

# Subconjuntos o indexación

true

septiembre 2022

## Contents

<b>Subconjuntos de datos</b>	<b>2</b>
<b>La indexación</b>	<b>2</b>
Extracción de elementos . . . . .	3
Extracción de variables . . . . .	4
Combinando la extracción . . . . .	5
Extracción en matrices . . . . .	5
Extracción con coincidencia parcial . . . . .	6
<b>Eliminando datos con NA</b>	<b>7</b>
Eliminando los casos con NAs de un data frame . . . . .	7
Eliminando NAs de un vector . . . . .	8
<b>La función subset</b>	<b>8</b>
<b>Ordenando vectores</b>	<b>10</b>
<b>Ejercicios</b>	<b>11</b>
Indicaciones . . . . .	11
Datos . . . . .	12
Informe . . . . .	15



Para regresar a Introducción R

---

Pueden descargar este documento en pdf haciendo clic aquí

---

## Subconjuntos de datos

Una vez creada una estructura de datos la flexibilidad de R para manipular los objetos es enorme, podemos seleccionar cualquier elemento o grupo de elementos y operar sobre ellos. Existen múltiples posibilidades para poder obtener subconjuntos de los datos. Intentaré aclarar algunas de esas posibilidades.

## La indexación

La manipulación de los objetos depende de la dimensionalidad de los objetos, usaremos los corchetes [] para acceder a las diferentes dimensiones, dentro de cada objeto podemos acceder a la dimensión filas, columnas y hojas. Cada uno de estos elementos separados por una coma. Así, si tengo un vector (una dimensión), solo necesito especificar la dimensión filas, si tengo una matriz o una data.frame necesito especificar dos dimensiones, filas y columnas. Finalmente, si tengo un arreglo, necesito especificar tres dimensiones, filas, columnas y hojas.

```
objeto[filas,columnas,hojas]
```

Veamos un ejemplo;

```
x <- 1:30 #Un vector  
x[8:12]
```

```
## [1] 8 9 10 11 12
```

```
y <- matrix(1:9,3,3) #Una matriz  
y[1:2,3]
```

```
## [1] 7 8
```

```
z <- array(1:27,c(3,3,3)) #Un arreglo  
z[2:3,2,3]
```

```
## [1] 23 24
```

Como vemos para el vector especificamos solamente las filas que queremos obtener (elementos del 8 al 12). Para la matriz le dimos dos dimensiones los elementos de la fila uno y dos de la columna 3. Finalmente, en el arreglo le pedimos los elementos de la fila dos y tres, de la columna dos y de la hoja tres.

Si queremos todos los elementos de alguna de las dimensiones dejamos el espacio en blanco. Así, si me interesa tener todas las filas de la segunda columna de la matriz usaríamos:

```
y[,2] #todas las filas de la columna dos
```

```
## [1] 4 5 6
```

```
y[2,] #todas las columnas de la fila dos
```

```
## [1] 2 5 8
```

## Extracción de elementos

La extracción de elementos del objeto puede realizarse de al menos 3 formas distintas.

1. Un vector de números naturales, usamos un vector numérico que extrae los elementos que se encuentran en la posición determinada por el vector numérico, si el vector de extracción es 2:3 extrae los elementos de la posición dos y tres.

```
x <- seq(1,40, by=2)
x #todos los elementos del vector

## [1] 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
```

```
x[1:3] #los elementos de la posición uno a la tres
```

```
## [1] 1 3 5
```

```
x[c(1,3,4)] #los elementos en la posición uno, tres y cuatro
```

```
## [1] 1 5 7
```

2. Un vector lógico de la misma longitud que el vector original. Podemos usar un vector lógico que permite extraer los elementos que cumplen la condición lógica. Podemos utilizar el mismo vector u otro vector.

```
cat <- sample(c("a","b"), 20, replace =TRUE)
vl <- x>10 #Generamos el vector lógico

x[vl] #Extraemos los datos en función del vector lógico creado
```

```
## [1] 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
```

```
x[cat=="a"] #Usamos directamente la operación lógica
```

```
## [1] 1 5 9 13 17 23 27 31 37 39
```

3. Un vector de números naturales negativos. Por medio de esta indexación se seleccionan todos los elementos del vector excepto los indicados con valores negativos.

```
x[-(1:3)] #elementos que no están en posición uno a tres

## [1] 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39

x[-c(1,3,4)] #elementos que no están en posición uno, tres y cuatro

## [1] 3 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
```

La indexación de las matrices, marco de datos y los arreglos se pueden acceder bajo los mismos principios, únicamente debemos recordar que debemos manejar más de una dimensión.

