USO DE HERRAMIENTAS INFORMÁTICAS PARA LA RECOPILACIÓN, ANÁLISIS E INTERPRETACIÓN DE DATOS DE INTERÉS EN LAS CIENCIAS BIOMÉDICAS

Módulo 1
Introducción al uso de R

Alfredo Rigalli
Maela Lupo
María Eugenia Chulibert
Mercedes Lombarte
Patricia Lupión

Centro Universitario de Estudios Medioambientales
Facultad de Ciencias Médicas
Universidad Nacional de Rosario

ISBN 978-987-86-0202-8

USO DE HERRAMIENTAS INFORMÁTICAS PARA LA RECOPILACIÓN, ANÁLISIS E INTERPRETACIÓN DE DATOS DE INTERÉS EN LAS CIENCIAS BIOMÉDICAS

MODULO 1

Introducción al uso de R

Alfredo Rigalli
Maela Lupo
María Eugenia Chulibert
Mercedes Lombarte
Patricia Lupión

Centro Universitario de Estudios Medioambientales
Facultad de Ciencias Médicas
Universidad Nacional de Rosario

Uso de herramientas informáticas para la
recopilación, análisis e interpretación de
datos de interés en las ciencias biomédicas :
módulo 1: introducción al uso de R / Alfredo
Rigalli ... [et al.]. - 1a edición para el alumno -
Rosario : Alfredo Rigalli, 2019.

Libro digital, PDF - (USO DE HERRAMIENTAS INFORMÁTICAS PARA LA RECOPILACIÓN, ANÁLISIS E INTERPRETACIÓN DE DATOS DE INTERÉS / Rigalli, Alfredo; Lupo, Maela; Lombarte, M.; 1)

Archivo Digital: descarga
ISBN 978-987-86-0202-8

1. Software Libre. 2. Bioestadísticas. I. Rigalli, Alfredo.
CDD 610.28

AUTORES

Chulibert, María Eugenia: Licenciada en nutrición. Estudiante del doctorado en Ciencias Biomédicas de la Facultad de Ciencias Médicas de la Universidad Nacional de Rosario. Becaria doctoral del CONICET.

Lombarte, Mercedes: Licenciada en biotecnología y Doctora en Ciencias Biomédicas. Investigadora del Laboratorio de Biología Ósea y del Centro Universitario de Estudios Medioambientales y docente de la cátedra de Química Biológica de la Facultad de Ciencias Médicas de la Universidad Nacional de Rosario.

Lupión, Patricia: Licenciada en biotecnología. Estudiante del doctorado en Ciencias Biomédicas de la Facultad de Ciencias Médicas de la Universidad Nacional de Rosario. Becaria doctoral del CONICET.

Lupo, Maela: Licenciada en biotecnología y Doctora en Ciencias Biomédicas. Investigadora del Laboratorio de Biología Ósea y del Centro Universitario de Estudios Medioambientales y docente de la cátedra de Química Biológica de la Facultad de Ciencias Médicas de la Universidad Nacional de Rosario.

Rigalli, Alfredo: Bioquímico y Doctor en bioquímica. Investigadora del Laboratorio de Biología Ósea y del Centro Universitario de Estudios Medioambientales y docente de la cátedra de Química Biológica de la Facultad de Ciencias Médicas de la Universidad Nacional de Rosario. Investigador independiente del Consejo de Investigaciones de la UNR y del CONICET

Table of Contents

ORGANIZACIÓN DE LA OBRA

Esta obra está dividida en módulos y clases. Cada módulo agrupa temas diferentes. Brevemente

Módulo 1: introducción al manejo de objetos y funciones en R.

Módulo 2: introducción al uso de bibliotecas gráficas.

Módulo 3: introducción a la estadística básica.

Módulo 4: análisis multivariado de datos numéricos y análisis especiales de datos.

Módulo 5: desarrollo de scripts y programación en R.

Cada módulo se divide en 9 clases, las cuales constan de tablas específicas para cada clase, así como de un vídeo y una ejercitación. Al final de las 9 clases existe un examen final del módulo.

Las clases llevarán el nombre Clase1- seguido de un número de 1-9 si son clases del módulo 1, por ejemplo. Así tendrá clases Clase2-3, Clase4-1, etc según sean la clase 3 del módulo 2 o la clase 1 del módulo 4.

Las planillas de cálculo en formatos ods o xls llevarán la denominación tablaR1-3.ods por ejemplo si es la planilla de cálculo para la clase 3 del módulo 1. En el interior de la planilla hallará tablas con los nombre tablaR131, tablaR132, tablaR133, etc. Todas tablas para el módulo 1 (primer número), de la clase 3 (segundo número) y el tercer número indica el número de tabla. Con estos nombres serán introducidos como objetos en el espacio de trabajo.

Al principio de cada clase hallará un link al vídeo sobre la clase y tendrá un link a la planilla de cálculo con las tablas para el desarrollo de la clase.

1. Clase1

Video: https://youtu.be/oCGaXSKGSLw

1.1. Aclaraciones general

En el texto de cada clase encontrará las explicaciones

Cuando mostremos un comando o rutina aparecerá luego del caracter ">", por ejemplo

> t.test(a)

Usted puede copiar la línea (sin el >), pegarla en la línea de comando de R y oprimir enter, de esta manera se ejecutará, la orden establecida por el comando.

Si en una línea de comando aparece el caracter #, esto hace que lo que sigue sea un comentario y aunque lo pegue en la línea de comando y oprima enter, no se ejecutará ninguna orden. Por ejemplo

> citation() # este comando da información cómo citar R en un trabajo

1.2. Arrancar R

Si trabaja en linux

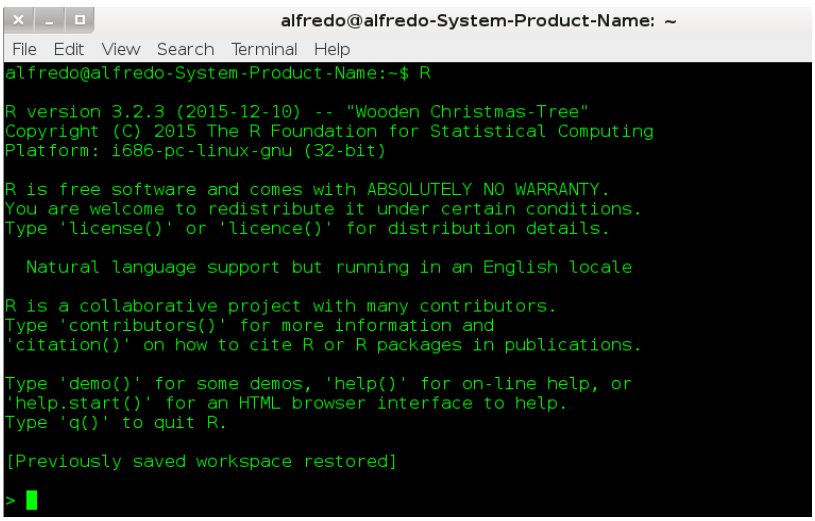
Abrir una terminal o consola

moverse al directorio que se desea trabajar, con comando cd

tipear R

>R

mostrará una bienvenida



Como podemos leer allí ya tenemos las primeras instrucciones

Al final aparece

[Previously saved workspace restored]

¿qué significa?

Que como arrancamos R en un directorio determinado, allí ya había datos nuestros y R ha cargado los mismos

al final puede observar

> ▉

Esa es la línea de comandos. Allí puede escribir o pegar comandos.

Si deseamos citar el entorno en un trabajo, tipeamos

> citation () # aparecerá lo de abajo o actualizado, para citar el R

R Development Core Team (2011). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

1.3. Salir del programa

Para salir de R y terminar la sesión de trabajo, en la línea de comando deberá escribir

>q()

Luego de oprimir enter, aparecerá en la pantalla

> Save workspace image? [y/n/c]:

si oprimimos ***y***, se guardarán los objetos creados y el trabajo realizado durante la sesión. Si opriminos ***n***, no se salvará nada del trabajo realizado y con ***c***, cancelamos y volvemos al espacio de trabajo.

1.4. Obtener información de la versión de R que utilizamos

Si deseamos conocer qué versión estamos utilizando en la línea de comando escribimos

> sessionInfo()

Luego de oprimir enter recibiremos la versión y fecha de lanzamiento de la misma

R version 2.14.0 (2011-10-31)

1.5. Conocer bibliotecas cargadas

R es un entorno que necesita bibliotecas para ejecutar sus acciones. A fines prácticos y de manera de optimizar su funcionamiento, R solo carga algunas bibliotecas básicas y luego el usuario debe cargar las bibliotecas que desea. ¿Como saber qué bibliotecas están cargadas? Podemos hacerlo con cualquiera de los dos comando que se muestran a continuación

>library()

Este comando indica los paquetes instalados y los path. se sale con 'q'

a continuación se muestra la respuesta obtenido al oprimir enter

```
Packages in library ‘/home/alfredo/R/i686-pc-linux-gnu-library/3.2’:

abind                   Combine Multidimensional Arrays
acepack                 ace() and avas() for selecting regression
                        transformations
agricolae               Statistical Procedures for Agricultural
                        Research
AlgDesign               Algorithmic Experimental Design
assertthat              Easy pre and post assertions.
audio                   Audio Interface for R
backports               Reimplementations of Functions Introduced Since
                        R-3.0.0
base64enc               Tools for base64 encoding
BB                      Solving and Optimizing Large-Scale Nonlinear
                        Systems
beepr                   Easily Play Notification Sounds on any Platform
BH                      Boost C++ Header Files
bitops                  Bitwise Operations
car                     Companion to Applied Regression
```

el siguiente comando da la lista de los paquetes cargados actualmente

>search()

1.6. Conocer el directorio de trabajo

Cada vez que iniciemos R, lo haremos dentro de un directorio, el cual podremos elegir. Una vez dentro del mismo, si deseamos conocerlo o recordarlos escribirmos

> getwd() #mostrará el directorio o workspace, en este caso el directorio se llama modulo1

[1] "/home/alfredo/alf/cursos/R/modulo1"

1.7. Ver los comando previos

R nos permite tener una archivo con los comandos previos, del día y los históricos, desde que iniciamos el espacio de trabajo

> history()

Este comando nos muestra los últimos 25 comandos usados. Se sale de la visión oprimento la letra ***q***

Si deseamos ver toda la historia de trabajo

> history(max.show=Inf)

El comando anterior muestra todos los comando. Se sale de la visión oprimiendo la letra ***q***

Recuerde que si desea dejar durante el trabajo comentarios puede hacerlo fácilmente anteponiendo a su escrito el símbolo ***#***

Por ejemplo

> # sesión del día 18/9/18. Día soleado, agradable para una sesión de R. Objetivos: análisis multivariado de datos de la base de datos aguas. Se realizarán PCA y MCA de los datos en

búsqueda de respuestas a los interrogantes de la reunión anterior. ¿Hay relación entre los componenetes químicos del agua y la provincia de origen? ¿Las personas están satisfechas con el agua que consumen y el precio que pagan por ella?

Este texto quedará en un archivo en su espacio de trabajo, junto con todo los análisis y resultados hallados.

1.8. Ver el espacio de trabajo

Si se desean ver los objetos que se hallan en el espacio de trabajo se usa el siguiente código

```
>ls()                #objects()  da la misma información
```

muestra los objetos, es decir estructuras que almacenan datos o análisis de estos.

1.9. Introducir datos en R

Los datos y los resultados se almacenan en R en lo que llamamos objetos. Dentro de los objetos para almacenar datos, los más comunes son

vectores

data.frame

matrices

Los objetos también pueden almacenar resultados de análisis realizados.

1.9.1. vectores

Un vector es un conjunto de varios datos. Por ejemplo supongamos que tuviera la glucemia de 4 ratas, podría guardar ellas en un vector.

1.9.1.1. Vectores con elementos numéricos

crear un vector con elementos numéricos

Supongamos que tenemos una serie de datos de tiempo: 1 seg, 2 seg, 3 seg, ... 5 segundos. Con estos datos creamos un vector que almacena los números. El código para crear el vector es

```
> a<-c(1,2,3,4,5)
```

"a" será el nombre con que se identifica el objeto y el mismo tendrá 5 números. Podemos pedir un detalle de este objeto con

```
> summary(a)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      1       2       3       3       4       5
```

en este caso no está mostrando el mínimo, máximo, mediana y los percentilos 25 y 75%

Podemos también ver que tipo de datos tiene el objeto. La función mode() permite obtener esta inforamación

```
> mode(a)
[1] "numeric"
```

podemos también pasar los datos a otro objeto "b" con la función assign(). Con el comando siguiente creamos un objeto llamado ***b***, que tendrá los mismos datos que el objeto ***a***

> assign("b"a)

o también con

> b<-a

como podemos ver ahora ***b*** también tiene la misma información

> b

[1] 1 2 3 4 5

1.9.1.2. Vectores con elementos string (secuencia de caracteres)

si se desea crear el objeto 'c' con elementos string, es decir elementos que sean combinaciones de letras: por ejemplo las letras con la que identifique cinco ratas de mis experimentos , aplicaremos el siguiente código.

> c<-c("x","y","z","u","v")

Es importante tener en cuenta que los elementos de un vector que sean strings deben introducirse entre comillas ("..."). También puede utilizarse el tilde ('.....')

Todos los comandos que necesiten escrituras entre comillas (".....") pueden escribirse entre tildes ('....'). La segunda opción es más ventajosa desde el punto de vista de eficiencia en el sentido que el tilde se utiliza sin oprimir mayúscula, mientra que la comilla requiere oprimir mayúscula y por lo tanto involucra dos teclas, aumentando el tiempo utilizado y la probabilidad de error. Por lo tanto el vector c, podría escribirse

> c<-c('x','y','z','u','v')

y puedo ver sus propiedades con los comandos, si lo deseara

> summary(c)

> mode(c)

1.9.1.3. Algunas funciones y operaciones con vectores (también aplicable a otros objetos)

max(a) #Buscar el elemento más grande del vector

>max(a)

[1] 5

min(a) #buscar el elemento más chico del vector

> min(a)

[1] 1

range(a) #buscar el máximo y mínimo, o sea el rango

> min(a)

[1] 1

> length(a) #buscar el número de elementos

[1] 5

> sum(a) #buscar la suma delos componentes del objeto

[1] 15

> prod(a) #mostrar el producto de los componentes del objeto

[1] 120

mean(a) #calcula la media

> mean(a)

median(a) #calcula la mediana

> median(a)

[1] 3

sd(x) #calcula desvio standard

> sd(a)

[1] 1.581139

var(a) #calcula la variancia

> var(a)

[1] 2.5

sort(a) #ordenar datos de un objeto, la propiedad decreasing me permite indicarle si en forma ascendente o descendente.

sort(a,decreasing= T or F)

> a

[1] 1 2 3 4 5

> sort(a,decreasing=T)

[1] 5 4 3 2 1

1.9.1.4. crear vector con secuencia y valores repetidos

x<-seq(1,33,1) # crea el vector que contiene los números entre 1 y 33 de a 1. Además en este caso el resultado del comando "seq" lo asignamos a un objeto que llamamos ***x.***

> x

[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33

> quince<-rep(15,37) #crea un vector con el número 15 repetido 37 veces

> quince

[1] 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15

[26] 15 15 15 15 15 15 15 15 15 15 15 15

1.9.1.5. redondeo y otros

creamos un vector d

> a

[1] 1 2 3 4 5

> d<-a/10 # en este caso d está formado por los elementos de a pero todos divididos por 10.

> d

[1] 0.1 0.2 0.3 0.4 0.5

'ceiling' # toma cada valor y regresa el primer entero no menor que el número

> ceiling(d)

[1] 1 1 1 1 1

floor # toma los valores y retorna el mayor entero posible, pero no mayor que el valor original

> floor(d)

[1] 0 0 0 0 0

trunc # retorna el entero de haber truncado el valor en la coma.

> trunc(d)

[1] 0 0 0 0 0

round #redondea en el numero de digitos que se desea.

> round(d,digits=0)

[1] 0 0 0 0 0

> round(d,digits=1)

[1] 0.1 0.2 0.3 0.4 0.5

1.9.1.6. generador números aleatorios

runif(10,0,1) #genera 10 números aleatorios entre 0 y 1

[1] 0.8304301 0.3168486 0.9059344 0.5761408 0.4335628 0.9095313 0.1411549

[8] 0.1893361 0.8885673 0.4636167

se puede combinar con otras funciones para obtener números enteros

trunc(runif(10,0,1)*10) #en este caso trunca los 10 numeros aleatorios generados entre 0 y 1, que previamente fueron multiplicados por 10.

[1] 2 1 5 4 9 0 9 9 2 0

> floor(runif(10,0,1000))

[1] 293 541 693 357 337 860 953 405 810 948 #qué hizo aca?

1.10. Visualizar espacio de trabajo

Si deseamos ver los objetos que se hallan en el espacio de trabajo

> ls()

> objects()

> objects(sorted=T or F)

1.11. Borrar el objeto

rm(a), #elimina el elemento a

remove(a) # elimina el elemento a

rm(list=ls()) # borra todos los objetos del espacio de trabajo

Si por error borramos algo, al salir del espacio de trabajo nos pedirá si queremos salvar los cambios y en dicho caso será conveniente colocar "n"

Tenga en cuenta que si borra un objeto no existe "undo".

2. Clase 2

Vídeo: https://youtu.be/N9MFknGml2g
tabla de datos: http://hdl.handle.net/2133/9450
Para esta clase usted trabajará con datos de una planilla de cálculo tablaR1-2.ods/xls, que debe descargar del link anterior.

2.1. Introducir datos en R

Los datos y los resultados se almacenan en en el espacio de trabajo de R en lo que llamamos objetos. Dentro de los objetos para almacenar datos, los más comunes son

vectores

data.frames

matrices

Cada uno de ellos tiene características que los hacen más ventajosos para ciertas aplicaciones.

2.1.1. Tablas: data frame

Los data.frame son tablas con columnas y filas. Pueden almacenar datos numéricos y caracteres y en general cualquier tipo de datos. Se asemeja a una planilla de cálculo. Una característica importante es que cada columna debe contener el mismo tipo de datos, por ejemplo una columna puede ser edad, es decir un dato numérico y para todas las filas, deberá ser numérico. Contrariamente una fila puede tener en cada columna datos de diferente tipo.

2.1.1.1. Crear data.frame

2.1.1.2. Ingreso de datos manualmente en R

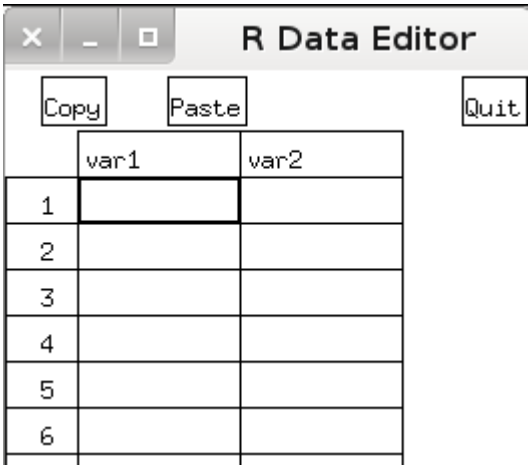
La forma más fácil es con el siguiente código y sirve para cuando debemos cargar datos directamente en R.

En prime lugar creamos un data.frame vacío con el siguiente código, en cuyo caso estamos creando un objeto del tipo data.frame con nombre ***a***

```
a<-data.frame()
```

Con el código siguiente abrimos el data.frame en una interfaz amigable, donde podemos ingresar datos y cambiar nombre a columnas.

```
a<-edit(a)
```



Haciendo click con el botón derecho del mouse sobre var1, var2, etc se despliega un pequeño menú que nos permite cambiar nombre de la columna y tipo de datos. Al crear el data.frame con el código anterior, R no coloca nombre a las columnas. Si desearamos crear el mismo data.frame pero teniendo nombre de columnas, por ejemplo que la columna 1 sea nombre y la segunda edad, podemos utilizar el código

```
> a<-data.frame(nombre="NA",edad="NA")
```

si vemos el contenido de ***a***

```
> a
  nombre edad
1     NA   NA
```

tenemos las dos columnas con la primer fila vacía. También podríamos ya crear el data.frame introduciendo el primer elemento de la tabla o en realidad cualquiera, ya que luego podremos fácilmente modificarlo.

```
> a<-data.frame(nombre="Pérez",edad=39)
```

vemos el contenido

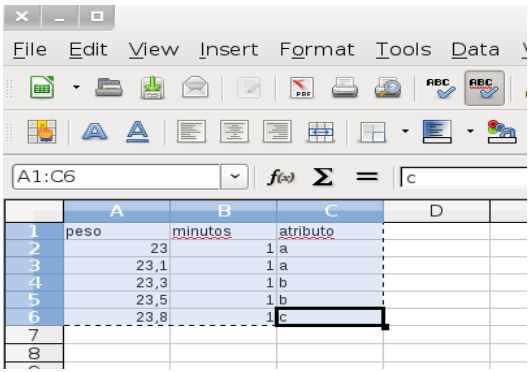
```
> a
  nombre edad
1 Pérez   39
```

Con el tiempo irá conociendo otras formas más eficientes, más rápidas y más adecuadas a cada situación que requiera.

2.1.1.3. Introducir al espacio de trabajo datos de una planilla de cálculo

Otra forma de introducir datos al espacio de trabajo es importándola desde una planilla de cálculo. Un detalle importante en este método es que debemos tener muy claro si el separador decimal utilizado en la planilla es "." o ",".

Procedimiento: Utilice para este ejercicio la tablaR121 de la planilla de cálculo tablaR1-2.xls/ods. Teniendo abierto R en nuestro espacio de trabajo y la planilla de cálculo, marcamos la parte de la planilla que queremos introducir, la copiamos al portapapeles,



luego vamos a R y escribimos el siguiente código

tablaR121<-read.table("clipboard", header=TRUE, dec=",")

Este código se utiliza si copiamos una planilla con los encabezamientos de columnas y los números están en coma decimal. Ya veremos problemas y soluciones que se pueden presentar en casos específicos.

Para ir familirizándonos con la terminología, en la línea de comando anterior, read.table() es una función de R que ejecuta algo, en este caso introduce datos desde el portapapeles a un objeto, en este caso llamado tablaR121. Dentro de la función figura header=.... , esto es un argumento de la función, que puede cambiar. En este caso, header=TRUE, le indica a la función que la primer línea de lo que hemos copiado son los encabezamientos de columnas y por ende los asignará a los nombres de columnas del data.frame. El argumento dec=",", le indica a la función read.table() que en nuestra planilla de cálculo la división decimal de los números se hace con una ",".

si luego pedimos el objeto creado "tablaR121"

> tablaR121

```
  peso minutos atributo
1 23.0       1       a
2 23.1       1       a
3 23.3       1       b
4 23.5       1       b
5 23.8       1       c
```

vemos los datos de la planilla de cálculo. Se han introducido con punto decimal.

2.1.1.4. Importar datos de una planilla de cálculo al espacio de trabajo

Se puede importar una planilla de cálculos completa. Si bien hay numerosas opciones y usted irá definiendo con el tiempo cual le resulta más adecuada, para comenzar se recomienda hacerlo a partir de archivos con extensión .csv (comma separated values). Seguramente usted tendrá algunas dificultades hasta hallar las condiciones óptimas. Se muestra un mecanismo en

este caso desarrollado en Calc (planilla de libreOffice).

1- a partir de la hoja tablaR121 de la planilla de cálculo tablaR1-2ods/xls, grabe el archivo en formato csv, con el nombre tablaR121.csv. En el proceso elija

Character set: Unicode

Field delimiter: {Tab}

text delimiter: "

2- desde R ejecute la siguiente linea de comandos.

tablaR121<-read.table("tablaR121.csv",header=TRUE,sep="\t",fileEncoding ="Unicode", dec=",")

header= TRUE (o FALSE) indica si se utilizarán los encabezados de columnas como títulos de las columnas de nuestro objeto o no

sep="\t" indica que utilizamos como Field delimiter {Tab}. Es decir quedan separados por tabulador los valores de cada columna dentro de una fila.

fileEncoding= "Unicode", indica que utilizamos ese modo de codificación en Character set.

dec=",", indica que en nuestra planilla utilizamos como punto decimal a la coma ",".

2.1.1.5. Creación de un data.frame a partir de vectores existentes en el espacio de trabajo

Si se desea que un data.frame tenga dos columnas, la primera con los datos del objeto "***a***" y la segunda con los datos del objeto "***c***", se asigna el nombre del objeto, por ejemplo "tabla1" y el siguiente código asigna las columnas. Por defecto R pone los vectores como columnas.

