

# Ingreso de Datos

*Santiago Lozano*

*29 de marzo de 2020*

## Introducción

Se puede obtener números y datos a través del teclado, portapapeles o de algún archivo externo, sabemos que para una variable con una cantidad de números pequeña, por ejemplo 10, es sencillo crear un vector

```
y <- c(6,7,3,4,8,5,6,2)
```

Tenemos varias opciones en R que se adecuan a tamaño de los datos que se quieran ingresar

## Ingreso de datos desde el teclado (por consola)

La opción `scan()` es útil en estos casos para pegar algunos números en un vector llamado `x`, así,

```
x <- scan()
```

después de esto en la consola aparecerá

1:

el cual actúa como un prompt que permite ingresar su primer número, después de haber ingresado este debemos oprimir enter

1: 6

2:

permitiendo ingresar el segundo número, de manera sucesiva realizamos este procedimiento, cuando hayamos ingresado nuestro último número oprimimos 2 veces enter para acabar el vector de números

1: 6

2: 7

3: 3

4: 4

5: 8

6: 5

7: 6

8: 2

9:

Read 8 items

También podemos usar `scan()` para pegar columnas de números desde el portapapeles, así, en una hoja de cálculo seleccione la columna de números de interés y cópiela, cuando aplique

```
x <- scan()
```

y aparezca

1:

en el 1. pegue la columna que estaba en el portapapeles

Aquí va img1

```
> x <- scan()
1: 2
   5
   3
```

Y Presionando enter

```
> x <- scan()
1: 2
2: 5
3: 3
4:
Read 3 items
```

Si queremos agregar más elementos desde el portapapeles, copiamos el conjunto de número a copiar y pegar, y lo ponemos sobre lo que ya tenemos

aquí va img2

```
> x <- scan()
1: 2
   5
   3
   4
   5
   4
```

Y oprimimos enter y

```
> x <- scan()
1: 2
2: 5
3: 3
4: 4
5: 5
6: 4
7:
Read 6 items
```

Copiando por filas se obtiene el mismo resultado

Otra forma de ingresar datos, pero ya de todo tipo, usamos la función `readline()`

```
mi.nombre <- readline(prompt="Ingrese su nombre: ")
mi.edad <- readline(prompt="Ingrese su edad: ")
```

corriendo la línea de `mi.nombre` obtendremos

```
> mi.nombre <- readline(prompt="Ingrese su nombre: ")
Ingrese su nombre:
```

Donde efectivamente ingresaríamos nuestro nombre

```
> mi.nombre <- readline(prompt="Ingrese su nombre: ")
Ingrese su nombre: Santiago
```

```
> mi.edad <- readline(prompt="Ingrese su edad: ")
Ingrese su edad: 26
```

Pero veamos que en esta última línea las variables se guardan como `character`

```
> typeof(mi.edad)
```

```
[1] "character"
```

con lo que un valor entero como una edad debe ser guardado con tipo integer

```
my.age <- as.integer(my.age)
```

y podemos imprimir finalmente

```
print(paste("Hola","mi nombre es",mi.nombre,"mi edad es",mi.edad,"años"))
```

obteniendo

```
[1] "Hola mi nombre es Santiago mi edad es 26 años"
```

## Ingreso de datos desde archivos

Una manera sencilla de obtener datos en R es trayéndolos desde algún archivo sobre el cuál usted obtuvo los datos, sobre distintas estructuras de consignación de datos (dataframes, textos, etc), en esta parte veremos ciertos tipos de formatos de datos consignados y como importarlos para cada caso.

## El directorio de trabajo

Es importante establecer un lugar en su PC bajo el cual R accederá y escribirá de una manera sencilla, para que en posteriores códigos, se puedan sólo mencionar el nombre del archivo y no toda la ruta. Para establecer el directorio de trabajo usamos la función `setwd()`

```
setwd("C:/Users/santiago/Documents/Progrmación en R/2020-I/PR10-Ingreso de Datos")
```

recuerde que aquí hay que cambiar los (backslash) por /.

Si estamos trabajando desde R y queremos saber cuál es nuestro directorio de trabajo usamos la función `getwd()`

```
getwd()
```

```
## [1] "C:/Users/santiago/Documents/Progrmación en R/2020-I/PR10-Ingreso de Datos"
```

El directorio de trabajo por defecto que trabajamos en R cuando no le hemos establecido el directorio de trabajo

Muchas veces queremos volver a nuestro directorio de trabajo por lo que es necesario guardar nuestro directorio anterior, R nos da la posibilidad de guardar este directorio de trabajo en una variable

```
dicr <- getwd()
```

y para volver al directorio de trabajo por defecto que es mis documentos hacemos

```
setwd("~")
```

y

```
getwd()
```

obtendremos

```
> getwd()
```

```
[1] "C:/Users/santiago/Documents"
```

y si queremos después volver al directorio que usabamos anteriormente

```
setwd(dicr)
```

si queremos ver los archivos que tenemos en nuestro directorio de trabajo, con el fin de asegurarnos que cierto archivo se encuentre en esa carpeta, hacemos

```
dir(dicr)

## [1] "1.png" "2.png"
## [3] "3-04-2020" "3.png"
## [5] "bowens.csv" "daphnia.txt"
## [7] "Datos ingreso de datos" "Datos ingreso de datos.zip"
## [9] "ex.txt" "img1.PNG"
## [11] "img2.PNG" "img3.PNG"
## [13] "Intima_ftable.txt" "Intima_Media2.txt"
## [15] "murders.txt" "PR10-Ingreso de Datos.Rmd"
## [17] "PR10-Ingreso_de_Datos.aux" "PR10-Ingreso_de_Datos.log"
## [19] "PR10-Ingreso_de_Datos.out" "PR10-Ingreso_de_Datos.pdf"
## [21] "PR10-Ingreso_de_Datos.Rmd" "PR10-Ingreso_de_Datos.synctex.gz"
## [23] "PR10-Ingreso_de_Datos.tex" "rt.txt"
## [25] "worms.txt" "yield.txt"
```

Para traer un archivo del directorio de trabajo usando el navegador del computador usamos `file.choose()` el cual puede ser usado en todas la funciones que nos permiten importar datos

```
data<-read.table(file.choose(),header=T)
```