## Extracción de variables

Los data.frames y las listas son estructuras especiales por lo cual tienen algunas formas de realizar subconjuntos de datos. Podemos acceder a las variables usando el **doble corchete** (`[[]]`), el **corchete simple** (`[]`) o el **signo de dólar** (`$`). Estos signos pueden usar con la **posición de la variable** (la posición en números) o el **nombre de la variable**. Sin embargo el uso de cada una de estas posibilidades genera salidas que son ligeramente diferentes.

### Uso de doble corchete

Cuando usamos el doble corchete, R nos devuelve la estructura más baja posible, así, si tenemos un data frame la salida cuando uso el doble corchete es un vector. En cambio, si uso el doble corchete en una lista, la salida me devolverá la estructura básica que se almacenó en esa posición.

```
yd <- data.frame(peso=rnorm(10,20,5), tipo=rep(c("a","b"),5),
                altura= rnorm(10, 165, 30))

z <- list(letters[1:4], matrix(letters[1:9],3,3), yd) # Creamos la lista
names(z) <- c("letra","mat","dfr") #le damos nombres

yd[["peso"]]; yd[[1]]

## [1] 13.09758 20.14862 15.94121 21.67508 25.06257 12.45905 21.95102 13.97916
## [9] 26.68982 20.89274

## [1] 13.09758 20.14862 15.94121 21.67508 25.06257 12.45905 21.95102 13.97916
## [9] 26.68982 20.89274
```

```
z[["letra"]]; z[[1]]

## [1] "a" "b" "c" "d"

## [1] "a" "b" "c" "d"
```

En este caso el uso de los dos corchetes nos devolvió la estructura más baja, un vector. En el caso de las listas esta estructura más baja depende de lo que tenga en esa posición. Mire que sucede si cambia *letra* por *mat*. El uso del doble corchete lo podemos usar tanto con los nombres como con la posición de la variable.

### Uso del corchete simple

El corchete simple podemos usarlo en los dos estructuras de objetos, data frames y listas. En este caso la salida de esta extracción nos devuelve un objeto con igual característica al objeto “madre”, así, si extraigo los datos de un data frame la salida será un data frame, si extraigo de una lista la salida será una lista.

```
ydpes <- yd["peso"]

zlet <- z["letra"]

class(ydpes)

## [1] "data.frame"
```

```
class(zlet)
```

```
## [1] "list"
```

Como vemos los elementos extraídos siguen siendo un data frame y una lista. Puede desplegarlos a los nuevos objetos creados.

### Uso del signo de dólar (\$)

El uso del signo de dólar se usa únicamente con el nombre. La salida de este tipo de selección es similar a lo que obtenemos con el uso del doble corchete.

```
yddp <- yd$peso  
zsdl <- z$letra
```

```
yddp
```

```
## [1] 13.09758 20.14862 15.94121 21.67508 25.06257 12.45905 21.95102 13.97916  
## [9] 26.68982 20.89274
```

```
zsdl
```

```
## [1] "a" "b" "c" "d"
```

Como vemos en este caso los objetos resultantes son vectores.

### Combinando la extracción

Como hemos visto hay muchas formas de extraer los datos, lo interesante de R es que puedo combinar las diferentes formas de extracción, por ejemplo puedo usar doble corchete y una expresión lógica, o un solo corchete y una ubicación.

```
yd["peso"][3:4,]
```

```
## [1] 15.94121 21.67508
```

```
yd[["peso"]][3:4]
```

```
## [1] 15.94121 21.67508
```

Como vemos estas dos expresiones me dan la misma salida, pero miren bien la expresión, es ligeramente diferente. Recuerden los corchetes simples me devuelve un data frame (el cual tiene dos dimensiones), mientras que el corchete simple me devuelve un vector (con una sola dimensión).

