

Práctica Programación R

Fernández Hernández Alberto

26/11/2020

Contents

Pregunta 1	2
Apartado 1	2
Apartado 2	3
Pregunta 2	4
Apartado 1	4
Apartado 2	8
Apartado 3	10
Apartado 4	14
Pregunta 3	17
Apartado 1	17
Apartado 2	19
Apartado 3	20
Apartado 4	21
Apartado 5	22
Pregunta 4	22
Apartado 1	23
Apartado 2	24
Apartado 3	24
Apartado 4	25

NOTA: salvo determinados apartados, la solución a cada uno de los ejercicios se ha llevado a cabo por medio de funciones definidas, integrando el código en una misma estructura.

Pregunta 1

Un boleto del sorteo de la ONCE consta de dos partes, la primera es un número de 4 dígitos y la segunda es un número de tres dígitos que forman la serie del boleto.

Aquí consideramos sólo el número, por ejemplo, 0209. Se pide:

Apartado 1

Genera todos los números que entran en el sorteo de la ONCE y mostrarlos con los cuatro dígitos.

Para generar todos los posibles números de cuatro dígitos, debemos pensar en todas las posibilidades de combinación. Dado que cada casilla puede valer un número comprendido entre 0 y 9, pudiendo repetirse en más de una ocasión, nos encontramos ante una **variación con repetición**. Por tanto, el número de posibles combinaciones es de:

$$10 * 10 * 10 * 10 = 10^4 = 10000$$

Considerando lo anterior, creamos una función denominada **generar_boletos**, cuyo único parámetro será el número de dígitos que forman la serie. En primer lugar, dado que cada número se puede repetir num_digitos veces (una por cada posición), repetimos la lista de posibles números (0-9) tantas veces como dígitos tenga el número. Finalmente, mediante la función *expand.grid* se genera un DataFrame con todas las posibles combinaciones a partir del vector anterior (de cara al apartado 2):

```
generar_boletos <- function(num.digitos) {  
  lista.combinaciones <- rep(list(seq(0,9)), num.digitos)  
  expand.grid(lista.combinaciones)  
}  
# Prueba  
df.combinaciones.sorteo <- generar_boletos(4)
```

Una vez ejecutada la función, echemos un primer vistazo al DataFrame:

```
head(df.combinaciones.sorteo, 5)
```

```
##   Var1 Var2 Var3 Var4  
## 1    0    0    0    0  
## 2    1    0    0    0  
## 3    2    0    0    0  
## 4    3    0    0    0  
## 5    4    0    0    0
```

```
tail(df.combinaciones.sorteo, 5)
```

```
##   Var1 Var2 Var3 Var4  
## 9996    5    9    9    9  
## 9997    6    9    9    9  
## 9998    7    9    9    9  
## 9999    8    9    9    9  
## 10000    9    9    9    9
```

```
# Comprobamos que el numero de filas es 10000  
nrow(df.combinaciones.sorteo)
```

```
## [1] 10000
```

Como podemos comprobar, el número de filas del DataFrame coincide con el número de posibles combinaciones (10000). Por otro lado, si comprobamos cuántas filas hay únicas (mediante la función *unique*, vemos que también coincide con el total de filas):

```
nrow(unique(df.combinaciones.sorteo))
```

```
## [1] 10000
```

Por último, si deseamos recuperar el total de combinaciones con los cuatro dígitos concatenados, mediante un *apply* aplicamos, a cada fila del DataFrame, la función *paste*, concatenando cada fila en una única cadena:

```
vector.combinaciones <- apply(df.combinaciones.sorteo, 1, paste, collapse = "")  
# Mostramos las primeras 50 combinaciones  
head(vector.combinaciones, 50)
```

```
## [1] "0000" "1000" "2000" "3000" "4000" "5000" "6000" "7000" "8000" "9000"  
## [11] "0100" "1100" "2100" "3100" "4100" "5100" "6100" "7100" "8100" "9100"  
## [21] "0200" "1200" "2200" "3200" "4200" "5200" "6200" "7200" "8200" "9200"  
## [31] "0300" "1300" "2300" "3300" "4300" "5300" "6300" "7300" "8300" "9300"  
## [41] "0400" "1400" "2400" "3400" "4400" "5400" "6400" "7400" "8400" "9400"
```

Apartado 2

¿Cuál es la suma de los números de un boleto que más se repite?

En primer lugar, para calcular la suma de los dígitos, y dado que se encuentran almacenados en un DataFrame, utilizaremos nuevamente la función *apply*, aplicando a nivel de fila la suma de todas sus columnas:

```
suma.combinaciones <- apply(df.combinaciones.sorteo, 1, sum)
```

```
# Ejemplo de salida  
suma.combinaciones[1:20]
```

```
## [1] 0 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 10
```

Por desgracia, R no dispone de una función que permita calcular la moda, por lo que habrá que diseñarla desde cero, pasando como parámetro el vector anterior con las sumas de cada combinación. Desde esta función creamos, en primer lugar, una tabla con las frecuencias de aparición de cada suma, empleando la función *table*. A continuación, obtenemos el/los índices con la/s suma/s de mayor frecuencia de aparición (*max*):

```
moda <- function(vector.suma) {  
  vector.suma.frecuencias <- table(vector.suma)  
  as.numeric(names(vector.suma.frecuencias)[vector.suma.frecuencias ==  
                                             max(vector.suma.frecuencias)])  
}
```

Una vez definida la función, realizamos la correspondiente prueba:

```
# Prueba funcion moda  
cat("La suma de los numeros que mas se repite es ", moda(suma.combinaciones))
```

```
## La suma de los numeros que mas se repite es 18
```

Esto último es posible comprobarlo mediante una pequeña tabla dinámica en Excel, contando el número de apariciones en *suma.combinaciones* :

Etiquetas de fila	Cuenta de SUMA
18	670
17	660
19	660
16	633
20	633
15	592
21	592
14	540

Figure 1: Salida tabla dinámica Excel

Pregunta 2

En la carpeta covid_19 hay una serie de archivos sobre el COVID-19 en España. Se pide:

Apartado 1

Leer los archivos “datos_provincias.csv”, “CodProv.txt” y “CodCCAA.dat”. Añade el código de la comunidad autónoma al fichero “datos_provincias.csv” (no manualmente).

En primer lugar, para leer cada archivo utilizamos la función `read.table` cuyos parámetros serán la ruta del fichero, el separador de cada columna (dado que los archivos emplean diferentes separadores como la coma o un tabulador), así como la cabecera (el cual debe estar a `TRUE` en todos los casos, ya que todos los archivos presentes contienen cabecera). Cabe remarcar el parámetro `na.strings` en `datos_provincias.csv`, ya que la provincia Navarra está denotada como NA, por lo que R lo interpreta como *Not Available* (vacío). Para evitarlo, indicamos que los campos *Not Available* corresponden con cadenas vacías en lugar de NA, literalmente:

```
# Comunidades Autonomas
cod.ccaa <- read.table("CodCCAA.csv", sep = "\t", header = TRUE)
head(cod.ccaa)
```

```
## Código Nombre.de.la.subdivisión.en.la.IS01. X
## 1 ES-AN Andalucía 0
## 2 ES-AR Aragón 1
## 3 ES-AS Asturias, Principado de 2
## 4 ES-CN Canarias 3
## 5 ES-CB Cantabria 4
## 6 ES-CM Castilla-La Mancha 5
```

```
#Dimensiones (Filas x Columnas)
dim(cod.ccaa)
```

```
## [1] 17 3
```

```
# Provincias
cod.prov <- read.table("CodProv.txt", sep = ",", header = TRUE)
head(cod.prov)
```

```
## Código Nombre.de.la.subdivisión.en.la.IS01 Comunidad.autónoma
## 1 ES-C A Coruña GA
## 2 ES-VI Araba PV
## 3 ES-AB Albacete CM
```

```
## 4   ES-A           Alicante/Alacant           VC
## 5   ES-AL           Almería                 AN
## 6   ES-O           Asturias                 AS
```

```
#Dimensiones (Filas x Columnas)
dim(cod.prov)
```

```
## [1] 50 3
```

```
# Datos provincias
# na.string = "" => Para no confundir NA (Navarra) con un valor NA
datos.provincias <- read.table("datos_provincias.csv", sep = ",", header = TRUE,
                               na.strings = "")
head(datos.provincias)
```

```
##   provincia_iso   fecha num_casos num_casos_prueba_pcr
## 1             A 2020-01-31         0                 0
## 2             AB 2020-01-31         0                 0
## 3             AL 2020-01-31         0                 0
## 4             AV 2020-01-31         0                 0
## 5             B 2020-01-31         1                 1
## 6             BA 2020-01-31         0                 0
##   num_casos_prueba_test_ac num_casos_prueba_otras
## 1                        0                      0
## 2                        0                      0
## 3                        0                      0
## 4                        0                      0
## 5                        0                      0
## 6                        0                      0
##   num_casos_prueba_desconocida
## 1                          0
## 2                          0
## 3                          0
## 4                          0
## 5                          0
## 6                          0
```

```
#Dimensiones (Filas x Columnas)
dim(datos.provincias)
```

```
## [1] 12324 7
```

Una vez leídos los ficheros, debemos añadir el código de comunidad autónoma, situado en *CodCCAA.csv*, al fichero *datos_provincias.csv*, sin tener que hacerlo manualmente. De forma previa, dado que los ficheros *CodCCAA.csv* y *CodProv.txt* contienen columnas con tildes, las renombramos:

```
colnames(cod.ccaa) <- c("Cod.Comunidad", "Nombre.Comunidad", "Num")
colnames(cod.prov) <- c("Codigo", "Nombre.de.la.subdivision.en.la.IS01",
                       "Comunidad.Autonoma")
```

Una vez renombradas dichas columnas, debemos preguntarnos ¿Cómo enlazamos ambos DataFrames? Debemos fijarnos en que el DataFrame *datos.provincias* tiene como columna el código de cada provincia, mientras que el DataFrame *cod.ccaa* no presenta ningún campo relacionado con la provincia, sino con las comunidades autónomas. Sin embargo, disponemos de un DataFrame intermedio (como si de una tabla intermedia se tratase en SQL): *cod.prov*, el cual relaciona la comunidad autónoma con la provincia.

