.S1)
```

```
[1] FALSE
```

2.4. USO DEL PAQUETE RODBC.

Si por el contrario los datos a los cuales deseamos realizar el análisis estadístico se encuentran en formato XLS (versión 2003 de Microsoft Excel), debemos de seguir los siguientes pasos (Ilustraremos el procedimiento con el archivo “contaminación_mexico.xls”):

- Instalar el paquete RODBC, con la siguiente instrucción `install.packages(c("RODBC"))` o desde el menú como en el caso de la instalación del paquete Foreign.
Con este procedimiento se instalan los paquetes directamente desde internet, es necesario para ello contar con una conexión a internet en el momento. Posteriormente se selecciona un mirror (un servidor desde el cual se descargarán los paquetes), y finalmente buscar el paquete deseado del listado.
- Cargar el paquete con la siguiente instrucción: `library(RODBC)`

```
> library(RODBC)
```

- Seleccionar el archivo (el cual puede contener más de una hoja de datos) “contaminación_mexico.xls”, con la instrucción:

```
> #datos.xls <- odbcConnectExcel(file.choose())
```

- Seleccionar la hoja en la cual se encuentran los datos

```
> #datoshoja1.xls <- sqlFetch(datos.xls, "contaminacion_mexico")
```

Con esta instrucción se indica la hoja en la cual se encuentran los datos con los que se desea trabajar (contaminación_mexico) o cargar en R. Siempre es necesario especificarlo.

- Realizar los análisis o cálculos correspondientes.

