

Lectura de Datos desde un fichero

Welton Vieira dos Santos

28/2/2020

Leyendo tablas de datos

- **read.table()**: para definir un data frame a partir de una tabla de datos contenida en un fichero
 - Este fichero puede estar guardado en nuestro ordenador o bien podemos conocer su URL. Sea cual sea el caso, se aplica la función al nombre del fichero o la dirección entre comillas

Carga de ficheros local

Como ejemplo, vamos a cargar un fichero de datos llamado *bulls.dat* que se encuentra en la carpeta *datos* de ese ambiente de trabajo.

```
#Cargar la tabla del fichero bulls.dat a la variable datos
datos <- read.table("../data/bulls.dat")
```

```
#Para visualizar el contenido de la variable datos
head(datos, 10)#Las 10 primeras filas
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9
1	1	2200	51.0	1128	70.9	7	0.25	54.8	1720
2	1	2250	51.9	1108	72.1	7	0.25	55.3	1575
3	1	1625	49.9	1011	71.6	6	0.15	53.1	1410
4	1	4600	53.1	993	68.9	8	0.35	56.4	1595
5	1	2150	51.2	996	68.6	7	0.25	55.0	1488
6	1	1225	49.2	985	71.4	6	0.15	51.4	1500
7	1	2250	51.0	959	72.1	7	0.20	54.0	1522
8	1	4000	51.5	1060	69.3	7	0.30	55.6	1765
9	1	1600	50.1	979	71.2	6	0.25	51.5	1365
10	1	1525	49.6	1083	75.8	6	0.30	54.6	1640

```
tail(datos, 10)#Las 10 últimas filas
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9
67	8	1550	51.2	1057	74.8	7	0.10	55.5	1520
68	8	1250	50.8	1040	74.5	6	0.10	55.8	1516
69	8	1350	52.7	1079	75.5	7	0.15	56.1	1595
70	8	1725	51.4	1034	71.2	7	0.10	56.0	1655
71	8	1750	50.7	1012	71.6	6	0.10	54.3	1480
72	8	1450	51.4	997	73.4	7	0.10	55.2	1454
73	8	1200	49.8	991	70.8	6	0.15	54.6	1475
74	8	1425	50.0	928	70.8	6	0.10	53.9	1375
75	8	1250	50.1	990	71.0	6	0.10	54.9	1564
76	8	1500	51.7	992	70.6	7	0.15	55.1	1458

Parámetros de la función `read.table()`

- **header = TRUE**: para indicar si la tabla que importamos tiene una primera fila con los nombres de las columnas. El valor por defecto es **FALSE**
- **col.names = c(...)**: para especificar el nombre de las columnas. No olvidar que cada nombre debe ir entre comillas.
- **sep**: para especificar las separaciones entre columnas en el fichero (si no es un espacio en blanco). Si es así, hay que introducir el parámetro pertinente entre comillas. Para verificar los tipos de separación que proporciona R utilice la función `?read.tables`
- **dec**: para especificar el signo que separa la parte entera de la decimal (si no es un punto. Si es así, hay que introducir el parámetro pertinente entre comillas)

```
df <- read.table("../data/bulls.dat",
                 header = FALSE,
                 col.names = c("breed", "sale_price", "shoulder",
                              "fat_free", "percent_ff", "frame_scale",
                              "back_fat", "sale_height", "sale_weight"))
names(df)

[1] "breed"      "sale_price" "shoulder"   "fat_free"   "percent_ff"
[6] "frame_scale" "back_fat"   "sale_height" "sale_weight"
```

Para verificar que los datos están bien cargados, utilizar la función para ver la estructura del data frame

```
str(df)

'data.frame':   76 obs. of  9 variables:
 $ breed      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ sale_price : int  2200 2250 1625 4600 2150 1225 2250 4000 1600 1525 ...
 $ shoulder   : num  51 51.9 49.9 53.1 51.2 49.2 51 51.5 50.1 49.6 ...
 $ fat_free   : int  1128 1108 1011 993 996 985 959 1060 979 1083 ...
 $ percent_ff : num  70.9 72.1 71.6 68.9 68.6 71.4 72.1 69.3 71.2 75.8 ...
 $ frame_scale: int   7 7 6 8 7 6 7 7 6 6 ...
 $ back_fat   : num   0.25 0.25 0.15 0.35 0.25 0.15 0.2 0.3 0.25 0.3 ...
 $ sale_height: num   54.8 55.3 53.1 56.4 55 51.4 54 55.6 51.5 54.6 ...
 $ sale_weight: int  1720 1575 1410 1595 1488 1500 1522 1765 1365 1640 ...
```

