Uso de herramientas informáticas para la recopilación, análisis e interpretación de datos de interés en las ciencias biomédicas

Contenidos

Continuación uso de data.frames....1
Head....1
Nombre de columnas....2
Transform....2
Eliminar/seleccionar/reordenar filas y/o columnas....3
eliminar/seleccionar filas....3
Eliminar/seleccionar columnas....5
Eliminar/seleccionar filas y columnas....6
Eliminar/seleccionar filas y columnas con condiciones....6
Reordenar filas y/o columnas....7
Tapply....8
Tapply con un factor....8
Tapply con mas de un factor....9
Trasponer un data frame....9
Transformar formato de variables....10
transformar variable numérica en factor....10
Transformar variable continua en factor con niveles....11
Transformar factores en variables numéricas....13
Obtener niveles de un factor de un data.frame....15
Agregar niveles de un factor....15
Operaciones básicas....16
logaritmo natural....16
logaritmo decimal....16
raíz cuadrada....17
funciones trigonométricas....17
Constantes....17
Funciones de tiempo....18
Información importante....21

Módulo 1 – clase 4

Continuación uso de data.frames

Trabajaremos con la tablaR141 de la planilla de cálculo tablaR1-4.ods/xls. Introdúzcala en su espacio de trabajo. Para ello utilice el siguiente código y lo aprendido en clases anteriores

```
> 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
```

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. Para el ejemplo siguiente le pedimos ver solo 2 filas.

```
> 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 por defecto

```
> 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
```

Nombre de columnas

La función names() nos da solo los nombre de las columnas del data.frame. Seguimos con el data.frame tablaR141, que mostramos nuevamente a continuación

```
> 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
```

ahora veamos solo los nombres de las columnas

```
> names(tablaR141)
```

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

Esta función no es útil para tablas pequeñas donde podemos ver todas las columnas. Pero se torna muy útil cuando las tablas son muy grandes.

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. Es decir si sobre escribimos el data.frame tablaR141 con el resultado hallado con transform()

```
> 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 del peso en la columna número 1 del data.frame. Para seguir con la tabla original deberá introducirla nuevamente.

Eliminar/seleccionar/reordenar filas y/o columnas

eliminar/seleccionar filas

Sigamos trabajando con el data.frame tablaR141. Si lo ha modificado puede volver a introducirlo, con el código ya aplicado

```
>  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
```

Para varias acciones siguientes tiene que tener en cuenta que si escribimos el nombre del data.frame seguido de corchete, por ejemplo tablaR141[filas,columnas], entre corchete se indica antes de la coma las filas. que se desean seleccionar y después de la coma, las columnas a seleccionar.
Por ejemplo si en el data.frame tablaR141 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, el data frame no se ha modificado. Siempre recuerde que si quiere que los cambios sean guardados reemplazando el data.frame original debe escribir el código y reasignarlo al mismo objeto. El siguiente sería el código a utilizar
> tablaR141<. tablaR141[c(1:4),]

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 números 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(1:4)
```

```
> 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
```

Vemos que la primer columna ahora tiene números correlativos.
Es útil en estos casos utilizar la función nrow(), que cuenta por nosotros el número de filas de un data.frame. Es especialmente útil si tenemos un data.frame con un gran número de filas. En lugar de utilizar el número 4, para indicarle la última fila utilizamos nrow(tabla2), quedando el código como se indica a continuación

```
> 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
```

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, dentro de los corchetes. Si antes de la coma no se pone nada es porque se desean seleccionar 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 y todas las filas, escribiremos

```
> 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
```

Eliminar/seleccionar filas y columnas

Con la misma metodología se pueden seleccionar filas y columna, simultáneamente. Supongamos que deseamos las filas 1,3 y 4 además las columna 1,3 y 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, ya que la sobreescribiremos). Si ya no deseamos la tabla original, la sobre escribimos con el siguiente código

```
> 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.