### Extracción en matrices

En el caso de las matrices la forma de extracción es distinta, aunque igual que en los otros casos se puede extraer con nombre o por posición, en este caso es necesario decir si el nombre es de las filas o de las columnas.

```
colnames(y) <- letters[1:3]
```

```
y["a"]
```

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

```
y[, "a"]; y[, 1]
```

```
## [1] 1 2 3
```

```
## [1] 1 2 3
```

Como vemos el uso del nombre con uno o dos corchetes no produce una extracción de las variables. Es necesario que aclare que me refiero a los nombres o la posición de las columnas, en este caso la salida es siempre un vector.

Otra forma de extraer los datos de las matrices es con la ubicación global, la posición global del elemento en la matriz se refiere al puesto contado en orden a partir del primer elemento, contando por filas. Veamos un ejemplo.

```
matC <- matrix(letters[1:9], 3,3)
```

```
matC
```

```
##      [,1] [,2] [,3]  
## [1,] "a"  "d"  "g"  
## [2,] "b"  "e"  "h"  
## [3,] "c"  "f"  "i"
```

```
# el elemento "a" se encuentra en la posición 1,  
# así que puedo extraer este elemento usando esta posición
```

```
matC[1]
```

```
## [1] "a"
```

```
# el elemento "e" se encuentra en la posición 5
```

```
matC[5]
```

```
## [1] "e"
```

Nótese que para extraer los elementos de la matriz por la posición global usamos un solo corchete y una sola dimensión.

## Extracción con coincidencia parcial

Como hemos visto el usar los nombres de las variables para poder extraer los datos nos permite tener mucha flexibilidad, sin embargo, algunas veces es posible que los nombres de las variables sean muy largos y complejos. Una de las ventajas que ofrece R es que la extracción de la variable se la puede hacer con una coincidencia parcial. Veamos un ejemplo.

```
names(yd)
```

```
## [1] "peso" "tipo" "altura"
```

```
yd$p
```

```
## [1] 13.09758 20.14862 15.94121 21.67508 25.06257 12.45905 21.95102 13.97916
```

```
## [9] 26.68982 20.89274
```

```
yd[["p", exact=FALSE]]
```

```
## [1] 13.09758 20.14862 15.94121 21.67508 25.06257 12.45905 21.95102 13.97916
```

```
## [9] 26.68982 20.89274
```

Como vemos en estos dos casos hemos podido extraer la variable que lleva de nombre peso pero solamente usando la primera letra. Si hay una coincidencia parcial en más de una variable R dará la extracción como nula (NULL).

## Eliminando datos con NA

Cuando trabajamos con datos, muchas veces estos datos no se encuentran completos y tenemos variables con datos no disponibles (**NA**; Not Available), estos datos pueden ser un problema cuando estamos haciendo un análisis por lo que algunas veces nos interesa quedarnos con el grupo de datos que tenemos los datos completos.

### Eliminando los casos con NAs de un data frame

La función `complete.cases` nos permite generar un vector lógico con las filas que tienen los datos completos (sin NA).

```
set.seed(34)
dtfna <- data.frame(a=sample(c(rnorm(10,23,5), rep(NA,5)), 10),
                    b=sample(c(rnorm(10,46,5), rep(NA,3)), 10),
                    c=sample(c(rnorm(10,2,5), rep(NA,3)), 10))

compdt <- complete.cases(dtfna)
compdt
```

```
## [1] TRUE FALSE FALSE FALSE FALSE TRUE TRUE TRUE FALSE FALSE
```

Ahora tenemos un vector que nos dice los casos en los que tenemos los datos completos. Podemos usar este vector para extraer los datos completos.

```
dtfna[compdt,]
```

```
##           a           b           c
## 1 22.30555 38.34615 -0.3365098
## 6 18.75493 48.10822 -2.3306438
## 7 19.26139 41.50276 11.8424693
## 8 28.33402 44.60130  4.2056921
```