- **stringsAsFactor**: para prohibir la transformación de las columnas de palabras en factores debemos usar **stringsAsFactors=FALSE** (ya que por defecto, R realiza dicha transformación)
- Para importar un fichero de una página web segura (cuyo url empiece con https), no podemos entrar directamente la dirección en `read.table()`; una solución es instalar y cargar el paquete *RCurl* y entonces usar la instrucción `**read.table(textConnection(getURL("url")), ...)`.

Ejemplos de Factores en un DF:

```
df3 <- read.table("https://maitra.public.iastate.edu/stat501/datasets/olive.dat")
str(df3)

'data.frame':   573 obs. of  9 variables:
 $ V1: Factor w/ 10 levels "1","2","3","4",...: 10 1 1 1 1 1 1 1 1 1 ...
 $ V2: Factor w/ 310 levels "1010","1020",...: 310 24 32 294 303 12 294 297 40 27 ...
 $ V3: Factor w/ 176 levels "100","101","102",...: 176 152 150 136 138 145 132 144 142 141 ...
 $ V4: Factor w/ 139 levels "152","156","158",...: 139 59 57 79 73 92 99 96 68 72 ...
 $ V5: Factor w/ 390 levels "6300","6367",...: 390 351 327 388 373 341 366 384 332 336 ...
 $ V6: Factor w/ 345 levels "1000","1002",...: 345 230 265 203 219 230 231 218 248 242 ...
 $ V7: Factor w/ 46 levels "0","10","15",...: 46 21 16 16 35 35 36 34 24 31 ...
```

```
$ V8: Factor w/ 77 levels "0","10","100",...: 77 38 39 41 56 58 48 34 42 61 ...
$ V9: Factor w/ 45 levels "1","10","11",...: 45 22 22 22 29 40 38 22 29 27 ...
```

Como se puede observar, según la base de datos *olive.dat* al cargar en R hay varias columnas de tipo factors. Si no quiere trabajar con factors, basta poner el argumento **stringsAsFactors=FALSE** para solucionar ese inconveniente.

```
df3 <- read.table("https://maitra.public.iastate.edu/stat501/datasets/olive.dat", stringsAsFactors = FALSE)
str(df3)
```

```
## 'data.frame':    573 obs. of  9 variables:
## $ V1: chr  "group.id" "1" "1" "1" ...
## $ V2: chr  "X1" "1075" "1088" "911" ...
## $ V3: chr  "X2" "75" "73" "54" ...
## $ V4: chr  "X3" "226" "224" "246" ...
## $ V5: chr  "X4" "7823" "7709" "8113" ...
## $ V6: chr  "X5" "672" "781" "549" ...
## $ V7: chr  "X6" "36" "31" "31" ...
## $ V8: chr  "X7" "60" "61" "63" ...
## $ V9: chr  "X8" "29" "29" "29" ...
```

Se observa que las columnas ya no son factores. También esa base de datos trae el nombre de las cabeceras. Entonces se puede decir que las importe correctamente.

```
df3 <- read.table("https://maitra.public.iastate.edu/stat501/datasets/olive.dat", stringsAsFactors = FALSE)
str(df3)
```

```
## 'data.frame':    572 obs. of  9 variables:
## $ group.id: int  1 1 1 1 1 1 1 1 1 1 ...
## $ X1      : int  1075 1088 911 966 1051 911 922 1100 1082 1037 ...
## $ X2      : int   75  73  54  57  67  49  66  61  60  55 ...
## $ X3      : int  226 224 246 240 259 268 264 235 239 213 ...
## $ X4      : int 7823 7709 8113 7952 7771 7924 7990 7728 7745 7944 ...
## $ X5      : int  672 781 549 619 672 678 618 734 709 633 ...
## $ X6      : int   36  31  31  50  50  51  49  39  46  26 ...
## $ X7      : int   60  61  63  78  80  70  56  64  83  52 ...
## $ X8      : int   29  29  29  35  46  44  29  35  33  30 ...
```

Ahora perfecto.