Por tanto, el objetivo será unir (inicialmente) *cod.ccaa* con *cod.prov* para finalmente unirlo con *datos.provincias*, no solo de cara al apartado 1 sino además para el apartado 2), donde pide los datos en función del código de

CCAA, por lo que una vez mezcladas las tres tablas pueden servirnos perfectamente para el resto de apartados. Sin embargo, si nos fijamos en el contenido de los DataFrames, tanto el campo Comunidad Autónoma como el código de Provincia presentan diferentes formatos en cada tabla (por ejemplo, el campo Cod.Autonoma en *cod.ccaa* empieza por las siglas ES- mientras que el campo correspondiente en *cod.prov* no, y lo mismo ocurre con el código de Provincia). Por tanto, antes de definir cualquier función debemos renombrar dichos campos, añadiendo las siglas ES-:

```
cod.prov <- transform(cod.prov, Comunidad.Autonoma =
  paste("ES-", Comunidad.Autonoma, sep = ""))
datos.provincias <- transform(datos.provincias, provincia_iso =
  paste("ES-", provincia_iso, sep = ""))
```

A continuación, definimos una función (`juntar_tres_dataframes`), que presenta como parámetros el vector con los DataFrames a unir, así como la lista de claves con las que unir dichas tablas:

```
juntar_tres_dataframes <- function(vector.df, vector.claves) {
  merge(merge(vector.df[1], vector.df[2], by.x = vector.claves[1],
    by.y = vector.claves[2], all.x = TRUE), vector.df[3],
    by.x = vector.claves[3], by.y = vector.claves[4], all.x = TRUE)
}
```

Mediante esta función, aplicaremos dos *merge* sobre el conjunto de DataFrames: uno entre los dos primeros DataFrames del vector pasados como parámetro (utilizando como claves las proporcionadas en *vector.claves*), así como un segundo y último *merge* entre el DataFrame anterior y el tercero situado en *vector.df*. Cabe destacar que el DataFrame *datos.provincias* contiene información de las Ciudades Autónomas de Ceuta y Melilla, las cuales **no están almacenadas en el resto de tablas**, por lo que al realizar ambos *merge* debemos conservar todas las filas de la tabla izquierda (mediante el parámetro *all.x = TRUE*) de forma que podamos mantener los datos sobre la evolución del COVID-19 en ambas ciudades.

Por tanto, una vez definida la función realizamos la llamada con los parámetros pertinentes, juntando en primer lugar *datos.provincias* con *cod.prov* y finalmente *cod.ccaa* :

```
datos.ccaa <- juntar_tres_dataframes(vector.df = list(datos.provincias, cod.prov, cod.ccaa),
  vector.claves = c("provincia_iso", "Codigo",
    "Comunidad.Autonoma", "Cod.Comunidad")
)
```

```
# Comprobemos que el numero de filas equivale al de datos.provincias
nrow(datos.ccaa) == nrow(datos.provincias)
```

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

```
# Consultemos las primeras filas
head(datos.ccaa)
```

```
##   Comunidad.Autonoma provincia_iso   fecha num_casos
## 1                ES-AN      ES-J 2020-05-03         1
## 2                ES-AN      ES-J 2020-09-15        75
## 3                ES-AN      ES-J 2020-05-28         3
## 4                ES-AN      ES-J 2020-04-07        32
## 5                ES-AN      ES-J 2020-03-17        51
## 6                ES-AN      ES-J 2020-05-24         0
##   num_casos_prueba_pcr num_casos_prueba_test_ac num_casos_prueba_otras
## 1                   0                   1                   0
## 2                  75                   0                   0
## 3                   3                   0                   0
## 4                  17                  15                   0
## 5                  49                   2                   0
```

```
## 6          0          0          0
## num_casos_prueba_desconocida Nombre.de.la.subdivision.en.la.IS01
## 1          0          Jaén
## 2          0          Jaén
## 3          0          Jaén
## 4          0          Jaén
## 5          0          Jaén
## 6          0          Jaén
## Nombre.Comunidad Num
## 1 Andalucía 0
## 2 Andalucía 0
## 3 Andalucía 0
## 4 Andalucía 0
## 5 Andalucía 0
## 6 Andalucía 0
```

Para comprobar que cada Provincia está en su correspondiente Comunidad Autónoma, mediante un *lapply* recuperamos cada Comunidad Autónoma, mostrando qué provincias tiene asociadas en el DataFrame. Debemos recordar que tanto Ceuta como Melilla presentan el campo Comunidad.Autonoma a NA, por lo que habrá que comprobarlo (*is.na*):

```
lapply(unique(datos.ccaa[, "Comunidad.Autonoma"]), function(x) {
  if (is.na(x)) {
    vector.provincias <- c(paste0(x, " => "),
                          unique(datos.ccaa[is.na(datos.ccaa["Comunidad.Autonoma"]),
                                              "provincia_iso"]))
  } else {
    vector.provincias <- c(paste0(x, " => "),
                          unique(datos.ccaa[datos.ccaa["Comunidad.Autonoma"] == x &
                                                  !is.na(datos.ccaa["Comunidad.Autonoma"]), "provincia_iso"]))
  }
  vector.provincias
})
```

```
## [[1]]
## [1] "ES-AN => " "ES-J"      "ES-AL"      "ES-CO"      "ES-GR"      "ES-CA"
## [7] "ES-H"      "ES-SE"      "ES-MA"
##
## [[2]]
## [1] "ES-AR => " "ES-TE"      "ES-HU"      "ES-Z"
##
## [[3]]
## [1] "ES-AS => " "ES-O"
##
## [[4]]
## [1] "ES-CB => " "ES-S"
##
## [[5]]
## [1] "ES-CL => " "ES-SO"      "ES-AV"      "ES-SG"      "ES-BU"
## [6] "ES-P"      "ES-ZA"      "ES-LE"      "ES-VA"      "ES-SA"
##
## [[6]]
## [1] "ES-CM => " "ES-CR"      "ES-TO"      "ES-AB"      "ES-CU"      "ES-GU"
##
## [[7]]
```

```
## [1] "ES-CN => " "ES-GC"      "ES-TF"
##
## [[8]]
## [1] "ES-CT => " "ES-GI"      "ES-B"      "ES-L"      "ES-T"
##
## [[9]]
## [1] "ES-EX => " "ES-BA"      "ES-CC"
##
## [[10]]
## [1] "ES-GA => " "ES-PO"      "ES-C"      "ES-OR"      "ES-LU"
##
## [[11]]
## [1] "ES-IB => " "ES-PM"
##
## [[12]]
## [1] "ES-MC => " "ES-MU"
##
## [[13]]
## [1] "ES-MD => " "ES-M"
##
## [[14]]
## [1] "ES-NC => " "ES-NA"
##
## [[15]]
## [1] "ES-PV => " "ES-VI"      "ES-SS"      "ES-BI"
##
## [[16]]
## [1] "ES-RI => " "ES-LO"
##
## [[17]]
## [1] "ES-VC => " "ES-V"      "ES-A"      "ES-CS"
##
## [[18]]
## [1] "NA => " "ES-ME"      "ES-CE"
```

Por tanto, vemos que cada provincia está asociada a su correspondiente comunidad, salvo Ceuta y Melilla (NA). Una vez tengamos el DataFrame generado, lo volcamos al fichero *datos_provincias.csv*, mediante la función *write.csv*, en lugar de *write.table* ya que existe una función específica que nos permite escribir directamente sobre archivos con extensión .csv:

```
write.csv(datos.ccaa[c("Comunidad.Autonoma", colnames(datos.provincias))],
          "datos_provincias.csv", row.names = FALSE)
```

Apartado 2

Selecciona los datos de la comunidad autónoma que te corresponda.

Para este apartado, ya disponemos del DataFrame con las tablas cruzadas, incluyendo el código de comunidad y el número de casos. Sin embargo, antes de continuar eliminaremos dos columnas redundantes: **Nombre.de.la.subdivision.en.la.ISO1** y **Nombre.Comunidad** (columnas 9 y 10, respectivamente), ya que disponemos de las columnas **Comunidad.Autonoma** y **provincia_iso**, evitando con ello información redundante:

```
datos.ccaa <- datos.ccaa[, -c(9, 10)]
```

Una vez eliminadas dichas columnas, mediante la función **seleccionar_datos_comunidad** filtraremos los

datos en función del código de comunidad, sumando los dígitos del DNI mod 17 (a través de la función `subset`), obteniendo el código con el mismo valor resultante:

```
seleccionar_datos_comunidad <- function(datos.ccaa, dni) {
  subset(datos.ccaa, Num == (dni %% 17))
}
```

```
# Prueba con Castilla y Leon (DNI = 12345678 mod 17 -> 6)
head(seleccionar_datos_comunidad(datos.ccaa, 12345678), 5)
```

```
##      Comunidad.Autonomia provincia_iso      fecha num_casos
## 3082          ES-CL          ES-S0 2020-05-24          0
## 3083          ES-CL          ES-S0 2020-03-11         20
## 3084          ES-CL          ES-S0 2020-05-15          3
## 3085          ES-CL          ES-AV 2020-04-27          2
## 3086          ES-CL          ES-S0 2020-02-23          0
##      num_casos_prueba_pcr num_casos_prueba_test_ac num_casos_prueba_otras
## 3082                0                0                0
## 3083               15                4                0
## 3084                3                0                0
## 3085                1                1                0
## 3086                0                0                0
##      num_casos_prueba_desconocida Num
## 3082                0 6
## 3083                1 6
## 3084                0 6
## 3085                0 6
## 3086                0 6
```

En nuestro caso, nos corresponde la Comunidad Autónoma de Cantabria:

```
# Prueba con Cantabria (DNI = 54003003 mod 17 -> 4)
datos.filtrado <- seleccionar_datos_comunidad(datos.ccaa, 54003003)
head(datos.filtrado, 5)
```

```
##      Comunidad.Autonomia provincia_iso      fecha num_casos
## 2845          ES-CB          ES-S 2020-02-14          0
## 2846          ES-CB          ES-S 2020-03-24         57
## 2847          ES-CB          ES-S 2020-05-22          1
## 2848          ES-CB          ES-S 2020-05-04          1
## 2849          ES-CB          ES-S 2020-05-03          3
##      num_casos_prueba_pcr num_casos_prueba_test_ac num_casos_prueba_otras
## 2845                0                0                0
## 2846               57                0                0
## 2847                1                0                0
## 2848                1                0                0
## 2849                3                0                0
##      num_casos_prueba_desconocida Num
## 2845                0 4
## 2846                0 4
## 2847                0 4
## 2848                0 4
## 2849                0 4
```

Apartado 3

Realizar un gráfico que muestre adecuadamente la evolución de los casos nuevos. Justifica el gráfico elegido.