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. Continuemos 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
```

Supongamos que 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
```

Veamos 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". Es decir que el signo == se utiliza para evaluar una condición o realizar una comparación.
Si hubiera escrito

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

estará reemplazando en la columna atributo sus elementos por el valor "a", como podemos comprobar a continuación.

```
> 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 los valores originales.

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"
```

Supongamos que deseamos reorganizar la tabla dejando primero atributo2 (columna 4), luego atributo (columna 3), peso (columna 1) y finalmente minutos (columna 2).
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 que llamamos 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
```

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 el funcionamiento de esta poderosa función. Puede resultar compleja al principio, pero su uso ayudará a comprender la lógica.

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 argumento indico cual es la columna para dividir a los elementos según nivel del factor. El segundo argumento siempre será un factor y no una variable numérica continua.

```
> tapply(tablaR141$peso,factor(tablaR141$atributo),mean)
```

```
    a     b     c
23.05 18.40 13.50
```

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

```
> tapply(tablaR141$peso,factor(tablaR141$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. Por esta razón en el valor del desvío estándar de c aparece NA (non available)

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(tablaR141$peso,factor(tablaR141$atributo),mean)
```

```
> mediapeso
```

```
    a     b     c
23.05 18.40 13.50
```

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 utiliza 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.

Trasponer un data frame

La 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(tablaR141[,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), paso a ser la fila 2.
La función t(), veremos que es de mayor aplicación en el uso de matrices.

Transformar formato de variables

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<-read.table('clipboard',header=TRUE,dec=',',sep='\t')
```

```
> 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(). Sin embargo no es una variable continua, ya que toma solo dos valores.

```
> 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 transformamos la variable dosis en factor y luego reemplazamos la columna dosis de tablaR142 estos valores factorizados. 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.

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 para esto 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
```

para mantener los datos originales de la tablaR141, creamos otro data.frame sobre el que haremos la modificaciones y al que llamamos 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 transformar la columna peso ($peso) en factor, tomaremos pesos de 10-20 como Q1 y pesos de 20-30 como Q2 y aplicaremos para ello la función cut(). Esta función permite transformar variables continuas en categóricas y posee diversos argumentos. El argumento breaks: indica los límites que se fijarán para dividir a la variable peso en este caso. El argumento label, contiene 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. A continuación vemos el código de la función cut() con el que reemplazamos la columna peso por sus valores factorizados. Como siempre estas funciones complejas, se simplifican con el uso cotidiano.

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

veamos el resultado obtenido

```
> 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
```

Como podemos ver los tres primeros valores han sido reemplazados por Q2, ya que los valores de peso son mayores o iguales a 20. Los dos últimos valores fueron reemplazados por Q1. Observamos con summary() el contenido actual de la columna peso.

```
> 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
```

Veremos más la utilidad de este tipo de cambio de variables en módulos siguientes

<u>Otras opciones de la función cut</u>

1- Intervalos de igual longitud

Si deseamos dividir en factores una variable continua, de manera que queden intervalos iguales, podemos hacerlo indicando simplemente el número de intervalos. Veamos con la 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
```

supongamos que deseamos dividir a los valores de glucosas en 4 intervalos iguales. Como podemos ver a continuación se le indica el número de intervalos y creamos un vector tablaR143glucosa

```
> tablaR142glucosa <- cut(tablaR142$glucosa, breaks=4)
```

si le pedimos los niveles de este vector, vemos que son cuatro y se nos indica el intervalo que abarca. Como verá cada intervalo tiene un paréntesis a la izquierda y un corchete a la derecha. Por ejemplo 1.23 es el límite entre el primero y el segundo intervalo. Esto indica que si existe entre los datos el valor 1.23 será incluido en el primer intervalo.

```
> levels(tablaR142glucosa)
```

```
[1] "(1.1,1.23]"  "(1.23,1.35]" "(1.35,1.48]" "(1.48,1.6]"
```

si queremos saber cuantos elementos tiene cada intervalo

```
> summary(tablaR142glucosa)
```

```
 (1.1,1.23] (1.23,1.35] (1.35,1.48]  (1.48,1.6]
          5           2           1           2
```

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.