Hasta ahora hemos utilizado la función **read.table()**, que es muy básica y sólo carga tablas. Para cargar otros tipos de archivos con extensión *.csv*, *.xls*, *.xlsx*, *.mtb* y *.spss* hay otras funciones más específicas.

Leyendo diferentes tipos de fichero

- **read.csv()**: para importar ficheros en formato CSV.
- **read.xls()** o **read.xlsx()**: para importar hojas de calculos tipo Excel u OpenOffice en formato XLS o XLSX, respectivamente. Se necesita el paquete *xlsx*.
- **read.mtb()**: para importar tablas de datos Minitab. Se necesita el paquete *foreign*.
- **read.spss()**: para importar tablas de datos SPSS. Se necesita el paquete *foreign*.

Los formatos *.mtb* y *.spss* son lenguajes de pago muy utilizados por programas estadísticos de pago muy utilizados anteriormente hasta que salió R.

Es preferible utilizar para guardar las tablas de datos el formato *.csv* porque se trata de un texto plano y no posee una estructura compleja como las otras anteriores comentadas y además evita muchísimos errores a la hora de cargar esos tipos de archivos.

Para mas detalles mirar la ayuda de R:

```
help.search("read")
```

```
starting httpd help server ... done
```

Guardar un Data Frame (DF)

Guardar en formato txt

```
write.table(df3, file = "oliva.txt", dec = ".")
```

Guardar en formato csv

```
write.csv(df3, file = "oliva.csv", sep = ",")
```

```
## Warning in write.csv(df3, file = "oliva.csv", sep = ","): attempt to set 'sep'
## ignored
```

Importante: Lo más aconsejable es al cargar de un fichero bajado de internet, es bueno guardarlo en un formato .text o .csv y despues tratarlo para que se pueda trabajar sin problemas.

```
df4 <- read.table("oliva.txt", dec = ".", header = TRUE)
head(df4)
```

	group.id	X1	X2	X3	X4	X5	X6	X7	X8
1	1	1075	75	226	7823	672	36	60	29
2	1	1088	73	224	7709	781	31	61	29
3	1	911	54	246	8113	549	31	63	29
4	1	966	57	240	7952	619	50	78	35
5	1	1051	67	259	7771	672	50	80	46
6	1	911	49	268	7924	678	51	70	44

Construyendo data frames

Se puede construir un data frame a partir de una vector de datos o variable.

- **data.frame(vector_1, ..., vector_n)**: para construir un data frame a partir de vectores introducidos en el orden en el que queremos disponer las columnas de la tabla
 - R considera del mismo tipo de datos todas las entradas de una columna de un data frame.
 - Las variables tomarán los nombres de los vectores. Estos nombres se pueden especificar en el argumento de **data_frame** entrando una construcción de la forma **nombre_variable=vector**
 - **rownames**: para especificar los identificadores de las filas.
 - También en esta función podemos hacer uso del parámetro **stringAsFactors** para evitar la transformación de las columnas de tipo palabra en factores.
- **fix(DataFrame)**: para crear / editar un data frame con el editor de datos.
- **names(DataFrame)**: para cambiar los nombres de las variables.
- **rownames(DataFrame)**: para modificar los identificadores de las filas. Han de ser todos diferentes.
- **dimnames(DataFrame) = list(vec_nom_fil, vec_nom_col)**: para modificar el nombre de las filas y de las columnas simultáneamente
- **DataFrame[num_fil,] = c(...)**: para añadir una fila a un data frame:

- Las filas que añadimos de esta manera son vectores, y por tanto sus entradas han de ser todas del mismo tipo.
- Si no añadimos las filas inmediatamente siguientes a la última fila del data frame, los valores entre su última fila y las que añadimos quedarán no definidos y aparecerán como NA
- Para evitar el problema anterior, vale más usar la función *rbind()* para concatenar el data frame con al nueva fila.