NOTA: para la realización de este apartado se utilizarán los datos filtrados por el DNI el cual tengo asociado: 54003003, correspondiente con la **Comunidad Autónoma de Cantabria**, aunque también puede emplearse el *dataset* completo o de cualquier otra comunidad gracias al dinamismo de las funciones.

Para este apartado (y de forma previa a su implementación gráfica), la mejor forma de representación sería **agrupando el número de casos por fecha**, de forma que podamos tener el total de casos diarios. Para ello, y de cara tanto al apartado 3 como 4, diseñamos una función denominada **agrupar_datos** que recibe como parámetros el DataFrame a agrupar (*datos.ccaa*), el campo sobre el que agrupar (*clave*), así como un vector de columnas a filtrar.

Para ello, mediante la función *by* agrupamos cada fila por el campo **clave** (pasado como parámetro), obteniendo sus valores únicos. En el resto de columnas indicadas por parámetro, mediante la función *lapply* realizaremos el sumatorio por cada una de ellas, concatenando los resultados a través de la función *cbind*, la cual es invocada gracias a la función *do.call*. Una vez obtenida la fila, eliminamos su nombre (*rownames*) ya que por defecto es la primera columna; además de agrupar cada una de ellas en un mismo DataFrame, aplicando la función *rbind*, incluyendo finalmente los nombres de columnas:

```
agrupar_datos <- function(datos.ccaa, clave, columnas) {
  agrupacion <- by(datos.ccaa, list(datos.ccaa[, clave]), function(fila) {
    data.frame(
      total = unique(fila[, clave]),
      do.call(cbind,
        lapply(columnas, function(columna) sum(fila[, columna]))
      )
    )
  })
  # Eliminamos los índices de fila (por defecto es la primera columna)
  rownames(agrupacion) <- NULL
  df.agrupado <- do.call(rbind, agrupacion)
  colnames(df.agrupado) <- c(clave, columnas)
  df.agrupado
}
```

Una vez definida la función, **agrupamos el número de casos (num_casos) por cada fecha** :

```
# Prueba agrupar_datos
datos.agrupados.fecha <- agrupar_datos(datos.filtrado, "fecha", "num_casos")
# Echamos un primer vistazo a los datos obtenidos
tail(datos.agrupados.fecha)
```

```
##           fecha num_casos
## 232 2020-09-18         48
## 233 2020-09-19          1
## 234 2020-09-20          0
## 235 2020-09-21          0
## 236 2020-09-22          0
## 237 2020-09-23          0
```

Una vez tengamos los datos agrupados, de cara a facilitar la representación gráfica cambiaremos el tipo de dato al formato fecha (*factor* a *Date*):

```
datos.agrupados.fecha[, "fecha"] <- as.Date(datos.agrupados.fecha[, "fecha"])
```

```
sapply(datos.agrupados.fecha, class)
```

```
##      fecha num_casos  
##      "Date" "integer"
```

Una vez preprocesado el DataFrame, es momento de realizar su representación gráfica. Lo primero que debemos pensar es qué tipo de gráfico elegir (gráfico de barras, gráfico de líneas...). Una primera opción sería aplicar un gráfico de barras vertical, donde cada barra muestre el número de casos por cada fecha. Esta opción, a simple vista, resultaría ser viable de no ser por un factor: **el elevado número de filas**.

```
nrow(datos.agrupados.fecha)
```

```
## [1] 237
```

Es decir, supondría tener que representar en un mismo eje 237 barras en las que se muestra la distribución del número de casos a lo largo del tiempo, lo que podría dificultar el entendimiento del gráfico. Por ello, una alternativa sería agrupar nuevamente los datos **por meses**, reduciendo con ello el número de barras a representar. Para ello, realizamos una agrupación por cada mes, mediante la función *format*, representando el diagrama de barras empleando la función *barplot* tal y como se muestra a continuación:

```
agrupacion_v2 <- by(datos.agrupados.fecha, list(format(datos.agrupados.fecha[, "fecha"], "%m")),  
  function(fila) {  
    data.frame(  
      mes = unique(format(fila[, "fecha"], "%m")),  
      num_casos = unique(sum(fila[, "num_casos"]))  
    )  
  })  
agrupacion_v2 <- do.call(rbind, agrupacion_v2)
```

```
barplot(agrupacion_v2[, "num_casos"], names = agrupacion_v2[, "mes"],
        ylab = "NUMERO DE CASOS", xlab = "MESES",
        main = "EVOLUCION NUMERO DE CASOS COVID-19")
```

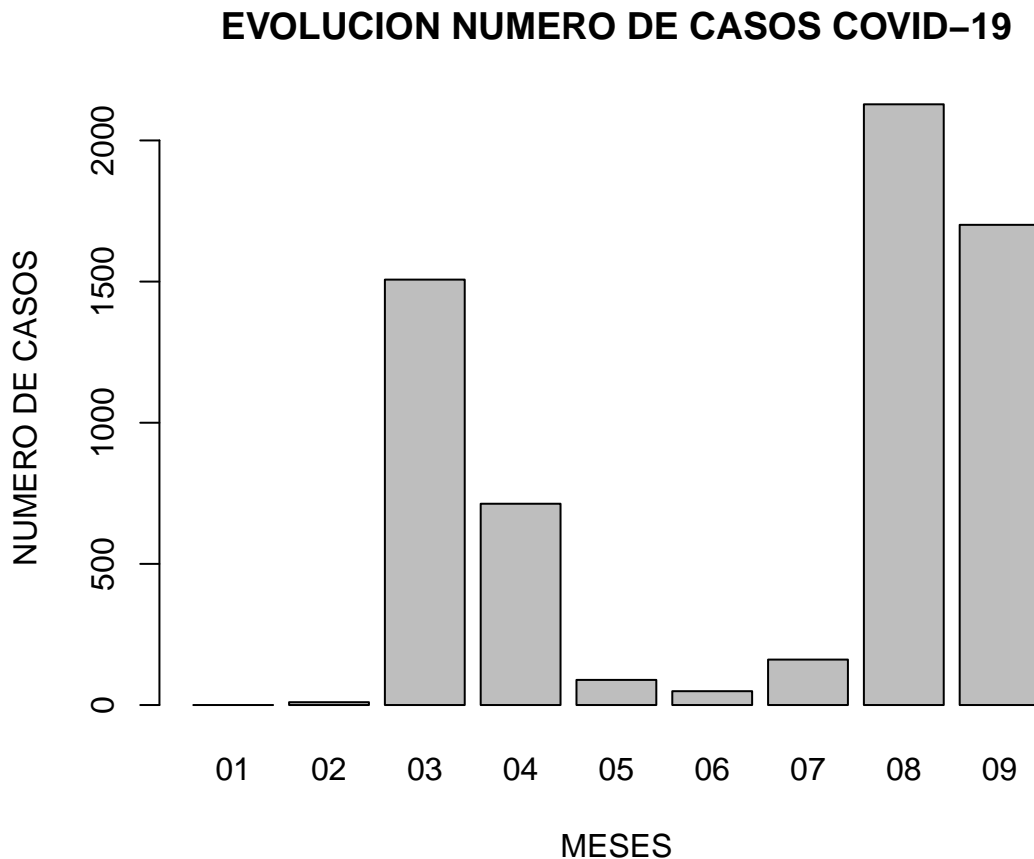


Figure 2: Gráfico de barras con la evolución del número de casos

Sin embargo, nos encontramos con un inconveniente: **¿Qué ha ocurrido en los meses de marzo, abril, agosto o septiembre?** Es decir, la incidencia acumulada ha sido alta, pero ¿Ha sido así durante todo el mes? ¿O ha habido semanas con un mayor repunte? A primera vista, con tan solo nueve barras somos incapaces de comprobarlo, es decir, perdemos información al agrupar.

Por el contrario, utilizar un gráfico de líneas puede resultar una mayor ventaja, permitiendo mostrar los cambios que sufre una variable **a lo largo del tiempo**, dado que lo que se desea representar es, al fin y al cabo, una serie temporal en el que se muestra una sucesión del número de casos por COVID-19, donde lo que se analiza es su tendencia (ascendente o descendente) entre días o semanas (no solo entre meses). Por el contrario, un gráfico de barras se emplea comunmente para comparar datos con un número limitado de categorías o grupos, sobretudo cuando la muestra no es muy grande.

Por ello, tanto para el apartado 3 como 4 creamos una función, denominada **imprimir_grafica**, recibiendo como parámetros el conjunto de datos a mostrar, las columnas empleadas para los ejes X e Y, el número de saltos a realizar en el eje X, así como el color del gráfico y un parámetro (**segmentos**) que permite, en caso de estar a TRUE, crear un segmento por cada fecha en el eje, uniéndolo con su correspondiente valor en el gráfico, además de mostrar la tendencia (de forma mucho más suavizada) del número de casos:

```
imprimir_grafica <- function(datos, eje.x, eje.y, saltos.eje.x, color, segmentos = FALSE) {
  par(mar=c(11,4,4,1), xaxt = "n")
  plot(x = datos[, eje.x], y = datos[, eje.y], type = "l",
```

```

ylim = range(pretty(c(0, datos[, eje.y]))),
main = "EVOLUCION NUMERO DE CASOS COVID-19",
xlab = "", ylab = "NUMERO DE CASOS", font.lab = 2,
col = color, las = 2)
par(xaxt = "s")
sec <- seq(datos[1, eje.x], datos[nrow(datos[eje.x]), eje.x],
          by = saltos.eje.x)

axis.Date(1, at = sec, format = "%Y-%m-%d", las = 2)

if (segmentos == TRUE) {
  segments(sec, 0, sec, subset(datos, datos[, eje.x] %in% sec)[, eje.y],
           lty = 2)
  points(x = subset(datos, datos[, eje.x] %in% sec), pch = 20)
  lines(smooth.spline(datos[, eje.x],
                     datos[, eje.y]),
        col = rgb(red = 0, green = 0, blue = 1, alpha = 0.7),
        lwd = 2)
}
mtext(text = "FECHA", side = 1, line = 6, font = 2)
}

```

De la función anterior, quisiera destacar algunos detalles relevantes:

1. Para poder apreciar en mayor medida el gráfico resultante, mediante la función *par* establecemos unos mayores márgenes de representación, además de eliminar temporalmente el eje X. Esto último lo hacemos para que el usuario, por parámetro, pueda elegir el número de divisiones (el número de fechas a mostrar en el eje X) de forma dinámica.
2. Por otro lado, el parámetro *las* se emplea para modificar la orientación de las etiquetas de los ejes (en horizontal).
3. Tras borrar el eje X con la función *par*, mediante la función *axis.Date* establecemos el eje de abcisas (concretamente en formato fecha), obteniendo una secuencia de fechas en función del parámetro (*saltos.eje.x*).
4. Una vez creado el eje de abcisas y si el usuario lo desea, mediante la función *segments* establecemos las rectas discontinuas para cada fecha en el eje X, empleando para ello la misma secuencia que con la función *axis.Date*. De este modo, el usuario podrá visualizar con mayor facilidad el número de casos en cada fecha.
5. Mediante la función *points* marcamos los números de casos anteriores.
6. Mediante la función *smooth.pline* del paquete *stats* se crea un gráfico “suavizado” con respecto a los datos originales, con el objetivo de visualizar (de forma más simplificada) la tendencia del número de casos a lo largo del gráfico.
7. Por último, mediante la función *mtext* establecemos el título del eje X.