Usted debe tener creados del módulo anterior los vectores ***a*** y ***c***, sino créelos nuevamente

```
a<-c(1,2,3,4,5)

c<-c("x","y","z","u","v")

> tabla1<-data.frame(a,c)

> tabla1
  a c
1 1 x
2 2 y
3 3 z
4 4 u
5 5 v
```

es equivalente a

```
> tabla1<-data.frame(cbind(a,c))     #cbind significa unidos por columnas
```

si utilizamos la función rbind, formará un data frame, pero cada vector ocupará una fila

```
> tabla2<-data.frame(rbind(a,c))          #rbind significa unidos por rows

> tabla2
```

```
  X1 X2 X3 X4 X5
a  1  2  3  4  5
c  x  y  z  u  v
```

2.1.1.6. crear nuevo data frame a partir de un data.frame existente

Si se desea formar otra tabla (tabla3) que tenga los mismos datos que tabla1, donde agregaremos una columna "y" con datos que son una función de "a", en este caso los valores de a+1

```
> tabla3<-data.frame(tabla1,y=a+1)
```

```
> tabla3
  a c y
1 1 x 2
2 2 y 3
3 3 z 4
4 4 u 5
5 5 v 6
```

2.1.1.7. Otra formas de crear data.frames

Si se le quiere poner nombre a las columnas, tenemos los vectores a e c

```
> tabla1<-data.frame(A=a,C=c)
```

```
> tabla1
  A C
1 1 x
2 2 y
3 3 z
4 4 u
5 5 v
```

2.1.1.8. Conocer las características del data frame

Un data.frame no tiene límite conocido de cantidad de filas y columnas, por lo que cuando son muy grandes tener una idea del tipo de datos almacenados es un problema. La función str() soluciona ese problema haciéndonos un resumen. Veamos por ejemplo para la tablaR121

```
> str(tablaR121)
```

```
'data.frame':        5 obs. of  3 variables:
 $ peso    : num  23 23.1 23.3 23.5 23.6
 $ minutos : int  1 1 1 1 1
 $ atributo: Factor w/ 3 levels "a","b","c": 1 1 2 3 2
```

La primer línea nos indica: 5 obs (5 filas) of 3 variables (3 columnas)

la segunda línea: $ peso: no indica que son datos numéricos y nos indica los valores. Cuando son muchos nos mostrará sólo una parte

La tercer línea $ minuto: nos indica que los datos son enteros (int)

La cuarta línea $ atributo: nos indica que es un factor, es decir una variable categórica, que en este caso tiene 3 niveles (3 levels) y nos indica los niveles existentes: a, b y c. Luego nos da el

orden en que se hallan en las primeras filas: 1,1,2,3,2 está indicando a,a,b,c,b

2.1.1.9. ver una columna

Retomemos el trabajo con el data.frame tablaR121. Visualicémoslo

```
> tablaR121
  peso minutos atributo
1 23.0       1        a
2 23.1       1        a
3 23.3       1        b
4 23.5       1        c
5 23.6       1        b
```

si deseáramos ver solo la columna peso utilizaremos el código

```
> tablaR121$peso
```

[1] 23.0 23.1 23.3 23.5 23.6

o la columna atributo

```
> tablaR121$atributo
```

[1] a a b c b

Levels: a b c

que nos muestra los datos consecutivos de la columna y luego los niveles existentes

Aclaración: cuando escribimos el nombre del data.frame seguido de $ y un nombre estamos haciendo un llamado a una columna del data.frame. Ej tablaR121$peso, llama la columna peso del data.frame tablaR121. Como veremos este código no sólo sirve para ver la columna.

2.1.1.10. Cambiar valores de un data.frame

Introduzcamos en nuestro espacio de trabajo la tablaR122 de nuestra planilla de cálculo tablaR1-2.ods/xls. Para ello utilicemos cualquier de los dos métodos vistos anteriormente.

por ejemplo

```
> tablaR122<-read.table("clipboard", header=TRUE, dec=",")
```

vemos los datos introducidos

```
> tablaR122
  peso minutos atributo atributo2
1 23.0       1        a     tubo3
2 23.1       1        a     tubo3
3 23.3       1        b     tubo4
4 13.5       1        c     tubo5
5 13.5       1        b     tubo5
```

para cambiar un número en una tabla se debe indicar numero de fila y columna. Por ejemplo si deseamos cambiar el valor 23.1 de la fila 2 y columna 1 por el valor 50, se escribe el nombre del data.frame y entre corchetes, la fila y la columna, como lo indica el código siguiente.

```
> tablaR122[2,1]<-50
```

vemos como quedó el data.frame

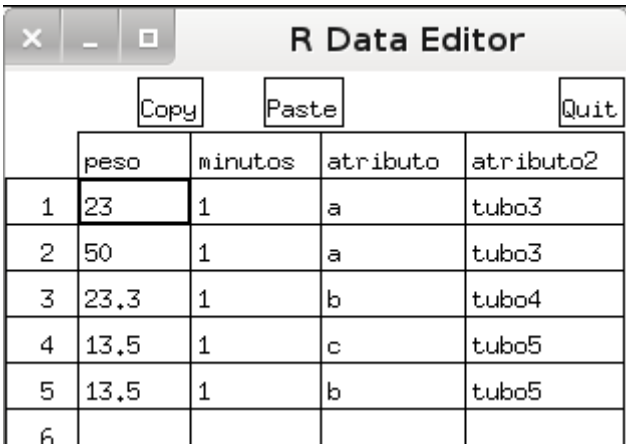
```
> tablaR122
  peso minutos atributo atributo2
1 23.0       1        a     tubo3
2 50.0       1        a     tubo3
3 23.3       1        b     tubo4
4 13.5       1        c     tubo5
5 13.5       1        b     tubo5
```

en amarillo observamos el cambio logrado.

otra forma más sencilla es utilizar el comando

```
> tablaR122<-edit(tablaR122)
```

que abrirá una ventana de edición en un formato muy amigable, que se muestra a continuación



en esta tabla cambiamos el valor y luego oprimimos "Quit". El comando mostrado permite abrir un data.frame, modificar los datos y guardar los cambios.

Si usted hubiera ejecutado el comando

```
> edit(tablaR122)
```

ingresará al mismo sitio, podrá modificar datos si desea, pero cuando oprima Quit, esto no se grabarán en el data.frame, ya que no los asignó nuevamente al objeto con "<-". El comando mencionado es muy útil cuando desea ver datos de un data.frame.

2.1.1.11. Guardar y cargar data.frame en espacio de trabajo

Supongamos que deseamos enviar datos en un data.frame o bien utilizarlos en otro espacio de trabajo. Un recurso interesante es grabarlo como un data.frame en un archivo independiente que luego puede ser introducido en otro espacio de trabajo

Tomemos el data.frame tablaR122 que estamos utilizando. Lo guardamos como un archivo con el siguiente código.

> save(tablaR122,file="tablaR122")

esto nos creará una archivo llamado tablaR122 que tendrá los datos del data frame, el cual quedará en el directorio en el que estamos trabajando. Puede comprobar esto utilizando un administrador de archivos.

Recuerde que para conocer en que directorio está trabajando puede ejecutar el comando.

> getwd()

cuyo resultado será algo como se describe a continuación.

[1] "/home/alfredo/alf/cursos/R/modulo1"

Si quisiera utilizar el data.frame en otro espacio de trabajo, copie el archivo al directorio donde se halla su espacio de trabajo y luego desde el espacio de trabajo ejecute

> load("tablaR122")

comprobará con la función ls() que ahora tiene un data.frame llamado tablaR122 con los datos.

Para comprobar en este momento como funciona, elimine el data.frame tablaR122 de su espacio de trabajo. Para ello ejecute

rm> (tablaR122)

luego compruebe si está el objeto tabla1, pidiendo el el objeto

> tablaR122

Error: object 'tablaR122' not found

Ahora vuelva a introducir el data.frame que borró, pero que anteriormente había salvado como un archivo.

> load("tablaR122")

pida a R el objeto introducido

> tablaR122

	peso	minutos	atributo	atributo2
1	23.0	1	a	tubo3
2	50.0	1	a	tubo3
3	23.3	1	b	tubo4
4	13.5	1	c	tubo5
5	13.5	1	b	tubo5

Advertirá que es un buen recurso para enviar datos a personas con las que se encuentra trabajando sobre un mismo proyecto.

2.1.2. Conocer el tipo de objeto

En nuestros espacios de trabajo se irán juntando objetos, por ende es prioritario definir una política de asignación de nombre. En el desarrollo de esta clase llamamos a los objetos

tablaR....., cuando son data.frames, por ejemplo. Usted puede elegir su propio y más intuitiva forma. Los nombres largos si bien son claros, al tener cientos de objetos puede ser dificultoso hallarlos. Independientemente de la eficiencia que tenga en el trabajo de nombrar objetos, R nos da el recurso de saber si es una data.frame, matriz o vector.
Para saber si es una matriz utilizamos la función is.matrix(), que nos devolverá TRUE o FALSE, si es o no. En el caso de tablaR122, la creamos como data.frame
> is.matrix(tablaR122)

[1] FALSE

La función is.data.frame() no devuelve TRUE o FALSE si es o no un data.frame

> is.data.frame(tablaR122)

[1] TRUE

la función is.vector(), nos devuelve TRUE o FALSE si es o no vector

> is.vector(tablaR122)

[1] FALSE

la función is.array(), nos devuelve TRUE o FALSE si es o no un arreglo

> is.array(tablaR122)

[1] FALSE

2.1.3. Convertir objetos

Supongamos que deseamos convertir el objeto tablaR122 que es un data.frame en una matriz. Para ello utilizamos el siguiente código
> tablaR122matrix<-as.matrix(tablaR122)

podemos comprobar que el objeto obtenido no es un data.frame, a pesar que tablaR122 lo era.

> is.data.frame(tablaR122matrix)

[1] FALSE

podemos verificar que el proceso lo transformó en un objeto del tipo matriz

> is.matrix(tablaR122matrix)

[1] TRUE

2.2. Detalle sobre nombres de objetos

Los nombres de los objetos no pueden llevar los siguientes caracteres ni espacios

~
!
@
#
$
%
^
&

*
(
)
{
}
_
+
:
"
<
>
?
,
.
/
;
'
[
]
-
=

Si bien pueden llevar números, estos no pueden iniciar el nombre del objeto. Es permitido un objeto llamado datos1, pero no 12perros.

Es muy sencillo de recordar, solo utilice palabras y números, evite letras acentuadas.

Si desea que un objeto refleje su contenido puede utilizar mayúsculas

por ejemplo

DatosDeExperimentoDelDia<-c(.....

3. Clase 3.

Vídeo: https://youtu.be/jZMa8litrmI
tabla de datos: http://hdl.handle.net/2133/9451

3.1. Continuación uso de data.frames

3.1.1. Reemplazo de valores de una data.frame

En el módulo anterior vimos el reemplazo de valores en un data.frame utilizando el comando edit() y asignando valores por filas y columnas. Veamos ahora otras formas útiles de realizar reemplazos.

3.1.1.1. Reemplazar valores de un nivel con la función gsub()

Cuando se desea realizar un reemplazo a lo largo de todo un data.frame o una columna de él, se puede optar por un método que busque lo que se quiera reemplazar y en su lugar colocar el valor deseado. Este método es similar a las funciones "find & replace" o "buscar y reemplazar" de procesadores de texto y planillas de cálculo. En este caso se utiliza la función gsub().

Utilizaremos la tablaR131 de la planilla de cálculo tablaR1-3.xls/ods adjunta a esta clase. En primer lugar introduzca esta tabla utilizando

```
> tablaR131<-read.table("clipboard",header=TRUE,dec=",",sep="\t")
```

vemos la tabla introducida

```
> tablaR131
  peso minutos atributo atributo2
1 23.0       1       a     tubo3
2 23.1       1       a     tubo3
3 23.3       1       b     tubo4
4 13.5       1       c     tubo5
5 13.5       1       b     tubo5
```

queremos reemplazar en la columna atributo2, "tubo" por "t". Para ello a la columna atributo debemos reemplazarla por los datos modificados. La función gsub() tendrá el siguiente formato.

```
> tablaR131$atributo2<-gsub("tubo","t",tablaR131$atributo2)
```

si observamos ahora la tabla

```
> tablaR131
  peso minutos atributo atributo2
1 23.0       1       a        t3
2 23.1       1       a        t3
3 23.3       1       b        t4
4 13.5       1       c        t5
5 13.5       1       b        t5
```

3.2. Obtener una resumen de un data.frame

Cuando deseamos conocer el contenido de una tabla en lo que se refiere a nombre de las columnas, tipo de variables y niveles de cada variable cualitativa, es recomendable la función summary()

```
> summary(tablaR131)
     peso           minutos      atributo  atributo2
 Min.   :13.50   Min.   :1    a:2       Length:5
 1st Qu.:13.50   1st Qu.:1    b:2       Class :character
 Median :23.00   Median :1    c:1       Mode  :character
 Mean   :19.28   Mean   :1
 3rd Qu.:23.10   3rd Qu.:1
 Max.   :23.30   Max.   :1
```

Las variables numéricas se indican con estadísticas descriptivas: min, max, mediana, media y cuartilos. Las cualitativas que se hallan factorizadas se indican los niveles, por ejemplo la varible atributo. Otras variables como atributo2 que tiene datos cualitativos pero no caracterizdos por nivel, se indica el tipo de datos: character. Podría, como veremos más adelante, transformarse en nivel del factor atributo2.

3.3. cambiar nombre de las columnas de un data frame

A menudo necesitamos cambiar nombre de las columnas

3.3.1. Utilizando la función names()

En el data.frame que acabamos de crear las columnas tienen los nombres: peso, minutos, atributo y atributo2

```
> tablaR131
  peso minutos atributo atributo2
1 23.0       1        a         t3
2 23.1       1        a         t3
3 23.3       1        b         t4
4 13.5       1        c         t5
5 13.5       1        b         t5
```

deseamos cambiar el nombre de la columna atributo2 por "atr"

```
> names(tablaR131)<-c("peso","minutos","atributo","atr")
```

```
> tablaR131
  peso minutos atributo atr
1 23.0       1        a          t3
2 23.1       1        a         t3
3 23.3       1        b         t4
4 13.5       1        c         t5
5 13.5       1        b         t5
```

Es importante notar que se deben indicar en el vector el nombre de todas las columnas aunque solo se cambie uno.

3.3.2. A través de edit

Se puede editar la tabla, en el modo modificación, es decir asignado edit() a la misma tabla o a otra, como en este caso tabla1

```
tabla1<-edit(tablaR131)
```

hacer click sobre el nombre de la columna y elegir cambiar nombre. Se debe borrar el nombre y escribir el deseado. Recuerde que para que el cambio sea permanente debe utilizar el código

indicado. Si utiliza el código

edit(tablaR131)

podrá hacer la modificación pero no quedará guardado en el objeto.

3.4. Insertar datos con nombre de columna

Si al data.frame “tablaR131” le deseo agregar una nueva columna, a partir de datos de un vector (vector1) y deseo que la columna se llame variable 3. El vector tiene que tener la misma cantidad de datos que las filas del data.frame.

Primeramente declaramos e inicializamos el vector llamado vector1

```
> vector1<-c("x","x","x","x","y")
```

lo agregamos a la tablaR131 con la función cbind (unir por columnas). Esta función agregará el vector1 como una columna a continuación del data.frame tablaR131, que en este caso introducimos con el nombre columna: variable3. PUede utilizarse el nombre que desee para la columna.

```
> tablaR131<-cbind(tablaR131,variable3=vector1)
```

```
> tablaR131
  peso minutos atributo atr variable3
1 23.0      1        a       t3        x
2 23.1      1        a       t3        x
3 23.3      1        b       t4        x
4 13.5      1        c       t5        x
5 13.5      1        b       t5        y
```

3.5. Ordenar datos en un data.frame

3.5.1. Ordenamiento creciente

Habitualmente necesitamos ordenar los datos de un data.frame en forma creciente o decreciente, por los valores de una o más columnas. Para ello utilizaremos la función order(). Trabajaremos con el data.frame tablaR131

```
> tablaR131
  peso minutos atributo atr variable3
1 23.0      1        a       t3        x
2 23.1      1        a       t3        x
3 23.3      1        b       t4        x
4 13.5      1        c       t5        x
5 13.5      1        b       t5        y
```

queremos ordenarlo primero por la columna peso (columna 1) y luego por atributo (columna 3).

```
> tablaR131[order(tablaR131[,1],tablaR131[,3]),]
  peso minutos atributo atr variable3
5 13.5      1        b  t5        y
4 13.5      1        c  t5        x
1 23.0      1        a  t3        x
```

2 23.1 1 a t3 x
3 23.3 1 b t4 x
también puede ejecutarse como
> tablaR131[order(tablaR131$peso,tablaR131$atributo),]

En ambos casos logré el ordenamiento, pero dicho orden es visible en pantalla, pero no se almacena en el data.frame. Si deseara ordenarlo y sobreescribir el data.frame debería utilizar

tablaR131<- tablaR131[order(tablaR131[,1],tablaR131[,3]),]

3.5.2. Ordenamiento creciente y decreciente

Se utiliza el caracter "-" delante de la columna que se desea ordenar en forma decreciente

```
> tablaR131[order(-tablaR131$peso,tablaR131$atributo),]
  peso minutos atributo atr variable3
3 23.3      1       b  t4        x
2 23.1      1       a  t3        x
1 23.0      1       a  t3        x
5 13.5      1       b  t5        y
4 13.5      1       c  t5        x
```

3.6. Operaciones con datos de una tabla

Veamos utilizando el objeto tablaR131 que hemos creado la aplicación de operaciones. El objeto lo hemos modificado así que podemos introducirlo nuevamente en el espacio de trabajo con el código conocido, copiando nuevamente los datos de la tabla.

tablaR131<-read.table("clipboard",header=TRUE,dec=",",sep="\t")

el objeto quedará

```
> tablaR131
  peso minutos atributo atributo2
1 23.0      1       a     tubo3
2 23.1      1       a     tubo3
3 23.3      1       b     tubo4
4 13.5      1       c     tubo5
5 13.5      1       b     tubo5
```

Si deseamos obtener la suma de los valores de la columna peso (columna 1)

```
> sum(tablaR131[,1])

[1] 96.4
```

Si deseamos la media de los datos de esa columna

```
> mean(tablaR131[,1])

[1] 19.28
```

Supongamos que deseamos la media de los datos de la columna 1, pero solo las 3 primeras filas

```
> mean(tablaR131[c(1:3),1])

[1] 23.13333
```

O la media de la primer columna y las 3 primeras filas junto con la última

```
> mean(tablaR131[c(1:3,5),1])
```

```
[1] 20.725
```

así se puede aplicar todas las funciones vistas anteriormente

3.7. Crear objetos a partir de un data.frame

3.7.1. Crear un vector a partir de data.frame

Muchas veces es necesario extraer datos de un data.frame y llevarlos a un vector. Sigamos trabajando con el mismo data.frame

```
> tablaR131
  peso minutos atributo atributo2
1 23.0       1       a     tubo3
2 23.1       1       a     tubo3
3 23.3       1       b     tubo4
4 13.5       1       c     tubo5
5 13.5       1       b     tubo5
> vectorpeso<-tablaR131$peso
```

veamos que contiene el vector

```
> vectorpeso
```

```
[1] 23.0 23.1 23.3 13.5 13.5
```

como vemos son los datos de la columna 1. Veamos ahora si es realmente un vector

```
> is.vector(vectorpeso)
```

```
[1] TRUE
```

3.7.2. crear un data.frame a partir de otro data.frama

3.7.2.1. de una sola columna

Muchas veces nos puede convenir tener un data.frame de una columna en lugar de un vector, en esta circunstancia aplicamos

```
> peso<-tablaR131[1]
```

vemos el data.frame peso, recientemente creado

```
> peso
  peso
1 23.0
2 23.1
3 23.3
4 13.5
5 13.5
```

comprobamos si es un data.frame

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

```
[1] TRUE
```

3.7.2.2. de más de una columna

si deseamos crear un data.frame con más de una columna, debemos indicar las columnas

```
> tablaR131
  peso minutos atributo atributo2
1 23.0       1        a     tubo3
2 23.1       1        a     tubo3
3 23.3       1        b     tubo4
4 13.5       1        c     tubo5
5 13.5       1        b     tubo5
> pesominuto<-tablaR131[,c(1:2)]
```

```
> pesominuto
  peso minutos
1 23.0       1
2 23.1       1
3 23.3       1
4 13.5       1
5 13.5       1
> is.data.frame(pesominuto)
```

```
[1] TRUE
```

3.7.2.3. con diferente número de filas

Para crear un data.frame que tenga diferente número de filas a partir de un data.frame ya creado, por ejemplo tablaR131

```
> tablaR131
  peso minutos atributo atributo2
1 23.0       1        a     tubo3
2 23.1       1        a     tubo3
3 23.3       1        b     tubo4
4 13.5       1        c     tubo5
5 13.5       1        b     tubo5
> tresfilas<-tablaR131[c(1:3),]
```

```
> tresfilas
  peso minutos atributo atributo2
1 23.0       1        a     tubo3
2 23.1       1        a     tubo3
3 23.3       1        b     tubo4
> is.data.frame(tresfilas)
```

```
[1] TRUE
```

3.7.2.4. Con diferente número de filas y columnas

Ahora combinemos lo aprendido. Supongamos que deseamos crear un nuevo data.frame que tenga las tres primeras filas, las dos primeras columnas y la última

```
> tablaR131
  peso minutos atributo atributo2
1 23.0       1       a     tubo3
2 23.1       1       a     tubo3
3 23.3       1       b     tubo4
4 13.5       1       c     tubo5
5 13.5       1       b     tubo5
> nuevodataframe<-tablaR131[c(1:3),c(1:2,4)]

> nuevodataframe
  peso minutos atributo2
1 23.0       1     tubo3
2 23.1       1     tubo3
3 23.3       1     tubo4
```

3.8. Conocer el tipo de datos en un data frame

por ejemplo para un data frame tablaR131

```
> tablaR131<-edit(tablaR131)        #se abre la tabla
```

hacer click botón izquierdo sobre la primer celda de cada columna. Allí figura el nombre de la variable, Real, Character, Change Name. Hacer click en real o character según corresponda. Si se desea cambiar el nombre hacer click en Change nombre y modificar

también se puede conocer con

```
> mode(tablaR131$peso)

[1] "numeric"
```

3.9. conocer numero de columnas y filas

Teniendo en cuenta el data.frame “tablaR131”, existen dos funciones para cumplir con este objetivo: nrow() y ncol() nos indican el número de filas y columnas, respectivamente

```
> tablaR131
  peso minutos atributo atributo2
1 23.0       1       a     tubo3
2 23.1       1       a     tubo3
3 23.3       1       b     tubo4
4 13.5       1       c     tubo5
5 13.5       1       b     tubo5
> nrow(tablaR131)

[1] 5

> ncol(tablaR131)

[1] 4
```

3.10. Conocer número de datos dentro de cada factor (table)

3.10.1. a un factor

En la tablaR131 podemos ver los valores de cada columna, pero en tablas más grandes esto es

imposible y debe recurrirse a formas analíticas. A continuación se muestra la forma de saber cuantos niveles tiene cada factor o columna. Supongamos que deseamos conocer para la columna atributo el número de cada una de los valores que aparecen.

```
> tablaR131
  peso minutos atributo atributo2
1 23.0       1        a     tubo3
2 23.1       1        a     tubo3
3 23.3       1        b     tubo4
4 13.5       1        c     tubo5
5 13.5       1        b     tubo5
```

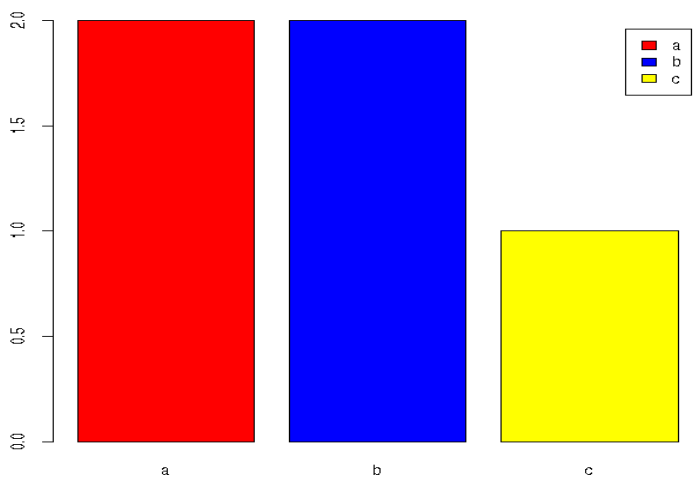
La función table() nos provee esta información . Supongamos que queremos saber cuantos elementos tienen nivel a, b o c, correspondiente a la columna atributo

```
> table(tablaR131$atributo)

a b c

2 2 1
```

Nos indica que hay dos elementos con a, dos con b y 1 con c.

Podemos obtener también esta información en forma gráfica con la función barplot().

```
> barplot(table(tablaR131$atributo),col=c("red","blue","yellow"),legend=c("a","b","c"))
```



El largo de cada barra indica el número de elementos con niveles a, b y c respectivamente.

Si se desea los porcentajes en lugar del número de unidades, la función prop.table() da esa información.

```
> prop.table(table(tablaR131$atributo))*100

 a  b  c
40 40 20
```

Nos indica que 40% de las filas tienen valor a, 40% valor b y 20 % valor c, para la columna atributo.

3.10.2. a dos factores

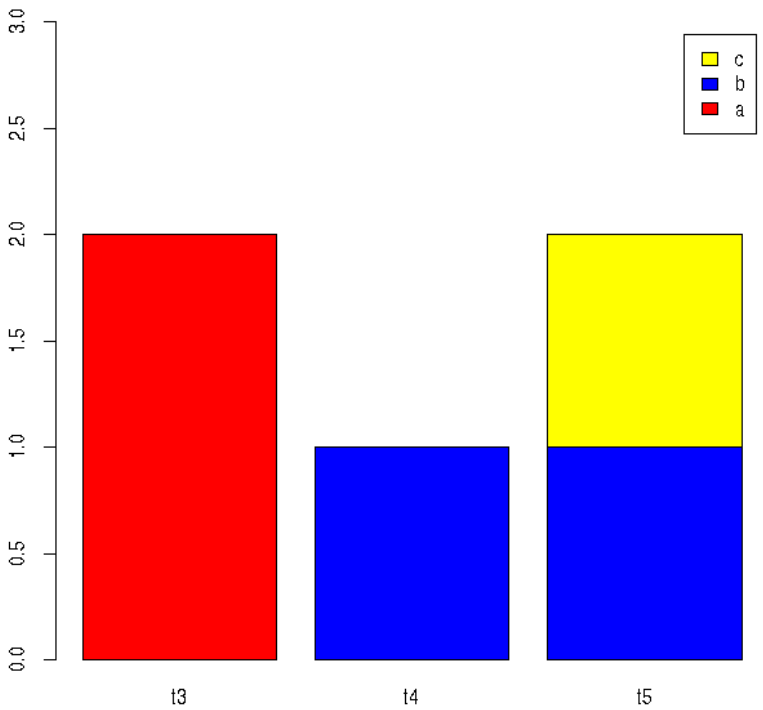
Si desearamos contar las filas por dos factores, por ejemplo atributo y atributo2, la función table() sigue siendo útil, pero le debo indicar las dos columnas

```
> table(tablaR131$atributo,tablaR131$atributo2)
   tubo3 tubo4 tubo5
 a     2     0     0
 b     0     1     1
 c     0     0     1
```

Nos indica que hay dos fila en que la variable atributo toma el a y simultaneamente atributo2 el valor tubo3. Pero no hay ninguna fila que simultaneamente atributo tome el valor b y atributo2 el valor tubo3.