Ejemplos:

```
#Vectores de notas de asignaturas
Algebra <- c(1,2,0,5,4,6,7,5,5,8)
Analysis <- c(3,3,2,7,9,5,6,8,5,6)
Statistics <- c(4,5,4,8,8,9,6,7,9,10)
```

```
#Contruir el data frame
grades <- data.frame(Alg = Algebra, An = Analysis, Stat = Statistics)
str(grades)
```

```
'data.frame': 10 obs. of 3 variables:
 $ Alg : num 1 2 0 5 4 6 7 5 5 8
 $ An : num 3 3 2 7 9 5 6 8 5 6
 $ Stat: num 4 5 4 8 8 9 6 7 9 10
```

```
gender <- c("H", "M", "M", "M", "H")
age <- c(23, 45, 20, 30, 18)
family <- c(2, 3, 4, 2, 5)
```

```
#Construir el data frame
df5 <- data.frame(genero = gender, edad = age, familia = family, stringsAsFactors = TRUE)
df5
```

	genero	edad	familia
1	H	23	2
2	M	45	3
3	M	20	4
4	M	30	2
5	H	18	5

```
str(df5)
```

```
'data.frame': 5 obs. of 3 variables:
 $ genero : Factor w/ 2 levels "H","M": 1 2 2 2 1
 $ edad : num 23 45 20 30 18
 $ familia: num 2 3 4 2 5
```

Para asignar nombres en las filas:

```
row.names(df5) <- c("P1", "P2", "P3", "P4", "P5")
df5
```

	genero	edad	familia
P1	H	23	2
P2	M	45	3
P3	M	20	4
P4	M	30	2
P5	H	18	5

Para modificar el nombre de las líneas y columnas a la vez:

```
dimnames(df5) <- list(
  c("Antonio", "Ricardo", "JuanGabriel", "Maria", "Margarita"),
  c("Sexo", "Años", "MiembrosFamilia")
)

df5
```

	Sexo	Años	MiembrosFamilia
Antonio	H	23	2
Ricardo	M	45	3
JuanGabriel	M	20	4
Maria	M	30	2
Margarita	H	18	5

Ejemplo del data frame grades creado anteriormente, vamos añadir una nueva columna de datos

```
Calculus <- c(5,4,6,2,1,0,7,8,9,6)
grades2 <- cbind(grades, Calculus)
head(grades2)
```

	Alg	An	Stat	Calculus
1	1	3	4	5
2	2	3	5	4
3	0	2	4	6
4	5	7	8	2
5	4	9	8	1
6	6	5	9	0

Se puede tambien acceder a las columnas directamente con el \$:

```
#Convertir la columna Sexo en caracter
df5$Sexo = as.character(df5$Sexo)
df5
```

	Sexo	Años	MiembrosFamilia
Antonio	H	23	2
Ricardo	M	45	3
JuanGabriel	M	20	4
Maria	M	30	2
Margarita	H	18	5

Se pasa de factor a caracter.

Se puede tambien añadir una nueva columna con esa tecnica

```
df5$Ingresos = c(1000,12000,15000,12000,20000)
df5
```

	Sexo	Años	MiembrosFamilia	Ingresos
Antonio	H	23	2	1000
Ricardo	M	45	3	12000
JuanGabriel	M	20	4	15000
Maria	M	30	2	12000
Margarita	H	18	5	20000

Cambiando los tipos de datos

Muy conocido como *casting* en otros lenguajes. Que es nada más o nada menos que cambiar un tipo de datos a otro tipo de datos.

- **as.character**: para transformar todos los datos de un objeto en palabras(strings).
- **as.integer**: para transformar todos los datos de un objeto a números enteros.
- **as.numeric**: para transformar todos los datos de un objeto a números reales.