Una vez definida la función, realizamos la prueba:

```
# Prueba imprimir_grafica
imprimir_grafica(datos.agrupados.fecha, "fecha", "num_casos", 14, "red", TRUE)
```

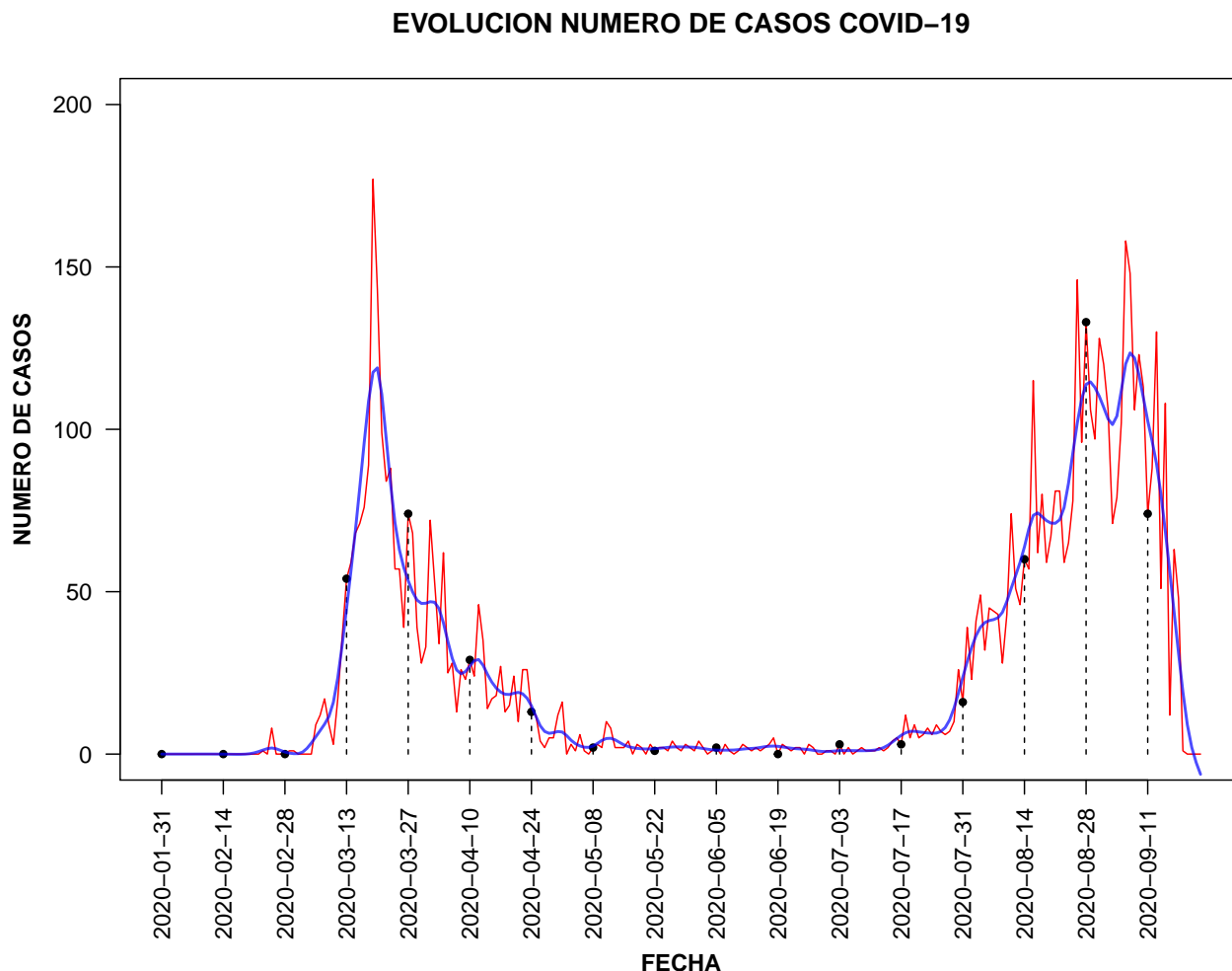


Figure 3: Gráfica con el número de casos en Cantabria

Analizando la evolución del número de casos (gráfico azul), podemos observar el elevado pico de contagios producido durante los primeros meses de pandemia, en especial cuando las medidas en Europa (y en concreto España), no eran lo suficientemente estrictas. Tras la declaración del estado de alarma el día 14 de marzo, los efectos del confinamiento se tradujeron en un descenso lento en el número de casos diarios, aunque no constantes, sino que observamos numerosos “repuntes” en el número de casos durante los meses de marzo y abril (algo que no podíamos comprobar con un diagrama de barras). No obstante, con la apertura gradual de fronteras y negocios la curva de contagios comenzó a crecer desde comienzos de verano, aunque de forma menos “apuntalada”, alcanzando en el mes de septiembre valores cercanos a los meses de abril y marzo aunque, al igual que en la “primera ola”, con continuos repuntes en el número de casos.

Cabe destacar, especialmente, la caída en el número de contagios a partir del día 22 de septiembre, lo que puede deberse a una falta de datos por parte de la Comunidad Autónoma, por lo que el *dataset* puede estar incompleto para dicho día.

Apartado 4

Presenta en un único gráfico la evolución de las distintas variables (columnas) por medio de un gráfico de líneas múltiples. Utiliza diferentes colores y añade una leyenda que muestre el

origen de cada línea.

Para este apartado, dado que disponemos de una función diseñada en el apartado anterior que permite agrupar las columnas de un DataFrame, llamaremos de nuevo a dicha función agrupando cada columna (casos nuevos, casos por pruebas PCR, casos por pruebas anticuerpos, otras pruebas y pruebas desconocidas) en función de la fecha. De este modo podremos conocer la evolución de cada variable a lo largo del tiempo:

```
casos.por.columnas <- agrupar_datos(datos.filtrado, "fecha", c("num_casos",
                    "num_casos_prueba_pcr",
                    "num_casos_prueba_test_ac", "num_casos_prueba_otras",
                    "num_casos_prueba_desconocida"))
# Modificamos nuevamente el campo fecha (factor -> Date)
casos.por.columnas[, "fecha"] <- as.Date(casos.por.columnas[, "fecha"])

# Mostramos las ultimas seis filas
tail(casos.por.columnas)
```

```
##          fecha num_casos num_casos_prueba_pcr num_casos_prueba_test_ac
## 232 2020-09-18         48              48              0
## 233 2020-09-19          1              1              0
## 234 2020-09-20          0              0              0
## 235 2020-09-21          0              0              0
## 236 2020-09-22          0              0              0
## 237 2020-09-23          0              0              0
##      num_casos_prueba_otras num_casos_prueba_desconocida
## 232                      0                      0
## 233                      0                      0
## 234                      0                      0
## 235                      0                      0
## 236                      0                      0
## 237                      0                      0
```

Una vez agrupado el DataFrame, debemos representar gráficamente cada columna con un color diferente. Para ello, creamos una función denominada **imprimir_multiples_lineas**, que recibe como parámetros tanto el DataFrame a representar gráficamente, los valores del eje X (campo fecha), del eje Y (columnas con el número de casos); además de un vector con la paleta de colores para cada línea y el número de saltos a realizar en el eje X.

En primer lugar, la función recupera la primera de las columnas pasadas como parámetro (num_casos) y establece el mismo gráfico que el del apartado anterior, llamando a la función **imprimir_grafica** (sin imprimir los segmentos como en el apartado anterior). A continuación, mediante la función *mapply* por cada pareja (columna, color) se añade una nueva línea al gráfico (*line*), asociando su correspondiente color. Finalmente, por medio de la función *legend* se muestra la leyenda en el gráfico, situándose en la parte superior:

```
imprimir_multiples_lineas <- function(datos, eje.x, eje.y, paleta, saltos.eje.x) {
  imprimir_grafica(datos, eje.x, eje.y[1], saltos.eje.x, paleta[1])

  mapply(FUN = function(x, y) {
    lines(datos[, eje.x], datos[, x], col = y, lwd = 2)
  }, eje.y, paleta)

  eje.y <- lapply(gsub('_', ' ', eje.y), toupper)
  legend(x= "top", legend = eje.y, fill = paleta, cex = 0.7, text.font = 2, bg = 'white')
}
```

A continuación, realizamos la correspondiente prueba, mostrando la evolución de cada columna a lo largo del tiempo:

```
# Prueba imprimir_múltiples_lineas
columnas <- c("num_casos", "num_casos_prueba_pcr", "num_casos_prueba_test_ac",
              "num_casos_prueba_otras", "num_casos_prueba_desconocida")
paleta <- c("red", "blue", "orange", "darkgreen", "purple")

imprimir_múltiples_lineas(casos.por.columnas, "fecha", columnas, paleta, 14)
```