Aquí va img3

## Ingreso de datos con read.table

```
yield <- read.table("yield.txt",header=T)
head(yield)
```

```
##   year wheat barley oats rye corn
## 1 1980   5.9    4.4  4.1 3.8  4.4
## 2 1981   5.8    4.4  4.3 3.7  4.1
## 3 1982   6.2    4.9  4.4 4.1  4.0
## 4 1983   6.4    4.7  4.3 3.7  4.1
## 5 1984   7.7    5.6  4.9 4.7  4.7
## 6 1985   6.3    5.0  4.6 4.6  4.3
```

Si quiere simplificar escritura, puede usar `'read.delim()'` pues esta omite el `header=T`

```
yield2 <- read.delim("yield.txt")
head(yield2)
```

```
##   year wheat barley oats rye corn
## 1 1980   5.9    4.4  4.1 3.8  4.4
## 2 1981   5.8    4.4  4.3 3.7  4.1
## 3 1982   6.2    4.9  4.4 4.1  4.0
## 4 1983   6.4    4.7  4.3 3.7  4.1
## 5 1984   7.7    5.6  4.9 4.7  4.7
## 6 1985   6.3    5.0  4.6 4.6  4.3
```

## Errores comunes en el read.table()

Es importante resaltar que en el `read.table()` podría fallar si hubo algunos espacios en los nombre de las variables de la fila 1 de un dataframe (la fila encabezado), por ejemplo con nombres como Nombre Campo,

pH soya, densidad del clima, o entre palabras con el mismo nivel de factor, es decir la misma característica de nombramiento cualitativo, por ejemplo si una de la característica de Nombre Campo es El cañadulzal, con lo que es óptimo reemplazar los espacios con punto ".", por ejemplo Nombre.Campo, pH.soya, etc, antes de guardar el archivo en Excel o cualquier programa que esté manejando

Algunas cosas a tener en cuenta son:

- La ruta y los nombres de los archivos siempre deben ir en comillas, por ejemplo "c:\\abc.txt".
- El argumento `header=T` dice si el archivo a importar contiene fila encabezado

Una de las causas más comunes de falla a la hora de importar es que el número de variables no coincide con el número de columnas de información, y eso se debe principalmente a que se ha dejado espacios en blanco en los nombres de las variables, por ejemplo

```
nombre dept poblacion hogar propietario carro
```

Es erróneo pues se esperan 6 columnas en nuestro dataframe, pero este sólo contiene 5, una forma correcta de establecerlos es mediante

```
nombre dept poblacion hogar propietario carro
```

Otra forma común de equivocación es que los archivos contienen espacios en blanco donde hay missing data, es necesario que en su archivo de origen se reemplacen estos por NA's

Existen archivos a importar que los nombres de las variables los separan por otro marcador distinto al espacio en blanco, por ejemplo por comas ",", estos archivos son en formato .csv los cuales corresponden a hojas de cálculo, donde cada elemento está separado mediante comas "," o en un caso poco común mediante punto y coma ";".

Para este caso es posible importarlos mediante `read.table()`, usando el argumento `sep=","`, o de forma más rápida y fácil `read.csv()`, también si está separado por tabulaciones (tab), que en nuestro contexto generalmente son cuatro espacios en blanco usamos `sep="\t"`

De esta manera es importante las características que posea su archivo para importar de manera que lo pueda hacer correctamente

```
bowens <- read.table("bowens.csv",header=T,sep=",")
head(bowens)
```

```
##           place east north
## 1      Abingdon   50    97
## 2   Admoor Copse   60    70
## 3    AERE Harwell   48    87
## 4   Agates Meadow   70    73
## 5    Aldermaston   59    65
## 6 Aldermaston Court   60    65
```

Sin embargo es más sencillo con

```
bowens2 <- read.csv("bowens.csv")
head(bowens2)
```

```
##           place east north
## 1      Abingdon   50    97
## 2   Admoor Copse   60    70
## 3    AERE Harwell   48    87
## 4   Agates Meadow   70    73
## 5    Aldermaston   59    65
## 6 Aldermaston Court   60    65
```

## Separadores y puntos decimales

El caracter separador de campos en `read.table()` es `sep=" "`, el cual corresponde a una separación de un espacio en blanco, sin embargo, usted puede usar más espacios para especificar la cantidad de espacios en blanco que los separan, se puede separar también por tab `\t` para un tab, se pueden usar más, por ejemplo `\t\t\t` corresponde a 3 tab, `\n` es una nueva línea.

Vea que todas las opciones de `read.table` tienen por defecto `header=TRUE`

También es importante que usted sepa que marca decimal se está usando en el archivo que va a importar pues R es cuidadoso en esta característica, así, si `dec=","`, está especificando que su marca decimal es coma, puede ser `dec="."` especificando el punto como marca decimal.

## Otras formas de importar

archivos separados por punto y coma

```
read.csv2("c:\\temp\\file.csv")
```

archivos separados por tab con marca decimal coma

```
read.delim2("c:\\temp\\file.txt")
```

## Otras cuestiones sobre importar datos

Cuando los nombres de sus variables usan espacios es útil usar como separadores de variables comas o punto y coma, por ejemplo cuando usamos “Reino Unido” o “Estado Unidos”

Si queremos especificar los nombres de nuestras filas `row.names` entonces una de las columnas del dataframe debe ser la que debemos especificar, lo podemos hacer especificando el número de la columna o también un dato character que diga el nombre de la columna `row.names=1`, `row.names="Pais"`, si `row.names` no se pone, por defecto la filas se ordenan con números en orden

Recordemos también que cuando `read.table()` se usa, las variables de tipo cualitativo quedan como factores, pero podemos especificar ciertas variables que queremos que no las ponga como factores

```
murders <- read.table("murders.txt",header=T)
head(murders)
```

```
##      state population murder region
## 1   Alabama      3615    15.1  South
## 2    Alaska       365    11.3   West
## 3   Arizona     2212     7.8   West
## 4  Arkansas     2110    10.1  South
## 5 California   21198    10.3   West
## 6  Colorado     2541     6.8   West
```

```
attach(murders)
class(region)
```

```
## [1] "factor"
```

```
murder <- read.table("murders.txt",header=T,as.is="region")
```

```
attach(murder)
```

```
## The following object is masked _by_ .GlobalEnv:
##
```

```
##      murder
## The following objects are masked from murders:
##
##      murder, population, region, state
class(region)

## [1] "character"
```