Si se desea graficar dicha información:

```
>
barplot(table(tablaR131$atributo,tablaR131$atributo2),col=c("red","blue","yellow"),legend=c(
"a","b","c"),ylim=c(0,3))
```



Puede hacerse a tres, cuatro y más factores si se desea

3.11. Trabajar con una columna de un data.frame

A continuación veremos cálculos realizados con una columna de un data.frame. Si aplico una operación a una columna de un data.frame, esta se aplicará a todos los elementos de esa columna. Veamos un ejemplo con el data.frame tablaR131

```
> tablaR131
  peso minutos atributo atributo2
1 23.0      1        a     tubo3
2 23.1      1        a     tubo3
3 23.3      1        b     tubo4
4 13.5      1        c     tubo5
5 13.5      1        b     tubo5
```

a la columna peso le sumaré el valor de la misma columna multiplicado por 0,1. Es decir que a la columna peso la incremento en 10%

```
> tablaR131$peso+0.1*tablaR131$peso

[1] 25.30 25.41 25.63 14.85 14.85
```

3.12. Seleccionar datos de un data frame

Es habitual que tengamos que seleccionar datos de un data.frame. Para ello existen diferentes mecanismos. Estos son muy útiles en grandes tablas de datos, cuando no podemos ver todos los valores. La función subset() es de mucha utilidad. Si bien esta función puede ser dificultosa en un comienzo es una herramienta poderosísima que recomendamos tener muy en cuenta.

3.12.1. Selección con un solo criterio

3.12.1.1. Con elección positiva

Supongamos que en la tablaR131 deseamos obtener las columnas pesos y atributo2, pero solo para aquello peso>20

```
> tablaR131
  peso minutos atributo atributo2
1 23.0      1        a     tubo3
2 23.1      1        a     tubo3
3 23.3      1        b     tubo4
4 13.5      1        c     tubo5
5 13.5      1        b     tubo5
```

En la función subset() se indican tres argumentos, separados por coma. El primer argumento es el nombre del data.frame, el segundo el valor que utilizamos de una columna como criterio de selección, el tercero las columnas que deseamos visualizar

```
> subset(tablaR131,peso>20, select=c(peso,atributo2))
  peso atributo2
1 23.0     tubo3
2 23.1     tubo3
3 23.3     tubo4
```

También podemos hacer selecciones indicando que valores no deberían incluirse, es decir indicar que los valores de una columna deben ser diferentes de algo. En el siguiente caso seleccionaremos todas las columnas, menos el peso, por ello en select se expresa -peso. Por

otra parte seleccionaremos las filas en las que atributo sea distinto de a (se usa != para indicar diferente)

```
> subset(tablaR131,atributo!="a", select=c(-peso))
  minutos atributo atributo2
3       1       b     tubo4
4       1       c     tubo5
5       1       b     tubo5
```

3.12.2. Selección a dos o más criterios

En la selección de elementos es común que se desee seleccionar por más de un criterio. Por ejemplo de una base de datos de alumnos deseo obtener aquellos que sean de sexo femenino y menores de 15 años. Para ello R tiene un código muy sencillo. Veamos los casos más comunes utilizando la función subset()

3.12.2.1. AND (&)

Cuando deseamos que en la selección se cumplan dos criterios simultáneamente utilizamos &.

En la tablaR131 seleccionaremos aquellas unidades con peso>10 y que simultáneamente el atributo tome el valor a

```
> tablaR131
  peso minutos atributo atributo2
1 23.0       1       a     tubo3
2 23.1       1       a     tubo3
3 23.3       1       b     tubo4
4 13.5       1       c     tubo5
5 13.5       1       b     tubo5
> subset(tablaR131,peso>10 & atributo=="a", select=c(peso,atributo2))
  peso atributo2
1 23.0     tubo3
2 23.1     tubo3
```

Note que en la condición, cuando se indica que alguna variable tome un valor igual a algo, el igual se expresa ==.

3.12.2.2. OR (|)

Utilizaremos también procesos de selección cuando queremos quedarnos con unidades experimentales que cumplen uno u otro requisito. Por ejemplo si de la tablaR131 quisiéramos a las unidades cuyo peso>20 o la variable atributo tome el valor c, utilizaremos el siguiente código, donde "|" indica OR.

```
> tablaR131
  peso minutos atributo atributo2
1 23.0       1       a     tubo3
2 23.1       1       a     tubo3
3 23.3       1       b     tubo4
4 13.5       1       c     tubo5
5 13.5       1       b     tubo5
> subset(tablaR131,peso>20 | atributo=="c", select=c(peso,atributo))
  peso atributo
```

```
1 23.0       a
2 23.1       a
3 23.3       b
4 13.5       c
```

3.12.2.3. AND y OR

También en la selección podemos combinar AND y OR. Por ejemplo si tuviéramos alumnos de los cuales queremos seleccionar los de sexo femenino y que a su vez sean argentinas o paraguayas.

Veamos en nuestra tabla, queremos las unidades cuyo peso>20 y que la columna atributo tome los valroes "c" o "a"

```
> tablaR131
  peso minutos atributo atributo2
1 23.0       1       a     tubo3
2 23.1       1       a     tubo3
3 23.3       1       b     tubo4
4 13.5       1       c     tubo5
5 13.5       1       b     tubo5
> subset(tablaR131,peso>20 & (atributo=="c"|atributo=="a"), select=c(peso,atributo))
  peso atributo
1 23.0       a
2 23.1       a
```

3.12.2.4. Otras combinaciones de la función subset()

En el código siguiente seleccionamos la columna peso, pero solo de aquellas filas donde peso supera el valor 20. Al anidarla con la función colMeans(), luego de hacer la selección de las filas deseada, se obtiene la suma de los valores

```
> colMeans(subset(tablaR131,peso>20,select=c(peso)))

    peso

23.13333
```

3.12.3. Otra forma de seleccionar datos

Una forma práctica es indicar entre [filas,columnas] las filas y las columnas a seleccionar

veamos en la tablaR131

```
> tablaR131
  peso minutos atributo atributo2
1 23.0       1       a     tubo3
2 23.1       1       a     tubo3
3 23.3       1       b     tubo4
4 13.5       1       c     tubo5
5 13.5       1       b     tubo5
```

deseamos seleccionar los valores para los cuales las filas tienen valores de peso>20. El código sería el siguiente

```
> tablaR131[tablaR131$peso>20,]
  peso minutos atributo atributo2
```

```
1 23.0      1       a     tubo3
2 23.1      1       a     tubo3
3 23.3      1       b     tubo4
```
Note que si deseamos todas las columnas, dentro del corchete, luego de la coma no colocamos nada. Entre corchetes, el primer término antes de la coma hace referencias a las filas y después de la coma hace referencia a las columnas.

si además de seleccionar las filas con peso>20 deseáramos solo las columnas 1 y 3, el código sería

```
> tablaR131[tablaR131$peso>20,c(1,3)]
  peso atributo
1 23.0       a
2 23.1       a
3 23.3       b
```
note que lo anterior también se puede hacer con

```
> subset(tablaR131,peso>20,select=c("peso","atributo"))
  peso atributo
1 23.0       a
2 23.1       a
3 23.3       b
```
Una forma más compleja es con la función tapply(), que veremos más adelante.

3.13. Fusionar dos o mas data.frame

Es común que debamos juntar dos o más data.frames. Se puede poner uno debajo del otro o uno a la par de otro. Veamos cada caso.

3.13.1. Uno debajo del otro

Para poder aplicar este procedimiento los data.frame tienen que tener igual número de columna, tipo de datos y nombre de la columna. Para hacer el ejercicio creamos dos data.frame a partir de tablaR131

```
> tablaR131
  peso minutos atributo atributo2
1 23.0      1       a     tubo3
2 23.1      1       a     tubo3
3 23.3      1       b     tubo4
4 13.5      1       c     tubo5
5 13.5      1       b     tubo5
```
creamos dos data.frames

el data.frame tabla1 tiene todas las columnas, pero solo las filas 1,2 y 3

```
> tabla1<-tablaR131[c(1:3),]
```

el data.frame tabla2 tiene las filas 4 y 5 y todas sus columnas

```
> tabla2<-tablaR131[c(4:5),]
```

lo verificamos

```
> tabla1
  peso minutos atributo atributo2
1 23.0       1        a     tubo3
2 23.1       1        a     tubo3
3 23.3       1        b     tubo4
> tabla2
  peso minutos atributo atributo2
4 13.5       1        c     tubo5
5 13.5       1        b     tubo5
```

Para fusionar nuevamente los dos data.frames utilizaremos la función rbind(). Con ella crearemos la tabla3. En este caso en particular fusionaremos los datos pero pondremos primero los datos de la tabla2 y luego la 1

```
> tabla3<-rbind(tabla2,tabla1)
```

```
> tabla3
   peso minutos atributo atributo2
4  13.5       1        c     tubo5
5  13.5       1        b     tubo5
3  23.0       1        a     tubo3
41 23.1       1        a     tubo3
51 23.3       1        b     tubo4
```

3.13.2. Uno a la par del otro

Para aplicar esta función los data.frame tienen que tener igual cantidad de líneas. Para lograr esto se utiliza la función cbind(). Para entender esta función partiremos la tablaR131 en dos partes

```
> tablaR131
  peso minutos atributo atributo2
1 23.0       1        a     tubo3
2 23.1       1        a     tubo3
3 23.3       1        b     tubo4
4 13.5       1        c     tubo5
5 13.5       1        b     tubo5
```

en primer lugar hacemos una tabla1 que contiene todas las filas de tablaR131 pero solo las dos primeras columnas.

```
> tabla1<-tablaR131[,c(1:2)]
```

la tabla2 tiene todas las filas, pero solo las columnas 3 y 4

```
> tabla2<-tablaR131[,c(3:4)]
```

veamos las tablas

```
> tabla1
  peso minutos
1 23.0       1
2 23.1       1
3 23.3       1
4 13.5       1
```

5 13.5 1
> tabla2
atributo atributo2
1 a tubo3
2 a tubo3
3 b tubo4
4 c tubo5
5 b tubo5

Ahora fusionaremos las tablas, colocando primero las columnas atributo y atributo 2

> tabla3<-cbind(tabla2,tabla1)

> tabla3<-cbind(tabla2,tabla1)

> tabla3
atributo atributo2 peso minutos
1 a tubo3 23.0 1
2 a tubo3 23.1 1
3 b tubo4 23.3 1
4 c tubo5 13.5 1
5 b tubo5 13.5 1

4. Clase 4

Vídeo: https://youtu.be/TIuAQWLktbo
Tabla de datos: http://hdl.handle.net/2133/9452

4.1. Continuación uso de data.frames

Trabajaremos con el siguiente data.frame (puede introducirla copiándola del archivo tablaR1-4.ods/xls, hoja tablaR141)

```
> tablaR141<-read.table("clipboard",header=TRUE,dec=",",sep="\t")
```

```
> tablaR141
  peso minutos atributo atributo2
1 23.0       1       a     tubo3
2 23.1       1       a     tubo3
3 23.3       1       b     tubo4
4 13.5       1       c     tubo5
5 13.5       1       b     tubo5
```

4.2. Head

La función head() nos permite obtener rápidamente información sobre el data.frame, particularmente nos da las columnas con sus nombres y los primeros elementos. Se complementa con la función summary(). El número luego de la coma indica la cantidad de filas que se desean ver

```
> head(tablaR141,2)
  peso minutos atributo atributo2
1 23.0       1       a     tubo3
2 23.1       1       a     tubo3
```

Si no se especifica número de filas, la función muestra 5 filas.

```
> head(tablaR141)
  peso minutos atributo atributo2
1 23.0       1       a     tubo3
2 23.1       1       a     tubo3
3 23.3       1       b     tubo4
4 13.5       1       c     tubo5
5 13.5       1       b     tubo5
```

4.3. Nombre de columnas

la función names() nos da solo los nombre de las columnas del data.frame. Seguimos con el data.frame tablaR141

```
> tablaR141
  peso minutos atributo atributo2
1 23.0       1       a     tubo3
2 23.1       1       a     tubo3
3 23.3       1       b     tubo4
4 13.5       1       c     tubo5
5 13.5       1       b     tubo5
> names(tablaR141)
```

```
[1] "peso"      "minutos"   "atributo"  "atributo2"
```

4.4. Transform

La función transform() sirve para transformar los valores de una columna de un data frame, aplicando alguna función. Si la tablaR141 es

```
> tablaR141
  peso minutos atributo atributo2
1 23.0       1        a     tubo3
2 23.1       1        a     tubo3
3 23.3       1        b     tubo4
4 13.5       1        c     tubo5
5 13.5       1        b     tubo5
```

el código siguiente, transformará y reemplazará la columna peso por el logaritmo del peso

```
transform(tablaR141,peso=log(peso))
      peso      minutos atributo atributo2
1 3.135494      1        a         tubo3
2 3.139833      1        a         tubo3
3 3.148453      1        b         tubo4
4 2.602690      1        c         tubo5
5 2.602690      1        b         tubo5
```

en el código anterior, la columna peso se modificó por aplicación del logaritmo, pero si pedimos el data.frame, podemos ver que no se ha modificado el valor del peso.

```
> tablaR141
  peso minutos atributo atributo2
1 23.0       1        a         tubo3
2 23.1       1        a         tubo3
3 23.3       1        b         tubo4
4 13.5       1        c         tubo5
5 13.5       1        b         tubo5
```

En cambio sí se modificará si la función anterior la asignamos nuevamente al objeto tablaR141 como vemos a continuación

```
> tablaR141<-transform(tablaR141,peso=log(peso))
```

```
> tablaR141
      peso minutos atributo atributo2
1 3.135494      1        a     tubo3
2 3.139833      1        a     tubo3
3 3.148453      1        b     tubo4
4 2.602690      1        c     tubo5
5 2.602690      1        b     tubo5
```

Tenga presente que si aplicó el código anterior no tiene más el peso sino el logaritmo en la columna 1. Para seguir con la tabla original deberá introducirla nuevamente.

4.5. Eliminar/seleccionar/reordenar filas y/o columnas

4.5.1. eliminar/seleccionar filas

sigamos trabajando con el data.frame tablaR141. Si lo ha modificado puede volver a introducirlo.

```
> tablaR141
  peso minutos atributo atributo2
1 23.0       1       a     tubo3
2 23.1       1       a     tubo3
3 23.3       1       b     tubo4
4 13.5       1       c     tubo5
5 13.5       1       b     tubo5
```

Si deseamos eliminar la fila 5, pero quedarnos con todas las columnas tiene que tener en cuenta que si escribimos tablaR141[filas,columnas], entre corchete se indica antes de la coma las filas. que se desean y después de la coma, las columnas

si deseamos eliminar la fila 5, utilizaremos el siguiente código

```
> tablaR141[c(1:4),]
  peso minutos atributo atributo2
1 23.0       1       a     tubo3
2 23.1       1       a     tubo3
3 23.3       1       b     tubo4
4 13.5       1       c     tubo5
```

como puede ver nos quedaron las filas 1,2,3 y 4 y todas las columnas. Al no especificar nada después de la coma, dentro del corchete, se interpreta que no se elimina ninguna columna. Con el código anterior eliminamos una fila, pero como no lo asignamos al data.frame, si bien vemos los cambios en el data frame no se han mantenido.

Si deseáramos mantener las filas 1, 3, 4 y 5, pero todas las columnas, escribiríamos

```
> tablaR141[c(1,3:5),]
  peso minutos atributo atributo2
1 23.0       1       a     tubo3
3 23.3       1       b     tubo4
4 13.5       1       c     tubo5
5 13.5       1       b     tubo5
```

Los códigos anteriores seleccionaron filas, pero los datos no han quedado grabados en ningún objeto. En cambio, si quisiéramos las filas 1 y 2 además de las 4 y 5, pero además quisiéramos guardarlas en otro objeto creamos un data.frame "tabla2"

```
> tabla2<-tablaR141[c(1:2,4:5),]

> tabla2
  peso minutos atributo atributo2
1 23.0       1       a     tubo3
2 23.1       1       a     tubo3
4 13.5       1       c     tubo5
5 13.5       1       b     tubo5
```

Se debe notar que las filas mantienen los números originales. Si bien esto puede ser una ventaja

a la hora de trabajar puede ser una complicación a la hora de ejecutar scripts (tema que veremos en módulos posteriores).

Si se desea reasignar números de filas, de manera que estos sean correlativos se puede utilizar la función row.names() con el siguiente código que asigna a los numeros de las filas los datos de un vector numérico que va desde 1 hasta el número de filas del data.frame, en este caso 4.

```
> row.names(tabla2)<-c(1:nrow(tabla2))

> tabla2
  peso minutos atributo atributo2
1 23.0      1        a       tubo3
2 23.1      1        a       tubo3
3 13.5      1        c       tubo5
4 13.5      1        b       tubo5
```

4.5.2. Eliminar/seleccionar columnas

Para seleccionar columnas se sigue el mismo procedimiento, teniendo en cuenta que las columnas que se desean seleccionar se especifican después de la coma. Si antes de la coma no se pone nada es porque se desean todas las filas. Veamos nuevamente la tablaR141

```
> tablaR141
  peso minutos atributo atributo2
1 23.0      1        a       tubo3
2 23.1      1        a       tubo3
3 23.3      1        b       tubo4
4 13.5      1        c       tubo5
5 13.5      1        b       tubo5
```

Si deseamos seleccionar todas las filas, pero solo las columnas 1 y 2 escribiremos

```
>  tablaR141[,c(1:2)]
  peso minutos
1 23.0      1
2 23.1      1
3 23.3      1
4 13.5      1
5 13.5      1
```

de la misma manera si deseamos las columna 1, 2 y 4, pero todas las filas

```
> tablaR141[,c(1:2,4)]
  peso minutos atributo2
1 23.0      1     tubo3
2 23.1      1     tubo3
3 23.3      1     tubo4
4 13.5      1     tubo5
5 13.5      1     tubo5
```

4.5.3. Eliminar/seleccionar filas y columnas

Con la misma metodología se pueden seleccionar filas y columna. Supongamos que deseamos las filas 1,3 y 4 además las columna 1,3,4, escribiremos

```
> tablaR141[c(1,3:4),c(1,3,4)]
```

```
  peso atributo atributo2
1 23.0       a     tubo3
3 23.3       b     tubo4
4 13.5       c     tubo5
```

si deseamos guardar los cambios en otra tabla, asignamos la selección de las filas y columnas a un nuevo objeto que llamamos en este caso tablanueva

```
> tablanueva<-  tablaR141[c(1,3:4),c(1,3,4)]
```

También podríamos guardarla en la misma tabla (pero perderemos la original)

```
> tablaR141<- > tablaR141[c(1,3:4),c(1,3,4)]
```

Siempre que cambie un data.frame y desee guardar los cambios en la misma tabla, esté seguro del procedimiento, ya que es irreversible (al menos de manera fácil). Siempre es recomendable grabar los cambios en otro objeto, al menos hasta estar seguro.

4.5.4. Eliminar/seleccionar filas y columnas con condiciones

Muchas veces no solo queremos seleccionar ciertas filas y columnas, sino que deseamos filas donde se cumplan ciertas condiciones. Supongamos el data.frame tablaR141

```
> tablaR141
  peso minutos atributo atributo2
1 23.0       1       a     tubo3
2 23.1       1       a     tubo3
3 23.3       1       b     tubo4
4 13.5       1       c     tubo5
5 13.5       1       b     tubo5
```

y deseamos todas las filas en las que se cumpla que el peso>20. Por otro lado queremos todas las columnas, para ello utilizamos el mismo mecanismo pero en antes de la coma, indicamos la columna y la condición, con el siguiente código

```
> tablaR141[tablaR141$peso>20,]
  peso minutos atributo atributo2
1 23.0       1       a        tubo3
2 23.1       1       a        tubo3
3 23.3       1       b        tubo4
```

una selección más sofisticada y exigente. Deseamos todas las columnas, pero solo aquellas filas que simultáneamente tienen peso>23 y la columna atributo= a. Es obvio de mirar la tabla que mayor que 23 son las filas 2 y 3, pero si simultáneamente queremos que atributo valga a, nos quedaremos solo con la fila 2. Comprobemos si el código siguiente lo hace

```
>  tablaR141[tablaR141$peso>23 & tablaR141$atributo=="a",]
  peso minutos atributo atributo2
2 23.1       1       a         tubo3
```

Una aclaración

Si usted utiliza como en el código anterior tablaR141$atributo== "a"

El doble igual (==) nos permite buscar en que filas la columna atributo tiene el valor "a"

Si hubiera escrito

```
> tablaR141$atributo="a"
```

estará reemplazando en la columna atributo sus elementos por el valor "a"

```
> tablaR141
  peso minutos atributo atributo2
1 23.0       1        a     tubo3
2 23.1       1        a     tubo3
3 23.3       1        a     tubo4
4 13.5       1        a     tubo5
5 13.5       1        a     tubo5
```

recuerde que este código modificó el data.frame. Deberá reintroducir tablaR141 si desea trabajar con valores originales.

4.5.5. Reordenar filas y/o columnas

Cuando deseamos reordenar filas o columnas, básicamente utilizamos las mismas herramientas que para seleccionar filas o columnas vistas anteriormente

Supongamos el data.frame tablaR141

```
> tablaR141
  peso minutos atributo atributo2
1 23.0       1        a      tubo3
2 23.1       1        a      tubo3
3 23.3       1        b      tubo4
4 13.5       1        c      tubo5
5 13.5       1        b      tubo5
```

con la función names() podemos ver los nombres de las columnas

```
> names(tablaR141)
```

```
[1] "peso"      "minutos"   "atributo"  "atributo2"
```

A continuación reordenaremos las columnas, colocándolas en el orden 4, 3, 1 y 2. Por otra parte para no afectar el data.frame original lo asignamos al data.frame tabla2

```
> tabla2<-tablaR141[,c(4,3,1,2)]
```

veamos tabla2

```
> tabla2
  atributo2 atributo peso minutos
1     tubo3        a 23.0       1
2     tubo3        a 23.1       1
3     tubo4        b 23.3       1
4     tubo5        c 13.5       1
5     tubo5        b 13.5       1
```

4.6. Tapply

La función tapply() nos permite calcular estadísticas como la media, desvío estándar, etc de una columna de un data.frame en función de una clasificación de otra columna. Veremos ejemplos para comprender esta poderosa función.

4.6.1. Tapply con un factor

trabajaremos sobre la tablaR141

```
> tablaR141
  peso minutos atributo atributo2
1 23.0       1        a       tubo3
2 23.1       1        a       tubo3
3 23.3       1        b       tubo4
4 13.5       1        c       tubo5
5 13.5       1        b       tubo5
```

Supongamos que queremos conocer la media de los pesos de aquellas unidades para las cuales atributo =a, atributo=b y atributo=c. Es decir calcular la media de los pesos para cada nivel de la columna atributo. El código para este caso se muestra a continuación. Dentro de la función tapply() hay tres argumentos separados por comas.El primero indica la columna del data.frame (en este caso peso) en la que quiero calcular el valor de la estadística indicada en el tercer argumento (mean en este caso). En el segundo indico cual es la columna para dividir a los elementos según nivel del factor.

```
> tapply(tablaR141$peso,factor(tablaR141$atributo),mean)
    a     b     c
23.05 18.40 13.50
```

Si quisieramos el desvío estándar, aplicaríamos tapply como se ve a continuación.

```
> tapply(tabla1$peso,factor(tabla1$atributo),sd)
         a          b          c
0.07071068 6.92964646         NA
```

Para el nivel c, no se puede calcular el desvío estándar porque con un solo valor el cálculo no es posible.

Como siempre se puede asignar los resultados a un objeto. En el ejemplo siguiente asignamos las medias a un vector al que llamamos mediapeso.

```
> mediapeso<-tapply(tablaR1411$peso,factor(tablaR141$atributo),mean)

> mediapeso
    a     b     c
23.05 18.40 13.50
```

4.6.2. Tapply con mas de un factor

La función tapply() puede aplicarse también a una columna de un data.frame pero clasificando por más de un factor o columna.

```
> tablaR141
  peso minutos atributo atributo2
1 23.0       1        a       tubo3
2 23.1       1        a       tubo3
3 23.3       1        b       tubo4
4 13.5       1        c       tubo5
5 13.5       1        b       tubo5
```

Supongamos que queremos las medias de los pesos agrupadas por niveles de las columna atributo y atributo2. En ese caso el segundo argumento se utilizada con la función list() donde

indico separados por comas las dos columnas utilizadas para la selección

```
> tapply(tablaR141$peso,list(tablaR141$atributo,tablaR141$atributo2),mean)
      tubo3   tubo4   tubo5
a     23.05   NA      NA
b       NA    23.3     13.5
c       NA    NA      13.5
```

Dada la escasa cantidad de valores, algunas de las combinaciones nos arrojan NA (no available) ya que no existe la combinación de factores. Por ejemplo no hay ningún elemento que tenga atributo= b y atributo2=tubo3

4.7. Trasponer un data frame

Esta función t(), permite pasar las filas a columnas y las columnas a filas. La función t() no se puede aplicar siempre, dado que un data.frame tiene que tener los mismos elementos en cada columna. Tomemos la tablaR141. Como tiene elementos de diferente tipo no la podemos transponer.

```
> tablaR141
  peso minutos atributo atributo2
1 23.0      1        a       tubo3
2 23.1      1        a       tubo3
3 23.3      1        b       tubo4
4 13.5      1        c       tubo5
5 13.5      1        b       tubo5
```

Pero, si seleccionemos a partir de tablaR141 aquellas columnas solo con números, es decir la 1 y 2

```
> tablaR141[,c(1:2)]
  peso minutos
1 23.0      1
2 23.1      1
3 23.3      1
4 13.5      1
5 13.5      1
```

si quisiéramos transponerla, en este caso podríamos hacerlo usando la función t().

```
> t(tabla1[,c(1:2)])
              [,1]   [,2]   [,3]   [,4]    [,5]
peso            23   23.1 23.3  13.5  13.5
minutos          1    1.0    1.0    1.0   1.0
```

Como podemos ver la columna 1 (peso) pasó a ser la fila 1. De la misma manera la columna 2 (minutos).