Para comprender este cambio, supongamos que tenemos un vector con números pero que los mismos han sido introducidos como caracteres, cosa que hacemos encerrando a los números entre comillas.

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

verifiquemos que los elementos del vector a son caracteres y no números como parecen. Esto lo podemos hacer con la función str().

```
> str(a)
```

```
 chr [1:3] "1" "0.01" "3.2"
```

Tal cual nos indica str(), los elementos son caracteres, que queda indicado por chr.

Para realmente convencernos que no son números intentemos sumar le primer y segundo elemento del vector

```
> a[1] + a[2]
```

como consecuencia de oprimir enter hallamos lo siguiente

```
Error in a[1] + a[2] : non-numeric argument to binary operator
```

Si deseamos transformar los elementos que son caracteres, 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, objetivos que podemos lograr con la función str() como lo hicimos antes o bien con la función mode()

```
> str(b)

 num [1:3] 1 0.01 3.2
```

que nos está indicando que los elementos están en formato numérico, indicado por num.
Podemos verificarlo con una suma de elementos del vector

```
> b[1] + b[2]
[1] 1.01
```

Con mode(), obtenemos una respuesta más directa a nuestro interrogante

```
> mode(b)

[1] "numeric"
```

los elementos de b, tienen formato numérico, cuando en el vector a, que le dio origen eran caracteres

```
> 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, cuando intentemos transformarlos en formato numérico. Veamos el vector siguiente con caracteres

```
> 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 la función 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.

Obtener niveles de un factor de un data.frame

Si tenemos un data.frame con columnas en formato de factores con varios niveles, siendo la tabla visible es sencillo ver el número de niveles de dicho factor. Por ejemplo para la columna o 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 aplicáramos levels() y factor(), simultáneamente. 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
```

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 el código anterior 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(), vemos el nivel d, que aparece 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
```

Operaciones básicas

A continuación veremos algunas operaciones matemáticas básicas y su forma de resolución con R

logaritmo natural

Para calcular logaritmos naturales (de base e) se utiliza la función log(). Podemos utilizar la función para calcular el logaritmo de un número en particular, por ejemplo

```
> log(7.3)
```

```
[1] 1.987874
```

Podemos aplicarlo también a todo los elementos de 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 logaritmo en base ***e*** del elemento de la fila 1, columna 1 de la tablaR141

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

```
[1] 3.135494
```

logaritmo decimal

Para calcular logaritmo decimal (de base 10) utilizamos la función log10() o log(valor,10), donde valor representa el número al que le deseamos calcular el logaritmo. 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

raíz cuadrada

La raíz cuadrada se obtiene con la función sqrt(). Para el ejemplo la aplicamos a toda la columna peso de la tablaR141

```
> sqrt(tablaR141$peso)
```

[1] 4.795832 4.806246 4.827007 3.674235 3.674235

Si quisiéramos calcular la raíz cuadrada del número 23, aplicaríamos
```
> sqrt(23)
```

[1] 4.795832

funciones trigonométricas

De la misma manera tenemos funciones trigonométricas. 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) son:
coseno: cos(x)
tangente: tan(x)
arcoseno: acos(x)
arcseno: asin(x)
arctangente: atan(x)
Por supuesto que R dispone de infinidad de funciones que iremos aplicando a lo largo del curso y usted mismo irá descubriendo a medida que las necesite. También podrá usted crear sus propias funciones.

Constantes

R tiene también constantes incorporadas. A continuación se indican el contenido de algunas de ellas
A continuación mostramos algunas constantes que se hallan en la biblioteca base, que se instala cuando realizamos la instalación de R en nuestra computadora. Para ello ejecute
```
> help(Constants)
```

obtendrá
Built-in Constants
Description:
    Constants built into R.
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. La tablaR141 tiene los siguientes datos
```
> 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
```
Es importante notar que en el código anterior se especificaron los primeros cinco meses, dado que el data.frame tablaR141 tiene solo 5 líneas. Si no hubiera especificado el número de meses, se intentarian agregar los doce meses y daría un error. Compruebelo
```
> tabla2<-cbind(tablaR141,mes=c(month.abb))

Error in data.frame(..., check.names = FALSE) :
  arguments imply differing number of rows: 5, 12
```