## Importar datos desde la web

Podemos importar datos directamente desde internet usando `read.table()` con la URL del sitio

```
data2 <- read.table("http://www.bio.ic.ac.uk/research/mjcraw/therbook/data/cancer.txt", header=T)
head(data2)

##      death treatment status
## 1         4      DrugA      1
## 2        26      DrugA      1
## 3         2      DrugA      1
## 4        25      DrugA      1
## 5         7      DrugA      1
## 6         6      DrugA      0
```

## Ejercicio

Cree una función en la que utilizando la ruta larga del archivo, sólo especifique el nombre del archivo y este haga todo el procedimiento

### `read.table` usando clicks

```
read.csv(file.choose())
read.csv2(file.choose())
read.delim(file.choose())
read.delim2(file.choose())
```

### `read.table()` desde el portapapeles

```
x <- read.table(file("clipboard"),sep="nt",header=TRUE,dec=" ")
```

## Importar Tablas de contingencia

Las tablas de contingencia son una manera cruzada de expresar la información de una población

	H	M
0	168	140
1	92	45

Estas tablas pueden ser importadas en R de manera sistemática usando `read-fable()`, con lo que

```
Intima.table <- read.fable("Intima_fable.txt",row.var.names=c("GENDER","tobacco"),col.vars=list("alcohol"))
fable(Intima.table)
```

```
##                alcohol nondrinker occasional drinker regular drinker
## GENDER tobacco
## M      non-smoker                6                19                7
##      former smoker                0                9                0
##      smoker                      1                6                5
## F      non-smoker               12               26                2
##      former smoker                3                5                1
##      smoker                      1                6                1
```

Aquí tenemos que `row.var.names` es un vector de caracteres que menciona los nombre de las variables fila, en caso de que estos no puedan ser determinados, `cols.vars` es una lista que da los nombres y niveles de las variables columna, en caso de que puedan ser determinadas, de igual forma esta función también usa `sep=` y `dec=` de la manera anteriormente dicha

## Importar datos usando scan()

Cuando se trata de dataframes `read.table()` es magnífico, pero cuando tratamos con archivos algo engorrosos como `rt.txt` vemos que

```
read.table("rt.txt",header = F)
```

```
Error in scan(file = file, what = what, sep = sep, quote = quote, dec = dec, : line 1 did not have 4 el
```

simplemente `read.table` precisa de una coincidencia con todas las líneas

Pero funciones como `scan()` y `readLines` pueden hacer esta labor

La función `scan()` lee los datos y los consigna en una lista o vector cuando es usado para leer un archivo. Es mucho menos amigable que `read.table()` pero es más flexible y por ende puede leer otras estructuras de archivos.

Por defecto `scan()` asume que se están ingresando datos de tipo double, pero se puede usar el argumento `what`, con el fin de organizar otros tipos de datos que estén en el documento (por ejemplo 'character', 'logical', 'integer', 'complex' o 'list').

Algo importante de `what` es que para especificar el tipo de dato que se quiere leer no se debe poner el nombre del tipo de dato si no una muestra del tipo, por ejemplo, si tenemos un dato de tipo complex, no usamos `complex` sino una muestra, `2+3i`.

Si `what` es una lista, se asume que las líneas del archivo son traídas cada una conteniendo campos, tantos ítems como `length(what)` tengan, es decir, `scan` retorna una lista donde cada elemento de esta lista es un vector con los datos respecto al lugar del dato dado en esa posición.

`what` también provee una forma de lectura de datos en forma columnar, donde si el correspondiente campo es NULL, es decir, vacío (""), corresponde a un dato de tipo character con lo que si en `what` creamos una lista de NULL's con cantidad de campos correspondiente a las columnas que se usan, obtendremos una lista donde cada elemento de esta es un vector con los valores de la respectiva columna

Por defecto tenemos que en `scan()` los campos están separados por espacio en blanco o por tabulación (tab), y si se quiere especificar la opción de separación simplemente se utiliza `sep=", "` para especificar la marca separadora entre comillas, aunque en sí, un campo está siempre delimitados por una marca de cambio de línea, a menos que esté entre comillas.

si `sep` es el establecido por defecto (" "), el caracter \ antes de una commilla hace que esta no tenga su efecto de establecer una frase como un campo ("Ella dijo \"QUE!!\" a el"), pero si `sep` no está establecido por defecto, los campos pueden ser puestos entre comillas en el estilo de un archivo .csv donde los separadores dentro de las comillas son ignorados y la comillas pueden ser puestas dentro del string doblandolas

Sin embargo, si `sep="\n"` se asume que se quiere leer la línea entera



Con `scan()` menudo queremos obviar las filas encabezado o algún texto irrelevante , así, el argumento `skip=1`, por ejemplo ignora la primera línea del documento

Ahora usemos `scan()` para importar un dataframe y veamos la ineficiencia de esta para tales casos

```
sn <- scan("worms.txt",skip=1,what=as.list(rep("",7)))
sn

## [[1]]
## [1] "Nashs.Field"      "Silwood.Bottom"  "Nursery.Field"
## [4] "Rush.Meadow"      "Gunness.Thicket" "Oak.Mead"
## [7] "Church.Field"     "Ashurst"         "The.Orchard"
## [10] "Rookery.Slope"    "Garden.Wood"     "North.Gravel"
## [13] "South.Gravel"     "Observatory.Ridge" "Pond.Field"
## [16] "Water.Meadow"     "Cheapside"       "Pound.Hill"
## [19] "Gravel.Pit"       "Farm.Wood"
##
## [[2]]
## [1] "3.6" "5.1" "2.8" "2.4" "3.8" "3.1" "3.5" "2.1" "1.9" "1.5" "2.9"
## [12] "3.3" "3.7" "1.8" "4.1" "3.9" "2.2" "4.4" "2.9" "0.8"
##
## [[3]]
## [1] "11" "2" "3" "5" "0" "2" "3" "0" "0" "4" "10" "1" "2" "6"
## [15] "0" "0" "8" "2" "1" "10"
##
## [[4]]
## [1] "Grassland" "Arable" "Grassland" "Meadow" "Scrub"
## [6] "Grassland" "Grassland" "Arable" "Orchard" "Grassland"
## [11] "Scrub" "Grassland" "Grassland" "Grassland" "Meadow"
## [16] "Meadow" "Scrub" "Arable" "Grassland" "Scrub"
##
## [[5]]
## [1] "4.1" "5.2" "4.3" "4.9" "4.2" "3.9" "4.2" "4.8" "5.7" "5" "5.2"
## [12] "4.1" "4" "3.8" "5" "4.9" "4.7" "4.5" "3.5" "5.1"
##
## [[6]]
## [1] "F" "F" "F" "T" "F" "F" "F" "F" "F" "T" "F" "F" "F" "F" "T" "T" "T"
## [18] "F" "F" "T"
##
## [[7]]
## [1] "4" "7" "2" "5" "6" "2" "3" "4" "9" "7" "8" "1" "2" "0" "6" "8" "4"
## [18] "5" "1" "3"
```