La función t(), veremos que es de mayor aplicación en el uso de matrices.

4.7.1. Transformar formato de variables

4.7.1.1. transformar variable numérica en factor

Habitualmente tenemos variables numéricas que no son variables continuas, sino que toman solo algunos valores. Tal es el caso de cuando se realiza un experimento por ejemplo con tres

edades de ratas: 30 días, 60 días y 90 días. Esta variable es discontinua y debemos tratarla como un factor con tres niveles.
Introducimos la tabla tablaR142

```
> tablaR142
   glucosa dosis
1      1.1     0
2      1.2     0
3      1.1     0
4      1.2     0
5      1.3     0
6      1.5    10
7      1.6    10
8      1.4    10
9      1.3    10
10     1.2    10
```

en esta tabla la dosis es una variable cuantitativa como lo muestra un summary()

```
> summary(tablaR142)
    glucosa          dosis
 Min.   :1.100     Min.   : 0
 1st Qu.:1.200     1st Qu.: 0
 Median :1.250     Median : 5
 Mean   :1.290     Mean   : 5
 3rd Qu.:1.375     3rd Qu.:10
 Max.   :1.600     Max.   :10
```

También podemos verificar esto con la función str(). Como vemos en el resultado siguiente, la columna dosis tiene datos numéricos enteros (int).

```
> str(tablaR142)

'data.frame':        10 obs. of  2 variables:

 $ glucosa: num  1.1 1.2 1.1 1.2 1.3 1.5 1.6 1.4 1.3 1.2

 $ dosis  : int  0 0 0 0 0 10 10 10 10 10
```

Para un correcto análisis esta variable debería factorizarse. Es decir transformarse en un factor con dos niveles: 0 y 10

para transformar en factor la columna dosis, utilizaremos la función as.factor() como se indica a continuación

```
> tablaR142$dosis<-as.factor(tablaR142$dosis)
```

con el comando anterior reemplazamos la columna dosis de tablaR142 por la misma columna pero transformada en factor con la función as.factor(). Si ahora aplicamos la función summary() veremos que la columna dosis está factorizada, indicándonos que 5 líneas de tabla tiene en nivel de dosis= 0 y otras 5 filas el nivel de dosis=10

```
> summary(tablaR142)
    glucosa          dosis
 Min.   :1.100     0 :5
```

```
 1st Qu.:1.200   10:5
 Median :1.250
 Mean   :1.290
 3rd Qu.:1.375
 Max.   :1.600
```
el summary() nos indica que dosis ahora tiene dos niveles cada uno de ellos con 5 elementos.

4.7.1.2. Transformar variable continua en factor con niveles

Algunas veces también es necesario transformar una variable continua en factor. Tal es el caso cuando se trabaja con variables como el peso. Por ejemplo en la tablaR141 tenemos pesos diversos pero podemos querer agruparlos por ejemplo en bajo peso a los que llamamos Q1 y alto peso a los que llamamos Q2

tomamos la tablaR141

```
tablaR141
  peso minutos atributo atributo2
1 23.0       1        a       tubo3
2 23.1       1        a       tubo3
3 23.3       1        b       tubo4
4 13.5       1        c       tubo5
5 13.5       1        b       tubo5
```
para mantener los datos originales de tabla1 creamos tabla2

```
> tabla2<-tablaR141
```

vemos el summary de esta tabla

```
> summary(tabla2)
     peso           minutos     atributo  atributo2
 Min.   :13.50    Min.   :1    a:2       tubo3:2
 1st Qu.:13.50    1st Qu.:1    b:2       tubo4:1
 Median :23.00    Median :1    c:1       tubo5:2
 Mean   :19.28    Mean   :1
 3rd Qu.:23.10    3rd Qu.:1
 Max.   :23.30    Max.   :1
```
Para tranformar la columna peso ($peso) en factor, tomaremos pesos de 10-20 como Q1 y pesos de 20-30 como Q2 y aplicaremos el siguiente código. El argumento breaks: indica los límites, label: los nombres asignados a cada unidad según su valor de peso respecto de los breaks. Más complejo son los argumentos right y include.lowest. Si colocamos right=FALSE (F) como está en el código, indica que si una unidad tuviera un peso de 20, sería incluído en el intervalo siguiente, es decir de 20 -30. Por otro lado el argumento include lowest=TRUE (T), indica que el valor 20 es incluido en el intervalo 20-30. En otras palabras, si analizamos el intervalo 10-20, include.lowest=T indica que el valor 10 será incluido en el intervalo mencionado. El argumento right=F indica que el valor 20 no será incluido en este intervalo y si en el siguiente.

Con el código siguiente reemplazamos la columna peso por sus valores factorizados

```
> tabla2$peso<-cut(tabla2$peso,breaks=c(10,20,30),label=c("Q1","Q2"),right=F,include.lowest=T)
```

Resultando

```
> tabla2
  peso minutos atributo atributo2
1   Q2       1        a     tubo3
2   Q2       1        a     tubo3
3   Q2       1        b     tubo4
4   Q1       1        c     tubo5
5   Q1       1        b     tubo5
```

```
> summary(tabla2)
 peso      minutos      atributo atributo2
 Q1:2   Min.   :1      a:2      tubo3:2
 Q2:3   1st Qu.:1      b:2      tubo4:1
        Median :1      c:1      tubo5:2
        Mean   :1
        3rd Qu.:1
        Max.   :1
```

4.7.1.3. Transformar factores en variables numéricas

Para transformar una variable no numérica en numérica se puede hacer con la función as.numeric() o as.double(). Por supuesto que la columna del data.frame debe tener números pero que hallan sido introducidos como caracteres, ya sea manualmente o en el proceso de importación. Esto puede ocurrir en ciertos casos, cuando las planillas de cálculos no están correctamente configuradas y al introducir números lo hace en formato de texto.

supongamos que tenemos un vector con números pero como caracter

```
> a<-c("1","0.01","3.2")
```

verifiquemos que los elementos de a son caracteres.

```
> str(a)
 chr [1:3] "1" "0.01" "3.2"
```

si deseamos transformar los elementos que son caracteres y no números, en formato numérico, podemos utilizar el siguiente código y su resultado asignarlo a un vector b

```
> b<-as.numeric(a)
> b
[1] 1.00 0.01 3.20
```

comprobamos el formato de sus elementos

```
> mode(b)
[1] "numeric"
> mode(a)
[1] "character"
```

La función as.double(), genera el mismo resultados

```
> c<-as.double(a)
> c
[1] 1.00 0.01 3.20
```

Si el vector tuviera caracteres que no sean números serán forzados a NA

```
> a<-c("1","0.01","3.2","B")
> b<-as.numeric(a)
Warning message:
NAs introduced by coercion
> b
[1] 1.00 0.01 3.20   NA
```

Un caso común que puede ocurrir cuando introducimos datos en R, es que nuestras tablas tengan separador decimal ",". Si esto no se lo indicamos con el argumento dec="," al utilizar read.table(), los datos ingresarán como caracteres. Si antes de transformar los datos en numéricos no tenemos la precaución de reemplazar "," por ".", al aplicar as.numeric() dichos valores serán tratados como caracteres y serán transformados en NA, perdiendo sus datos. Estos problemas se tornan complejos y difíciles de percibir cuando las tablas exceden el tamaño de las pantalla de visualización de datos. El certero conocimiento de uso de cada función evitará problemas graves en el manejo de grandes volúmenes de datos.

4.7.2. Obtener niveles de un factor de un data.frame

Si tenemos un data.frame con factores con varios niveles, siendo la tabla visible es sencillo ver el número de niveles de dicho factor. Por ejemplo para la variable atributo de tablaR141, vemos que son 3 niveles: a, b y c. No será tan evidente en tablas que superen los 100 datos. Para ver los niveles de un factor, R nos proporciona la función levels() y factor()

tablaR141

	peso	minutos	atributo	atributo2
1	23.0	1	a	tubo3
2	23.1	1	a	tubo3
3	23.3	1	b	tubo4
4	13.5	1	c	tubo5
5	13.5	1	b	tubo5

levels() nos indica los niveles, sin indicarnos cuantas unidades pertenecen a cada nivel

```
> levels(tablaR141$atributo)
[1] "a" "b" "c"
```

factor() nos muestra todos los elementos y nos resume cuantos niveles existen

```
> factor(tablaR141$atributo)
[1] a a b c b
Levels: a b c
```

la función summary() nos muestra los niveles y cuantos unidades hay por nivel de cada factor,

es decir que summary() da la misma información que si aplicaramos levels() y factor(). Aunque summary() parece reemplazar a las otras dos, no es así y en algunas circunstancias puede ser mejor el uso de una por sobre la otra función. Veamos summary()

```
> summary(tablaR141)
      peso           minutos      atributo    atributo2
 Min.   :13.50      Min.   :1      a:2       tubo3:2
 1st Qu.:13.50      1st Qu.:1      b:2       tubo4:1
 Median :23.00      Median :1      c:1       tubo5:2
 Mean   :19.28      Mean   :1
 3rd Qu.:23.10      3rd Qu.:1
 Max.   :23.30      Max.   :1
```

4.7.3. Agregar niveles de un factor

Supongamos la tablaR141 que tiene los siguientes niveles: a, b, c, para el factor atributo. Si deseáramos agregar un nuevo nivel puede ser una circunstancia problemática. No tenemos problemas si lo hacemos con la función edit(), pero sí cuando lo hacemos por otros mecanismos, por ejemplo como veremos más adelante a través de la ejecución de scripts.

Cuando se utilizan scripts puede traer dificultad, por lo tanto para esta circunstancias conviene crear primero el nivel del factor. Supongamos que deseamos agregar un nivel "d"

La misma función levels() nos sirve para crear niveles, aun cuando este quede vacío de elementos

Utilizamos el código:

```
> levels(tablaR141$atributo)<-c(levels(tablaR141$atributo),"d")
```

con esto agregamos el nivel "d". El mismo no aparece en la tabla, ya que ninguna línea contenía dicho valor

```
> tablaR141
  peso minutos atributo atributo2
1 23.0       1        a     tubo3
2 23.1       1        a     tubo3
3 23.3       1        b     tubo4
4 13.5       1        c     tubo5
5 13.5       1        b     tubo5
```

sin embargo en un summary(), sí lo vemos con 0 elementos.

```
> summary(tablaR141)
      peso           minutos      atributo  atributo2
 Min.   : 13.50      Min.   :1    a:2       tubo3:2
 1st Qu.:13.50   1   st Qu.:1    b:2       tubo4:1
 Median :23.00      Median :1    c:1       tubo5:2
 Mean   :19.28      Mean   :1    d:0
 3rd Qu.:23.10      3rd Qu.:1
 Max.   :23.30      Max.   :1
```

4.7.4. Operaciones básicas

4.7.4.1. logaritmo natural

Para calcular logaritmos naturales (de base e) se utiliza la función log(). Para aplicarlo a todo la columna peso de la tablaR141

```
> log(tablaR141$peso)

[1] 3.135494 3.139833 3.148453 2.602690 2.602690
```

el código siguiente halla el log del elemento de la fila 1, columna 1 de la tablaR141

```
> log(tablaR141[1,1])

[1] 3.135494
```

4.7.4.2. logaritmo decimal

Para calcular logaritmo decimal (de base 10) utilizamos la función log10() o log(valor,10). El siguiente código calcula el logaritmo de base 10 de todos los elementos de la columna peso del data.frame tablaR141

```
> log10(tablaR141$peso)

[1] 1.361728 1.363612 1.367356 1.130334 1.130334
```

que es equivalente al siguiente código, donde la base es colocada como segundo argumento de la función.

```
> log(tablaR141$peso,10)

[1] 1.361728 1.363612 1.367356 1.130334 1.130334
```

4.7.4.3. raíz cuadrada

La función sqrt() calcula la raíz cuadrada de un número. Para el ejemplo la aplicamos a toda la columna peso de la tablaR141

```
> sqrt(tablaR11$peso)

[1] 4.795832 4.806246 4.827007 3.674235 3.674235
```

Pero por ejemplo si quisiéramos calcular la raiz cuadrada del número 23 aplicaríamos

```
> sqrt(23)

[1] 4.795832
```

4.7.4.4. funciones trigonométricas

De la misma manera tenemos funciones trigonómetricas. Veamos la aplicación de la función sin() (seno) de la columna peso de tablaR141

```
> sin(tablaR141$peso)

[1] -0.8462204 -0.8951874 -0.9658882  0.8037844  0.8037844
```

otras funciones trigonométricas que podemos aplicar a un número o conjunto de números (x)

coseno: cos(x)

tangente: tan(x)

arcoseno: acos(x)

arcseno: asin(x)

arctangente: atan(x)

4.7.5. Constantes

R tiene también constantes. A continuación se indican el contenido
Built-in Constants: package:base

Usage:

```
  LETTERS          # da las letras del abecedario en mayúscula
  letters          # da las letras del abecedario en minúscula
  month.abb        # da abreviatura de 3 letras para los meses del año
  month.name       # da los nombres de los meses del año
  pi               # relación del radio y la circunferencia, número pi: 3,14.....
```

por ejemplo si deseamos introducir el número π

```
> pi
[1] 3.141593
```

introduzcamos por ejemplo una columna con los meses a la tablaR141, creando la tabla2

```
> tablaR141
  peso minutos atributo atributo2
1 23.0      1       a     tubo3
2 23.1      1       a     tubo3
3 23.3      1       b     tubo4
4 13.5      1       c     tubo5
5 13.5      1       b     tubo5
```

para ello utilizamos el comando

```
> tabla2<-cbind(tablaR141,mes=c(month.abb[1:5]))
```

```
> tabla2
  peso minutos atributo atributo2 mes
1 23.0      1       a       tubo3   Jan
2 23.1      1       a       tubo3   Feb
3 23.3      1       b       tubo4   Mar
4 13.5      1       c       tubo5   Apr
5 13.5      1       b       tubo5   May
```

4.7.6. Funciones de tiempo

Los datos que representan tiempos: segundos, minutos, días, etc. Si bien son sencillos de introducir, ya que se pueden hacer en formato de texto, luego no permitirán operar con ellos. Es decir si deseamos sumar o restar tiempos, estando como caracteres será imposible. A la hora de graficar funciones a lo largo del tiempo es necesario que estos datos tengan un formato especial. Veremos como obtenerlos en ese formato. En primer lugar veamos algunas funciones de tiempo propias de R.
Algunas funciones de tiempo

Para obtener la fecha como día de la semana, mes, día del mes, hora:minuto:segundos, año utilizamos la función date()

```
> date()
[1] "Wed Nov 19 08:16:46 2014"
```

Para obtenerla en el formato año-mes-día hora:minuto:segundo ubicación horaria, utilizamos la función Sys.time()

```
> Sys.time()
[1] "2014-11-19 08:16:51 ART"
```

Para obtener solo año-mes-día del mes, utilizamos Sys.Date()

```
> Sys.Date()
[1] "2014-11-19"
```

Veamos ahora una planilla de cálculo en la cual colocamos fechas. Dichos datos los pasamos a R resultando un data.frame llamado tablaR143 (utilice la tablaR143 de la planilla de cálculo tablaR1-4.ods/xls)

```
> tablaR143<-read.table("clipboard",header=TRUE,dec=",",sep="\t")
> tablaR143
    fecha peso
1 01/02/16   20
2 03/02/16   25
3 05/02/16   28
4 17/02/16   35
5 18/02/16   40
6 20/02/16   42
7 25/02/16   48
8 10/03/16   56

> summary(tablaR143)
      fecha        peso
 01/02/16:1   Min.   :20.00
 03/02/16:1   1st Qu.:27.25
 05/02/16:1   Median :37.50
 10/03/16:1   Mean   :36.75
 17/02/16:1   3rd Qu.:43.50
 18/02/16:1   Max.   :56.00
 (Other) :2
```

vemos que la columna fecha está manejada como factor.

Si intentamos hacer una diferencia entre dos valores de la columna fecha de la tablaR143

```
> tablaR143[2,1]-tablaR143[1,1]
```

nos da un valor NA

```
[1] NA
Warning message:
In Ops.factor(tablaR143[2, 1], tablaR143[1, 1]) :
  ‘-’ not meaningful for factors
```

y nos indica que "-" no tiene sentido si tenemos factores. Lo que intentamos hacer anteriormente es como que hubiéramos intentado restar "diabético" – "normal".

Revisamos nuevamente nuestra tablaR143

```
> summary(tablaR143)
```

```
      fecha        peso
 01/02/16:1   Min.   :20.00
 03/02/16:1   1st Qu.:27.25
 05/02/19:1   Median :37.50
 10/03/16:1   Mean   :36.75
 17/02/16:1   3rd Qu.:43.50
 18/02/16:1   Max.   :56.00
```

la fecha está factorizada, donde cada fecha aparece una vez.

Similar información nos da la función str()

```
> str(tablaR143)
'data.frame':        8 obs. of  2 variables:
 $ fecha: Factor w/ 8 levels "01/02/16","03/02/16",..: 1 2 3 5 6 7 8 4
 $ peso : int  20 25 28 35 40 42 48 56
```

Entonces, ¿como operar con fechas.?

En primer lugar se deben pasar al formato entendible por R . Como las fechas las tenemos escrita como dia/mes/año, esto se lo indicamos con "%d/%m/%Y y utilizamos la función strptime()

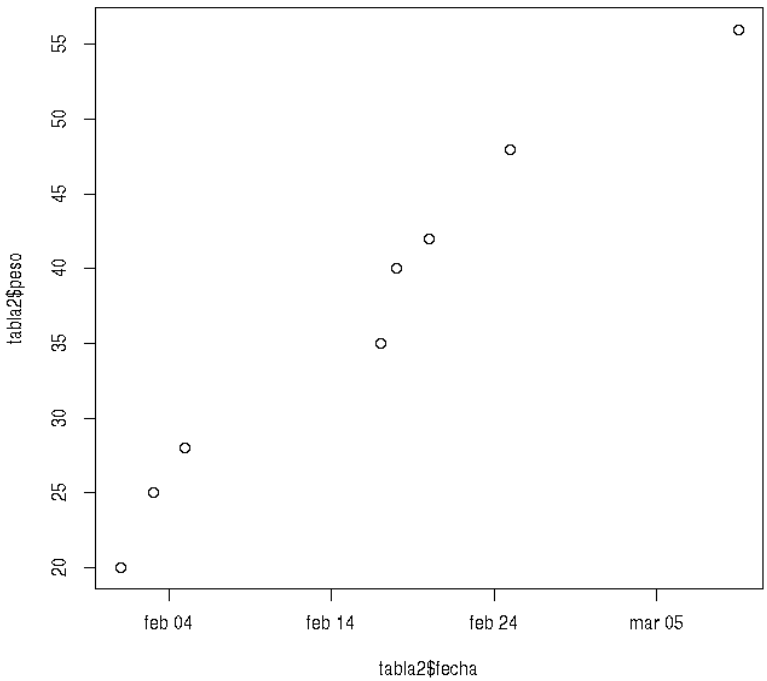
```
> tablaR143$fecha<-strptime(tablaR143$fecha,"%d/%m/%Y")
> tablaR143
    fecha peso
1 16-02-01   20
2 16-02-03   25
3 16-02-05   28
4 16-02-17   35
5 16-02-18   40
6 16-02-20   42
7 16-02-25   48
8 16-03-10   56
```

Modificó el formato pasándolo a año-mes-día. Con este formato podemos operar. Si restamos el dato de la fila 2 menos el datos de la fila 1, que claramente son dos días

```
> tablaR143[2,1]-tablaR143[1,1]
Time difference of 2 days
```

Con esta transformación también podremos graficar tiempos, manteniendo la relación entre ellos.

```
> plot(tablaR143$fecha,tablaR143$peso)
```



También podemos calcular diferencias de tiempo en diversas unidades con la función difftime(), indicando el argumento units.

```
> difftime(tablaR1143[4,1],tablaR143[1,1],units="mins")
Time difference of 23040 mins
```

units puede tomar valores: "auto", "secs", "mins", "hours", "days", "weeks

5. Clase 5

Vídeo: https://youtu.be/D0D4VxKvx5c

Tabla de datos: http://hdl.handle.net/2133/9453

5.1. Bibliotecas (packages)

5.1.1. Cargar bibliotecas

La ejecución de ciertas funciones necesitan bibliotecas especiales, que se deben cargar en el momento de utilizar la aplicación.

Si bien esto puede ser una contra de R, es una fortaleza, ya que solo tiene en memoria aquellas cosas que estamos utilizando y no todas las bibliotecas que la mayoría quedarían inactivas.

Supongamos que deseamos hacer una análisis ROC y para ello utilizamos la función roc() aplicada a un objeto ***a***, al aplicar la función seguramente obtendríamos un mensaje de error

> roc(a)

Error: could not find function "roc"

R nos indica que no reconoce la función roc() y esto se debe a que la biblioteca pROC que contiene dicha función no está instalada. Por lo tanto debemos cargar la biblioteca. Para ello se utiliza la función library()

library(pROC)

luego de este comando queda la biblioteca instalada y podremos ejecutar sus funciones. Por supuesto para que la función library() funcione, la biblioteca pROC debe estar instalada en nuestra computadora.

5.1.2. Conocer bibliotecas cargadas

Función getOption(), nos da las bibliotecas cargadas como base de R
> getOption("defaultPackages")

en nuestro caso por ejemplo

> getOption("defaultPackages")

[1] "datasets" "utils" "grDevices" "graphics" "stats" "methods"

estas son las bibliotecas cargas default al instalar R

Con la función library() conocemos los paquetes instalados en nuestra computadora y los path. Para salir la función library oprima q

>library()

nos mostrará una larga tabla de la que solo mostramos una parte

Packages in library ‘/home/alfredo/R/i686-pc-linux-gnu-library/3.2’:

abind Combine Multidimensional Arrays

acepack ace() and avas() for selecting regression transformations

agricolae Statistical Procedures for Agricultural Research

AlgDesign Algorithmic Experimental Design

assertthat Easy pre and post assertions.

..... continua

Para conocer las bibliotecas cargadas en el espacio de trabajo utilizamos la función search(), que da las bibliotecas cargadas en el espacio de trabajo

```
>search()                                #da las paquetes actualmente cargados
[1] ".GlobalEnv"        "package:stats"     "package:graphics"
[4] "package:grDevices" "package:utils"     "package:datasets"
[7] "package:methods"   "Autoloads"         "package:base"
```

La función .libPaths() indica donde están instalados los paquetes disponibles

```
> .libPaths()
[1] "/home/alfredo/R/i686-pc-linux-gnu-library/2.14"
[2] "/usr/local/lib/R/site-library"
[3] "/usr/local/lib/R/library"
```

La función installed.packages() nos muestra los paquetes instalados. Se puede colocar el retorno de la función en un data.frame y luego editarlo si queremos ver su contenido

```
> a<-installed.packages()
```

luego

```
edit(a)
```

podremos ver su contenido, ver versiones, dependencias, imports, etc.

5.1.3. instalar y eliminar paquetes

Conocido un paquete que se puede encontrar en internet, desde el prompt de R se utiliza la función install.packages("nombre de la biblioteca"). Si no anda el código siguiente intentarlo nuevamente actuando como “root”. Para eso en el prompt de linux escribir 'su' y luego la contraseña de root.

Si es usuario de R bajo windows todo será más fácil. Las instrucciones siguientes son más necesarias para usuarios de linux.

Por ejemplo si desearamos instalar la biblioteca pROC, en la línea de comando escribiremos

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

pedirá que elijamos un “mirror”, hay dos en Argentina, pero pueden ser otros. Luego de elegido, el programa lo instala solo y quedará listo para su uso.

Si tuviera problemas agregue el argumento dependencies en la función install.packages()

```
> install.packages("pROC", dependencies=TRUE)
```

busca automáticamente las dependencias necesarias para que pROC funciones. Se entiende por dependencias otras bibliotecas que pROC requiere para su funcionamiento.

también se puede instalar indicando el repositorio

>install.packages("randomForest", repos="http://cran.cnr.berkeley.edu")

5.1.4. Remover un paquete

Para remover un paquete utilizamos la función remove.packages(). Tenga cuidado que esta función remueve el paquete de nuestro disco duro y ya no lo tendemos disponible. Si deseáramos utilizar nuevamente la biblioteca deberemos ejecutar nuevamente install.packages(). El comando general es

> remove.packages("nombre de la biblioteca")

Si por ejemplo queremos remover el paquete pROC, escribiremos la siguiente línea.

> remove.packages("pROC")

5.1.5. conocer versión de un paquete

> packageVersion("FactoMineR")

[1] ‘1.25’

1.25 es la versión de la biblioteca FactoMineR

5.1.6. Instalar paquetes a partir de tar.gz

Si la función intall.packages() no ha funcionado, puede hacerlo desde los archivos tar.gz. Para ello deberá descargar los paquetes requeridos, los cuales supongamos que quedan en la carpeta Downloads, luego del proceso de descarga. Siempre si trabaja en Windows será más fácil la búsqueda de biblitecas, los pasos siguientes son para usuarios de Linux.
Los pasos a seguir para buscar una biblioteca son:
En primer lugar busque la dirección

https://cran.r-project.org/

hacer click en packages

luego hacer click en

Table of available packages, sorted by name

buscar el paquete deseado y hacer click sobre el vínculo , por ejemplo lme4

hacer click sobre el nombre del paquete, lleva a características del mismo

buscar Package source y hacer click en lme_1.1-11.tar.gz (que es la última versión)

se descargará en la carpeta Downloads de su computadora o en la que usted tenga configurado.