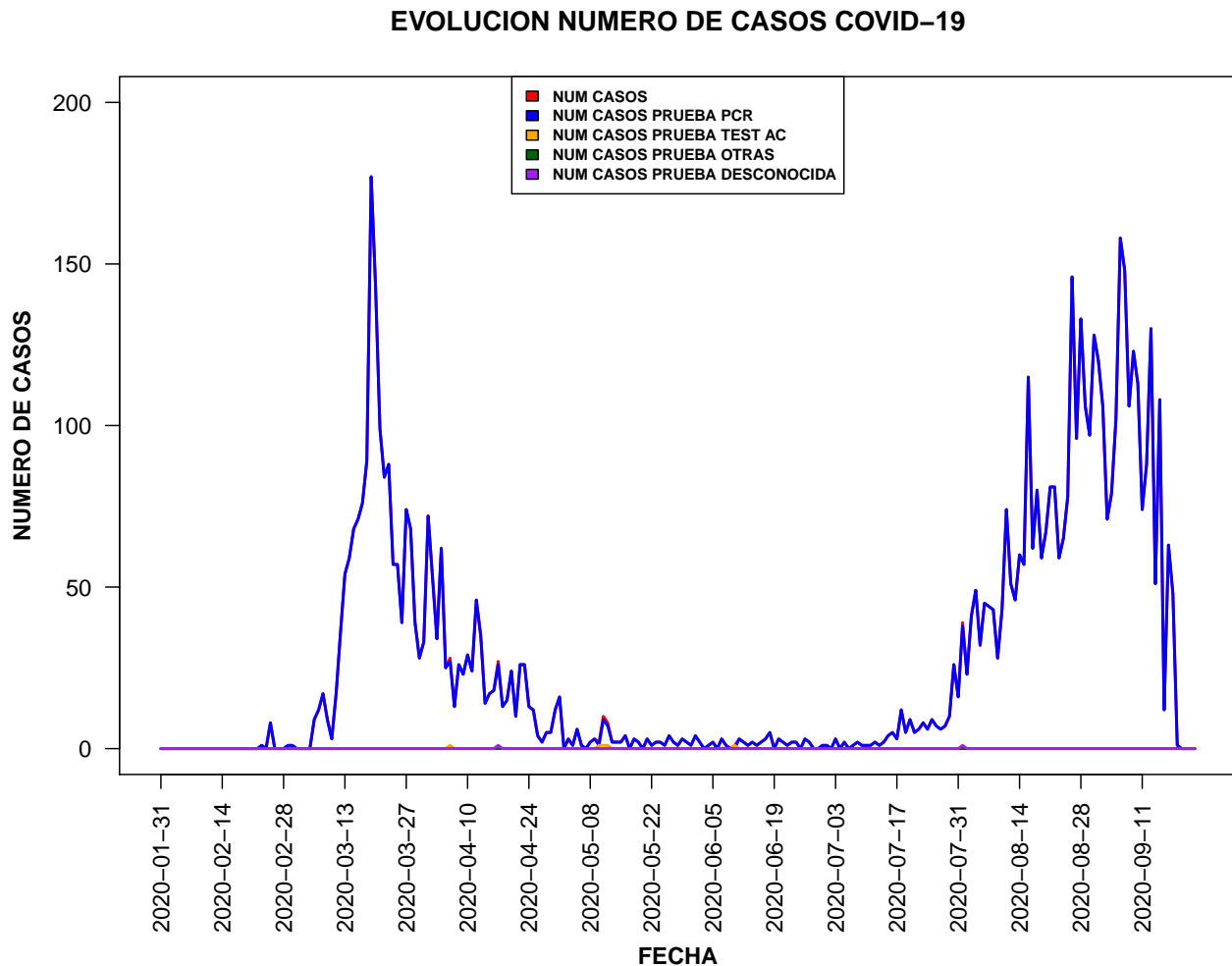


Figure 4: Gráfica con el número de casos de cada categoría en Cantabria

Analizando la gráfica resultante podemos comprobar cómo desde el comienzo de la pandemia la mayoría de los casos por COVID-19 son diagnosticados por pruebas PCR. Por el contrario, otras pruebas como los Anti-cuerpos, “Otras” o “Desconocida” apenas se emplearon en la Comunidad Autónoma, salvo casos excepcionales.

Para demostrar que cada columna se está mostrando correctamente, realizamos una pequeña prueba a nivel nacional, **con los datos de todas las CCAA:**


```
casos.por.columnas <- agrupar_datos(datos.ccaa, "fecha", columnas)
casos.por.columnas[, "fecha"] <- as.Date(casos.por.columnas[, "fecha"])

imprimir_multiples_lineas(casos.por.columnas, "fecha", columnas, paleta, 14)
```

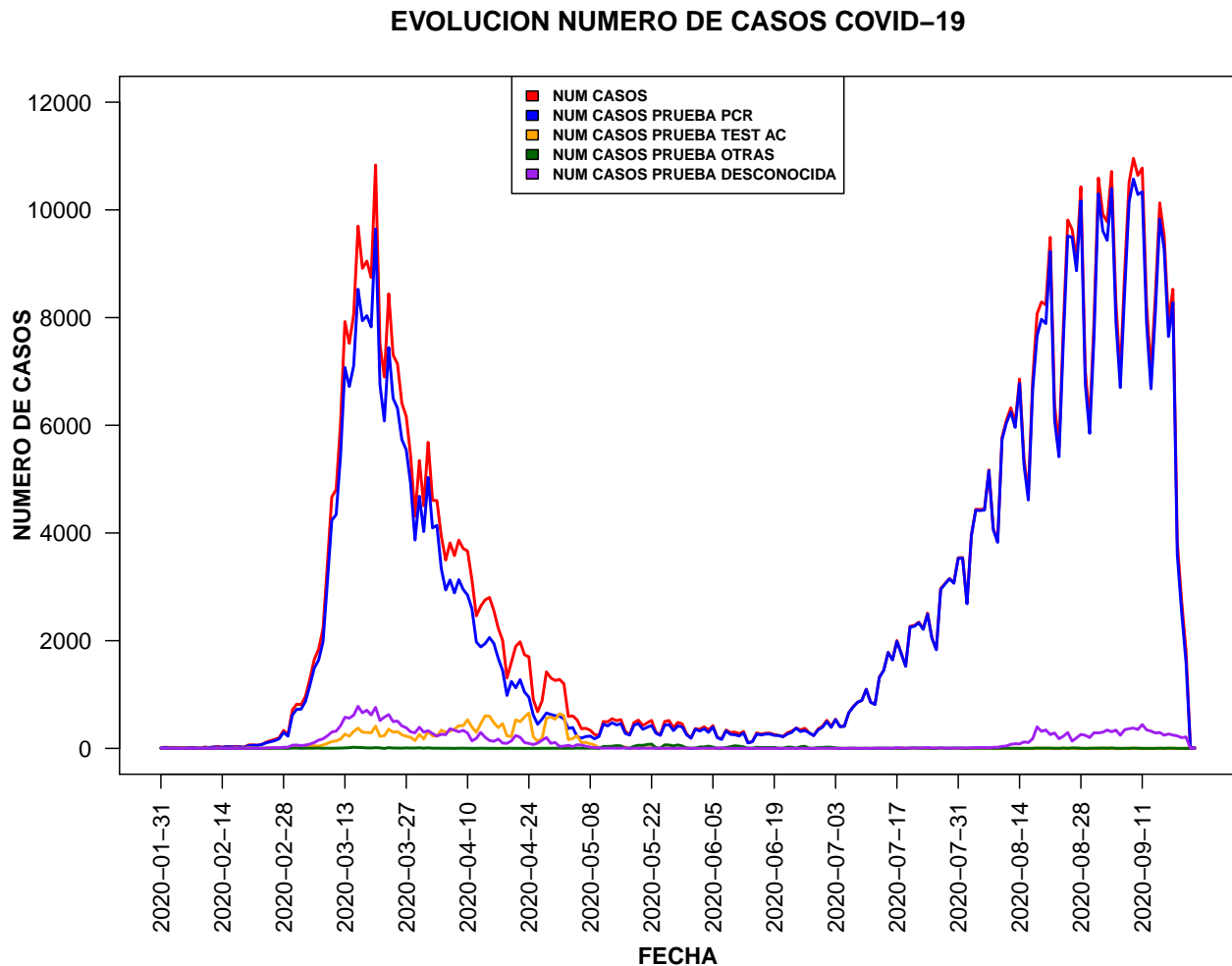


Figure 5: Gráfica con el número de casos de cada categoría en España

Pregunta 3

Consideramos un fichero de datos en formato SAS de nombre “punt.sas7bdat” que contiene datos sobre alumnos matriculados en diversos cursos.

Apartado 1

Importa el fichero de datos y guárdalo en un objeto de nombre `punt`. Comprueba la estructura del objeto `punt`. Si es necesario conviértelo en un data frame.

Hasta ahora, hemos trabajado con ficheros de extensiones `.txt` o separados por comas (`.csv`). En este apartado, nos encontramos con una nueva extensión: `sas7bdat`. Una posible opción sería leer dicho fichero por medio de las funciones del paquete `base` como `scan` o incluso `read.table` :

```
read.table("Punt.sas7bdat")
```

```
## Warning in read.table("Punt.sas7bdat"): line 1 appears to contain embedded
## nulls
```

```
## Warning in read.table("Punt.sas7bdat"): line 2 appears to contain embedded
## nulls
```

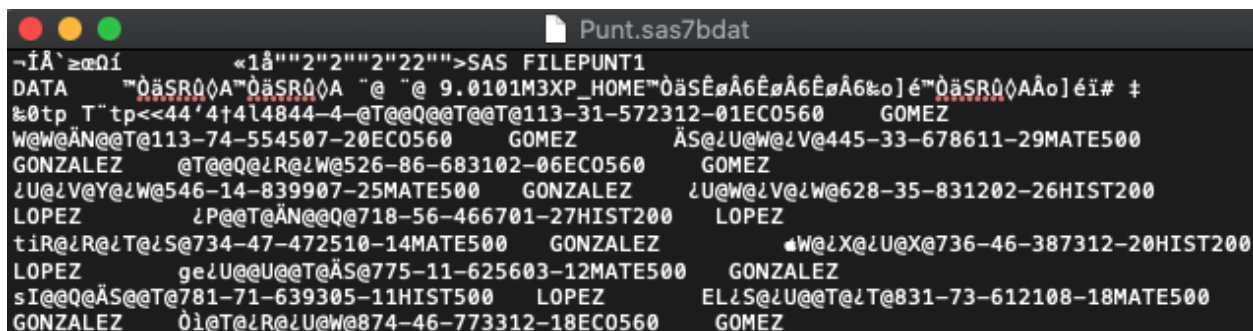
```
## Warning in read.table("Punt.sas7bdat"): line 3 appears to contain embedded
## nulls
```

```
## Warning in read.table("Punt.sas7bdat"): line 4 appears to contain embedded
## nulls
```

```
## Warning in read.table("Punt.sas7bdat"): line 5 appears to contain embedded
## nulls
```

```
## Error in scan(file = file, what = what, sep = sep, quote = quote, dec = dec, : line 2 did not have 1
```