Viendo aquí que todos los vectores son character

```
sapply(FUN = class, X = sn)

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

Aquí, sabiendo que worms tiene 7 columnas, usamos `what` para especificar este propósito.

Ahora, si yo quiero identificar los tipos de datos de cada columna

```
sn2 <- scan("worms.txt",skip=1,what=list("",0,as.integer(0),"",0,TRUE,as.integer(0)))
sn2

## [[1]]
```

```
## [1] "Nashs.Field"      "Silwood.Bottom"   "Nursery.Field"
## [4] "Rush.Meadow"      "Gunness.Thicket"  "Oak.Mead"
## [7] "Church.Field"     "Ashurst"          "The.Orchard"
## [10] "Rookery.Slope"    "Garden.Wood"      "North.Gravel"
## [13] "South.Gravel"     "Observatory.Ridge" "Pond.Field"
## [16] "Water.Meadow"     "Cheapside"        "Pound.Hill"
## [19] "Gravel.Pit"       "Farm.Wood"
##
## [[2]]
## [1] 3.6 5.1 2.8 2.4 3.8 3.1 3.5 2.1 1.9 1.5 2.9 3.3 3.7 1.8 4.1 3.9 2.2
## [18] 4.4 2.9 0.8
##
## [[3]]
## [1] 11 2 3 5 0 2 3 0 0 4 10 1 2 6 0 0 8 2 1 10
##
## [[4]]
## [1] "Grassland" "Arable"      "Grassland" "Meadow"      "Scrub"
## [6] "Grassland" "Grassland" "Arable"     "Orchard"     "Grassland"
## [11] "Scrub"      "Grassland" "Grassland" "Grassland" "Meadow"
## [16] "Meadow"     "Scrub"      "Arable"     "Grassland" "Scrub"
##
## [[5]]
## [1] 4.1 5.2 4.3 4.9 4.2 3.9 4.2 4.8 5.7 5.0 5.2 4.1 4.0 3.8 5.0 4.9 4.7
## [18] 4.5 3.5 5.1
##
## [[6]]
## [1] FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE TRUE FALSE
## [12] FALSE FALSE FALSE TRUE TRUE TRUE FALSE FALSE TRUE
##
## [[7]]
## [1] 4 7 2 5 6 2 3 4 9 7 8 1 2 0 6 8 4 5 1 3
```

y aquí

```
sapply(FUN = class, X = sn2)
```

```
## [1] "character" "numeric"    "integer"    "character" "numeric"    "logical"
## [7] "integer"
```

Para convertir esta lista en un dataframe usamos `as.data.frame`, con lo que

```
data3 <- as.data.frame(sn)
```

Y para poner los nombres de las variables hacemos, con lo que usando `scan()` sacamos la primera fila y nos ayudamos del argumento `nlines=1`, el cual nos especifica que solo queremos leer la primera línea y quitamos `skip=1`, así

```
header <- scan("worms.txt",nlines=1,what=as.list(rep("",7)))
header
```

```
## [[1]]
## [1] "Field.Name"
##
## [[2]]
## [1] "Area"
##
## [[3]]
```

```
## [1] "Slope"
##
## [[4]]
## [1] "Vegetation"
##
## [[5]]
## [1] "Soil.pH"
##
## [[6]]
## [1] "Damp"
##
## [[7]]
## [1] "Worm.density"
```

Pero lo queremos como vector, así que la función `unlist()` sirve para estos propósitos

```
rn <- unlist(header)
rn
```

```
## [1] "Field.Name" "Area" "Slope" "Vegetation"
## [5] "Soil.pH" "Damp" "Worm.density"
```

Y finalmente agregamos nuestros nombres de columnas

```
names(data3)<-rn
head(data3)
```

```
##      Field.Name Area Slope Vegetation Soil.pH Damp Worm.density
## 1   Nashs.Field  3.6   11  Grassland   4.1    F         4
## 2  Silwood.Bottom 5.1    2   Arable    5.2    F         7
## 3   Nursery.Field 2.8    3  Grassland   4.3    F         2
## 4    Rush.Meadow  2.4    5   Meadow    4.9    T         5
## 5  Gunness.Thicket 3.8    0   Scrub     4.2    F         6
## 6      Oak.Mead  3.1    2  Grassland   3.9    F         2
```

Como vemos, importar un dataframe es demasiado tedioso mediante `scan()`, claramente `read.table()` es mucho mejor

## `scan()` estructuras de datos complicadas

Teniendo en cuenta esta estructura vemos que

TITLE extra line 2 3 5 7 11 13 17

```
scan("ex.txt", skip = 1, quiet = TRUE)
```

```
## [1]  2  3  5  7 11 13 17
```

```
scan("ex.txt", skip = 1)
```

```
## [1]  2  3  5  7 11 13 17
```

donde el la primera no me muestra el Read 7 items y en la segunda si, gracias a 'quiet=TRUE

```
scan("ex.txt", skip = 1, nlines = 1)
```

```
## [1] 2 3 5 7
```

```
scan("ex.txt",skip=1, what = list(0,as.integer(0),3+2i))
```

```
## Warning in scan("ex.txt", skip = 1, what = list(0, as.integer(0), 3 +
## (0+2i))): número de items leídos no es múltiplo del número de columnas
```