Si deseo una versión anterior oprima Old Sources

Aparecerán todas las versiones y haga click sobre la desea y se descargará en Downloads. Puede ser necesario utilizar versiones anteriores de un paquete en función de que versiones tenga de los paquetes en uso y de la versión de R en ejecución. Suponiendo entonces que la biblioteca descargada fue lme41_1.1-11.tar.gz, para instalarlo ejecutamos en la consola en R el

siguiente comando

install.packages("~/Downloads/lme4_1.1-11.tar.gz",repos=NULL,type="source",dependencies=TRUE)

luego ingreso a R, ya debería estar instalado. Lo cargo en el espacio de trabajo con el comando ya visto

library(nombredelpaquete)

debería funcionar.

5.1.7. Instalación sin descargar en Downloads

Una alternativa de instalación de paquetes es utilizando la dirección de internet. Por ejemplo si coloco el mouse sobre la versión 0.4-7 (Package source: pbkrtest_0.4-7.tar.gz), oprimimos el botón derecho del mouse y copiamos la dirección

https://cran.r-project.org/src/contrib/pbkrtest_0.4-7.tar.gz

y luego asignamos dicha dirección a un objeto que llamamos por ejemplo packageurl

packageurl <- "https://cran.r-project.org/src/contrib/Archive/pbkrtest/pbkrtest_0.4-4.tar.gz"

y finalmente instalo el paquete

install.packages(packageurl, repos=NULL, type="source")

5.1.8. Actualizar paquetes

La función update.packages("nombre del paquete") sirve para actualizar bibliotecas. Si no anda el código siguiente intentarlo nuevamente actuando como “root”. Para eso en el prompt de linux escribir 'su' y luego la contraseña de root.

update.packages(“nombredelpaquete”)

5.2. Siguiendo con data.frame

5.2.1. Eliminar niveles de factores no usados

Cuando tenemos un data.frame con columnas que son factores con más de un nivel, puede ocurrir que al eliminar algunas filas, queden niveles sin elementos. Esto puede ser ventajoso en ciertas circunstancias pero complicarnos en otras. Veremos como eliminar niveles de un factor

Introduzcamos en nuestro espacio de trabajo la tablaR151 de la planilla de cálculo tablaR1-5.ods/xls

```
> tablaR151<-read.table("clipboard",header=TRUE,sep="\t",dec=",")

> tablaR151
  id var1 var2
1  a    l    z
2  b    l    z
3  c    l    z
4  d    l    z
5  e    l    z
6  f    m    z
7  g    m    d
```

```
8  h    n    d
9  i    n    d
```
creamos un data.frame d1, eliminando algunas filas

```
> d1<-tablaR151[c(1:5),]
```

```
> d1
  id var1 var2
1  a    l    z
2  b    l    z
3  c    l    z
4  d    l    z
5  e    l    z
```
Si vemos un summary() de nuestra tabla original: tablaR151

```
> summary(tablaR151)
      id    var1  var2
 a     :1   l:5    d:3
 b     :1   m:2   z:6
 c     :1   n:2
 d     :1
 e     :1
 f     :1
 (Other):3
```

podemos comprobar la cantidad de líneas que contienen un valor de un dado factor. Por ejemplo para la variable var2, existen 3 líneas en que toma el valor d y 6 en que toma el valor z.

Si hacemos un summary() del nuevo data.frame d1

```
> summary(d1)
      id    var1  var2
 a     :1   l:5    d:0
 b     :1   m:0   z:5
 c     :1   n:0
 d     :1
 e     :1
 f     :0
 (Other):0
```
Vemos que hay varios niveles de los factores que quedaron sin elementos. Podemos eliminarlos si fuera necesario. Ya veremos situaciones en que es útil retenerlos y las cuales conviene eliminarlos.

Si deseamos eliminar los niveles sin elementos, debemos cargar la biblioteca gdata.

```
> library(gdata)
```

Si deseamos eliminar todos los niveles con 0 elemento y asignar el resultado a otro data.frame que llamamos d2 ejecutamos

```
> d2<-drop.levels(d1)
```

```
> summary(d2)
 id    var1  var2
 a:1   l:5   z:5
 b:1
 c:1
 d:1
 e:1
```
también podríamos eliminar niveles de factores por columnas. Supongamos que solo deseamos eliminar los niveles vacíos del factor var1 y asignarlo al mismo data.frame

```
> summary(d1)
      id    var1  var2
 a     :1   l:5    d:0
 b     :1          z:5
 c     :1
 d     :1
 e     :1
 f     :0
 (Other):0
```
también se pueden eliminar niveles de un factor directamente sin ninguna biblioteca extra, utilizando la función droplevels() que se halla la biblioteca base que se instala al instalar por primera vez R.

Veamos entonces partiendo nuevamente de la tablaR151

```
> tablaR151
  id var1 var2
1  a    l    z
2  b    l    z
3  c    l    z
4  d    l    z
5  e    l    z
6  f    m    z
7  g    m    d
8  h    n    d
9  i    n    d
```
generamos nuevamente el data.frame d1, en el que eliminamos algunas lineas

```
> d1<-tablaR151[c(1:5),]
```

obtenemos así

```
> d1
  id var1 var2
1  a    l    z
2  b    l    z
3  c    l    z
4  d    l    z
5  e    l    z
```
si pedimos un sumary de d1

```
> summary(d1)
      id    var1  var2
 a     :1   l:5   d:0
 b     :1   m:0   z:5
 c     :1   n:0
 d     :1
 e     :1
 f     :0
 (Other):0
```

vemos que tiene factores vacíos en id, var1 y var2. Entonces eliminamos esos niveles de cada factor con dropleves() y reasignamos el resultado a d1

```
> d1<-droplevels(d1)
```

si ahora pedimos un summary() de d1

```
> summary(d1)
 id    var1  var2
 a:1   l:5   z:5
 b:1
 c:1
 d:1
 e:1
```

vemos que hemos eliminado los niveles sin elementos.

Si solo deseáramos eliminar los niveles sin elementos de la columna id, procedemos de la siguiente manera. En primer lugar partamos nuevamente de la tablaR151

```
> tablaR151
  id var1 var2
1  a    l    z
2  b    l    z
3  c    l    z
4  d    l    z
5  e    l    z
6  f    m    z
7  g    m    d
8  h    n    d
9  i    n    d
```

generamos nuevamente el data.frame d1, en el que eliminamos algunas filas

```
> d1<-tablaR151[c(1:5),]
```

obtenemos así

```
> d1
  id var1 var2
1  a    l    z
2  b    l    z
3  c    l    z
4  d    l    z
```

5 e l z
si pedimos un sumary de d1

```
> summary(d1)
      id    var1  var2
 a     :1   l:5   d:0
 b     :1   m:0   z:5
 c     :1   n:0
 d     :1
 e     :1
 f     :0
 (Other):0
```

vemos que tiene factores vacíos en id, var1 y var2. Ahora eliminaremos solo los nivel de la columna id utilizando dropleves() y reasignamos el resultado a d1

```
> d1$id<-droplevels(d1$id)
```

si ahora pedimos un summary() de d1

```
> summary(d1)
  id    var1  var2
 a:1   l:5   d:0
 b:1   m:0   z:5
 c:1   n:0
 d:1
 e:1
```

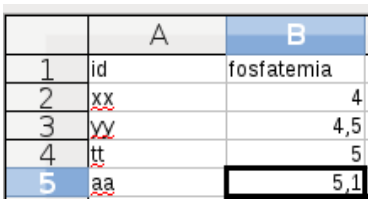
vemos que hemos eliminado los niveles sin elementos solo de la columna id.

5.2.2. Unir data.frames

Ya hemos visto como unir data.frames utilizando las funciones cbind() y rbind(). Estas funciones requieren que las cantidades de filas o columnas sean iguales y además tengan sus columnas el mismo nombre.
R provee otras alternativas que pueden ser útiles en diversas situaciones. En situaciones especiales puede tener tablas que tengan una columna común identificadora. En las figuras siguientes se muestran tres tablas, donde se tabulan valores de glucemia, calcemia y fosfatemia para diferentes pacientes. Estas tablas tienen en común que la columna A tiene a los pacientes y donde vemos que algunos (como el xx y el yy) se hallan en las tres tablas, pero otros (como el ww, tt, etc) no están presentes en todas las tablas.

	A	B
1	id	glucemia
2	xx	1
3	yy	1,1
4	zz	1,2
5	ww	0,9

	A	B
1	id	calcemia
2	xx	10
3	yy	10,1
4	zz	
5	tt	9
6	uu	9,5



	A	B
1	id	fosfatemia
2	xx	4
3	yy	4,5
4	tt	5
5	aa	5,1

Deseamos hacer una sola tabla. Este trabajo sería tedioso si quisiéramos hacerlos a mano y tendríamos un alto grado de probabilidad de error, especialmente si las tablas son grandes.

Utilice para esta sección las tablaR152, tablaR153 y tablaR154 del archivo tablaR1-5.ods/xls. Introduzca los valores con la función read.table() como ya ha aprendido.

```
> tablaR152<-read.table("clipboard",header=TRUE,sep="\t",dec=",")
```

```
> tablaR153<-read.table("clipboard",header=TRUE,sep="\t",dec=",")
```

```
> tablaR154<-read.table("clipboard",header=TRUE,sep="\t",dec=",")
```

Comprobamos que su contenido sea el correcto

```
> tablaR152
  id glucemia
1 xx      1.0
2 yy      1.1
3 zz      1.2
4 ww      0.9
> tablaR153
  id calcemia
1 xx     10.0
2 yy     10.1
3 zz       NA
4 tt      9.0
5 uu      9.5


> tablaR154
  id fosfatemia
1 xx        4.0
2 yy        4.5
3 tt        5.0
4 aa        5.1
```

Los tres data.frame tienen una columna identificatoria "id". Deseamos juntar las tablas y que asigne a cada paciente sus valores de glucemia, calcemia y fosfatemia, dejando vacío si faltara algún valor. Para ello utilizaremos la función merge(), con la que juntaremos los tres data.frame anteriores. Debe tener la precaución de ir uniendo de a dos data.frames. Para ello creamos un nuevo data.frame que llamamos tablaR en el que vamos recibiendo los datos de cada fusión

Fusionamos tablaR152 y tablaR153

```
> tablaR<-merge(tablaR152,tablaR153,by="id",all.x=TRUE,all.y=TRUE)
```

luego agregamos a tablaR el data.frame tablaR154

```
> tablaR<-merge(tablaR,tablaR154,by="id",all.x=TRUE,all.y=TRUE)
```

Como podemos ver en tablaR quedaron todos los pacientes con sus mediciones, asignando NA a aquellas mediciones que no tienen valor para un dado paciente.

```
> tablaR
  id glucemia calcemia fosfatemia
1 ww      0.9       NA         NA
2 xx      1.0     10.0        4.0
3 yy      1.1     10.1        4.5
4 zz      1.2       NA         NA
5 tt       NA      9.0        5.0
6 uu       NA      9.5         NA
7 aa       NA       NA        5.1
```

Los argumentos all.x y all.y nos permiten hacer selecciones. Cuando el valor es TRUE, la tabla será una fusión de todos los datos. Sin embargo si colocamos un argumento en FALSE eliminaremos elementos. Para comprender el efecto comparemos la fusión de solo dos de los data.frame: tablaR152 y tablaR153

```
> tablaR152
  id glucemia
1 xx      1.0
2 yy      1.1
3 zz      1.2
4 ww      0.9

> tablaR153
  id calcemia
1 xx     10.0
2 yy     10.1
3 zz       NA
4 tt      9.0
5 uu      9.5
> tablaR<-merge(tablaR152,tablaR153,by="id",all.x=FALSE,all.y=FALSE)

> tablaR
  id glucemia calcemia
1 xx      1.0     10.0
2 yy      1.1     10.1
3 zz      1.2       NA
```

vemos que tablaR solo nos dejó los datos de los individuos xx, yy, zz ya que son los único tres que figuran en las dos tablas. Los individuos ww,tt y uu fueron eliminados por estar en una sola de las tablas utilizadas.

5.3. Escritura y autocompletado

Esta es una ayuda muy grande. Cuando estamos escribiendo, antes de terminar de escribir el

nombre de una función o un objeto oprimimos Tab y R autocompletará el nombre o bien nos mostrará con dos Tab consecutivos, todas las funciones u objetos que comienzan con ese nombre.
Por ejemplo si sabemos que una función comienza con "read" pero no conocemos el comando completo podemos escribir en la línea de comando
> read
oprimimos dos veces el tabulador {tab} {tab} y veremos toda las funciones que comienzan con read

readBin	read.csv	read.delim	read.fortran	readline	readRenviron
readChar	read.csv2	read.delim2	read.ftable	readLines	read.socket
readCitationFile	read.dcf	read.DIF	read.fwf	readRDS	read.table

5.4. Busque de funciones por string

Si el recurso anterior no hubiera funcionado, debido a que solo lo hace con los primeros caracteres de la función, tenemos la función apropos(). Esta función es útil cuando recordamos alguna cadena de caracteres sin saber realmente si está al inicio o en el medio del nombre de la función. Supongamos que recordamos que una función de utilidad en alguna parte de su nombre tenía "chi", utilizamos el siguiente comando

```
apropos("chi")                #buscará todo lo que contenga chi

[1] "ChickWeight" "chickwts"    "chisq.test"  "dchisq"      ".Machine"

[6] "pchisq"      "qchisq"      "rchisq"
```

5.5. Comprobar si un objeto existe

Muchas veces los espacios de trabajo nos han crecido de manera desmedida y recordar o encontrar un objeto puede ser dificultoso. Existen varios recursos.
Uno de ellos es pedir que nos liste los objetos de nuestros espacio de trabajo con la función ls()

```
> ls()
 [1] "a"          "aa"         "anova"      "aorkcontrol" "aov"
 [6] "b"          "dd"         "DD"         "df"          "f"
```

Si la lista de objetos es muy grande, puede ser engorroso y pasar por desapercibido un objeto. En dicho caso se puede utilizar la función exist(). Veamos la aplicación a un objeto que existe en el listado anterior y uno que no.

```
exists("dd")

TRUE

exists("ddd")

FALSE
```

por supuesto también se puede escribir el nombre del objeto

```
> a
```

y si este existe nos dará su valor

5.6. Ayudas

Permite buscar ayudas acerca de funciones. Solo hay que escribir

help(nombredelafunción, paquete,etc)
y R nos mostrará una detallada, aunque a veces completa ayuda al respecto, además de vínculos a funciones o paquetes relacionados. Además tendremos ejemplos para practicar dichas funciones y familiarizarnos.
Si buscamos una función en particular pero la ayuda no nos da respuesta podemos utilizar el siguiente código
??nombredeloquebusco
por ejemplo si busco "catch"
??catch
R me mostrará todos los paquetes que pueden tener algo relacionado a lo que queremos.

6. Crear archivos de ayuda

Muchas veces deseamos dejar registros de detalles de un día para otro, qué hicimos, qué debemos hacer, etc. El R.History es una memoria de lo hecho, pero ciertos detalles no nos quedan tan accesibles. Crear en el espacio de trabajo nuestra propia ayuda puede ser de utilidad

> ayuda<-file("ayuda.txt")

Este comando crea en el espacio y en el directorio de trabajo un archivo txt

luego escribimos desde la consola

> writeLines(c("ultimas cosas realizadas el 23/9: cargado de datos","\n","quedo para hacer mañana: analisis"),ayuda)

> close(ayuda)

> file.show("ayuda.txt") #nos mostrará que hicimos y que obligaciones tenemos.

Otra forma más práctica es antes de finalizar escribir lineas con el caracter #

por ejemplo

mañana tengo que hacer los análisis de las variancias del grupo A

luego cuando regrese al espacio de trabajo, si deseo saber que quedaba por hacer, escribo

history()

y veré todo lo hecho con las líneas finales que indican lo que dejé como mensaje.

6.1. Concatenación de string

La concatenación de strings o unión de elementos compuestos por caracteres es muchas veces necesario como veremos más adelante. Por ejemplo si tenemos dos vector: a y b, que tienen cada uno de ellos un elemento string, por ejemplo

a<-c("casa")

b<-c("bonita")

queremos por alguna necesidad un objeto que tenga los dos elementos

c<-paste(a,b,sep=" ")

en sep="" se puede colocar cualquier caracter que se desee para separar

si no se coloca sep, por defecto coloca un espacio en blanco, de igual manera que si colocamos sep=" ",

```
c
```

```
[1] "casa bonita"
```

7. Clase 6

Vídeo: https://youtu.be/t8dIglw1kpo

Tabla de datos: http://hdl.handle.net/2133/9454

7.1. Importación de datos desde planillas de cálculo (casos especiales y dificultades)

Revisión y aclaración sobre introducción de datos desde planillas de cálculos. Veremos varios ejemplos y analizaremos los errores y las posibles soluciones. Para seguir esta explicación utilice las tablas provistas en el archivo tablaR1-6.xls o tablaR1-6.ods

7.1.1. Solución a problemas de punto decimal

La tablaR161 tiene comas (",") como separador entero-decimal. Esto suele traer problemas si no se está alerta. Es más, el problema puede pasar desapercibido y provocar errores de interpretación. Como R importa los datos utilizando "." como separador entero-decimal, si no se toman medidas adecuadas los datos serán introducido como caracteres, perdiendo de esta manera la posibilidad de operar con los datos.

Utilizamos el comando siguiente sin ningún argumento

```
> tablaR161<-read.table("clipboard",header=TRUE)
```

realizamos las comprobaciones correspondientes para ver si la importación fue correcta

```
> tablaR161
  id   peso
1 aa  1,5
2 bb  1,3
3 cc  1,5
4 dd  1,1
```

al parecer ha sido introducida la tabla de manera correcta. Sin embargo si realizamos una exploración más detallada con la función summary()

```
> summary(tablaR161)
  id       peso
 aa:1     1,1:1
 bb:1     1,3:1
 cc:1     1,5:2
 dd:1
```

vemos que el peso ha sido introducido como factores y no podremos operar con ellos o calcular estadísticas descriptiva correspondiente a datos numéricos: media, mediana, etc. Recuerde que cuando los datos son incorporados como números, al pedir summary() de dichos datos nos mostrará media, mediana, etc.

Comprobamos que si bien la columna peso tiene datos, estos no son números reales. Aplicamos una operación que los requiere, a modo de comprobación. Por ejemplo intentamos calcular la media de la columna peso

```
> mean(tablaR161$peso)

[1] NA

Warning message:

In mean.default(tablaR161$peso) :

  argument is not numeric or logical: returning NAurning NA
```

R nos alerta conun Warning, indicándonos que no se puede aplicar la función mean() debido a el argumento no es numérico.

El comando correcto para evitar este problema sería indicar con un argumento que el separador decimal es ",". Para ello utilizamos el argumento "dec=",".

```
> tablaR161<-read.table("clipboard",header=TRUE,dec=",")
```

podemos comprobar la importación de los datos

```
> tablaR161
  id peso
1 aa  1.5
2 bb  1.3
3 cc  1.5
4 dd  1.1
```

y comprobemos el tipo de datos

```
> summary(tablaR161)
  id         peso
 aa:1    Min.   :1.10
 bb:1    1st Qu.:1.25
 cc:1    Median :1.40
 dd:1    Mean   :1.35
         3rd Qu.:1.50
         Max.   :1.50
```

como podemos ver, la función summary() nos indica que la columna peso ha sido introducida como números, mostrándonos ciertas estadísticas como min, max, mediana, media, etc. valores que no podría mostrar si no fueran números los introducidos.

Comprobamos que así lo es, haciendo una operación

```
> mean(tablaR161$peso)

[1] 1.35
```

Aclaración importante: es conveniente siempre colocar el argumento dec="." o dec="," según sea su separador. Si lo toma como costumbre le evitará inconvenientes.

7.1.2. Solución a problemas de celdas vacías

Las celdas vacías en una tabla suelen traer problemas a la hora de introducir los datos en el

espacio de trabajo. La tablaR162 tiene celdas vacías, si bien es un problema manejable, hay que estar alerta a la hora de la importación.

Si aplicamos el comando considerando de manera correcta el separador parte entera-decimal

```
> tablaR162<-read.table("clipboard",header=TRUE,dec=",")

Error in scan(file, what, nmax, sep, dec, quote, skip, nlines, na.strings,  :

  line 2 did not have 2 elements
```

Nos da un error indicándonos que la línea 2 no tiene 2 elementos como debería tenerlos. Cláramente R, en estas condiciones no detecta que en la celda de la columna 2 fila 3 no hay dato. Como consecuencia R no importa el dato, cosa que podemos comprobar

```
> tablaR162

Error: object 'tablaR162' not found
```

Para lograr introducir igualmente la tablaR162, debemos indicarle esto y lo hacemos incluyendo el argumento sep= "\t", con el siguiente comando

```
> tablaR162<-read.table("clipboard",header=TRUE,dec=",",sep="\t")
```

vemos la tabla

```
> tablaR162
  id    peso
1 aa   1.5
2 bb   NA
3 cc   1.5
4 dd   1.1
```

comprobamos que en la fila 2 segunda columna, R ha colocado un NA, que indica no available.

Si pedimos un summary() de la tabla

```
> summary(tablaR162)
  id         peso
 aa:1   Min.   :1.100
 bb:1   1st Qu.:1.300
 cc:1   Median :1.500
 dd:1   Mean   :1.367
        3rd Qu.:1.500
        Max.   :1.500
        NA's   :1
```

los comandos y salidas anteriores nos indican que la columna peso es numérica y que hay un NA (not available) en la fila 2, columna 2.

7.1.3. Valores de campos con caracteres que tienen espacios.

Cuando una celda tiene caracteres que contienen espacios, se presenta un problema que impide la importación de la tabla. En la tablaR163 la primer línea tiene el valor "a a" en la primer fila columna 1. Si no se le aclara R interpretará que la primer fila tiene tres elementos: a, a y 1,5.

Si importamos con el comando siguiente

```
> tablaR163<-read.table("clipboard",header=TRUE,dec=",")

Error in scan(file, what, nmax, sep, dec, quote, skip, nlines, na.strings,  :

  line 2 did not have 3 elements
```

nos da un error en la línea 2, ya que interpreta que hay 3 elementos por línea, luego de leer la primer línea de la tabla.

Se soluciona este problema con el mismo argumento que utilizamos para espacios vacíos: sep="\t"

```
> tablaR163<-read.table("clipboard",header=TRUE,dec=",",sep="\t")
```

vemos la tabla, en la que se reproduce el contenido de la planilla de cálculo

```
> tablaR163
   id peso
1 a a  1.5
2  bb  1.3
3  cc  1.5
4  dd  1.1
```

y un summary() de la tablaR163

```
> summary(tablaR163)
   id        peso
 a a:1   Min.   :1.10
 bb :1   1st Qu.:1.25
 cc :1   Median :1.40
 dd :1   Mean   :1.35
         3rd Qu.:1.50
         Max.   :1.50
```

nos garantiza la perfecta introducción.

Aclaración: Si su planilla de cálculo no tuviera espacios vacíos o campos con caracteres con espacios, si coloca el argumento sep="\t", no le ocasionará errores. Se sugiere siempre introducir ese argumento, si se necesita estará presente y si no se necesita, se ignora.

7.1.4. Problemas con acentos

Si tenemos una tabla donde hay campos de caracteres y tienen acento, es un problema ya que luego de introducido nos pueden aparecer campos con signos extraños. Una solución es escribir sin acentos, pero en algunos caso puede ser problemático. Introduzcamos la tablaR164 utilizando los códigos conocidos.

```
> tablaR164<-read.table("clipboard",header=TRUE,dec=",",sep="\t")
```

si vemos la tabla, identificamos algunos signos extraños resaltados en este caso en amarillo.

```
> tablaR164
          id glucemia localidad
1 juan perez     1.10 El ortondo
2  tomas pel     1.50 Beraveb\xfa
```

3	rita alejandra fomez	0.98	casilda
4	anastasia krim	1.20	Villada
5	guido alvarez	1.10	Casilda
6	segundo cuad	0.97	ciudad de casilda
7	mairo ereta	0.90	Galvez
8	maria alan	0.90	Perez
9	juan fonten	1.10	Perez
10	luz crusa	1.58	p\xe9rez
11	maria luz brunati	1.54	Rosario
12	pedro pani	1.10	rosario
13	nora natan	1.10	ciudad de Rosario
14	pedro adrian lumani	1.50	San Lorenzo

vemos que los acentos no han aparecido en el data.frame y sí otros códigos en su lugar. Para solucionar el problema debemos fijar el argumento encoding= "" en valores específicos. A continuación se muestra la solución al problema, utilizando el argumento encoding, con el valor 'latin1'. Quizás tenga que probar varios valores del argumento encoding para lograr en su computadora un correcto ingreso. Esto depende de sistemas operativos, versiones existentes y configuraciones del sistema

```
> tablaR164<-read.table("clipboard",header=TRUE,dec=",",sep="\t",encoding="latin1")
```

Si vemos ahora la tabla notamos que ya no existen los signos extraños y aparecen los acentos.

```
> tablaR164
```

	id	glucemia	localidad
1	juan perez	1.10	El ortondo
2	tomas pel	1.50	Beravebú
3	rita alejandra fomez	0.98	casilda
4	anastasia krim	1.20	Villada
5	guido alvarez	1.10	Casilda
6	segundo cuad	0.97	ciudad de casilda
7	mairo ereta	0.90	Galvez
8	maria alan	0.90	Perez
9	juan fonten	1.10	Perez
10	luz crusa	1.58	pérez
11	maria luz brunati	1.54	Rosario
12	pedro pani	1.10	rosario
13	nora natan	1.10	ciudad de Rosario
14	pedro adrian lumani	1.50	San Lorenzo

7.1.5. Conclusiones:

Para evitar problemas con tablas con celdas vacías, con cadenas de caracteres con espacios o acentos se recomienda
1- introducir siempre el argumento sep="\t"

```
> tablaR161<-read.table("clipboard",header=TRUE,dec=",",sep="\t")
```

2- luego de introducir una tabla (por ejemplo llamada "tablaR161") ejecute los siguientes comandos para realizar auditoría de los datos.