Más sobre sub-data frames

Algo que tiene que saber es que cuando se genera un sub-data frame de un frame, ese heredará los tipos del data frame, por ejemplo:

```
gender <- c("H", "M", "M", "M", "H")
age <- c(23, 45, 20, 30, 18)
family <- c(2, 3, 4, 2, 5)

#Construir el data frame
df5 <- data.frame(genero = gender, edad = age, familia = family, stringsAsFactors = TRUE)

#Creando el sub-data frame
df_m <- df5[df5$genero == "M", ]
df_m
```

	genero	edad	familia
2	M	45	3
3	M	20	4
4	M	30	2

```
str(df_m)
```

```
'data.frame':  3 obs. of  3 variables:
 $ genero : Factor w/ 2 levels "H","M": 2 2 2
 $ edad   : num  45 20 30
 $ familia: num  3 4 2
```

El ejemplo anterior se crea un sub-data frame donde el genero sólo tiene mujeres, pero cuando miro su estructura se puede observar que la opción de hombres sigue estando en la estructura del factor. Para solucionar eso vamos hacer uso de algunas funciones que se describirán abajo para cada caso.

Tidyverse

Es una librería que facilita muchísimo el uso de filtrado de los datos.

- **droplevels(DataFrame)**: para borrar los niveles sobrantes de todos los factores, ya que las columnas que son factores heredan en los sub-data frames todos los niveles del factor original, aunque no aparezcan en el trozo que hemos extraído.
- **select(DataFrame, parámetros)**: para especificar que queremos extraer de un data frame:
 - **starts_with("x")**: extrae del data frame las variables cuyo nombre empieza con la palabra "x".
 - **ends_with("x")**: extrae del data frame las variables cuyo nombre termina con la palabra "x".
 - **contains_with("x")**: extrae del data frame las variables cuyo nombre contiene la palabra "x".
 - Se necesita el paquete **dplyr** o mejor aún **tidyverse**

Del ejemplo anterior para solucionar el problema de heredar variables que no estan siendo utilizadas en el factor.

```
df_m <- droplevels(df_m)
df_m
```

```
  genero edad familia
2      M   45      3
3      M   20      4
4      M   30      2
```

```
str(df_m)
```

```
'data.frame':  3 obs. of  3 variables:
 $ genero : Factor w/ 1 level "M": 1 1 1
 $ edad   : num  45 20 30
 $ familia: num  3 4 2
```

Como se observa, ahora sólo tiene mujeres dentro del factor.

Para utilizar **tidyverse** hay que instalarlo y despues cargarlo.

Para instalar:

```
#install.packages("tidyverse", dependencies = T)
```

Para cargar la librería:

```
library(tidyverse)
```

```
-- Attaching packages -----
v ggplot2 3.2.1      v purrr   0.3.3
v tibble  2.1.3      v dplyr   0.8.4
v tidyr   1.0.2      v stringr 1.4.0
v readr   1.3.1      v forcats 0.5.0
```

Warning: package 'forcats' was built under R version 3.6.3

```
-- Conflicts -----
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

Ejemplo del uso de **tidverse**

```
library(tidyverse)
iris_petal <- select(iris, starts_with("Petal"))
head(iris_petal)
```

```
  Petal.Length Petal.Width
1          1.4          0.2
2          1.4          0.2
3          1.3          0.2
4          1.5          0.2
5          1.4          0.2
6          1.7          0.4
```

Subset de R

- **subset(DataFrame, condición, select=columnas)**: para extraer del data frame las filas que cumplen la condición y las columnas especificadas.

- Si queremos todas las filas, no hay que especificar ninguna condición.
- Si queremos todas las columnas, no hace falta especificar el parámetro **select**
- Las variables en la condición se especifican con su nombre, sin añadir antes el nombre del data frame.

```
subset(iris, Species == "setosa") -> setosa
head(setosa)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

```
str(setosa)
```

```
'data.frame': 50 obs. of 5 variables:
 $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
```

Como ha generado tres niveles y solo necesitamos de uno.

```
setosa <- droplevels(setosa)
str(setosa)
```

```
## 'data.frame': 50 obs. of 5 variables:
## $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
## $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
## $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
## $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
## $ Species : Factor w/ 1 level "setosa": 1 1 1 1 1 1 1 1 1 1 ...
```

```
setosa
```