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. Estas funciones son muy importantes cuando realice programación en R y requiera ejecutar acciones a tiempos definidos.
Para obtener la fecha como día de la semana, mes, día del mes, hora:minuto:segundos, año utilizamos la función date()
```
> date()

[1] "Mon Sep 13 17:59:17 2021"
```

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

```
[1] "2021-09-13 17:59:28 -03"
```

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

```
> Sys.Date()
```

```
[1] "2019-10-02"
```

La función siguiente le dará el horario al que está ajustada su computadora

```
> Sys.timezone()
```

```
[1] "America/Argentina/Buenos_Aires"
```

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, esto lo identificamos ya que cada valor de fecha es seguido por dos puntos y el número 1, indicándonos que cada fecha se presenta solo una vez.

Si intentamos hacer una diferencia entre dos valores de la columna fecha de la tablaR143, por ejemplo entre el valor de la fila 2 y la fila 1

```
> 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
```

str() nos indica que la columna fecha es un factor con 8 niveles.
Entonces, ¿como operar con fechas.?
En primer lugar se deben pasar los valores al formato de fecha entendible por R. Para ello utilizamos la función strptime(). Esta función tiene dos argumentos básicos: uno de ellos es el set de datos a transformar, en nuestro caso tablaR143$fecha, el otro es el formato en que se halla escrito. Como en este caso las fechas las tenemos escrita como día/mes/año, esto se lo indicamos a R con el siguiente argumento: "%d/%m/%Y. Si en su computadora aparecen en otro orden, cambie el orden de %d/%m/%Y

```
> 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
```

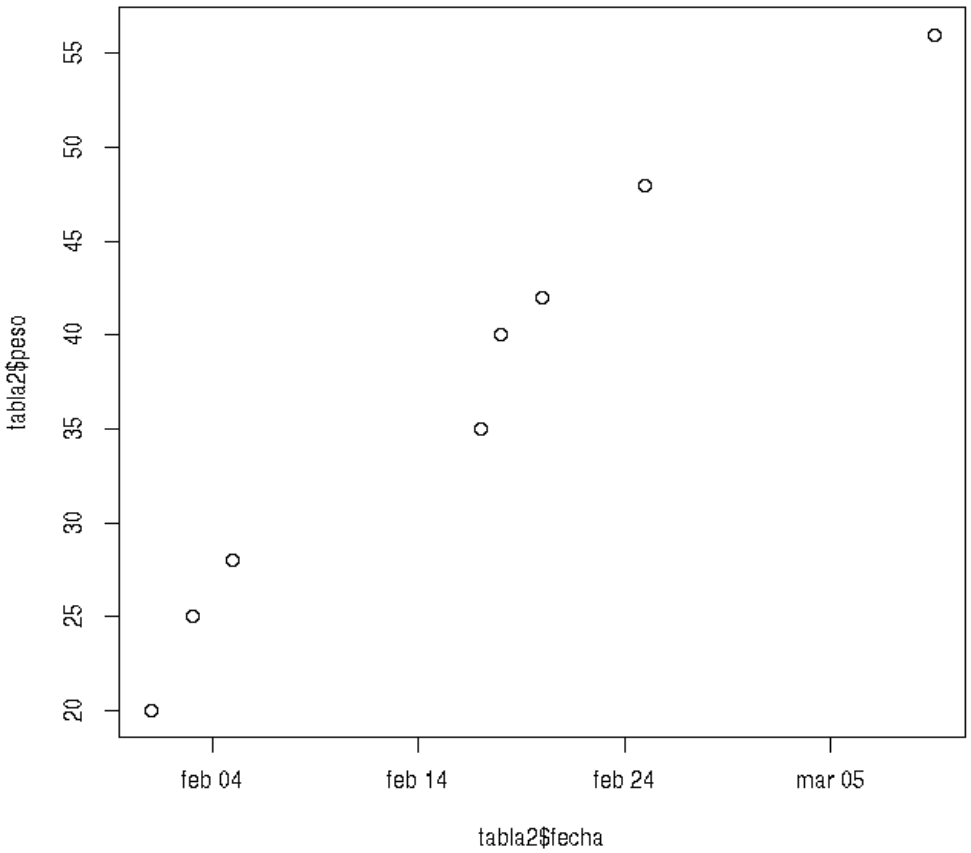
Vemos que la función strptime() modificó el formato pasándolo a año-mes-día. Con este formato podemos operar. Si restamos el dato de la fila 2 menos el datos de la fila 1, que claramente son dos días

```
> 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(tablaR143[4,1],tablaR143[1,1],units="mins")
```