2.1- que le permitan revisar los nombres de las columnas, deben coincidir con la planilla de

cálculo

```
> names(tablaR161)
```

```
[1] "id"   "peso"
```

2.2- que le permite ver los nombres de columnas y los primeros datos

```
> head(tablaR161)
  id peso
1 aa  1.5
2 bb  1.3
3 cc  1.5
4 dd  1.1
```

2.3- que le permite ver columnas y tipo de datos

```
> summary(tablaR161)
  id         peso
 aa:1   Min.   :1.10
 bb:1   1st Qu.:1.25
 cc:1   Median :1.40
 dd:1   Mean   :1.35
        3rd Qu.:1.50
        Max.   :1.50
```

2.4- que le permite chequear el número de columnas

```
> ncol(tablaR161)
```

```
[1] 2
```

2.5- que le permite chequear el número de filas

```
> nrow(tablaR161)
```

```
[1] 4
```

El tiempo invertido en comprobar la correcta introducción de datos, lo recuperará inmediatamente y garantizará un inicio correcto del análisis. Es muy desalentador darse cuenta luego de un tiempo que los datos bajo análisis no son adecuados.

7.2. Importación de datos desde archivos de texto

Tablas. Puede ocurrir que sus datos estén en un archivo de texto con extensión .odt o .doc y se hallen en formato de tabla. Puede convertir la tabla a formato de texto con la función del procesador de texto. Luego copiarla e importarla como hizo con planillas de cálculo. Utilizaremos la tabla del archivo tablatextoR1-6.doc/odt

por ejemplo tenemos la tabla mencionada

tablatextoR161

id	peso
aa	80,5
bb	78,0
cc	23,8

la convertimos en texto con la función del procesador de texto convert .... Table to text. Esto es para writer de LibreOffice, pero debe ser similar en todos los procesadores. Con esto resulta

id peso
aa 80,5
bb 78,0
cc 23,8

marcamos la tabla y la copiamos al portapapeles, luego ejecutamos el código siguiente

```
> tablatextoR161<-read.table("clipboard",header=TRUE,sep="\t",dec=",")
Warning message:
In read.table("clipboard", header = TRUE, sep = "\t", dec = ",") :
  incomplete final line found by readTableHeader on 'clipboard'
```

ignore este warning(). Veamos la tabla introducida.

```
> tablatextoR161
  id peso
1 aa 80.5
2 bb 78.0
3 cc 23.8
```

Si tuviera simplemente un texto, no en formato tabla, puede transformarlo en tabla con las funciones que le provee su procesador y luego proceder como se indicó anteriormente.

Otra alternativa es eliminar todos los separadores, dejando espacio entre las columnas

```
> tablatextoR161<-read.table("clipboard",header=TRUE,dec=",",sep="")
Warning message:
In read.table("clipboard", header = TRUE, dec = ",", sep = "") :
  incomplete final line found by readTableHeader on 'clipboard'
>tablatextoR161
  id peso
1 aa 80.5
2 bb 78.0
3 cc 23.8
```

```
> summary(e)
 id         peso
 aa:1   Min.   :23.80
 bb:1   1st Qu.:50.90
 cc:1   Median :78.00
        Mean   :60.77
        3rd Qu.:79.25
        Max.   :80.50
```

7.3. Funciones útiles

En clases anteriores vimos algunas funciones y estadísticas muy útiles. Recordémoslas y agreguemos otras

introduzcamos la tablaR165 de la planilla de cálculo tablaR1-6.ods/xls

```
> tablaR165<-read.table("clipboard",header=TRUE,dec=",",sep="\t")
```

```
> tablaR165
   id glucemia
1   aa     1.10
2    b     1.50
3   cc     0.98
4    d     1.20
5    e     1.10
6   ef     0.97
7   tt     0.90
8   rt     0.90
9   yr     1.10
10  er     1.58
11  ee     1.54
12 wer     1.10
13   b     1.10
14   h     1.50
15   y     0.98
16   u     1.20
17   i     1.10
18   o     0.97
19   j     0.90
20   h     0.90
21   r     1.10
22  dd     1.58
23   w     1.54
24   s     1.10
```

Revisemos las funciones y estadísticas conocidas, que aplicaremos a una de las columnas

la suma de todos los datos de la columna glucemia

```
> sum(tablaR165$glucemia)
```

[1] 27.94

la media de los datos de glucemia

> mean(tablaR165$glucemia)

[1] 1.164167

la variancia

> var(tablaR165$glucemia)

[1] 0.05703406

la mediana

> median(tablaR165$glucemia)

[1] 1.1

el valor máximo

> max(tablaR165$glucemia)

[1] 1.58

y el mínimo de glucemia hallado en la tabla

> min(tablaR165$glucemia)

[1] 0.9

el mínimo y el máximo, es decir el rango de valores

> range(tablaR165$glucemia)

[1] 0.90 1.58

Ahora agregaremos otras muy útiles a la hora de una descripción de muestras.

7.3.1. percentilos

Si queremos los percentilos más comunes

> quantile(tablaR165$glucemia)

0% 25% 50% 75% 100%

0.9000 0.9775 1.1000 1.2750 1.5800

Los percentilos 0% y 100% son los extremos, como vemos con la función range() y el percentilo 50% es la mediana, que podemos comprobar con la función median()

> range(tablaR165$glucemia)

[1] 0.90 1.58

> median(tablaR165$glucemia)

[1] 1.1

también es posible calcular uno de los percentilos, indicando la cantidad de valores menores

que él en nuestra muestra

```
> quantile(tablaR165$glucemia,probs=0.95)
  95%
1.574
```

7.3.2. Intervalo intercuartilos

El intervalo intercuartilo es una estadística útil indicándonos la diferencia entre el cuartilo 75 y el 25%

```
> IQR(tablaR165$glucemia)
[1] 0.2975
```

es la diferencia entre el 3er y 1er cuartilo

```
> quantile(tablaR165$glucemia,probs=0.75)-quantile(tablaR165$glucemia,probs=0.25)
   75%
0.2975
```

7.3.3. Moda o modo

Se denomina moda o modo al valor de la variable que más más veces se presenta.

R no tiene una "in-built function" para calcular la moda, pero nos podemos arreglar de la siguiente manera. Veamos primero la variable numérica glucemia en nuestra tablaR165. Transformamos la variable glucemia en factor, asignándola a un nuevo objeto (solo a fines de no afectar nuestros datos). El objeto la llamaremos mode y aplicaremos la función as.factor() a la columna glucemia. Con esta función los valores de glucemia pasarán a ser niveles de un factor.

```
> mode<-data.frame(as.factor(tablaR165$glucemia))
> summary(mode)
 as.factor.tablaR165.glucemia.
 1.1    :8
 0.9    :4
 0.97   :2
 0.98   :2
 1.2    :2
 1.5    :2
 (Other):4
```

como podemos ver en el summary(), el valor 1,1 se presenta 8 veces, siendo el más frecuente y por lo tanto la moda.

7.3.4. Mediana de las desviaciones absolutas

La mediana de las desviaciones absolutas es una medida robusta de la dispersión de datos. Calcula la sumatoria de los valores absolutos de los desvíos de cada valor respecto de la mediana. No es tan influenciada como el desvío estándar por valores extremos. Se calcula con

la función mad()

```
> mad(tablaR165$glucemia, center=median(tablaR165$glucemia))

[1] 0.185325
```

7.4. Exportar un data.frame

Para sacar un data.frame de R y llevarlo por ejemplo a un archivo o a otro espacio de trabajo, básicamente tenemos dos formas de exportar los datos. Una de ellas es hacia una planilla de cálculos en formato .csv con el función write.table() y otra es hacia un archivo de datos del tipo .RData con la función save()

7.4.1. write.table ()

Supongamos que hemos realizado trabajos de recopilación, ordenamiento y análisis de datos, de manera que (como es recomendado) todo quedó en un data.frame. Puede ocurrir que lo desee exportar hacia otro tipo de archivos, por ejemplo una planilla de cálculo.

Tomamos el data.frame tablaR164 que tenemos en nuestro espacio de trabajo

```
> tablaR164
                      id glucemia       localidad
1              juan perez     1.10      El ortondo
2               tomas pel     1.50        Beravebú
3  rita alejandra fomez      0.98          casilda
4          anastasia krim     1.20         Villada
5           guido alvarez     1.10          Casilda
6           segundo cuad      0.97  ciudad de casilda
7             mairo ereta     0.90          Galvez
8              maria alan     0.90           Perez
9              juan fonten    1.10           Perez
10               luz crusa    1.58          pérez
11     maria luz brunati      1.54         Rosario
12             pedro pani     1.10         rosario
13             nora natan     1.10  ciudad de Rosario
14 pedro adrian lumani       1.50       San Lorenzo
```

Para exportarlo utilizamos el siguiente código

```
> write.table(tablaR164, file = "tablaR164.csv", sep = "\t",eol="\n", dec=".", row.names=F, col.names=T)
```

Argumentos:

tablaR164: nombre del data.frame a exportar

"tablaR164.csv" nombre con que se exportará la planilla, la que la obtendremos en formato csv.

sep="\t" , serán los valores separados por tabulador

eol= "\n" #end of line. Al final de cada línea del data.frame comenzará una nueva fila

dec="." se exportará con formato parte entera separa de decimal por ".".

row.names=FALSE no pondrá los números de líneas.

col.names=TRUE colocará como nombre de columnas los nombres de columnas del data frame.

En el directorio donde se halla su espacio de trabajo hallará un archivo con el nombre tablaR164.csv. Luego el archivo puede abrirse desde cualquier programa de planillas de cálculo o procesador de texto, ya que fue exportado como csv (comma separated values)

7.4.2. save ()

Con la función save() podemos exportar datos en formato .RData, de manera que al enviar el archivo puede ser directamente introducido a otro espacio de trabajo con la función load().

Exportemos el data.frame tablaR164

```
> save(tablaR164, file = "tablaR164.RData")
```

con el comando anterior lo que estamos haciendo es tomar el data.frame tablaR164 y exportarlo como un archivo tablaR164.RData, que se hallará en el directorio de nuestro espacio de trabajo. El archivo .RData, puede ser oculto por lo que en algunos casos puede ocurrir que requiera configurar su máquina para verlos. Si trabaja en linux oprima simultáneamente alt+H y los verá u ocultará.

Luego de ejecutar el comando compruebe que en el directorio se halla el archivo tablaR164.RData.

Ahora veremos como importar al espacio de trabajo este archivo. Como en nuestro espacio de trabajo está la tablaR164, la debemos borrar

```
> rm(tablaR164)
```

compruebe que no se halla en su espacio de trabajo. Para ello ejecute

```
> tablaR164
```

Error: object 'tablaR164' not found

comprobamos que tablaR164 ya no existe. Pero tenemos el archivo generado anteriormente, entonces iniciamos la importación. Esto se realiza con la función load()

```
> load("tablaR164.RData")
```

llame ahora

```
> tablaR164
```

	id	glucemia	localidad
1	juan perez	1.10	El ortondo
2	tomas pel	1.50	Beravebú
3	rita alejandra fomez	0.98	casilda
4	anastasia krim	1.20	Villada
5	guido alvarez	1.10	Casilda
6	segundo cuad	0.97	ciudad de casilda
7	mairo ereta	0.90	Galvez
8	maria alan	0.90	Perez
9	juan fonten	1.10	Perez
10	luz crusa	1.58	pérez

11	maria luz brunati	1.54	Rosario
12	pedro pani	1.10	rosario
13	nora natan	1.10	ciudad de Rosario
14	pedro adrian lumani	1.50	San Lorenzo

Este método es práctico en el sentido que si tiene que enviar datos a otra persona o bien mover datos de una espacio de trabajo a otro, no tendrá incompatibilidades que suelen presentarse al exportar a planillas de cálculo, especialmente si son de diferentes sistemas operativos.

8. Clase 7

Vídeo: https://youtu.be/9cMntwlzf4c

Tabla de datos: http://hdl.handle.net/2133/9455

8.1. Guardar cambios del espacio de trabajo mientras se trabaja

Cuando salimos del espacio de trabajo con la función q(), R nos preguntará
Save workspace image? [y/n/c]:
Si oprimimos y, se guardarán todos los cambios y saldremos del espacio de trabajo. Sii oprimimos n, no se guardarán, pero igualmente saldremos del espacio de trabajo y por ende perderemos cambios realizados en objetos o nuevos objetos creados. Oprimiendo c volvemos al espacio de trabajo. Recuerde que todo lo que hizo durante una sesión de trabajo solo se guardará en este punto. Esto demanda un riesgo importante ya que cualquier falla que implique el cierre prematuro del programa hará que se nos pierda todo lo trabajado. Se recomienda cada tanto ejecutar la función save.image(), que guarda el trabajo realizado
> save.image()

De esta manera si sale del espacio de trabajo, todo lo que ejecuto hasta save.image() quedará guardado.

8.2. Ver los comando previos

Muchas veces es de utilidad recordar o utilizar comandos anteriores. Estos comando quedan guardados en un archivo oculto .Rhistory que se halla en el mismo directorio de trabajo. Podemos desde el espacio de trabajo ver estos comandos con la función history()
> history()

Este comando muestra los últimos 25 comandos usados

Para salir luego de esta pantalla debe escribir q.

Si desea ver todos los comando realizados utilice la misma función con el argumento max.show=inf

> history(max.show=Inf)

Con este argumento la función history() muestra todos los comando.

Si deseamos guardar los comandos durante la ejecución y no esperar a salir de R podemos utilizar el comando

> savehistory(file=".Rhistory")

Habitualmente cuando se carga el espacio de trabajo se carga el archivo .Rhistory. En caso que no hubiera ocurrido debemos ejecutar la función loadhistory()

.

> loadhistory(file=".Rhistory")

R de manera automática guarda una archivo .Rhistory, si al salir de espacio con la función q(), aceptamos guardar el espacio de trabajo..

Usted puede querer guardar la historia de comandos del día por ejemplo con el nombre .R30-10-17

Este archivo solo tendrá los comandos del día. Puede ser bueno para tener las cosas ordenadas por fecha

8.3. Unir dos o más .RData

Muchas veces nos han quedado objetos en más de un data.frame y los necesitamos para el trabajo en que estamos ubicados, que es diferente de otro que tiene nuestros objetos de interés.

Para entender este tema arranque R dentro de un directorio que puede llamar Clase1. Cree allí dos objetos del tipo vector a los que llamaremos clase1 y clase2.

clase1<-c(1,1)

clase2<-c(2,2)

Salga de R y del directorio, arranque R nuevamente pero dentro de otro directorio que podemos llamar clase2. Cree allí dos objetos del tipo vector que llamaremos vector2a y vector2b

vector2a<-c("a","b")

vector2b<-c("f","g")

Si arrancamos R por ejemplo en directorio clase 2, con la función ls() podemos comprobar que tenemos los objetos mencionados.

> ls()

[1] "vector2a" "vector2b"

comprobamos el directorio de trabajo con

> getwd()

[1] "/home/alfredo/alf/cursos/R/clase2"

Como deseamos fusionar este espacio de trabajo con otro en que tenemos los elementos clase1 y clase2, lo primero que tenemos que hacer es pasar al directorio clase1, para ellos ejecutamos el comando

> setwd("/home/alfredo/alf/cursos/R/modulo1/clase1")

Ahora estamos en el mismo espacio de trabajo, pero hemos entrado al directorio clase1, cosa que podemos comprobar

> getwd()

[1] "/home/alfredo/alf/cursos/R/modulo1/clase1"

cargamos el espacio de trabajo que se halla en clase1

> load(".RData")

Comprobamos ahora la composición de nuestro espacio de trabajo

> ls()

[1] "clase1" "clase2" "vector2a" "vector2b"

vemos que tenemos los objetos de los .RData de los directorios clase1 y clase2

Decidimos en que .RData y directorio deseamos que quede nuestro .RData fusionado

Si deseamos que los objetos de los dos .RData queden en clase2, debemos pasar a ese directorio

> setwd("/home/alfredo/alf/cursos/R/modulo1/clase2")

y luego grabar

ya sea con save.image()

o bien al salir colocar guardar.

Si abrimos nuevamente el espacio de trabajo veremos los objetos de ambos .RData.

8.4. Problemas con datos faltantes (NA)

Los datos faltantes en un data.frame suele ser un problema, para lo cual conviene estar preparado y saber como resolverlo. Conocer su existencia es importante a la hora de cálculos

8.4.1. Conocer si un objetos tiene datos faltantes (NA)

Es importante saber reconocer si un objeto tiene o no elementos faltantes, que aparecen como NA. Veamos como procederemos y para ello importamos datos de tablaR171

```
> tablaR171<-read.table("clipboard",header=TRUE,dec=",",sep="\t")
> tablaR171
  peso minutos atributo
1 23.10      1       a
2 23.00      1       a
3 23.32      1       b
4 23.85      1       c
5    NA      1       c
```

Obviamente en la tablaR171 vemos la ausencia de un dato en la columna 1, sin embargo en tablas de miles de datos será imposible verlos y deberemos recurrir a funciones que nos permitan ver lo que a los ojos no es evidente. La función na.fail() es una solución

```
> na.fail(tablaR171)
Error in na.fail.default(tablaR171) : missing values in object
```

Está indicando que hay datos faltantes.

Si ahora importamos datos de tablaR172 que no tiene datos faltantes

```
> tablaR172<-read.table("clipboard",header=TRUE,dec=",",sep="\t")
```

vemos la tabla introducida

```
> tablaR172
  peso minutos atributo
1 23.10      1       a
2 23.00      1       a
3 23.32      1       b
4 23.85      1       c
5 28.00      1       c
```

ahora ejecutamos na.fail()

```
> na.fail(tablaR172)
```

```
  peso minutos atributo
1 23.10       1       a
2 23.00       1       a
3 23.32       1       b
4 23.85       1       c
5 28.00       1       c
```

devuelve la tabla, esto nos indica que no hay NA.

8.4.2. Remover NA de operaciones

La presencia de valores NA en filas y columnas puede tener efectos indeseables. Por ejemplo si queremos calcular una suma y entre los datos hay un NA, el resultado de la operación nos dará como resultado NA, lo cual es obvio. Pero si queremos realizar igual la operación con los valores diferentes de NA, la función na.rm() es la solución para excluir del cálculo a aquellos valores NA

Veamos unos ejemplos. En primer lugar calculemos la media de los pesos de la tablaR172

```
> mean(tablaR172$peso)
[1] 24.254
```

Como la columna peso de la tablaR172 no tiene valores faltantes, el cálculo no tiene ninguna dificultad. Sin embargo si realizamos la misma operación con la columna peso de la tablaR171, es resultado no es el mismo

```
> mean(tablaR171$peso)
[1] NA
```

como uno de los valores de la columna peso en esta tabla es NA, el resultado de la media es NA. Para excluir del cálculo los valores NA se utiliza el argumento na.rm() con el siguiente código.

```
> mean(tablaR171$peso,na.rm=TRUE)
[1] 23.3175
```

Es decir que calculó la media con los valores disponibles, pero sin conosiderar el valor NA existente en la columna.

8.4.3. Eliminación de filas con NA

Cuando tenemos valores NA en una columna y deseamos hacer una operación, ya hemos visto la dificultad y como salvarla con el argumento na.rm. Otra alternativa, es omitir la fila que contiene NA y para ello podemos utilizar la función na.omit() que nos creará una tabla sin la fila que contiene NA. Volvamos a la tablaR171

```
> tablaR171
  peso minutos atributo
1 23.10       1       a
2 23.00       1       a
3 23.32       1       b
4 23.85       1       c
5    NA       1       c
```

la función na.omit() nos permite omitir la fila con NAs.

```
> na.omit(tablaR171)
  peso minutos atributo
1 23.10       1       a
2 23.00       1       a
3 23.32       1       b
```

```
4 23.85      1       c
```
Si quisiéramos hacer la suma de los valores de la columna con NA, podemos anidar las funciones sum() y na.omit().
```
> sum(na.omit(tablaR171$peso))
[1] 93.27
```
El resultado que se obtiene es el mismo que si utilizáramos
```
> sum(tablaR171$peso,na.rm=TRUE)
[1] 93.27
```
La función na.exclude() da el mismo resultado que la función na.omit() y que el argumento na.rm=TRUE
```
> na.exclude(tabla71)
  peso   minutos atributo
1 23.10      1       a
2 23.00      1       a
3 23.32      1       b
4 23.85      1       c
```
8.4.4. Importar una tabla con filas vacías

Ya hemos visto como solucionar problemas de importar tablas con valores faltantes. El mismo inconveniente con la misma solución es una fila completa vacía. Pero nos quedará una línea con todos valores NA. Importamos la tablaR173

```
> tablaR173
  peso minutos atributo
1 23.10      1       a
2 23.00      1       a
3    NA     NA
4 23.32      1       b
5 23.85      1       c
6    NA      1       c
```
En la tabla observamos que R coloca NA a aquellas celdas en que falta un valor numérico. Si el elemento de la celda es un caracter no coloca nada.

Si queremos eliminar la fila podemos utilizar varios recursos. Entre lo novedoso tenemos la función remove.all.na.rows(), que elimina la fila con todos valores NA y remove.na.rows() que remueve filas que contiene al menos un NA. Estas funciones requieren la biblioteca fts.

```
> remove.all.na.rows(tablaR173)
  peso minutos atributo
1 23.10      1       a
2 23.00      1       a
3    NA     NA
4 23.32      1       b
5 23.85      1       c
6    NA      1       c
```
vemos que no eliminó ninguna fila, ya que no hay filas con todos NA.
En cambio
```
> remove.na.rows(tablaR173)
  peso minutos atributo
```

```
1 23.10      1       a
2 23.00      1       a
4 23.32      1       b
5 23.85      1       c
```
eliminó las filas 3 y 6 que tenían al menos un NA.

8.4.5. Remplazar valores NA en una tabla

Utilicemos la tablaR173 para ver el cambio de valores NA.

```
> tablaR173
  peso minutos atributo
1 23.10      1       a
2 23.00      1       a
3    NA     NA
4 23.32      1       b
5 23.85      1       c
6    NA      1       c
```
como ya hemos visto tiene varios valores NA. Veamos esto también en un summary()

```
> summary(tablaR173)
          peso       minutos  a   tributo
 Min.   : 23.00   Min.   :1        :1
 1st Qu.: 23.07   1st Qu.:1       a:2
 Median :23.21    Median :1       b:1
 Mean   : 23.32   Mean   :1       c:2
 3rd Qu.: 23.45   3rd Qu.:1
 Max.   : 23.85   Max.   :1
 NA's   :  2      NA's   :1
```

vemos que la columna atributo tiene un valor vacío sin NA. Como la columna atributo está expresada como factor, lo que se deduce por el análisis de summary. Nos muestra el número de celdas con cada factor.

Si deseamos reemplazar los NA por otra valor. Por ejemplo deseamos reemplazar los valores NA de la columna peso por 25 y los de la columna minuto por 0, procedemos de la siguiente manera, utilizando la función is.na(). Esta función arroja el valor TRUE si hay un valor NA. En el código siguiente evalua la columna peso, si hay un valor NA lo reemplaza por 25, si no es NA, lo deja inalterado.

```
> tablaR173$peso[is.na(tablaR173$peso)]<-25
> tablaR173
  peso minutos atributo
1 23.10      1       a
2 23.00      1       a
3 25.00     NA
4 23.32      1       b
5 23.85      1       c
6 25.00      1       c
```
lo mismo hacemos con el valor NA de la columna minuto

```
> tablaR173$minutos[is.na(tablaR173$minutos)]<-0
> tablaR173
```

```
  peso minutos atributo
1 23.10       1       a
2 23.00       1       a
3 25.00       0
4 23.32       1       b
5 23.85       1       c
6 25.00       1       c
```
si intentamos cambiar el valor vacío de la columna atributo por el valor d.
```
> tablaR173$atributo[is.na(tablaR173$atributo)]<-"d"
Warning message:
In `[<-.factor`(`*tmp*`, is.na(tablaR173$atributo), value = c(2L,  :
  invalid factor level, NA generated
```
como podemos ver no lo ha reemplazado
```
> tablaR173
  peso minutos atributo
1 23.10       1       a
2 23.00       1       a
3 25.00       0
4 23.32       1       b
5 23.85       1       c
6 25.00       1       c
```
La solución es cambiar la columna de factor a caracter con la función as.character()
```
> tablaR173$atributo<-as.character(tablaR173$atributo)
```
la tabla no ha cambiado y en el lugar vacío existe en realidad un caracter vacío, que podemos representar como "". Entonces podemos hacer un reemplazo con el siguiente código, suponiendo que donde está el valor vacío deseamos incorporar el valor d.
```
> tablaR173$atributo[tablaR173$atributo==""]<-"d"
```
vemos entonces el contenido de la tablaR173
```
> tablaR173
  peso minutos atributo
1 23.10       1       a
2 23.00       1       a
3 25.00       0       d
4 23.32       1       b
5 23.85       1       c
6 25.00       1       c
```
vemos que el reemplazo se ha logrado con éxito.

8.5. Funciones con strings

Un string es un elemento formado por caracteres. Por ejemplo: casa, proteína, pero también puede ser un número si éste fue introducido como caracter. R proporciona funciones que permiten trabajar con strings, sin embargo para poder aplicarlas es importante que la columna del data.frame se halle en formato character.

8.5.1. Utilización de comillas

Supongamos que necesitamos introducir una variable que lleva comillas. Como las comillas son utilizadas habitualmente para ciertas funciones como por ejemplos dec=",", el ingreso de comillas en lugares inadecuados dará origen a errores. Para hacerlo evidente pensemos que

debemos introducir en una variable a el valor **proteina"1"**

```
> a<-"proteina"1""
Error: unexpected numeric constant in "a<-"proteina"1"
```

la solución es separar el valor a introducir proteina"1" en: proteina - "- 1 -" y utilizar la función paste() que empalmará los caracteres. Si se desea introducir una comilla dentro de un string este debe encerrarse entre '.

```
> a<-paste("proteina","","1","",sep=" ")
> a
[1] "proteina \" 1 \""
```

Se desaconseja utilizar variables string que tengan comillas (") o apóstrofes ('). Si las utiliza solo ganará en errores y para nada en claridad y velocidad.

8.5.2. Números de caracteres de un string.

En algunas situaciones deseamos conocer el número de caracteres de un string para ellos utilizamos la función nchar().