```
## [[1]]
## [1] 2 7 17
##
## [[2]]
## [1] 3 11 NA
##
## [[3]]
## [1] 5+0i 13+0i NA
```

```
scan("ex.txt",skip=1, what = list("", "", ""))
```

```
## Warning in scan("ex.txt", skip = 1, what = list("", "", "")): número de
## items leídos no es múltiplo del número de columnas
```

```
## [[1]]
## [1] "2" "7" "17"
##
## [[2]]
## [1] "3" "11" ""
##
## [[3]]
## [1] "5" "13" ""
```

```
138
27 44
19 20 345 48
115 2366
59
```

El archivo `rt` tiene la siguiente estructura, donde cada fila hace referencia a los ID de los vecinos de 5 individuos

así, para importar los datos

```
scan("rt.txt")
```

```
## [1] 138 27 44 19 20 345 48 115 2366 59
```

```
scan("rt.txt",sep="\n")
```

```
## [1] 138 2744 192034548 1152366 59
```

Vemos que cada elemento está separado por el cambio de línea

```
scan("rt.txt",sep="\t")
```

```
## [1] 138 NA NA NA 27 44 NA NA 19 20 345 48 115 2366
## [15] NA NA 59 NA NA NA
```

Vemos que en esta forma de separación es la que mejor nos conviene, pues se conservan la integridad de los números y al menos sabemos que cada 4 números cambiamos de individuo.

Imaginen que queremos una lista donde cada elemento de ella corresponde a un individuo

Para buscar esto procedemos de la siguiente manera

```
length(scan("rt.txt",sep="\n"))
```

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

sabemos que tenemos 5 individuos

```
length(scan("rt.txt",sep="\t"))
```

```
## [1] 20
```

Aquí vemos que tenemos  $20/5=4$  items por individuo, aunque sean NA's

Así, para encontrar los vecinos del primer individuo, hacemos

```
scan("rt.txt",sep="\t")[1:4]
```

```
## [1] 138 NA NA NA
```

Haciendo este procedimiento de forma sistemática hacemos

```
lista <- list()
for(i in 1:5){
  lista[[i]]<-scan("rt.txt",sep="\t",quiet=T)[(4*i-3):(4*i)]
}
lista
```

```
## [[1]]
## [1] 138 NA NA NA
##
## [[2]]
## [1] 27 44 NA NA
##
## [[3]]
## [1] 19 20 345 48
##
## [[4]]
## [1] 115 2366 NA NA
##
## [[5]]
## [1] 59 NA NA NA
```

ahora quitando los NA's y cambiando a número

```
for(i in 1:5){
  lista[[i]]<-as.numeric(na.omit(lista[[i]]))
}
lista
```

```
## [[1]]
## [1] 138
##
## [[2]]
## [1] 27 44
##
## [[3]]
## [1] 19 20 345 48
##
## [[4]]
## [1] 115 2366
##
## [[5]]
## [1] 59
```

O en una línea

```
sapply(1:5, function(i)
as.numeric(na.omit(
scan("rt.txt",sep="\t",quiet=T)[(4*i-3):
(4*i)])))
```

```
## [[1]]
## [1] 138
##
## [[2]]
## [1] 27 44
##
## [[3]]
## [1] 19 20 345 48
##
## [[4]]
## [1] 115 2366
##
## [[5]]
## [1] 59
```

## Lectura de datos con readLines

Una alternativa bastante poderosa diferente a `scan()` es `readLines()`, el cual consigna la lectura de datos por línea, en un vector de caracteres

```
lworms<- readLines("worms.txt")
```

```
## Warning in readLines("worms.txt"): incomplete final line found on
## 'worms.txt'
```

```
lworms

## [1] "Field.Name\tArea\tSlope\tVegetation\tSoil.pH\tDamp\tWorm.density"
## [2] "Nashs.Field\t3.6\t11\tGrassland\t4.1\tF\t4"
## [3] "Silwood.Bottom\t5.1\t2\tArable\t5.2\tF\t7"
## [4] "Nursery.Field\t2.8\t3\tGrassland\t4.3\tF\t2"
## [5] "Rush.Meadow\t2.4\t5\tMeadow\t4.9\tT\t5"
## [6] "Gunness.Thicket\t3.8\t0\tScrub\t4.2\tF\t6"
## [7] "Oak.Mead\t3.1\t2\tGrassland\t3.9\tF\t2"
## [8] "Church.Field\t3.5\t3\tGrassland\t4.2\tF\t3"
## [9] "Ashurst\t2.1\t0\tArable\t4.8\tF\t4"
## [10] "The.Orchard\t1.9\t0\tOrchard\t5.7\tF\t9"
## [11] "Rookery.Slope\t1.5\t4\tGrassland\t5\tT\t7"
## [12] "Garden.Wood\t2.9\t10\tScrub\t5.2\tF\t8"
## [13] "North.Gravel\t3.3\t1\tGrassland\t4.1\tF\t1"
## [14] "South.Gravel\t3.7\t2\tGrassland\t4\tF\t2"
## [15] "Observatory.Ridge\t1.8\t6\tGrassland\t3.8\tF\t0"
## [16] "Pond.Field\t4.1\t0\tMeadow\t5\tT\t6"
## [17] "Water.Meadow\t3.9\t0\tMeadow\t4.9\tT\t8"
## [18] "Cheapside\t2.2\t8\tScrub\t4.7\tT\t4"
## [19] "Pound.Hill\t4.4\t2\tArable\t4.5\tF\t5"
## [20] "Gravel.Pit\t2.9\t1\tGrassland\t3.5\tF\t1"
## [21] "Farm.Wood\t0.8\t10\tScrub\t5.1\tT\t3"
```

Vemos que `readLines` crea un vector donde cada elemento de este es una línea del archivo que se está leyendo,

esta en formato character, y en este caso vemos que la separación está dada en tabulacin \t, el vector como está aquí desplegado está algo sucio, es nuestro deber limpiarlo de manera que logremos extraer la información relevante