Time difference of 23040 mins

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

Siempre es un desafió interesante, resolver problemas de tiempo. Con las funciones básicas descriptas tendrá un buen punto de partida.

Información importante

El manejo correcto de tiempos es imprescindible para ciertas aplicaciones en informática, especialmente cuando ellos no son solo un registro sino que se utilizan para hacer cálculos. Por ejemplo en un laboratorio podemos tener un número importante de reactivos, con fechas de vencimiento. Si nuestro stock está informatizado utilizando estrategias de R y deseamos que R nos avise un mes antes del vencimiento, deberemos poder hacer calculos con esas fechas y si están mal manejados los resultados pueden ser letales. Si bien estas aplicacions de R las veremos en los módulos 5 y 8, el manejo adecuado de fechas será planteado en este sitio.

Independientemente que existan otros mecanismos se plantea como mecanismo efecto, ingresar a las fechas como caracteres y luego transformarlas con strptime() en formato legible por R.

Analicemos el siguiente ejemplo

Supongamos que hemos diseñado un software escrito en R que lleva el stock de nuestro droguero en el que incorporamos nuevos reactivos y sustancias químicas. Deseamos que R nos avise 10 días antes de la fecha de vencimiento de cada reactivo. Para este caso será suficiente con calcular los días que existen entre la fecha de vencimiento del reactivo y la fecha actual.

Para evitar problemas lo mejor es ingresar todos los datos de fechas como caracter
Supongamos que un reactivo que ingresó al droguero vence el 20 de octubre de 2021. Nos convendrá ingresar este dato como

```
vencimiento<- as.character("20-10-2021")
```

Por otra parte nuestro sistema tendrá cada día la fecha actual con la función Sys.Date(), que incorporamos a una variable que llamamos fechaactual

```
fechaactual<-as.character(Sys.Date())
```

de esta manera ambos datos están como caracter. Por supuesto que no podemos hacer ningún cálculo con esto, pero los transformaremos en el momento de calcular la diferencia de tiempo, utilizando la función strptime con el siguiente código
Analicemos la forma en que está escrito vencimiento

```
> vencimiento
```

```
[1] "20-10-2021"
```

```
> str(vencimiento)
```

```
 chr "20-10-2021"
```

está escrito en el formato dia-mes-año que equivale a %d-%m-%Y
Los códigos para hacer referencia a cada parte de un tiempo o fecha lo puede hallar en

```
> help(strptime)
```

Con respecto a la fechaactual

```
> fechaactual
```

```
[1] "2021-09-14"
```

```
> str(fechaactual)
```

```
 chr "2021-09-14"
```

está escrito en formato: año-mes-dia, que equivale a %Y-%m-%d
Entonces para hallar la diferencia y que nos indique cuantos días faltan para el vencimiento utilizamos el siguiente código. De esta manera R entiende ambas fechas aunque estén en formatos diferentes

```
> as.numeric(strptime(vencimiento, "%d-%m-%Y") - strptime(fechaactual,"%Y-%m-%d"))
```

```
[1] 36
```

nos está indicando que aun faltan 36 días para el vencimiento.
Como veremos en módulos 5 y 8 usted podrá realizar un script que calcule diariamente estas diferencias y que le avise por pantalla, a través de un archivo o bien enviando un mail, cuando falte un día preestablecido de días.