Podemos tener el mismo resultado de filtrado usando la función `na.omit`, esta función nos devuelve una matriz con los datos completos. La diferencia de esta función con la función `complete.cases` usada anteriormente, es que la primera es una función lógica, nos devuelve un vector lógico de los casos completos y no, la función `na.omit` es una función que elimina los NAs de una matriz.

```
na.omit(dtfna)
```

```
##           a           b           c
## 1 22.30555 38.34615 -0.3365098
## 6 18.75493 48.10822 -2.3306438
## 7 19.26139 41.50276 11.8424693
## 8 28.33402 44.60130  4.2056921
```

## Eliminando NAs de un vector

Cuando lo que nos interesa es extraer los datos completos de un vector usamos la función `is.na` esta función nos permite ubicar los datos con NA y con esto extraer los datos completos.

```
vecna <- sample(c(rnorm(10,46,5), rep(NA,3)), 10)
naV <- is.na(vecna)
```

```
vecna; naV
```

```
## [1] 45.57938 50.81343      NA 40.86374 53.01523      NA 42.21978 52.32222
## [9] 40.87128 45.96671
```

```
## [1] FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
```

El vector lógico nos muestra donde están los NAs bueno realmente nos interesa que datos nos son NA, así que usaremos el opuesto con la ayuda de `!`.

```
naV <- !is.na(vecna) #Busca los que no son NA
vecna[naV]
```

```
## [1] 45.57938 50.81343 40.86374 53.01523 42.21978 52.32222 40.87128 45.96671
```

Ahora tenemos los datos sin NA.

## La función subset

La indexación es la forma más formal de seleccionar filas y columnas de un marco de datos o una matriz, sin embargo, muchas veces la indexación puede tornarse engorrosa. El seleccionar unas filas de unas columnas definidas puede implicar generar diferentes índices para cada dimensión.

La función `subset` proporciona una forma más efectiva y sencilla de seleccionar filas y columnas. Esta función permite, seleccionar unas columnas y generar filtros lógicos para las filas.



```
#Incluimos una columna en el data frame de nivel
set.seed(345)
dtfna$nivel <- sample(c("ab", "bc", "cf"), 10, replace=T)

#Seleccionamos las columnas a y c
#filtramos los casos que corresponden a "ab"

subset(dtfna, select= c(a, c), subset= nivel=="ab" )
```

```
##           a           c
## 1  22.30555 -0.3365098
## 5  20.72254 -1.7139867
## 7  19.26139 11.8424693
## 8  28.33402  4.2056921
## 10 22.96270         NA
```

Como podemos observar el código para seleccionar columnas y hacer filtros es muy sencillo. Una cosa importante es que en esta función los nombres de las columnas, que normalmente esperaríamos que estén con comillas, son escritas directamente sin comillas.

Puesto que nuestra matriz de salida tiene NAs, deberíamos hacer un filtro para eliminar los NAs.

```
subset(dtfna, select= c(a, c), subset= nivel=="ab" &
       complete.cases(dtfna))
```

```
##           a           c
## 1  22.30555 -0.3365098
## 7  19.26139 11.8424693
## 8  28.33402  4.2056921
```

La función `subset` nos ofrece realizar filtros con facilidad y es muy flexible. Una funcionalidad interesante de esta función es nos permite eliminar columnas usando el nombre, algo que no es posible con otras funciones.

```
subset(dtfna, select= -c(b, nivel), subset= nivel=="ab" &
       complete.cases(dtfna))
```

```
##           a           c
## 1  22.30555 -0.3365098
## 7  19.26139 11.8424693
## 8  28.33402  4.2056921
```

Finalmente, podemos usar los nombres de las variables almacenadas en un vector para realizar la selección.

```
sna <- c("a", "b")
subset(dtfna, select= sna, subset= nivel=="ab" &
       complete.cases(dtfna))
```

```
##           a           b
## 1  22.30555 38.34615
## 7  19.26139 41.50276
## 8  28.33402 44.60130
```