```
stworms <- strsplit(lworms, "\t")
stworms
```

```
## [[1]]
## [1] "Field.Name" "Area" "Slope" "Vegetation"
## [5] "Soil.pH" "Damp" "Worm.density"
##
## [[2]]
## [1] "Nashs.Field" "3.6" "11" "Grassland" "4.1"
## [6] "F" "4"
##
## [[3]]
## [1] "Silwood.Bottom" "5.1" "2" "Arable"
## [5] "5.2" "F" "7"
##
## [[4]]
## [1] "Nursery.Field" "2.8" "3" "Grassland"
## [5] "4.3" "F" "2"
##
## [[5]]
## [1] "Rush.Meadow" "2.4" "5" "Meadow" "4.9"
## [6] "T" "5"
##
## [[6]]
## [1] "Gunness.Thicket" "3.8" "0" "Scrub"
## [5] "4.2" "F" "6"
##
## [[7]]
## [1] "Oak.Mead" "3.1" "2" "Grassland" "3.9" "F"
## [7] "2"
##
## [[8]]
## [1] "Church.Field" "3.5" "3" "Grassland"
## [5] "4.2" "F" "3"
##
## [[9]]
## [1] "Ashurst" "2.1" "0" "Arable" "4.8" "F" "4"
##
## [[10]]
## [1] "The.Orchard" "1.9" "0" "Orchard" "5.7"
## [6] "F" "9"
##
## [[11]]
## [1] "Rookery.Slope" "1.5" "4" "Grassland"
## [5] "5" "T" "7"
##
## [[12]]
## [1] "Garden.Wood" "2.9" "10" "Scrub" "5.2"
## [6] "F" "8"
##
## [[13]]
```

```
## [1] "North.Gravel" "3.3"          "1"          "Grassland"
## [5] "4.1"          "F"          "1"
##
## [[14]]
## [1] "South.Gravel" "3.7"          "2"          "Grassland"
## [5] "4"          "F"          "2"
##
## [[15]]
## [1] "Observatory.Ridge" "1.8"          "6"
## [4] "Grassland"        "3.8"          "F"
## [7] "0"
##
## [[16]]
## [1] "Pond.Field" "4.1"          "0"          "Meadow"      "5"
## [6] "T"          "6"
##
## [[17]]
## [1] "Water.Meadow" "3.9"          "0"          "Meadow"
## [5] "4.9"          "T"          "8"
##
## [[18]]
## [1] "Cheapside" "2.2"          "8"          "Scrub"        "4.7"      "T"
## [7] "4"
##
## [[19]]
## [1] "Pound.Hill" "4.4"          "2"          "Arable"        "4.5"
## [6] "F"          "5"
##
## [[20]]
## [1] "Gravel.Pit" "2.9"          "1"          "Grassland"     "3.5"
## [6] "F"          "1"
##
## [[21]]
## [1] "Farm.Wood" "0.8"          "10"         "Scrub"        "5.1"      "T"
## [7] "3"
```

de esta tenemos ya cada palabra separada por un elemento de un vector que a su vez separa cada línea en na lista, tomemos esta lista como un vector

```
vcworms <- unlist(stworms)
vcworms
```

```
## [1] "Field.Name"      "Area"          "Slope"
## [4] "Vegetation"      "Soil.pH"       "Damp"
## [7] "Worm.density"    "Nashs.Field"   "3.6"
## [10] "11"              "Grassland"     "4.1"
## [13] "F"               "4"             "Silwood.Bottom"
## [16] "5.1"             "2"             "Arable"
## [19] "5.2"             "F"             "7"
## [22] "Nursery.Field"   "2.8"           "3"
## [25] "Grassland"       "4.3"           "F"
## [28] "2"               "Rush.Meadow"   "2.4"
## [31] "5"               "Meadow"        "4.9"
## [34] "T"               "5"             "Gunness.Thicket"
## [37] "3.8"             "0"             "Scrub"
```



```
## [40] "4.2"          "F"            "6"
## [43] "Oak.Mead"     "3.1"          "2"
## [46] "Grassland"    "3.9"          "F"
## [49] "2"            "Church.Field" "3.5"
## [52] "3"            "Grassland"    "4.2"
## [55] "F"            "3"            "Ashurst"
## [58] "2.1"          "0"            "Arable"
## [61] "4.8"          "F"            "4"
## [64] "The.Orchard"  "1.9"          "0"
## [67] "Orchard"      "5.7"          "F"
## [70] "9"            "Rookery.Slope" "1.5"
## [73] "4"            "Grassland"    "5"
## [76] "T"            "7"            "Garden.Wood"
## [79] "2.9"          "10"           "Scrub"
## [82] "5.2"          "F"            "8"
## [85] "North.Gravel" "3.3"          "1"
## [88] "Grassland"    "4.1"          "F"
## [91] "1"            "South.Gravel" "3.7"
## [94] "2"            "Grassland"    "4"
## [97] "F"            "2"            "Observatory.Ridge"
## [100] "1.8"          "6"            "Grassland"
## [103] "3.8"          "F"            "0"
## [106] "Pond.Field"   "4.1"          "0"
## [109] "Meadow"       "5"            "T"
## [112] "6"            "Water.Meadow" "3.9"
## [115] "0"            "Meadow"       "4.9"
## [118] "T"            "8"            "Cheapside"
## [121] "2.2"          "8"            "Scrub"
## [124] "4.7"          "T"            "4"
## [127] "Pound.Hill"   "4.4"          "2"
## [130] "Arable"       "4.5"          "F"
## [133] "5"            "Gravel.Pit"   "2.9"
## [136] "1"            "Grassland"    "3.5"
## [139] "F"            "1"            "Farm.Wood"
## [142] "0.8"          "10"           "Scrub"
## [145] "5.1"          "T"            "3"
```

aquí ya tenemos en una lista cada entrada del dataframe, y como este está organizado de manera ordenada, podemos consignarlo en una matriz

```
dim(vcworms) <- c(7,21)
```

```
vcworms
```

```
##      [,1]      [,2]      [,3]      [,4]
## [1,] "Field.Name" "Nashs.Field" "Silwood.Bottom" "Nursery.Field"
## [2,] "Area"       "3.6"       "5.1"       "2.8"
## [3,] "Slope"      "11"        "2"         "3"
## [4,] "Vegetation" "Grassland" "Arable"     "Grassland"
## [5,] "Soil.pH"    "4.1"       "5.2"       "4.3"
## [6,] "Damp"       "F"         "F"         "F"
## [7,] "Worm.density" "4"         "7"         "2"
##      [,5]      [,6]      [,7]      [,8]      [,9]
## [1,] "Rush.Meadow" "Gunness.Thicket" "Oak.Mead" "Church.Field" "Ashurst"
## [2,] "2.4"        "3.8"        "3.1"     "3.5"     "2.1"
## [3,] "5"          "0"          "2"       "3"       "0"
```