Sin embargo, dicha función no permite leer el contenido del archivo. Esto es debido a que el fichero *Punt.sas7bdat* **NO** contiene la información en texto plano, por lo que su contenido no es legible a simple vista:



```

Punt.sas7bdat
<1â"2"2"2"22">SAS FILEPUNT1
DATA "0âSR0A"0âSR0A " @ " 9.0101M3XP_HOME"0âSÊA6ÊA6ÊA6ko]é"0âSR0AÂo]éi# +
&0tp T`tp<<44'41414844-4-@T@@Q@@T@@T@113-31-572312-01EC0560 GOMEZ
W@W@A@N@T@113-74-554507-20EC0560 GOMEZ AS@U@W@V@445-33-678611-29MATE500
GONZALEZ @T@@Q@R@W@526-86-683102-06EC0560 GOMEZ
U@U@V@W@546-14-839907-25MATE500 GONZALEZ U@W@V@W@628-35-831202-26HIST200
LOPEZ U@P@T@A@N@Q@718-56-466701-27HIST200 LOPEZ
tiR@R@T@S@734-47-472510-14MATE500 GONZALEZ W@X@U@X@736-46-387312-20HIST200
LOPEZ geU@U@T@A@775-11-625603-12MATE500 GONZALEZ
sI@Q@A@S@T@781-71-639305-11HIST500 LOPEZ ELU@U@T@T@831-73-612108-18MATE500
GONZALEZ 0i@T@R@U@W@874-46-773312-18EC0560 GOMEZ

```

Figure 6: Contenido del fichero Punt.sas7bdat

Esto último supone que las funciones base de R no nos permiten leer su contenido. Por ello, **y como único caso excepcional a lo largo de la práctica**, recurriremos a una librería externa denominada *sas7bdat* (**Click para acceder al sitio web de CRAN**) . Para instalar el paquete, empleamos la función *install.packages*. Una vez instalado, debemos importarlo por medio de la función *library* :

```
install.packages("sas7bdat", repos = "http://cran.us.r-project.org")
```

```
# Una vez instalado, lo cargamos
```

```
library(sas7bdat)
```

```
# Vemos que aparece, junto con las librerías estandar de R
```

```
(.packages())
```

```
## [1] "sas7bdat" "stats" "graphics" "grDevices" "utils" "datasets"
```

```
## [7] "methods" "base"
```

Una vez cargado el paquete, ya podemos utilizar la función *read.sas7bdat*, la cual permite leer un fichero SAS en formato binario de forma más cómoda que las librerías de R base, almacenando su contenido (por defecto en formato *DataFrame*) en una variable denominada *punt*:

```
punt <- read.sas7bdat("Punt.sas7bdat")
```

```
# Comprobamos que se trata, efectivamente, de un DataFrame
```

```
class(punt)
```

```
## [1] "data.frame"
# Una vez cargado, echamos un primer vistazo a las filas...
head(punt)

##          SEGSOC ENROLLED  COURSE  TEACHER TEST1 TEST2 TEST3 TEST4
## 1 113-31-5723    12-01  ECO560   GOMEZ    81    69    81    81
## 2 113-74-5545    07-20  ECO560   GOMEZ    92    92    61    81
## 3 445-33-6786    11-29 MATE500 GONZALEZ    78    87    92    91
## 4 526-86-6831    02-06  ECO560   GOMEZ    81    69    75    95
## 5 546-14-8399    07-25 MATE500 GONZALEZ    87    91   100    95
## 6 628-35-8312    02-26 HIST200  LOPEZ    87    92    91    95

# ... Y a las columnas ...
sapply(punt, class)
```

```
##      SEGSOC  ENROLLED   COURSE  TEACHER   TEST1   TEST2   TEST3
## "factor" "factor" "factor" "factor" "numeric" "numeric" "numeric"
##      TEST4
## "numeric"
```

Analizando las columnas, podemos comprobar cómo los campos de cadenas de caracteres (SEGSOC, ENROLLED, COURSE y TEACHER) se codifican como *factor*.

Apartado 2

Obtener una nueva variable overall que de la puntuación media de los cuatro test para cada estudiante suponiendo que el último test se pondera el doble.

Para este apartado creamos nuevamente una función, denominada **calcular_total_puntuacion**, que recibe como parámetros el DataFrame con las puntuaciones, la columna con el identificador del alumno así como un vector con las puntuaciones de cada test. Sobre dicho DataFrame aplicamos la función *cbind*, concatenando las columnas pasadas como parámetro junto con una nueva columna (OVERALL) con la puntuación media de cada alumno. Para su cálculo, mediante la función *apply* aplicamos la función *mean* sobre las n - 1 primeras columnas y, con la n-ésima columna, añade su valor multiplicado por dos:

```
calcular_total_puntuacion <- function(puntuaciones, id.alumno, columnas.test) {
  cbind(puntuaciones[c(id.alumno, columnas.test)],
        OVERALL = apply(puntuaciones[columnas.test], 1, function(x) {
          mean(c((x[1:length(x) - 1]), 2*x[length(x)]))
        }))
}
```

Una vez definida la función, realizamos la prueba, almacenando el resultado en una variable denominada *overall* :

```
overall <- calcular_total_puntuacion(punt, "SEGSOC", c("TEST1", "TEST2", "TEST3", "TEST4"))
overall
```

```
##          SEGSOC TEST1 TEST2 TEST3 TEST4 OVERALL
## 1 113-31-5723    81    69    81    81   98.25
## 2 113-74-5545    92    92    61    81  101.75
## 3 445-33-6786    78    87    92    91  109.75
## 4 526-86-6831    81    69    75    95  103.75
## 5 546-14-8399    87    91   100    95  117.00
## 6 628-35-8312    87    92    91    95  115.00
## 7 718-56-4667    67    81    61    69   86.75
## 8 734-47-4725    72    75    83    79   97.00
```

```
## 9 736-46-3873 92 99 87 96 117.50
## 10 775-11-6256 87 85 81 78 102.25
## 11 781-71-6393 50 69 78 81 89.75
## 12 831-73-6121 79 87 81 83 103.25
## 13 874-46-7733 81 75 87 92 106.75
## 14 778-83-8458 81 69 81 81 98.25
## 15 113-31-5974 81 69 81 81 98.25
## 16 017-75-4391 67 81 61 69 86.75
## 17 017-87-4637 67 81 61 69 86.75
## 18 124-29-7872 79 87 81 83 103.25
## 19 939-72-7137 79 87 81 83 103.25
```

Apartado 3

Formar una nueva variable denominada **start** compuesta por el mes y día de **ENROLLED** y por el año corriente y presenta en pantalla las variables **SEGSOC**, **COURSE** y **start**.

De forma similar al apartado 2), definimos una función denominada **anadir_columna_fecha**, que recibe como parámetros las puntuaciones en formato **DataFrame**, la columna compuesta por el día y mes (**dia.mes**), así como un vector con el resto de columnas a proyectar (en nuestro caso **SEGSOC** y **COURSE**). Sobre dicha función aplicamos nuevamente **cbind**, concatenando las columnas anteriores con una nueva, denominada **START**, formada por el año corriente (mediante la función **format(Sys.Date(), "%Y")**), así como el mes y día del parámetro **dia.mes**:

```
anadir_columna_fecha <- function(puntuaciones, dia.mes, columnas) {
  cbind(puntuaciones[columnas], START = apply(puntuaciones[dia.mes], 1, function(x) {
    paste0(format(Sys.Date(), "%Y"), "-", x)
  }))
}
```

Una vez definida la función, realizamos la prueba con las columnas **ENROLLED**, **SEGSOC** y **COURSE**, almacenando el resultado en una variable denominada **start** :

```
start <- anadir_columna_fecha(punt, "ENROLLED", c("SEGSOC", "COURSE"))
start
```

```
##          SEGSOC      COURSE      START
## 1 113-31-5723    EC0560 2020-12-01
## 2 113-74-5545    EC0560 2020-07-20
## 3 445-33-6786    MATE500 2020-11-29
## 4 526-86-6831    EC0560 2020-02-06
## 5 546-14-8399    MATE500 2020-07-25
## 6 628-35-8312    HIST200 2020-02-26
## 7 718-56-4667    HIST200 2020-01-27
## 8 734-47-4725    MATE500 2020-10-14
## 9 736-46-3873    HIST200 2020-12-20
## 10 775-11-6256    MATE500 2020-03-12
## 11 781-71-6393    HIST500 2020-05-11
## 12 831-73-6121    MATE500 2020-08-18
## 13 874-46-7733    EC0560 2020-12-18
## 14 778-83-8458    CIE500 2020-07-01
## 15 113-31-5974    FRANCE200 2020-02-01
## 16 017-75-4391    FRANCE200 2020-03-14
## 17 017-87-4637    EC0560 2020-06-14
## 18 124-29-7872    CIE500 2020-08-18
## 19 939-72-7137    CIE500 2020-09-29
```

Apartado 4

Formar un nuevo data frame de nombre `level500` que contenga los estudiantes cuyo curso acaba en 500. Crear dos nuevas variables carácter, una de nombre `subject` con el código de curso (parte literal) y otra de nombre `level` con el número del curso (parte numérica).

Para este apartado, la función diseñada (denominada `dividir_columna`) contiene como parámetros el `DataFrame` con las puntuaciones de cada alumno, el número de curso a filtrar, así como la columna donde aplicar el filtro (`COURSE`).

Inicialmente, debemos recuperar aquellas filas del `DataFrame` cuyo valor en la columna `COURSE` acabe en “500”. Para ello, mediante la función `grep` aplicamos una **expresión regular** a todas las filas del `DataFrame`, filtrando aquellas que acaben en 500 (para indicarlo mediante una expresión regular, bastaría con concatenar el valor pasado como parámetro junto con el símbolo “\$”). Una vez filtradas las filas, mediante la función `transform` añadimos las dos nuevas columnas. Para ello, mediante la función `gsub` aplicamos la expresión regular “([A-Za-z]+)([0-9]+)” a la columna `COURSE`, lo que permite extraer por un lado (`SUBJECT`) el código de curso, expresión formada por las letras del alfabeto ([A-Za-z]+); así como la columna `LEVEL`, que contiene la parte numérica del curso ([0-9]+).