## Ordenando vectores

Muchas veces cuando estamos trabajando con nuestros datos es necesario tener la posibilidad de ordenar nuestros datos. R tiene un par de funciones que nos dan la posibilidad de ordenar un vector; `order` y `sort`

```
z <- sample(letters, 20)
```

```
z
```

```
## [1] "z" "s" "v" "e" "c" "w" "y" "m" "o" "a" "p" "l" "t" "k" "h" "g" "r" "i" "b"
## [20] "j"
```

```
sort(z, decreasing = FALSE)
```

```
## [1] "a" "b" "c" "e" "g" "h" "i" "j" "k" "l" "m" "o" "p" "r" "s" "t" "v" "w" "y"
## [20] "z"
```

```
order(z, decreasing=FALSE)
```

```
## [1] 10 19 5 4 16 15 18 20 14 12 8 9 11 17 2 13 3 6 7 1
```

La diferencia entre estas dos funciones es que `sort` ordena los elementos en orden ascendente. Mientras que `order` genera un vector con la posición de cada elemento, si cambiamos el argumento de `decreasing` a `TRUE` entonces el orden será decreciente.

Ahora, podemos usar esa función para ordenar un data frame. Veamos cómo podemos usar estas funciones para ordenar nuestro data frame.

```
dtfna[order(dtfna["nivel"]),]
```

```
## Warning in xtfm.data.frame(x): cannot xtfm data frames
```

```
##      a      b      c nivel
## 1 22.30555 38.34615 -0.3365098 ab
## 5 20.72254      NA -1.7139867 ab
## 7 19.26139 41.50276 11.8424693 ab
## 8 28.33402 44.60130 4.2056921 ab
## 10 22.96270 36.78669      NA ab
## 6 18.75493 48.10822 -2.3306438 bc
## 9      NA      NA      NA bc
## 2      NA 49.90684 -7.1889879 cf
## 3 20.12376      NA 5.5618703 cf
## 4      NA 40.59035      NA cf
```

Nuestra matriz ahora está ordenada por el nivel. Ahora, nos podría interesar ordenar nuestra matriz usando más de una variable.

```
dtfna[order(dtfna["nivel"], dtfna["a"]),]
```

```
## Warning in xtfm.data.frame(x): cannot xtfm data frames
```

```
## Warning in xtfm.data.frame(x): cannot xtfm data frames
```

##	a	b	c	nivel
## 7	19.26139	41.50276	11.8424693	ab
## 5	20.72254	NA	-1.7139867	ab
## 1	22.30555	38.34615	-0.3365098	ab
## 10	22.96270	36.78669	NA	ab
## 8	28.33402	44.60130	4.2056921	ab
## 6	18.75493	48.10822	-2.3306438	bc
## 9	NA	NA	NA	bc
## 3	20.12376	NA	5.5618703	cf
## 2	NA	49.90684	-7.1889879	cf
## 4	NA	40.59035	NA	cf

Otra forma es usar la función `with`, esta función nos permite seleccionar un objeto y con ese generar una operación.

```
dtfna[with(dtfna, order(nivel, a)),]
```

##	a	b	c	nivel
## 7	19.26139	41.50276	11.8424693	ab
## 5	20.72254	NA	-1.7139867	ab
## 1	22.30555	38.34615	-0.3365098	ab
## 10	22.96270	36.78669	NA	ab
## 8	28.33402	44.60130	4.2056921	ab
## 6	18.75493	48.10822	-2.3306438	bc
## 9	NA	NA	NA	bc
## 3	20.12376	NA	5.5618703	cf
## 2	NA	49.90684	-7.1889879	cf
## 4	NA	40.59035	NA	cf

## Ejercicios

### Indicaciones

1. Lea los cuatro conjuntos de datos desde internet según las direcciones mostradas en la sección *Datos*. Es importante que cargue los datos y les asigne los nombres correctamente como se explica en esa sección
2. El primer objeto que usaremos es el objeto *dinic* (objeto 1), con este objeto siga los siguientes pasos:
  - a. Extraer como un `data.frame` las columnas 1 y 2 del objeto 1.
  - b. Use el la columna 2 para extraer todos los elementos que son menores a 1300.
  - c. Extraiga la columna 1 como un vector y ordenar este vector usando el vector “ord” de forma ascendente (`decreasing = False`).
3. Use la función `cat` e imprima el vector extraído en el punto anterior y el vector que obtenga de extraer la posición global de la matriz “nobj” en las posiciones: 1, 10, 12, 14. Verifique que el objeto *nobj* es una matriz.
4. Use el mensaje que obtiene del paso anterior para definir el objeto que usará para seguir con la tarea. Con este objeto (objeto 2) siga los siguientes pasos:

- a. Extraiga un data.frame con todas las variables, donde la columna 2 es igual a "a".
  - b. Ordene los datos del data frame generado en 4.a de forma ascendente usando la columna 3.
  - c. Extraiga como un vector los datos de la columna 1.
5. Utilice el vector extraído en el punto 4.c para extraer como vector la columna 1 del objeto que no ha sido usado hasta este punto (objeto 3). El vector extraído en el punto 4.c le dará la posición de los elementos a extraer en el objeto 3. Asigne el vector extraído con el nombre "mensaje".
6. Extraiga todos los datos donde ord es mayor a 4 del objeto 2.
- a. Ordene de forma descendente la matriz resultante del paso 6, usando las columnas; 3 y 2. Mantenga el orden definido para ordenar primero 3 y luego 2.
  - b. Del data.frame obtenido en el punto 6.a extraiga los dos primeros datos de la columna 1.
  - c. Use el objeto obtenido en 6.b para extraer la columna 2 del objeto 3. Guarde el objeto resultante como "califica".
7. Extraiga la columna 3 del objeto 3, donde el objeto creado en 6.d tiene el máximo valor (use la función max).
- a. Filtre el objeto 3 donde la columna 3 es igual al objeto creado en el punto 7.
  - b. Ordene de forma ascendente el data.frame resultante en el punto 7.1 usando la columna 5
  - c. Extraiga la columna 4 como vector.
  - d. Siga las indicaciones que se dan en ese vector.

## Datos

Esta es la tabla de datos **dinic** y puede descargarla desde el siguiente enlace: <https://raw.githubusercontent.com/Ciespinosa/Subconjuntos/master/dinic.csv>. La podemos ver a continuación.

```
##          mens.ord.ale
## 1      s;2160;0.005103427
## 2      e;1035;0.005478586
## 3      u;1305;0.012597976
## 4     NA;1350;0.021924786
## 5      y;1755;0.028660718
## 6      o;247.5;0.033454388
## 7     q;1822.5;0.037613588
## 8      g;157.5;0.043795999
## 9      I;877.5;0.056256697
## 10     u;1890;0.057278895
## 11     i;652.5;0.068514682
## 12     e;1462.5;0.093549199
## 13      x;45;0.098118947
## 14     as;2070;0.107866659
## 15     a;1372.5;0.123456227
## 16      l;225;0.136221049
## 17      s;1395;0.140795863
## 18     a;112.5;0.143437896
## 19     NA;202.5;0.145626556
## 20     w;1800;0.147786545
## 21      s;405;0.159353059
## 22     t;1192.5;0.165091829
## 23     t;1732.5;0.168213538
## 24     a;990;0.179728237
```

```

## 25    NA;427.5;0.20840887
## 26    a;337.5;0.223929239
## 27    p;1935;0.229471715
## 28    r;1710;0.234626575
## 29    o;607.5;0.240483678
## 30    i;697.5;0.261993447
## 31    h;1867.5;0.272254071
## 32    j;1980;0.274008672
## 33    e;450;0.279828513
## 34    o;1102.5;0.282326186
## 35    d;315;0.283412239
## 36    t;67.5;0.289598909
## 37    u;810;0.308890601
## 38    i;1912.5;0.334129212
## 39    s;2205;0.345295328
## 40    e;22.5;0.348262121
## 41    d;1665;0.360474856
## 42    a;180;0.363028631
## 43    NA;765;0.365049253
## 44    q;787.5;0.371354804
## 45    e;2047.5;0.375126334
## 46    NA;1012.5;0.382185312
## 47    o;1215;0.388531546
## 48    f;1642.5;0.39418503
## 49    e;832.5;0.399153965
## 50    s;270;0.402040657
## 51    g;1440;0.421470749
## 52    b;1125;0.426688861
## 53    j;1597.5;0.429935643
## 54    i;135;0.433497426
## 55    i;945;0.442758509
## 56    e;2025;0.450959441
## 57    h;1507.5;0.45836541
## 58    e;1687.5;0.477319994
## 59    d;1360;0.485233605
## 60    d;2092.5;0.489400111
## 61    NA;1080;0.500219779
## 62    m;1327.5;0.515526104
## 63    n;900;0.521331373
## 64    d;2250;0.523956376
## 65    m;2002.5;0.565374739
## 66    d;1417.5;0.59049546
## 67    p;585;0.615207946
## 68    n;1350;0.629581905
## 69    n;472.5;0.633198157
## 70    t;1485;0.636991933
## 71    y;1530;0.641341531
## 72    w;2227.5;0.642888533
## 73    d;1552.5;0.659645633
## 74    d;922.5;0.686376777
## 75    o;1957.5;0.692608737
## 76    r;90;0.696548551
## 77    o;382.5;0.705568097