##	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
## 1	5.1	3.5	1.4	0.2	setosa
## 2	4.9	3.0	1.4	0.2	setosa
## 3	4.7	3.2	1.3	0.2	setosa
## 4	4.6	3.1	1.5	0.2	setosa
## 5	5.0	3.6	1.4	0.2	setosa
## 6	5.4	3.9	1.7	0.4	setosa
## 7	4.6	3.4	1.4	0.3	setosa
## 8	5.0	3.4	1.5	0.2	setosa
## 9	4.4	2.9	1.4	0.2	setosa
## 10	4.9	3.1	1.5	0.1	setosa
## 11	5.4	3.7	1.5	0.2	setosa
## 12	4.8	3.4	1.6	0.2	setosa
## 13	4.8	3.0	1.4	0.1	setosa
## 14	4.3	3.0	1.1	0.1	setosa
## 15	5.8	4.0	1.2	0.2	setosa
## 16	5.7	4.4	1.5	0.4	setosa
## 17	5.4	3.9	1.3	0.4	setosa

```
## 18      5.1      3.5      1.4      0.3 setosa
## 19      5.7      3.8      1.7      0.3 setosa
## 20      5.1      3.8      1.5      0.3 setosa
## 21      5.4      3.4      1.7      0.2 setosa
## 22      5.1      3.7      1.5      0.4 setosa
## 23      4.6      3.6      1.0      0.2 setosa
## 24      5.1      3.3      1.7      0.5 setosa
## 25      4.8      3.4      1.9      0.2 setosa
## 26      5.0      3.0      1.6      0.2 setosa
## 27      5.0      3.4      1.6      0.4 setosa
## 28      5.2      3.5      1.5      0.2 setosa
## 29      5.2      3.4      1.4      0.2 setosa
## 30      4.7      3.2      1.6      0.2 setosa
## 31      4.8      3.1      1.6      0.2 setosa
## 32      5.4      3.4      1.5      0.4 setosa
## 33      5.2      4.1      1.5      0.1 setosa
## 34      5.5      4.2      1.4      0.2 setosa
## 35      4.9      3.1      1.5      0.2 setosa
## 36      5.0      3.2      1.2      0.2 setosa
## 37      5.5      3.5      1.3      0.2 setosa
## 38      4.9      3.6      1.4      0.1 setosa
## 39      4.4      3.0      1.3      0.2 setosa
## 40      5.1      3.4      1.5      0.2 setosa
## 41      5.0      3.5      1.3      0.3 setosa
## 42      4.5      2.3      1.3      0.3 setosa
## 43      4.4      3.2      1.3      0.2 setosa
## 44      5.0      3.5      1.6      0.6 setosa
## 45      5.1      3.8      1.9      0.4 setosa
## 46      4.8      3.0      1.4      0.3 setosa
## 47      5.1      3.8      1.6      0.2 setosa
## 48      4.6      3.2      1.4      0.2 setosa
## 49      5.3      3.7      1.5      0.2 setosa
## 50      5.0      3.3      1.4      0.2 setosa
```

Hay un detalle que tiene que tener tambien en cuenta son los nombres de las columnas, por ejemplo:

```
subset(iris, Species == "versicolor") -> versicolor
head(versicolor)
```

```
      Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
51          7.0         3.2         4.7         1.4 versicolor
52          6.4         3.2         4.5         1.5 versicolor
53          6.9         3.1         4.9         1.5 versicolor
54          5.5         2.3         4.0         1.3 versicolor
55          6.5         2.8         4.6         1.5 versicolor
56          5.7         2.8         4.5         1.3 versicolor
```

Se observa que los nombres de las filas empiezan en 51 y los más lógico sería empezar en 1. Para solucionar eso hay que cambiar los nombres de la filas.

```
rownames(versicolor) <- 1:nrow(versicolor)
head(versicolor)
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 1          7.0         3.2         4.7         1.4 versicolor
## 2          6.4         3.2         4.5         1.5 versicolor
## 3          6.9         3.1         4.9         1.5 versicolor
```

## 4	5.5	2.3	4.0	1.3 versicolor
## 5	6.5	2.8	4.6	1.5 versicolor
## 6	5.7	2.8	4.5	1.3 versicolor

Ahor si los nombres de las filas estan de acuerdo con datos de un sub-data frame extraido.