```
## [4,] "Meadow"      "Scrub"      "Grassland" "Grassland"  "Arable"
## [5,] "4.9"        "4.2"        "3.9"       "4.2"       "4.8"
## [6,] "T"         "F"         "F"         "F"         "F"
## [7,] "5"         "6"         "2"         "3"         "4"
##      [,10]      [,11]      [,12]      [,13]
## [1,] "The.Orchard" "Rookery.Slope" "Garden.Wood" "North.Gravel"
## [2,] "1.9"       "1.5"       "2.9"       "3.3"
## [3,] "0"        "4"        "10"       "1"
## [4,] "Orchard"   "Grassland" "Scrub"    "Grassland"
## [5,] "5.7"      "5"        "5.2"     "4.1"
## [6,] "F"       "T"       "F"       "F"
## [7,] "9"       "7"       "8"       "1"
##      [,14]      [,15]      [,16]      [,17]
## [1,] "South.Gravel" "Observatory.Ridge" "Pond.Field" "Water.Meadow"
## [2,] "3.7"        "1.8"        "4.1"       "3.9"
## [3,] "2"         "6"         "0"        "0"
## [4,] "Grassland"  "Grassland"  "Meadow"   "Meadow"
## [5,] "4"         "3.8"       "5"        "4.9"
## [6,] "F"        "F"        "T"        "T"
## [7,] "2"        "0"        "6"        "8"
##      [,18]      [,19]      [,20]      [,21]
## [1,] "Cheapside" "Pound.Hill" "Gravel.Pit" "Farm.Wood"
## [2,] "2.2"       "4.4"       "2.9"       "0.8"
## [3,] "8"        "2"        "1"        "10"
## [4,] "Scrub"    "Arable"    "Grassland" "Scrub"
## [5,] "4.7"     "4.5"     "3.5"     "5.1"
## [6,] "T"       "F"       "F"       "T"
## [7,] "4"      "5"      "1"      "3"
```

transponiendo,y quitando la fila encabezado

```
t(vcworms)[-1,]
```

```
##      [,1]      [,2] [,3] [,4]      [,5] [,6] [,7]
## [1,] "Nashs.Field"    "3.6" "11" "Grassland" "4.1" "F"  "4"
## [2,] "Silwood.Bottom" "5.1" "2"  "Arable"    "5.2" "F"  "7"
## [3,] "Nursery.Field"  "2.8" "3"  "Grassland" "4.3" "F"  "2"
## [4,] "Rush.Meadow"    "2.4" "5"  "Meadow"    "4.9" "T"  "5"
## [5,] "Gunness.Thicket" "3.8" "0"  "Scrub"     "4.2" "F"  "6"
## [6,] "Oak.Mead"       "3.1" "2"  "Grassland" "3.9" "F"  "2"
## [7,] "Church.Field"   "3.5" "3"  "Grassland" "4.2" "F"  "3"
## [8,] "Ashurst"        "2.1" "0"  "Arable"    "4.8" "F"  "4"
## [9,] "The.Orchard"     "1.9" "0"  "Orchard"   "5.7" "F"  "9"
## [10,] "Rookery.Slope"  "1.5" "4"  "Grassland" "5"   "T"  "7"
## [11,] "Garden.Wood"    "2.9" "10" "Scrub"     "5.2" "F"  "8"
## [12,] "North.Gravel"  "3.3" "1"  "Grassland" "4.1" "F"  "1"
## [13,] "South.Gravel"  "3.7" "2"  "Grassland" "4"   "F"  "2"
## [14,] "Observatory.Ridge" "1.8" "6"  "Grassland" "3.8" "F"  "0"
## [15,] "Pond.Field"     "4.1" "0"  "Meadow"    "5"   "T"  "6"
## [16,] "Water.Meadow"   "3.9" "0"  "Meadow"    "4.9" "T"  "8"
## [17,] "Cheapside"      "2.2" "8"  "Scrub"     "4.7" "T"  "4"
## [18,] "Pound.Hill"     "4.4" "2"  "Arable"    "4.5" "F"  "5"
## [19,] "Gravel.Pit"     "2.9" "1"  "Grassland" "3.5" "F"  "1"
## [20,] "Farm.Wood"      "0.8" "10" "Scrub"     "5.1" "T"  "3"
```

y ya convirtiendo el dataframe

```
dataf <- as.data.frame(t(vcworms)[-1,])
dataf
```

```
##           V1 V2 V3           V4 V5 V6 V7
## 1     Nashs.Field 3.6 11 Grassland 4.1 F 4
## 2     Silwood.Bottom 5.1 2   Arable 5.2 F 7
## 3     Nursery.Field 2.8 3 Grassland 4.3 F 2
## 4       Rush.Meadow 2.4 5   Meadow 4.9 T 5
## 5   Gunness.Thicket 3.8 0   Scrub 4.2 F 6
## 6       Oak.Mead 3.1 2 Grassland 3.9 F 2
## 7     Church.Field 3.5 3 Grassland 4.2 F 3
## 8       Ashurst 2.1 0   Arable 4.8 F 4
## 9     The.Orchard 1.9 0   Orchard 5.7 F 9
## 10    Rookery.Slope 1.5 4 Grassland 5 T 7
## 11    Garden.Wood 2.9 10   Scrub 5.2 F 8
## 12    North.Gravel 3.3 1 Grassland 4.1 F 1
## 13    South.Gravel 3.7 2 Grassland 4 F 2
## 14 Observatory.Ridge 1.8 6 Grassland 3.8 F 0
## 15      Pond.Field 4.1 0   Meadow 5 T 6
## 16    Water.Meadow 3.9 0   Meadow 4.9 T 8
## 17      Cheapside 2.2 8   Scrub 4.7 T 4
## 18      Pound.Hill 4.4 2   Arable 4.5 F 5
## 19      Gravel.Pit 2.9 1 Grassland 3.5 F 1
## 20      Farm.Wood 0.8 10   Scrub 5.1 T 3
```

y agregando los nombres de las variables