```

```
## 78 h;2182.5;0.713349962
## 79 l;517.5;0.724233336
## 80 a;1777.5;0.728265759
## 81 n;742.5;0.740051765
## 82 t;360;0.762520164
## 83 NA;292.5;0.765278796
## 84 r;2115;0.7685465
## 85 c;967.5;0.773720245
## 86 NA;562.5;0.775930782
## 87 NA;855;0.802127776
## 88 c;675;0.838675139
## 89 NA;495;0.85803998
## 90 f;1845;0.873379155
## 91 f;2137.5;0.902137606
## 92 a;540;0.913305082
## 93 s;630;0.926886661
## 94 NA;1237.5;0.927665126
## 95 e;1170;0.932163397
## 96 b;1620;0.938074133
## 97 l;1057.5;0.971142547
## 98 j;1147.5;0.995212824
## 99 o;720;0.996153687
## 100 u;1575;0.9966088
```

Otra tabla de datos que usaremos durante este ejercicio es **nobj** y puede descargarla desde el siguiente enlace: <https://raw.githubusercontent.com/Ciespinosa/Subconjuntos/master/nobj.csv>. La podemos ver a continuación.

```
##   var1 var2 var3
## 1    d    a    r
## 2    d    f    u
## 3    g    h    e
## 4    e    h    m
## 5    d    n    r
```

La tercera tabla de datos que usaremos es **dinal** y puede descargarla desde el siguiente enlace: <https://raw.githubusercontent.com/Ciespinosa/Subconjuntos/master/dinal.csv>. La podemos ver a continuación.

```
##           men.cal..grup.menfin.ordfinal
## 1           ella;8;c;use la función cat;30
## 2                usted;7;c;para grabar;36
## 3           nosotros;10;b;use la función sink;45
## 4  ellos;6;c;imprima los objetos en el orden;;25
## 5                ha;6;c;califica;45
## 6           ella;4;d;mensaje;64
## 7           de;6;b;para generar ;48
## 8           realizado ;4;d;reporte;35
## 9           los;2;b;un reporte;52
## 10          comenzado;3;f;función sink;56
## 11  un;4;b;imprima los objetos en el orden;;54
## 12          organizado;2;g;mensaje;71
## 13          una;3;b;mensaje;65
## 14          trabajo de;10;b;califica;73
```

Finalmente, la última tabla que usaremos durante este ejercicio es **dnum**, esta tabla la puede descargar desde el siguiente enlace: <https://raw.githubusercontent.com/Ciespinosa/Subconjuntos/master/dnum.csv>. La podemos ver a continuación.

```
##      loc grup ord
## 1      2    a   1
## 2      5    a   2
## 3      8    a   3
## 4     11    a   4
## 5     14    a   5
## 6     45    b   1
## 7     23    b   2
## 8      2    b   3
## 9      1    b   4
## 10    12    b   5
## 11     2    b   6
## 12     3    c   5
## 13    14    c   4
## 14    NA    c   3
## 15    NA    c   2
## 16    NA    c   1
```

## Informe

El informe debe contener los códigos usados para el desarrollo de la práctica y el reporte generado.