Por último, para convertir ambas variables en tipo carácter (en lugar de factor o numérico), a cada nueva columna le aplicamos la función `as.character` (mediante `lapply`), reestableciendo los números de fila:

```
dividir_columna <- function(puntuaciones, codigo, columna) {
  punt.filtrado <- puntuaciones[grep(paste0(codigo,"$"), puntuaciones[, columna]), ]
  regex <- "([A-Za-z]+)([0-9]+)"
  punt.filtrado <- transform(punt.filtrado,
                             SUBJECT = gsub(regex, replacement = "\\1" ,
                                              x = punt.filtrado[, columna]),
                             LEVEL = gsub(regex, replacement = "\\2" ,
                                           x = punt.filtrado[, columna])
  )
  punt.filtrado[, c("SUBJECT", "LEVEL")] <- lapply(punt.filtrado[, c("SUBJECT", "LEVEL")],
                                                  as.character)

  rownames(punt.filtrado) <- NULL
  punt.filtrado
}
```

Una vez definida la función, creamos el nuevo `DataFrame` (`level500`), filtrando aquellos estudiantes cuyo curso acaba en 500, además de las dos nuevas variables (`SUBJECT`, `LEVEL`):

```
# Prueba dividir_columna
level500 <- dividir_columna(punt, "500", "COURSE")
level500
```

##	SEGSOC	ENROLLED	COURSE	TEACHER	TEST1	TEST2	TEST3	TEST4	SUBJECT
## 1	445-33-6786	11-29	MATE500	GONZALEZ	78	87	92	91	MATE
## 2	546-14-8399	07-25	MATE500	GONZALEZ	87	91	100	95	MATE
## 3	734-47-4725	10-14	MATE500	GONZALEZ	72	75	83	79	MATE
## 4	775-11-6256	03-12	MATE500	GONZALEZ	87	85	81	78	MATE
## 5	781-71-6393	05-11	HIST500	LOPEZ	50	69	78	81	HIST
## 6	831-73-6121	08-18	MATE500	GONZALEZ	79	87	81	83	MATE
## 7	778-83-8458	07-01	CIE500	MARTINEZ	81	69	81	81	CIE
## 8	124-29-7872	08-18	CIE500	MARTINEZ	79	87	81	83	CIE
## 9	939-72-7137	09-29	CIE500	MARTINEZ	79	87	81	83	CIE
##	LEVEL								
## 1	500								
## 2	500								
## 3	500								

```
# Comprobamos el tipo de dato de las nuevas columnas
sapply(level500, class)
```

Como podemos observar en la ejecución anterior, las columnas SUBJECT y LEVEL se codifican como *character*.

Escribe la información de level500 en un fichero ASCII de nombre “level500.dat”.

```
write.table(level500, file = "level500.dat", row.names = FALSE)
```

Figure 7: Contenido de level500.dat

La siguiente tabla representa puntuaciones de sensación de ardor para 16 sujetos en un estudio para probar un nuevo hidrogel. La primera columna da el número del sujeto. Las siguientes columnas dan la puntuación de sensación de ardor (en una escala de 1 a 4) para semanas 1 (S1) a 7 (S7). (La matriz de datos se encuentra en “matriz.R”)

```
puntuaciones.hidrogel <- structure(c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,  
    16, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 3,  
    2, 1, 1, 1, 1, 1, 1, 2, 1, 4, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1,  
    2, 3, 3, 1, 2, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 2, 3, 4, 3, 1, 2,  
    1, 1, 4, 1, 1, 4, 4, 1, 1), .Dim = c(16L, 8L))  
  
puntuaciones.hidrogel
```

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

Cabe destacar que, a diferencia de los apartados anteriores, ya no disponemos de nombres de columnas, por lo que habrá que utilizar índices numéricos.

Apartado 1

Para la semana S_7 , calcule el vector $(f_1, 1-f_1, f_2, 1-f_2, f_3, 1-f_3, f_4, 1-f_4)$, donde f_i es la frecuencia de la modalidad i (1,2,3,4) observada en la semana S_7 sobre los 16 sujetos. (Sugerencia: use las funciones `tabulate()`, `cbind()`, `t()` y `as.vector()`)

En primer lugar, de cara al apartado 2), creamos una función denominada **calcular_vector_frecuencias** que recibe como parámetro la columna directamente con sus puntuaciones, además del número total de puntuaciones (en nuestro caso 4). Inicialmente, mediante la función `tabulate` obtenemos el número de repeticiones de cada puntuación en la columna pasada como parámetro. A continuación, una vez obtenido el número de repeticiones calculamos la frecuencia **relativa**¹ de cada uno, dividiéndolo entre la longitud de la columna **puntuaciones**. Por último, mediante la función `rbind` (equivalente a `t(cbind())`) unimos los vectores de frecuencias ($f, 1 - f$), concatenando cada pareja.

Sin embargo, puede ocurrir que una puntuación no aparezca en la columna (por ejemplo, en las semanas 1 y 3 solo aparece la puntuación 1), por lo que por cada puntuación faltante añadimos las parejas (0,1), es decir, la frecuencia de aparición es 0 y por tanto, $1 - f = 1$.

```
calcular_vector_frecuencias <- function(puntuaciones, categorias) {
  frecuencias <- tabulate(puntuaciones) / length(puntuaciones)
  c(as.vector(rbind(frecuencias, 1 - frecuencias)), rep(c(0,1),
    categorias - length(frecuencias)))
}
```

Una vez creada la función, calculamos el vector para la semana S_7 . Dado que los números de fila (N_r) forman parte de la matriz como una columna más, a cada índice habrá que sumarle 1:

```
# Prueba con la semana 7 (columna 7 + 1)
calcular_vector_frecuencias(puntuaciones.hidrogel[,8], 4)
```

```
## [1] 0.500 0.500 0.125 0.875 0.125 0.875 0.250 0.750
```

Si nos fijamos en la matriz, la puntuación 1 se repite 8 veces ($f = 8/16 = 0.5$; $1 - f = 0.5$). Por otro lado, la puntuación 2 se repite en dos ocasiones ($f = 2/16 = 1/8 = 0.125$, $1 - f = 0.875$); la puntuación 3, por su parte,

¹Para el ejercicio se ha trabajado con la frecuencia **relativa** (f_i) aunque también puede emplearse la frecuencia absoluta (n_i), con la diferencia de que en lugar de restar $1 - f_i$ sería $16 - f_i$.

aparece 2 veces ($f = 2/16 = 1/8 = 0.125$, $1 - f = 0.875$), y finalmente la puntuación 4 aparece en 4 ocasiones ($f = 4/16 = 1/4 = 0.250$, $1 - f = 0.750$). Por tanto, podemos comprobar que las parejas de frecuencias (f , $1 - f$) coinciden.

Apartado 2

Ahora, use la función `apply()` para hacer el mismo cálculo para todas las demás semanas. Almacene el resultado en una matriz.

Por medio de la función `apply` aplicamos la función anterior a cada columna de la matriz, salvo la primera columna, que contiene los números de fila. Una vez aplicada la función, creamos un único vector a través de la función `unlist` con el que obtener, a continuación, una matriz de 8 filas (parejas de frecuencias (f , $1 - f$) x número de posibles puntuaciones = 2×4). Para facilitar su lectura, a cada nombre de columna le asignaremos el día de la semana, mientras que a cada fila le asignaremos la frecuencia de cada puntuación:

```
matriz.frecuencias <- matrix(unlist(apply(
  puntuaciones.hidrogel[, -1], 2, calcular_vector_frecuencias, categorias = 4)),
  nrow = 8)

rownames(matriz.frecuencias) <- c("1", "1-f1", "2", "1-f2", "3", "1-f3", "4", "1-f4")
colnames(matriz.frecuencias) <- c("S1", "S2", "S3",
                                   "S4", "S5", "S6", "S7")

# Comprobamos que se trata, efectivamente, de una matriz
class(matriz.frecuencias)
```

```
## [1] "matrix"
```

```
# Mostramos su contenido
```

```
matriz.frecuencias
```

```
##      S1      S2 S3      S4      S5      S6      S7
## 1      1 0.9375  1 0.8125 0.7500 0.6250 0.500
## 1-f1  0 0.0625  0 0.1875 0.2500 0.3750 0.500
## 2      0 0.0625  0 0.0625 0.1875 0.1250 0.125
## 1-f2  1 0.9375  1 0.9375 0.8125 0.8750 0.875
## 3      0 0.0000  0 0.1250 0.0000 0.1875 0.125
## 1-f3  1 1.0000  1 0.8750 1.0000 0.8125 0.875
## 4      0 0.0000  0 0.0000 0.0625 0.0625 0.250
## 1-f4  1 1.0000  1 1.0000 0.9375 0.9375 0.750
```

Apartado 3

Utilice la función `barplot()` y el argumento `col = c("black", "white")` en esta matriz. El gráfico que se obtiene ofrece una descripción general de la evolución de la Sensación de ardor con el tiempo.

Mediante la función `barplot`, mostramos el diagrama de barras además de incluir una leyenda en la parte derecha del gráfico (modificando el margen derecho para no solaparlo):

```
par(xpd = TRUE, mar = par()$mar + c(0,0,0,4))
barplot(matriz.frecuencias, col = c("black", "white"),
  main = "EVOLUCION SENSACION DE ARDOR CON EL TIEMPO")
legend(x= "topright", legend = c("FREC. ARDOR", "FREC. SIN ARDOR"),
  inset = c(-0.3, 0), fill = c("black", "white"),
  cex = 0.6, text.font = 2, bg = 'white')
```


EVOLUCION SENSACION DE ARDOR CON EL TIEMPO

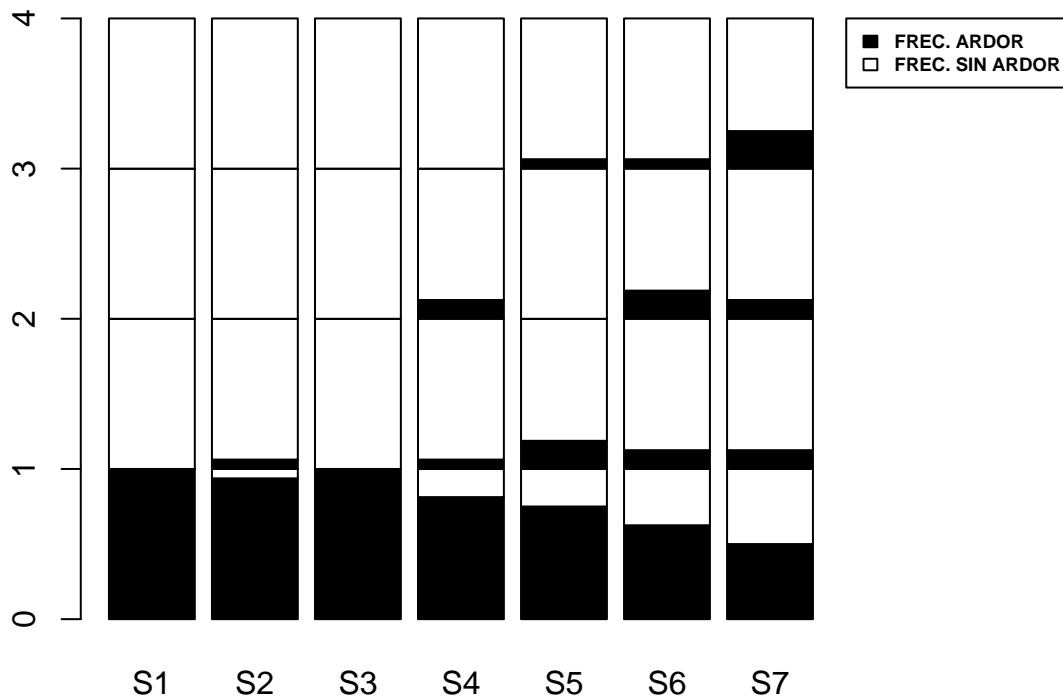


Figure 8: Gráfico de barras con la evolución de la sensación de ardor con el tiempo (I)

A través del gráfico anterior, podemos comprobar como, con el transcurso de las semanas, la sensación de ardor al emplear el hidrogel aumenta, concretamente a partir de la cuarta semana, donde aproximadamente una cuarta parte de los usuarios aumentaron su puntuación a 2 y 3.