```
names(dataf) <- t(vcworms)[1,]
dataf
```

```
##           Field.Name Area Slope Vegetation Soil.pH Damp Worm.density
## 1     Nashs.Field 3.6 11 Grassland 4.1 F 4
## 2     Silwood.Bottom 5.1 2   Arable 5.2 F 7
## 3     Nursery.Field 2.8 3 Grassland 4.3 F 2
## 4       Rush.Meadow 2.4 5   Meadow 4.9 T 5
## 5   Gunness.Thicket 3.8 0   Scrub 4.2 F 6
## 6       Oak.Mead 3.1 2 Grassland 3.9 F 2
## 7     Church.Field 3.5 3 Grassland 4.2 F 3
## 8       Ashurst 2.1 0   Arable 4.8 F 4
## 9     The.Orchard 1.9 0   Orchard 5.7 F 9
## 10    Rookery.Slope 1.5 4 Grassland 5 T 7
## 11    Garden.Wood 2.9 10   Scrub 5.2 F 8
## 12    North.Gravel 3.3 1 Grassland 4.1 F 1
## 13    South.Gravel 3.7 2 Grassland 4 F 2
## 14 Observatory.Ridge 1.8 6 Grassland 3.8 F 0
## 15      Pond.Field 4.1 0   Meadow 5 T 6
## 16    Water.Meadow 3.9 0   Meadow 4.9 T 8
## 17      Cheapside 2.2 8   Scrub 4.7 T 4
## 18      Pound.Hill 4.4 2   Arable 4.5 F 5
## 19      Gravel.Pit 2.9 1 Grassland 3.5 F 1
## 20      Farm.Wood 0.8 10   Scrub 5.1 T 3
```

ya lo que queda es tomar cada columna y convertirla en su respectivo tipo de dato

Ahora tomemos el archivo `rt.txt`

```
x <- readLines("rt.txt")
x
```

```
## [1] "138\t\t\t"      "27\t44\t\t"      "19\t20\t345\t48" "115\t2366\t\t"
## [5] "59\t\t\t\t"
```

así, esta función desplegó un vector de 5 elemento correspondientes a cada línea del archivo, como podemos ver, cada elemento está separado por tabulaciones, separando cada palabra

```
lx <- strsplit(x,"\t")
lx
```

```
## [[1]]
## [1] "138" "" ""
##
## [[2]]
## [1] "27" "44" ""
##
## [[3]]
## [1] "19" "20" "345" "48"
##
## [[4]]
## [1] "115" "2366" ""
##
## [[5]]
## [1] "59" "" ""
```

```
xl <- lapply(lx,as.numeric)
xl
```

```
## [[1]]
## [1] 138 NA NA
##
## [[2]]
## [1] 27 44 NA
##
## [[3]]
## [1] 19 20 345 48
##
## [[4]]
## [1] 115 2366 NA
##
## [[5]]
## [1] 59 NA NA
```

y terminando de limpiar el archivo

```
xf <- sapply(1:5,function(i) as.numeric(na.omit(xl[[i]])))
xf
```

```
## [[1]]
## [1] 138
##
## [[2]]
## [1] 27 44
##
## [[3]]
## [1] 19 20 345 48
```

```
##
## [[4]]
## [1] 115 2366
##
## [[5]]
## [1] 59
```

## Fallas al usar attach en un dataframe

Traigamos el dataframe a analizar

```
murder <- read.table("murders.txt",header=T,as.is="region")
```

y apliquemos attach()

```
attach(murder)
```

```
## The following object is masked _by_ .GlobalEnv:
##
##      murder
##
## The following objects are masked from murder (pos = 3):
##
##      murder, population, region, state
##
## The following objects are masked from murders:
##
##      murder, population, region, state
```

El suceso en este caso es que tenemos el dataframe `murder` y una variable llamada `murder`, y puede causar el siguiente problema

```
head(murder)
```

```
##      state population murder region
## 1  Alabama      3615   15.1  South
## 2  Alaska       365   11.3   West
## 3  Arizona     2212    7.8   West
## 4  Arkansas     2110   10.1  South
## 5 California   21198   10.3   West
## 6  Colorado     2541    6.8   West
```

```
table(murder)
```

```
table(murder$murder)
```

```
##
## 1.4 1.7 2.3 2.4 2.7 2.9 3 3.1 3.3 4.2 4.3 4.5 5 5.2 5.3
## 1 1 2 1 1 1 1 1 1 2 1 1 2 1 1
## 5.5 6.1 6.2 6.4 6.7 6.8 6.9 7.1 7.4 7.8 8.5 9.3 9.5 9.7 10.1
## 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1
## 10.3 10.6 10.7 10.9 11 11.1 11.3 11.5 11.6 12.2 12.5 13.2 13.9 15.1
## 2 1 1 1 1 2 1 1 1 1 1 1 1 1
```

## Enmascaramiento

Usamos la función `attach` para que se pueda acceder directamente a las variables dentro de un dataframe por su nombre. Técnicamente, esto significa que el dataframe se adjunta a la ruta de búsqueda R, de modo

que R busca el dataframe cuando evalúa una variable

Así que el problema de enmascaramiento ocurre cuando

- Se llama al mismo dataframe dos veces
- puede que usted tenga dos dataframes con el mismo nombre de variable pero distinto

La causa más común de enmascaramiento ocurre con nombres de variables simples como `x` e `y`. Es muy fácil terminar con múltiples variables del mismo nombre dentro de una sola sesión que significan cosas totalmente diferentes.

Así el mensaje de peligro debe alertar al usuario de las variables que se están enmascarando para que se prevenga de usar variables con el mismo nombre, pues esto podría conducir a problemas de hacer algún análisis que con la variable incorrecta y obtener resultados inesperados

Algunas prevenciones para evitar este problema pueden ser

- Usar nombres de variables más largo y explicativos
- no calcular variables con el mismo nombre de variables dentro de un dataframe
- usar siempre `detach` del dataframe después de usarlo
- remover variables calculadas después de usarlas
- usar `$` en vez de `attach()`

## Formatos de input y output

Los formatos para espaciado generalmente van en comillas

`\n` nueva línea

`\r` retorno de carro

`\t` tabulación

`\b` borrado

`\a \f \v`