Introducimos a nuestro espacio de trabajo la tablaR174

```
> tablaR174<-read.table("clipboard",header=TRUE,dec=",",sep="\t",encoding="latin1")
```

vemos el contenido de la tabla

```
> tablaR174
              nombre Nacionalidad
1         juán pérez   Argentino
2       Pedro Cáseres   Argentino
3        Joaquin Cruz    Brasileño
4    Santiago Bernabeu  Paraguayo
5 María Sanchez Hidalgo  Paraguayo
```

y pedimos un resumen de la misma con la función summary()

```
> summary(tablaR174)
              nombre              Nacionalidad
 Joaquin Cruz             :1   Argentino:2
 juán pérez               :1   Brasileño:1
 María Sanchez Hidalgo:1   Paraguayo:2
 Pedro Cáseres            :1
 Santiago Bernabeu        :1
```

Por la salida producida por summary(), vemos que tanto la columna nombre como Nacionalidad han sido considerada como factores. Si deseamos conocer el número de caracteres de algunos de los elementos de alguna columna, en primer lugar debemos transformar el dato en character, especialmente si este es una columna de un data.frame. La función nchar() no funciona con factores sino con caracteres. Pasamos a caracter la columna nombre

```
 > tablaR174$nombre<-as.character(tablaR174$nombre)
```

pedimos un summary()

```
> summary(tablaR174)
   nombre            Nacionalidad
 Length:5           Argentino:2
 Class :character   Brasileño:1
 Mode  :character   Paraguayo:2
```

Comprobada que la columna nombre está compuesta por elementos del formato character o

string, podemos ver el número de caracteres que forma cada elemento.

```
> nchar(tablaR174[1,1])
[1] 10
```

El número de caracteres de "juán pérez" es 10.

Si pedimos el número de caracteres de la fila 1, sin discriminar columna, nos dará el valor de ambas columnas

```
> nchar(tablaR174[1,])
    nombre Nacionalidad
        10           1
```

Sin embargo como la columna nacionalidad está como factor, nos indica 1 solo caracter, lo cual no es cierto. Si quisiéramos el número de caracteres de la columna nacionalidad, deberíamos pasarla a factor.

Si queremos el número de caracteres de cada valor de la columna 1

```
> nchar(tablaR174[,1])
[1] 10 13 12 17 21
```

Esta función es útil en unidades más avanzadas de este curso.

8.5.3. substr

A veces es necesario seleccionar una parte de un string, para ellos tenemos la función substr()

Por ejemplo si queremos seleccionar los tres primeros caracteres del elemento de la fila 1 y columna 1, es decir del elemento "juán pérez".

```
> substr(tablaR174[1,1],1,3)
[1] "juá"
```

tablaR174[1,1], indica que la función substr() será aplicada al elemento de la primer fila y primer columna. El valor 1 que se halla a continuación, indica desde donde comienza la selección. En este caso es el primer caracter del string que se halla en posición [1,1]. El segundo número, hasta que caracter seleccionamos, en este caso hasta el tercer caracter. Siempre el primer número tiene que ser menor que el segundo. En el caso que invirtamos los número no nos seleccionará ningún caracter.

y si la selección solo quisiera la primer letra

```
> substr(tablaR174[1,1],1,1)
[1] "j"
```

8.5.4. Cambiar caracteres de minúscula a mayúscula

En algunas situaciones es útil ingresar caracteres pero deseamos que siempre queden en mayúscula independientemente como utilicemos el teclado. La función toupper() logra este propósito. Pasemos a mayúscula los nombres de la columna nombre de la tablaR174 y reasignemos dichos cambios a la columna del data.frame.

```
> tablaR174$nombre<-toupper(tablaR174$nombre)
```

veamos el cambio

```
> tablaR174
              nombre                Nacionalidad
1          JUÁN PÉREZ               Argentino
2        PEDRO CÁSERES              Argentino
3         JOAQUIN CRUZ              Brasileño
4     SANTIAGO BERNABEU             Paraguayo
5 MARÍA SANCHEZ HIDALGO             Paraguayo
```

8.5.5. Cambiar a letra minúscula

Si deseamos convertir datos de mayúscula a minúscula, La función tolower() logra este objetivo. Reconvirtamos a minúscula los valores de la primer columna de la tablaR174, recién modificados

```
> tablaR174$nombre<-tolower(tablaR174$nombre)
> tablaR174
                nombre Nacionalidad
1          juán pérez    Argentino
2        pedro cáseres    Argentino
3         joaquin cruz     Brasileño
4     santiago bernabeu     Paraguayo
5 maría sanchez hidalgo     Paraguayo
```

8.5.6. Cortar cadenas de caracteres

La función strtrim() permite cortar un string en una dada posición. Supongamos que deseamos truncar los nombres de la nacionalidades en el cuarto caracter, para ello utilizamos la función strtrim() con el siguiente código, recordando que las funciones siempre actúan sobre caracteres, por ello en el ejemplo anidamos strtrim() con as.character()

```
> tablaR174$Nacionalidad<-strtrim(as.character(tablaR174$Nacionalidad),4)
```

vemos el cambio realizado en la columna Nacionalidad. Todos los elementos quedaron de 4 caracteres.

```
> tablaR174
                nombre Nacionalidad
1          juán pérez         Arge
2        pedro cáseres         Arge
3         joaquin cruz         Bras
4     santiago bernabeu         Para
5 maría sanchez hidalgo         Para
```

8.5.7. gsub

Si deseamos reemplazar una parte de un caracter utilizamos la función gsub().
Sigamos el trabajo con la tablaR174 que hemos modificado.

```
> tablaR174
                nombre Nacionalidad
1          juán pérez         Arge
2        pedro cáseres         Arge
3         joaquin cruz         Bras
4     santiago bernabeu         Para
5 maría sanchez hidalgo         Para
```

Deseamos reemplazar la A de la columna Nacionalidad por a (minúscula)

```
> tablaR174$Nacionalidad<-gsub("A","a",as.character(tablaR174$Nacionalidad))
```

vemos el resultado logrado

```
> tablaR174
   nombre Nacionalidad
1 juán pérez         arge
2 pedro cáseres         arge
3 joaquin cruz         Bras
```

```
4  santiago bernabeu          Para
5 maría sanchez hidalgo       Para
```
Si quisieramos eliminar el espacio entre nombre y apellido de los elementos de la columna nombre utilizamos el siguiente código
```
> tablaR174$nombre<-gsub(" ","",as.character(tablaR174$nombre))
```
vemos el resultado
```
> tablaR174
    nombre                Nacionalidad
1  juánpérez                  arge
2  PedroCáseres               arge
3  JoaquinCruz                Bras
4  SantiagoBernabeu           Para
5 MaríaSanchezHidalgo         Para
```
8.6. Matrices

Las matrices son un tipo de objetos que tienen todos elementos de un mismo tipo. Por ejemplo una matriz de números enteros, etc. Son de mucha utilidad a la hora de realizar gráficos en tres dimensiones y en algunos cálculos que se requiere gran velocidad de procesamiento, como en el caso de manejo de imágenes en formato de texto.

8.6.1. Crear una matriz

Con la función matrix() creamos una matriz vacía con el número de filas y columna deseada. Veamos un ejemplo con 3 filas y tres columnas
```
> matriz1<-matrix(data = NA, nrow = 3, ncol = 3, byrow = FALSE)
```
la función byrow=FALSE, indica que cuando vayamos ingresando valores se irán colocando siguiendo las columnas. Primero se llenará la columna 1 y luego se seguirá por la 2 y luego la 3. Veamos como quedó constituida la matriz
```
> matriz1
     [,1] [,2] [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
[3,]   NA   NA   NA
```

La matriz tiene nueve elementos, en este caso vacíos. A continuación creamos un vector con 9 elementos que luego utilizaremos para llenar la matriz2 que también tendrá 3 fijas y tres columnas y que llenaremos siguiendo las columnas
```
> a<-c(1,2,3,4,5,6,7,8,9)
> matriz2<-matrix(data = a, nrow = 3, ncol = 3, byrow = FALSE)
```
vemos la matriz2 y comprobamos como han ingresado los valores del vector a.
```
> matriz2
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```
nrow: numero de filas
ncol: numero de columnas
byrow= FALSE o TRUE. Si FALSE, la matriz se llena por columna

8.6.2. Reemplazo de valores en una matriz

Si deseamos hacer el reemplazo de un valor de una matriz, por ejemplo el de fila 1, columna 2 o ingresar un valor en una matriz vacía, por ejemplo en la matriz1, recientemente creada, procederemos de manera similar a un data.frame. En primer lugar veamos matriz1

```
> matriz1
      [,1]  [,2] [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
[3,]   NA   NA   NA
```

ingresamos ahora el valor 10 en fila 1 y columna 2 con el siguiente código

```
> matriz1[1,2]<-10
```

comprobamos el ingreso del dato

```
> matriz1
    [,1] [,2] [,3]
[1,]   NA   10   NA
[2,]   NA   NA   NA
[3,]   NA   NA   NA
```

también se puede hacer con la función edit() que nos abrirá una interfaz más amigable, segura y rápida, si es correctamente utilizada.

```
matriz1<-edit(matriz1)
```

8.6.3. Introducir una matriz desde planillas de cálculo

Para introducir una matriz en nuestro espacio de trabajo se utiliza la función read.table() que ya hemos utilizado para introducir los data.frame. Utilizamos la función as.matrix() para forzar el ingreso del objeto como matriz

```
> tablaR175<-as.matrix(read.table("clipboard",dec=","))
> tablaR175
     V1   V2    V3
[1,] 0.92 1.25  3.10
[2,] 0.97 1.10  2.10
[3,] 0.87 0.89 82.00
[4,] 0.89 0.85 25.00
[5,] 0.87 0.85  0.82
[6,] 0.89 1.00 21.00
```

Como habrá notado no lleva el argumento header. Esto se debe a que las columnas y filas no tienen nombres especiales. Todos son elementos del mismo tipo con un orden en particular de acuerdo a la aplicación.

Ingresemos también la matriz tablaR176

```
> tablaR176<-as.matrix(read.table("clipboard",dec=","))
> tablaR176
    V1 V2 V3
[1,]  1 10  1
[2,]  1 11  1
[3,]  2 15  1
[4,]  3  2  2
[5,]  4  8  4
```

[6,] 4 4 1
comprobemos si estos elementos son matrices.
> is.matrix(tablaR175)
[1] TRUE
> is.matrix(tablaR176)
[1] TRUE

8.6.4. operaciones con matrices

Para que se puedan hacer operaciones con matrices tiene que existir cierta relación entre las filas y columnas de ellas. No cualquier operación es posible. Las matrices tablaR175 y tablaR176 tienen igual número de filas y columnas, lo que permitirá prácticamente cualquier operación

8.6.4.1. Producto de matrices.

tomemos las matrices
> tablaR175
V1 V2 V3
[1,] 0.92 1.25 3.10
[2,] 0.97 1.10 2.10
[3,] 0.87 0.89 82.00
[4,] 0.89 0.85 25.00
[5,] 0.87 0.85 0.82
[6,] 0.89 1.00 21.00
> tablaR176
V1 V2 V3
[1,] 1 10 1
[2,] 1 11 1
[3,] 2 15 1
[4,] 3 2 2
[5,] 4 8 4
[6,] 4 4 1
y realizamos el producto de ellas
> producto<-tablaR175*tablaR176
vemos el resultado
> producto
V1 V2 V3
[1,] 0.92 12.50 3.10
[2,] 0.97 12.10 2.10
[3,] 1.74 13.35 82.00
[4,] 2.67 1.70 50.00
[5,] 3.48 6.80 3.28
[6,] 3.56 4.00 21.00
como se puede ver cada elemento de la matriz producto es la multiplicación de los elementos respectivos de las matrices que intervinieron en el producto.
Si multiplicamos una matriz por un número, esto equivale a multiplicar por dicho número a cada elemento de la matriz.
> producto<-tablaR175*10
> producto

```
     V1   V2    V3
[1,] 9.2 12.5  31.0
[2,] 9.7 11.0  21.0
[3,] 8.7  8.9 820.0
[4,] 8.9  8.5 250.0
[5,] 8.7  8.5   8.2
[6,] 8.9 10.0 210.0
```

8.6.5. Transformar matriz en data.frame y viceversa

8.6.5.1. convertir data.frame en matriz

En algunas circunstancias podemos tener un data.frame que deseamos convertir a matriz. Por ejemplo, introduzcamos nuevamente la tablaR175, como un data.frame.

```
> tablaR175<-read.table("clipboard",dec=",",sep="\t")
```

Como no utilizamos as.matrix(), R introduce por elección el objeto como data.frame.
comprobamos que lo ingresamos como data.frame

```
> is.data.frame(tablaR175)
[1] TRUE
```

veamos ahora el contenido de tablaR175

```
> tablaR175
   V1   V2    V3
1 0.92 1.25  3.10
2 0.97 1.10  2.10
3 0.87 0.89 82.00
4 0.89 0.85 25.00
5 0.87 0.85  0.82
6 0.89 1.00 21.00
```

Si deseamos transformar tablaR175 en matriz, utilizamos la función as.matrix()

```
> tablaR175<-as.matrix(tablaR175)
```

comprobamos si sigue siendo data.frame

```
> is.data.frame(tablaR175)
[1] FALSE
```

que comprobamos que no es así. Entonces comprobamos si es matriz con is.matrix()

```
> is.matrix(tablaR175)
[1] TRUE
```

lo cual es correcto.

8.6.5.2. Transformar matriz en data.frame

Tomemos la matriz tablaR175 y comprobemos su formato

```
is.matrix(tablaR175)
[1] TRUE
```

Vemos que es una matriz, pero podemos necesitarla en formato data.frame. Para convertirla en data.frame utilizamos la función as.data.frame()

```
> tablaR175<-as.data.frame(tablaR175)
> is.matrix(tablaR175)
[1] FALSE
```

comprobamos que dejó de ser matriz

```
> is.data.frame(tablaR175)
[1] TRUE
```

y ahora es un data.frame

8.6.6. Crear matriz a partir de vectores

Se pueden crear matrices a partir de vectores. Estos vectores tienen que tener cierta relación en el número de elementos. En este caso trabajaremos con dos vectores de igual longitud. Las funciones que se utilizan son cbind y rbind como para formar data.frames.

8.6.6.1. Con cbind

creamos dos vectores

```
> a<-c(1,2,3,4,5,6,7,8,9)
> b<-c(2,2,2,2,2,2,2,2,2)
```

creamos ahora una matriz que llamamos matrizcbind

```
> matrizcbind<-as.matrix(cbind(a,b))
```

Con cbind unimos a y b como si fueran columnas y con as.matrix, obligamos a formar un objeto del tipo matrix. Vemos la matriz y comprobamos lo comentado en recientemente

```
> matrizcbind
     a b
 [1,] 1 2
 [2,] 2 2
 [3,] 3 2
 [4,] 4 2
 [5,] 5 2
 [6,] 6 2
 [7,] 7 2
 [8,] 8 2
 [9,] 9 2
> is.matrix(matrizcbind)
[1] TRUE
```

con lo cual comprobamos que el objeto es una matriz

8.6.6.2. con rbind

crearemos una matriz con rbind(), es decir uniendo vectores como si fueran filas

```
> matrizrbind<-as.matrix(rbind(a,b))
```

comprobamos que los vectores a y b pasaron a la matriz formando parte de sus filas.

```
> matrizrbind
  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
a    1    2    3    4    5    6    7    8    9
b    2    2    2    2    2    2    2    2    2
```

verificamos a continuación si realmente es una matriz

```
> is.matrix(matrizrbind)
[1] TRUE
```

9. Clase 8

Vídeo: https://youtu.be/g2ExVhCQQZs

Tabla de Datos: http://hdl.handle.net/2133/9456

Estadísticas de filas y columnas de data.frames y matrices

En clases anteriores hemos utilizados estadísticas de filas y columnas. Revisemos estos conceptos para luego introducir otros.

Ingresamos al espacio de trabajo la tablaR181 de la planilla de cálculo tablaR1-8.ods/xls que usted dispone para esta clase.

```
> tablaR181<-read.table("clipboard",header=TRUE,dec=",",sep="\t")
```

```
> tablaR181
   peso edad sexo
1    53   25    f
2    58   26    f
3    59   24    m
4    96   59    m
5    78   54    m
6    98   56    m
7    58   58    f
8    59   58    f
9    85   59    f
10   58   53    m
11   84   65    f
12   85   56    m
13   59   54    f
14   85   45    m
15   48   45    m
16   58   48    m
```

pedimos como es adecuado para asegurarnos de la importación de los datos, un summary()

```
> summary(tablaR181)
      peso            edad           sexo
 Min.   : 48.00   Min.   :24.00   f:7
 1st Qu.: 58.00   1st Qu.:45.00   m:9
 Median :59.00   Median :54.00
 Mean   : 70.06   Mean   :49.06
 3rd Qu.: 85.00   3rd Qu.:58.00
 Max.   : 98.00   Max.   :65.00
```

Es lo que esperamos de la importación! A continuación haremos uso de algunas funciones y comprobaciones.

Haremos antes de iniciar la clase de hoy un breve repaso de funciones aprendidas. A continuación usted recibe en cada ítem la indicación de hacer algo y deberá elegir la función adecuada. Si no la recuerda hallará las mismas al final de este documento.

[i]- compruebe que tablaR181 es un data.frame.

[ii]- compruebe que el data.frame tablaR181 tiene 16 filas.

[iii]- compruebe que tiene tres columnas.

[iv]-compruebe que la columna 3 del data.frame tablaR181 tiene dos niveles del factor en estudio.

[v]- obtenga la media de la columna 1 del data.frame tablaR181.

[vi]- obtenga la mediana de la columna 2 del data.frame tablaR181.

[vii]- obtenga el percentilo del 95% de la edad.

[viii]- obtenga el rango de los pesos.

[ix]- obtenga el desvío estándar de las edades.

[x]- obtenga la variancia de los pesos.

[xi]- obtenga la edad del individuo más liviano (de menor peso).

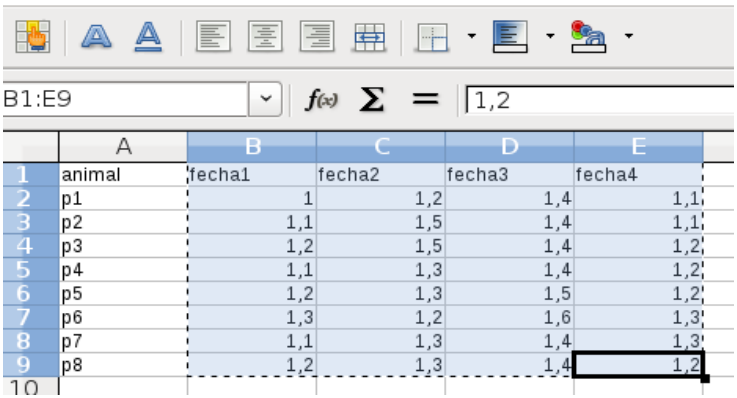
[xii]- halle la edad del individuo de mayor edad.

Veremos ahora otras funciones, de mayor aplicación en casos que se trabajen con matrices numéricas o data.frames solo con datos numéricos.

Veamos el ejemplo de la tablaR182. La misma corresponde a valores de glucosa en sangre de una serie de animales indicado su código en la primer columna: p1-p8. Como puede observarse en la tabla son todos valores numéricos, de la misma variable, aunque separados por animal (filas) y fechas (columnas). Cada valor pertenece a un animal y a una fecha de medición en particular.

En casos como este puede ser necesario conocer estadísticas de columnas y filas.

Será conveniente entonces introducir los datos de manera que solo nos queden en la tabla valores numéricos de la variable en estudio. Para ello hacemos la siguiente selección



	A	B	C	D	E
1	animal	fecha1	fecha2	fecha3	fecha4
2	p1	1	1,2	1,4	1,1
3	p2	1,1	1,5	1,4	1,1
4	p3	1,2	1,5	1,4	1,2
5	p4	1,1	1,3	1,4	1,2
6	p5	1,2	1,3	1,5	1,2
7	p6	1,3	1,2	1,6	1,3
8	p7	1,1	1,3	1,4	1,3
9	p8	1,2	1,3	1,4	1,2
10					

y para introducirla en el espacio de trabajo utilizamos el siguiente comando

> tablaR182<-

read.table("clipboard",header=TRUE,row.names=c("p1","p2","p3","p4","p5","p6","p7","p8"),dec=",",sep="\t")

con el argumento row.names, colocamos como nombre de la fila los códigos de cada animal, de manera que no se crea una columna con los códigos de cada animal. De esta manera tenemos todas filas y columnas con valores numéricos.

Vemos como quedó

```
> tablaR182
   fecha1 fecha2 fecha3 fecha4
p1    1.0    1.2    1.4    1.1
p2    1.1    1.5    1.4    1.1
p3    1.2    1.5    1.4    1.2
p4    1.1    1.3    1.4    1.2
p5    1.2    1.3    1.5    1.2
p6    1.3    1.2    1.6    1.3
p7    1.1    1.3    1.4    1.3
p8    1.2    1.3    1.4    1.2
```

Si bien podemos utilizar algunas funciones ya conocidas, la biblioteca "fBasics" amplia bastante las posibilidades. Puede descargarla desde cualquiera de los repositorios como hemos aprendido en clases anteriores. Esta biblioteca provee algunas funciones interesantes para estos casos. Nos brinda estadísticas de filas y columnas. Veamos su aplicación a columnas.

9.1.1. colStats()

nos permite calcular diferentes funciones de todas las columnas. Para calcular la media aplicamos el siguiente código

```
> colStats(tablaR182,mean)
```

nos da la media de cada columna

```
fecha1   fecha2    fecha3    fecha4
1.1500   1.3250    1.4375    1.2000
```

Si deseamos el desvío estándar el comando a escribir será

```
> colStats(tablaR182,sd)
```

nos permite calcular el desvío estándar de las columnas

```
   fecha1      fecha2          fecha3          fecha4
0.09258201  0.11649647  0.07440238  0.07559289
```

Para la variancia el procedimiento es similar

```
> colStats(tablaR182,var)
```

para calcular la variancias por columnas

```
    fecha1       fecha2          fecha3          fecha4
0.008571429 0.013571429 0.005535714 0.005714286
```

lo mismo que para hallar el mínimo de cada columna

> colStats(tablaR182,min)

buscar el mínimo valor dentro de cada columna

fecha1 fecha2 fecha3 fecha4

1.0 1.2 1.4 1.1

o el máximo de cada columna

> colStats(tablaR182,max)

busca el máximo valor de cada columna

fecha1 fecha2 fecha3 fecha4

fecha1 fecha2 fecha3 fecha4

1.3 1.5 1.6 1.3

1.4

también podemos hallar el rango de valores (mínimo y máximo)

> colStats(tablaR182,range)

busca el mínimo y el máximo valor de cada columna, es decir el rango de valores

fecha1 fecha2 fecha3 fecha4

[1,] 1.0 1.2 1.4 1.1

[2,] 1.3 1.5 1.6 1.3

para la mediana

> colStats(tablaR182,median)

calcula la mediana de cada columna

fecha1 fecha2 fecha3 fecha4

1.151.30 1.40 1.20

1.16

Veamos ahora estadísticas de filas. Para ello utilizamos la función rowStats() de la biblioteca fBasics

9.1.2. rowStats()

Para estadísticas de filas tenemos opciones similares.

Para la media

```
> rowStats(tablaR182,mean)
   p1    p2    p3    p4    p5    p6    p7    p8
1.175 1.275 1.325 1.250 1.300 1.350 1.275 1.275
```

para el desvío estándar

```
> rowStats(tablaR182,sd)
        p1          p2          p3          p4          p5          p6          p7          p8
0.170782 0.206155 0.150000 0.129099 0.141421 0.173205 0.125830  0.0957427
```
la variancia

```
> rowStats(tablaR182,var)
          p1          p2          p3          p4          p5          p6          p7          p8
0.0291666 0.0425000 0.0225000 0.0166666 0.02000000 0.03000000 0.01583333 0.00916667
```
la mediana

```
> rowStats(tablaR182,median)
          p1       p2     p3     p4   p5   p6    p7    p8
           1.151.25 1.30 1.25 1.25 1.30 1.30 1.25
```
y el rango

```
> rowStats(tablaR182,range)
      p1  p2  p3  p4  p5  p6  p7  p8
[1,] 1.0 1.1 1.2 1.1 1.2 1.2 1.1 1.2
[2,] 1.4 1.5 1.5 1.4 1.5 1.6 1.4 1.4
```

o algunos percentilos

```
> rowStats(tablaR182,quantile)
       p1      p2    p3      p4      p5    p6    p7      p8
0%   1.000 1.100 1.200 1.100 1.20 1.200 1.100 1.200
25%  1.075 1.100 1.200 1.175 1.20 1.275 1.250 1.200
50%  1.150 1.250 1.300 1.250 1.25 1.300 1.300 1.250
75%  1.250 1.425 1.425 1.325 1.35 1.375 1.325 1.325
100% 1.400 1.500 1.500 1.400 1.50 1.600 1.400 1.400
```

9.1.3. Generación de objetos con secuencias numéricas

Muchas veces tenemos que generar vectores, columnas o filas de data.frames, con secuencias establecidas de números. Por ejemplo, hemos realizado la obtención de 50 muestras cada 2 minutos, por lo tanto nuestros tiempos serán 0, 2, 4, ...etc.
Veremos a continuación algunos recursos al respecto

La función seq()

Esta función permite generar secuencias numéricas. Tiene diversas formas de expresarla, las cuales se muestran con ejemplos

```
> seq(1,5,2)
[1] 1 3 5
```

En el caso anterior, el primer número es desde donde parte la secuencia, el segundo número hasta donde llega y el tercero el escalón o incremento entre dos números consecutivos.
Veamos otros ejemplos similares

```
> seq(1,7,2)
[1] 1 3 5 7
> seq(1,7,3)
[1] 1 4 7
```

También podemos generar secuencias decrecientes.