Apartado 4

Cambie el gráfico anterior para que las barras que representan las frecuencias estén en rojo. Los números de las semanas deben estar en azul y en la parte superior del gráfico en lugar del fondo. Los números de modalidad deben estar a la izquierda, en azul. Agrega un título al gráfico.

Para resolver este apartado creamos una función denominada `mostrar_frecuencias`, que recibe como parámetros, además de la matriz, el ancho de cada conjunto de barras apiladas, así como el espacio entre cada una y el título del gráfico:

```
mostrar_frecuencias <- function(matriz, ancho, espacio, titulo) {
  par(xpd = TRUE, mar = c(0,4,5,6))
  barplot(matriz, col = c("red", "white"), width = ancho,
    space = espacio, xaxt = "n", yaxt = "n",
    ylab = "PUNTUACION", cex.lab = 1.25)

  v <- Reduce(function(v, x) v + 2 * ancho / 2 + espacio,
    x=numeric(length(colnames(matriz)) - 1),
    init=ancho / 2 + espacio,
    accumulate=TRUE)
  axis(side = 3, at = v, colnames(matriz), col.axis = "blue", font = 2, tick = FALSE)
```

```

w <- Reduce(function(w, x) w + 2 * ancho / 2,
             x=numeric(length(row.names(matriz)) / 2 - 1),
             init=ancho / 2,
             accumulate=TRUE)
axis(side = 2, at = w, row.names(matriz)[seq(1,length(row.names(matriz)),2)],
     col.axis = "blue", cex = 125, font = 2, tick = FALSE)

title(titulo, line = 4, cex.main = 1.25)
legend(x= "topright", legend = c("FREC. ARDOR", "FREC. SIN ARDOR"),
      inset = c(-0.2, 0), fill = c("red", "white"),
      cex = 0.6, text.font = 2, bg = 'white')
}

```

1. **Las barras que representen las frecuencias deben estar en rojo.** Para que las barras que indican las frecuencias estén en color rojo, al realizar la función *barplot* modificamos el vector de colores, cambiándose a ("red", "white"), además de ajustar los márgenes para poder añadir tanto los ejes como la leyenda en el gráfico:

```

par(xpd = TRUE, mar = c(0,4,5,6))
barplot(matriz, col = c("red", "white"), width = ancho,
       space = espacio, xaxt = "n", yaxt = "n",
       ylab = "PUNTUACION", cex.lab = 1.25)

```

2. **Los números de las semanas deben estar en azul y en la parte superior del gráfico en lugar del fondo.** Si nos fijamos en la función *barplot* anterior, vemos que tanto el parámetro *xaxt* como *yaxt* están a "n", lo que significa que por el momento no se muestran los ejes. Esto es debido a que mediante la función *axis* que ofrece el paquete *graphics* es posible crear ejes de una forma mucho más personalizada. Por ello, para crear el eje X utilizamos dicha función, situándolo en la parte superior (*side = 3*). Por otra parte, para denotar la posición en el eje X donde debe situarse cada número de semana, mediante la función *Reduce* calculamos la distancia de separación entre cada conjunto de barras apiladas, en base al ancho y espacio establecidos por parámetro. Finalmente, coloreamos de azul el eje mediante el parámetro *col.axis*.

```

v <- Reduce(function(v, x) v + 2 * ancho / 2 + espacio,
             x=numeric(length(colnames(matriz)) - 1),
             init=ancho / 2 + espacio,
             accumulate=TRUE)
axis(side = 3, at = v, colnames(matriz), col.axis = "blue", font = 2, tick = FALSE)

```

3. **Los números de modalidad deben estar a la izquierda, en azul. Agrega un título al gráfico.** De forma similar al eje X, mediante la función *axis* creamos el eje Y (*side = 2*). Para denotar la posición donde debe situarse cada modalidad, nuevamente mediante una función *Reduce* calculamos la distancia de separación entre cada valor (dado que en el eje Y no existe separación entre barras, sólo se tiene en cuenta el ancho). Finalmente, mediante la función *title* añadimos el título al gráfico, situándolo en la parte superior (*line = 3*), así como una leyenda en la parte superior derecha:

```

w <- Reduce(function(w, x) w + 2 * ancho / 2,
             x=numeric(length(row.names(matriz)) / 2 - 1),
             init=ancho / 2,
             accumulate=TRUE)
axis(side = 2, at = w, row.names(matriz)[seq(1,length(row.names(matriz)),2)],
     col.axis = "blue", cex = 125, font = 2, tick = FALSE)

title(titulo, line = 4, cex.main = 1.25)
legend(x= "topright", legend = c("FREC. ARDOR", "FREC. SIN ARDOR"),
      inset = c(-0.2, 0), fill = c("red", "white"),

```

```
cex = 0.6, text.font = 2, bg = 'white')
```

Si nos fijamos en la función *axis* podemos observar que, del conjunto de nombres de fila que previamente hemos asociado a la matriz:

$$(1, 1 - f_1, 2, 1 - f_2, 3, 1 - f_3, 4, 1 - f_4)$$

Sólo extraemos aquellos situados en las posiciones impares, esto es, **los números de modalidad:**

```
row.names(matriz)[seq(1,length(row.names(matriz)),2)]
```

Una vez creada la función, realizamos la prueba final, empleando como ancho de barra 1 y espacio 0.2 entre barra y barra (eje X):

```
mostrar_frecuencias(matriz.frecuencias, 1, 0.2,  
"FRECUENCIAS DE ARDOR DE HIDROGEL - SEMANAS 1 A LA 7")
```

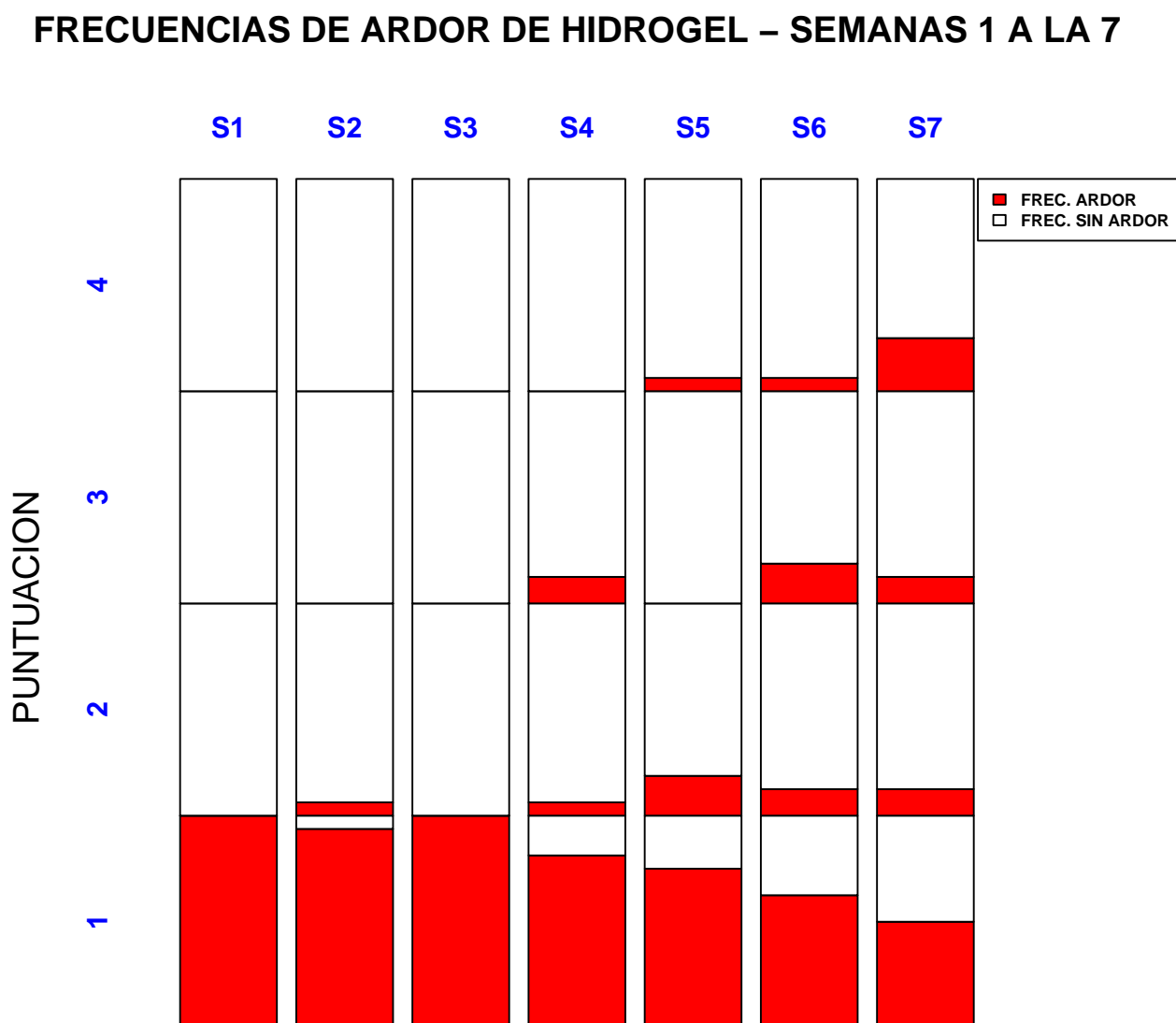


Figure 9: Gráfico de barras con la evolución de sensación de ardor con el tiempo (II)