```
> seq(10,1,-2)
[1] 10  8  6  4  2
```

Con la misma función podemos utilizar otro formato, en el cual indicamos con length: la cantidad de números, from: desde donde partimos y to: hasta donde llegamos. En este caso el escalón lo elige la función, ya que no se lo hemos pedido expresamente.

```
> seq(length=5,from=0, to=1)
[1] 0.00 0.25 0.50 0.75 1.00
> seq(length=23,from=0, to=1)
 [1] 0.00000000 0.04545455 0.09090909 0.13636364 0.18181818 0.22727273
 [7] 0.27272727 0.31818182 0.36363636 0.40909091 0.45454545 0.50000000
[13] 0.54545455 0.59090909 0.63636364 0.68181818 0.72727273 0.77272727
[19] 0.81818182 0.86363636 0.90909091 0.95454545 1.00000000
```

En el siguiente caso le indicamos la cantidad de números a generar con length. From: indica desde donde partimos y by: el escalón que será positivo si generaremos números crecientes o negativo si los números a generar son decrecientes. Obviamente en este caso, la función automáticamente elige donde finalizar la secuencia.

```
> seq(length=5,from=100, by=-7)
[1] 100  93  86  79  72
```

Existen incompatibilidad entre algunos parámetros, los que darán error. Por ejemplo, en el caso siguiente es una secuencias decreciente, pero el escalón es positivo.

```
> seq(from=100, to=70,by=10)
Error en seq.default(from = 100, to = 70, by = 10) :
  wrong sign in 'by' argument
```

para ser correcta debería ser

```
> seq(from=100, to=70,by=-10)
[1] 100  90  80  70
```

se puede asignar una secuencia a un objeto, en este caso lo llamamos s4

```
> s4<-seq(1,10,by=3)
```

cuando pidamos s4, obtendremos la secuencia

```
> s4
[1]  1  4  7 10
```

la forma más fácil de generar una secuencia numérica, por ejemplo de -10 hasta 10 en saltos de 0,1

```
a<-seq(-10,10,0.1)
```

En este caso no indicamos nada, entonces el primer número es de donde partimos, el segundo hasta donde llegamos y el tercero el escalón. Pero, podemos escribirla sin escalón

```
a<-seq(1,100)
```

Si no se aclara el tercer número, es decir no indicamos el escalón, la función asigna automáticamente al escalón el valor 1.

La función rep()

Esta función permite permite repetir un objeto en particular, no necesariamente numérico.
En el caso siguiente repetiremos 3 veces el objeto s4, creado anteriormente

```
> rep(s4,times=3)
 [1]  1  4  7 10  1  4  7 10  1  4  7 10
```

En este caso la misma función produce repeticiones de un objeto una dada cantidad de veces, pero agrupando sus elementos.

```
> rep(s4,each=3)
 [1]  1  1  1  4  4  4  7  7  7 10 10 10
```

9.1.3.1. Generación secuencias con distribuciones aleatorias uniformes y normales

Ya vimos la función seq() que genera secuencias de elementos de un objeto de R. Sin embargo en algunas oportunidades se requieren datos generados con cierta aleatoriedad o regularidad. A continuación desarrollaremos algunas funciones útiles

La función runif()

Esta función genera secuencias aleatorias con distribución uniforme, es decir sin tener una moda. Vemos por ejemplo

```
> runif(5,0,5)
[1] 3.751868 3.256647 3.356387 3.692737 2.871605
```

El primer número de la función runif(), en este caso 5, indica que se generarán 5 números aleatorios. El segundo y el tercero: 0 y 5, indican los límites entre los cuales estarán los números.

Es una buena función para realizar sorteos, por ejemplo para asignar unidades experimentales aleatoriamente a cada grupo experimental.

Se puede obviamente combinar con otras funciones. Por ejemplo si deseáramos generar 200 números enteros entre 0 y 1000, podríamos anidar la función runif() con la función round()

```
> round(runif(200,0,1000),digits=0)
  [1] 847 873 940 805 879 177 219 466 894 879 831 410 289 446 890 448  63 848
 [19] 234 661 893 466 707 571 987 485 991 766 431 472 975 543 950 324 127 115
 [37] 440 561 200 494 869 559 211 232 772 740 935 399 437 463  48 718 421 435
 [55] 118 580 489 581 314  56 360 167  64 957 755 578 189 628 253 228  92 485
 [73] 358 334 956  68 694 989 459 129 626 689 700 912 900 408 387 838 610 757
 [91] 312 571 264 585 394 494 784 509  55 769 300  85  71 152 883 236 920 814
[109] 925 402 731 999 957 164 158 868 404  71 753 927 831 207 503 824 611 938
[127] 198   5  61 544  59 205 786 818 117 404  27 877 487  69 446 109 173 643
[145] 566  62 221  58 294 161 284 508 273 167 210 148 945 978 852 580 541 599
[163] 745 603 888 593 140 399 617 475 202 171 122 137 113 901 361 801 380 646
[181]  86 724 685 376 818 929  63 993 587 159 884 354 623 642 637 895 714 229
[199] 196 766
```

La función rnorm()

Esta función genera números aleatorios pero los mismo tienen distribución normal.

Para mejor comprensión compararemos los datos obtenidos con seq(), runif() y rnorm().

Generamos una secuencia numérica que asignamos a un objeto A. Esta secuencia tendrá números comprendidos entre 1 y 100 con saltos de a una unidad.

```
> A<-seq(1,100)
```

vemos el objeto A

```
> A
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
[19] 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
```

```
 [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
 [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
 [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
 [91]  91  92  93  94  95  96  97  98  99 100
```

ahora generamos una secuencias aleatoria con distribución uniforme de 100 números comprendidos entre 0 y 100, que asignamos al objeto B

```
> B<-round(runif(100,0,100),digits=0)
```

sus valores serán diferentes a los mostrados en este texto ya que es aleatoria la generación de los mismos

vemos el objeto B

```
> B
  [1] 33  4 35 64 49 10 86 62 30 39  1 13 78 47 36 46 23 43
 [19] 81 39 82 72 40 43 57 25 44 96  2 33 61 81 81 38 86 52
 [37] 90 18 30 86 99 82 77 41 79 99 70 61 75 33  9 36 24 89
 [55] 65  8 24  4 65 86 100 44 75 65 58 30  1 31  4 66  5 72
 [73]  4 74 74 46 74 51 62 72 53 92 99 77 52 92 46 34 92 93
 [91] 90  7 33 47 94 97 17  3 38 47
```

por último generamos una secuencia aleatoria de 100 números con distribución normal y la asignamos al objeto C

```
> C<- round(rnorm(100,50,25),digits=0)
```

El primer número dentro del paréntesis indica el número de datos a generar, el segundo la media de la distribución normal y el tercero (25 en este caso) el desvío estándar.

sus valores serán diferentes a los mostrados en este texto ya que es aleatoria la generación de los mismos

vemos este objeto

```
> C
  [1] 82 43 67 93 31 85 27 33 94 33 48 39 30 49 84 23 38 40
 [19] 57 59 72 53 65 12 59 65 63 94 84 14 37 55 17 26 44 41
 [37] 78 25 12 33 43 50 47 40 105 68 40 47 42 43  6 52 56 69
 [55] 41 63  6 35  2 35 12 51 28 43 75 26 25 24 55 32 59 27
 [73] 48 34 57 46 34 48 43 74 61 43 47 71  0 30 33 47 29 122
 [91] 46 57 37 88 27 75 24 77 66 88
```

Construimos ahora un data.frame con los datos de los objetos A, B y C

```
> ABC<-data.frame(A,B,C)
```

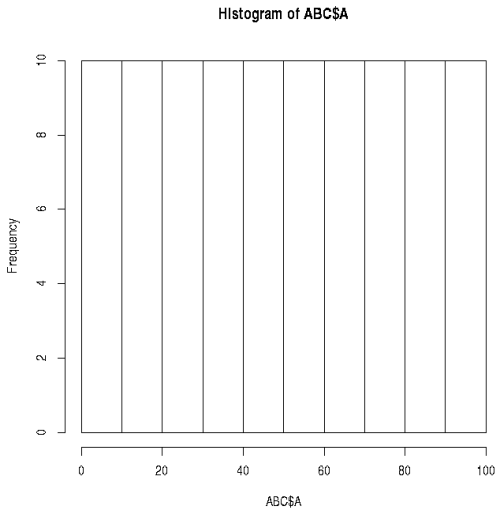
pedimos un head de ABC

```
> head(ABC)
  A  B  C
1 1 33 82
2 2  4 43
3 3 35 67
4 4 64 93
5 5 49 31
6 6 10 85
```

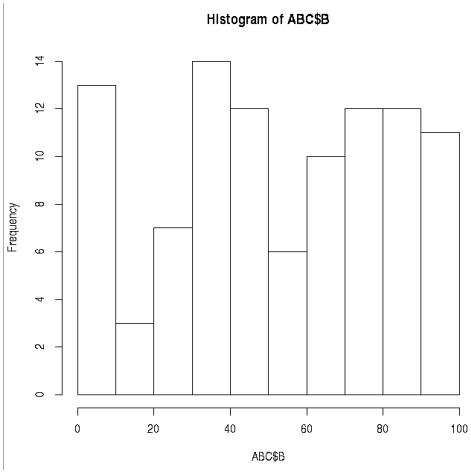
obviamente el objeto ABC tiene 100 filas, ya que cada elemento A, B y C tienen 100 filas.

Ahora graficaremos cada uno de estas columnas. primero veamos el histograma de la columna A que corresponde a número naturales correlativos de 1 a 100

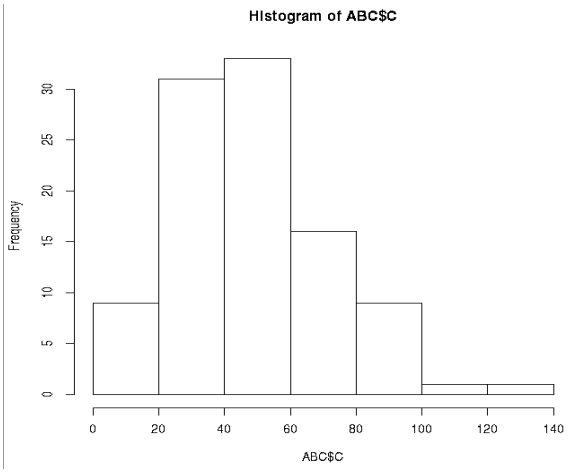
```
> hist(ABC$A)
```



nos indica que hay 10 número por cada intervalo de amplitud 10. Tal cual corresponde a lo que generamos.
Si realizamos el histograma de los datos del objeto B
hist(ABC$B)



Observamos que como consecuencia de la aleatoriedad, no existe el mismo número de datos en cada intervalo. Para el caso del elemento C
hist(ABC$C)



Podemos comprobar una concentración de los datos en el intervalo 40-60, que es donde se halla la media que tiene el valor 50

9.1.4. Definición de funciones de inicio

9.1.4.1. .First()

.First() es una función útil para iniciar un espacio de trabajo con orientación a funciones o bibliotecas especiales. Es decir en ella podemos cargar cierta información útil para un espacio de trabajo en especial. Supongamos que en un espacio de trabajo debemos trabajar diariamente con las funciones provistas por fBasics. Si bien no es mucho trabajo escribir library(fBasics), sería bueno ya tenerla cargada cuando se accede a dicho espacio de trabajo y no tener que hacerlo cada vez que iniciemos una sesión de trabajo.

Se puede poner en la función .First, esta biblioteca. El código siguiente nos indica como hacerlo.

```
.First<-function() {library(fBasics)}
```

cuando ejecute R en dicho espacio de trabajo , se cargará automáticamente la biblioteca fBasics (o la que desee)

Por supuesto se pueden colocar más de una biblioteca, como vemos en el siguiente ejemplo donde colocamos las bibliotecas pROC, fts y fBasics.

```
.First<-function(){
library(pROC)
library(fts)
library(fBasics)}
```

Si no deseara más el uso de .First(), puede eliminarla como cualquier objeto con la función rm()

```
rm(.First)
```

Cuando arranquemos nuevamente en este espacio de trabajo no se ejecutará más.

9.2. Listas

Las listas son objetos de utilidad en algunos casos especiales. Hasta ahora conocemos varios objetos que almacenan datos: vectores, data.frames, matrices y listas. Según el trabajo puede ser más o menos útil cada uno de ellos.
Veremos una aplicación y para ellos utilizaremos una tabla con gran cantidad de datos.
Introduzca en su espacio de trabajo la tablaR183
tablaR183<-read.table("clipboard",header=TRUE,dec=",",sep="\t")

Realizamos comprobaciones de rutina

```
> names(tablaR183)
[1] "codigo"       "nombre"       "clasificacion" "tipo"
[5] "ph"           "conduct"      "fosfato"
> head(tablaR183)
   codigo   nombre   clasificacion      tipo    ph    conduct fosfato
1  AR1A     fanta    otras              normal  3.62   0.37     6.76
2  AR1B     cocacola cola               normal  2.63   0.65    55.17
3  AR1C     h20h     otras              diet    3.63   0.59    20.85
4  AR2A     levite   aguasaborizada     normal 3.39    0.37    54.88
5  AR3B     awafrut  aguasaborizada     normal 3.04    0.58     0.70
6  AR3C     levite   aguasaborizada     normal 3.00    0.72     0.26
> summary(tablaR183)
    codigo       nombre        clasificacion        tipo           ph
 am1    : 1   cocacola:10   aguasaborizada:10   diet  :16    Min.   :2.180
 am2    : 1   h20h    : 5   cola          :11   normal:42    1st Qu.:2.855
 am3    : 1   levite  : 4   deportiva     : 6                Median :3.020
 am4    : 1   citric  : 3   jugoliquido   : 5                Mean   :3.049
 am5    : 1   clight  : 3   jugopolvo     :10                3rd Qu.:3.325
 am6    : 1   gatorade: 3   otras         :16                Max.   :3.790
 (Other):52   (Other) :30
   conduct          fosfato
 Min.   :0.3700   Min.   :  0.16
 1st Qu.:0.6800   1st Qu.: 31.97
 Median :0.8550   Median : 55.24
 Mean   :0.9167   Mean   :204.19
 3rd Qu.:1.0650   3rd Qu.:270.75
 Max.   :2.0550   Max.   :957.00
```

A modo de ejemplo de una forma de utilización de listas, analicemos la siguiente situación. Supongamos que quisiéramos calcular media, desvío estándar, mediana y rango, haciendo grupos y para diversas variables. En este caso calcularemos las estadísticas mencionadas para los diferentes niveles del factor clasificación, que vemos a continuación con la función levels()

```
> levels(tablaR183$clasificacion)
[1] "aguasaborizada" "cola"          "deportiva"      "jugoliquido"
[5] "jugopolvo"      "otras"
```

Por supuesto con un poco de ingenio y trabajo, ya estamos en condiciones de cumplir con el objetivo propuesto, al menos por 10 formas diferentes (quiere pensar cuales podrían ser?)
Acá la propondré un método bastante efectivo y claro con el uso de listas.
Primero hagamos un ejemplo sencillo.

Supongamos que deseáramos obtener las medias de pH de cada nivel dentro de clasificación. Para ello creamos una variable que llamaremos A, a la que le asignamos cada nivel del factor "clasificacion"
recordemos los niveles
> levels(tablaR183$clasificacion)
[1] "aguasaborizada" "cola" "deportiva" "jugoliquido"
[5] "jugopolvo" "otras"
comenzamos con el nivel aguasaborizadas
> A<-"aguasaborizada"
creamos una lista
> list(clasificacion=A,media=mean(tablaR183$ph[tablaR183$clasificacion==A]))
al ejecutar el comando anterior obtenemos como salida el nivel del factor clasificación
$clasificacion
[1] "aguasaborizada"
y el valor de su media
$media
[1] 3.19
Para calcular las medias del pH para los otros niveles solo tenemos que redefinir la variable A. Siguiendo con el nivel siguiente: "cola"
luego solo tenemos que redefinir A, para el otro nivel y así sucesivamente
> A<-"cola"
luego ejecutamos el mismo código, salvo que ahora para la función list() A ya no es más aguasaborizadas sino cola
> list(clasificacion=A,media=mean(tablaR183$ph[tablaR183$clasificacion==A]))
$clasificacion
[1] "cola"
$media
[1] 2.579545

Si deseamos hacer muchos cálculos a la vez, por ejemplo: media, desvío estándar, mediana, rango. Solo ampliamos la lista
>
list(clasificacion=A,media=mean(tablaR183$ph[tablaR183$clasificacion==A],na.rm=TRUE),desvioestandar=sd(tablaR183$ph[tablaR183$clasificacion==A],na.rm=TRUE),mediana=median(tablaR183$ph[tablaR183$clasificacion==A],na.rm=TRUE),rango=range(tablaR183$ph[tablaR183$clasificacion==A],na.rm=TRUE))
$clasificacion
[1] "cola"
$media
[1] 2.579545
$desvioestandar
[1] 0.235568
$mediana
[1] 2.605
$rango
[1] 2.18 3.02
Veremos más adelante que esto se puede articular con data.frames, scripts, etc.

10. Clase 9

Tabla de datos: http://hdl.handle.net/2133/9457

10.1. Ejercicio de aplicación de conceptos previos

A continuación se realizará una serie de ejercicios tendientes a revisar los comandos utilizados a lo largo de este módulo. En cada caso tendrá una instrucción a cumplir, trate de resolverla y en caso que no lo pueda hacer utilizando los contenidos de las clases 1-8, haga click en el número de la pregunta y hallará al final el código necesario para hacerlo. Puede ocurrir que halle otro camino diferente al propuesto, pero que igualmente sea efectivo.
Para su facilidad en esta clase el orden de las instrucciones se realizará indicando la clase tratada. En el examen correspondiente al módulo será sin dicha orientación

10.1.1. Clase 1

[xiii]- Busque la forma de citar a R en un trabajo científico.
[xiv]- compruebe las bibliotecas cargadas y disponibles en su espacio de trabajo.
[xv]- compruebe si hay objetos en su espacio de trabajo.
[xvi]- compruebe el directorio en que se halla su espacio de trabajo actual.
[xvii]- cree un vector llamado perros, donde los elementos sean caniche, dálmata, pastor inglés.
[xviii]- compruebe que el vector perros contiene los elementos mencionados en dicho orden.
[xix]- compruebe qué tipo de elementos constituyen su vector (numéricos, caracteres).
[xx]- El vector perros está ordenado en forma alfanumérica creciente, ordénelo en forma decreciente de manera que quede modificado de manera permanente el vector perros.
[xxi]- El contenido del vector perros asígnelo a otro objeto llamado misperros y elimine el vector perros.

10.1.2. Clase 2

[xxii]- Cree un data.frame llamado cervezas, que no contenga datos en el momento de la creación.
[xxiii]- Introduzca en su espacio de trabajo la TablaR1-9 que ha recibido con esta clase con diferentes extensiones (.ods, .xls, .csv), utilizando copy/paste (portapapeles).
[xxiv]- Introduzca en su espacio de trabajo la TablaR1-9 pero sin utilizar copy/paste, sino utilizando el archivo y asígnele a dicha tabla en su espacio de trabajo el nombre clase19.
[xxv]- Cree un data.frame llamado mascotas que tenga dos columnas con los nombres de los perros que tiene en su vector (misperros o perros). Una de las columnas debe tener los perros ordenados en forma creciente (nombre de la columna creciente) y la otra en forma decreciente (nombre de la columna: decreciente) .
[xxvi]- Ejecute un comando que le permita ver los nombres de cada columna y algunas de las primeras lineas del data.frame clase19.
[xxvii]- Verificar que el objeto clase19 es un data.frame y que no es un vector.

10.1.3. Clase 3

[xxviii]- En la columna codigo del data.frame tabla19 reemplace en aquellos que tienen como parte del mismo AR por MQ
[xxix]- Obtenga la mediana de la columna de la columna ph.
[xxx]- Ordene las filas del data.frame tabla19 según valores crecientes de acideztitulable.
[xxxi]- En un solo comando calcule la media y el desvío estándar de la columna ph del data.frame tabla19
[xxxii]- cree un data.frame tabla19bis a partir del data.frame tabla19, que solo contenga la columna código y el pH.

[xxxiii]- calcule cuantos valores tiene el data.frame tabla19 incluyendo aun aquellos que no tienen datos (NA).
[xxxiv]- El data.frame tabla19 tiene una columna llamada tipo que tiene datos de tipo factor. Identique cuales son los niveles del factor y cuantas lineas del data.frame tienen cada factor.
[xxxv]- Seleccione solo la columna codigo y ph de aquellas filas del data.frame tabla19 en que el ph es menor que 3
[xxxvi]- Seleccionar del data.frame tabla19 aquellas filas en que el ph es menor que 3 y la acideztitulable>50

10.1.4. Clase 4

[xxxvii]- infórmese solo de los nombres de las columnas del data.frame tabla19.
[xxxviii]- en un solo comando calcule las medias de los valores de ph para cada uno de los niveles de la columna tipo.
[xxxix]- transforme la variable ph (que es numérica) en factor, estratificando los valores en rangos. (por ejemplo de a 1 unidad de pH) y agregue al final del data.frame tabla19 una columna con dichos valores.
[xl]- Pídale a su sistema que le diga la fecha y hora.

10.1.5. Clase 5

[xli]- cargue la biblioteca fBasics o gdata.
[xlii]- escriba el código que le permita buscar los nombres de todas las funciones que en su nombre tienen la palabra "test".
[xliii]- Busque ayuda sobre las funciones de la biblioteca fBasics.

10.1.6. Clase 6

[xliv]- calcule la suma de los valores de acideztitulable del data.frame tabla19.
[xlv]- calcule el máximo y el mínimo de los valores de la columna ph del data.frame tabla19.
[xlvi]- Calcule los percentilos más comunes de la columna fosfato del data.frame tabla19.
[xlvii]- Exporte los datos del data.frame tabla19 con el nombre tabla19exp, en el formato csv, sin incluir los números de las filas pero si los nombres de las columnas.

10.1.7. Clase 7

[xlviii]- infórmese del directorio en que se halla su espacio de trabajo.
[xlix]- cambie de directorio.
[l]- infórmese del número de caracteres de los valores contenidos en cada fila de la columna clasificacion del data.frame tabla19.
[li]- Cree una columna al final del data.frame tabla19 que contenga las dos primeras letras de los valores de la columna clasificacion y llame a esa columna abreviaturaclasificacion.

10.1.8. Clase 8

[lii]- genere un vector que contenga 20 números aleatorios enteros entre 0 y 100 .
[liii]- genera una secuencia de números enteros crecientes de 0 a 20.
[liv]- obtenga una serie de 300 datos, que supuestamente provienen de una población normal con media=3 y desvío estándar= 0,1 y grafíquelo en un histograma.

```
i     > is.data.frame(tablaR181)
ii    > nrow(tablaR181)
iii   > ncol(tablaR181)
iv    > levels(tablaR181$sexo)
v     > mean(tablaR181$peso)
vi    > median(tablaR181$edad)
vii   > quantile(tablaR181$edad,probs=0.95)
viii  > range(tablaR181$peso)
ix    > sd(tablaR181$edad)
x     > var(tablaR181$peso)
xi    > tablaR181[tablaR181$peso==min(tablaR181$peso),2]
xii   > max(tablaR181$edad)
xiii  citation()
xiv   search()
xv    ls()
xvi   getwd()
xvii  perros<-c("caniche", "dálmata","pastor inglés")
xviii perros
xix
> str(perros)
chr [1:3] "caniche" "dálmata" "pastor inglés"
> mode(perros)
[1] "character"
> summary(perros)
   Length     Class      Mode
        3 character character
xx    perros<-sort(perros,decreasing=T)
xxi
> misperros<-perros
> rm(perros)
xxii  cervezas<-data.frame()
xxiii tabla19<-read.table("clipboard",header=TRUE,sep="\t",dec=",")
xxiv                clase19<-read.table("TablaR1-9.csv",header=TRUE,sep="\t",fileEncoding
      ="Unicode", dec=",")
xxv         mascotas<-data.frame(cbind(creciente=sort(misperros,decreasing=F),decreciente=
      sort(misperros,decreasing=T)))
xxvihead(tabla19,3) o > tabla19[c(1:3),]
xxvii is.data.frame(tabla19)
> is.vector(tabla19)
xxviii tabla19$codigo<-gsub("AR","MQ",tabla19$codigo)
xxix median(tabla19$ph)
> summary(tabla19)
> quantile(tabla19$ph,probs=0.5)

xxx   tabla19[order(tabla19$acideztitulable),]
xxxi c(mean(tabla19$ph),sd(tabla19$ph))
```

```
xxxii tabla19bis<-tabla19[,c(1,6)]
xxxiii nrow(tabla19)*ncol(tabla19)
xxxiv table(tabla19$tipo)
xxxvsubset(tabla19,ph<3,select=c(codigo,ph))  o > tabla19[tabla19$ph<3,c(1,6)]
xxxvi subset(tabla19,ph<3 & acideztitulable>50)
xxxviinames(tabla19)
xxxviiitapply(tabla19$ph,factor(tabla19$tipo),mean)
xxxixtabla19<-
      cbind(tabla19,categoriaph=cut(tabla19$ph,breaks=c(0,1,2,3,4,6),label=c("ph<1","ph<2
      ","ph<3","ph<4","ph<6"),right=F,include.lowest=T))
xl    date()
xli   library(gdata)
xlii  apropos("test")
xliii help(fBasics)
xliv sum(tabla19$acideztitulable,na.rm=TRUE)
xlv   min(tabla19$ph) - max(tabla19$ph)
xlvi quantile(tabla19$fosfato)
xlviiwrite.table(tabla19, file = "tabla19exp.csv", sep = "\t",eol="\n", dec=".", row.names=F,
      col.names=T)
xlviiisetwd()
xlix setwd(.....irá el nombre al path del directorio al cual desea ir)
l     nchar(as.vector(tabla19$clasificacion))
li     tabla19<-cbind(tabla19,abreviaturaclasificacion=substr(tabla19$clasificacion,1,2))
lii    trunc(runif(20,0,100))
liii  seq(0,20)
liv   hist(rnorm(300,3,0.1))
```