# Data Frames

Una matriz solo puede contener elementos del mismo tipo (lógico, numérico, string, fecha). Debido a esta limitación surgen los Data Frame. Un Data Frame es una tabla donde las columnas son las variables y las filas observaciones.

Cada columna puede ser de un tipo distinto, pero dentro de cada columna solo puede haber un tipo de dato.

**Crear un DF, añadir filas y columnas.**

df <- data.frame(col1 = vector1, col2 = vector2) # Creación de un DF poniendo nombre a las columnas. Las filas no suelen tener nombre.

df <- read.csv("filename.csv", header = T) # Creamos un DF leyendo los datos desde un fichero. Header=T el DF tiene nombre en las col.

df <- rbind(df, data.frame(col1 = 22, col2 = 5, col3 = 202, col4 = 4, row.names=c("seat"))) # Añadir una fila al final del DF.

df$newcolumn <- rep(1, nrow(df)) # Añadimos una columna creando un vector de unos, por cada fila del DF.

df[, 'copyofhp'] <- df$hp # Añadimos una columna copiando otra que ya existía (hp).

df$hp.gear <- df$hp / df$gear. # Añadimos una columna que es el rdo de una operación entre otras dos columnas.

df <- cbind(df, vector) # Añadimos una columna que es un vector (importante que el nº de filas se igual en el vecto y en el DF).

nombres <- c("Columna A", "Columna B", "Columna C");

colnames(dataSet)[5:7] <- nombres # Renombrar columnas usando el vector nombres que acabamos de crear.

Rownames(dataset) <- dataset[,1] # Poner nombre a las filas

**Indexar** (ojo, si indexas una columna obtienes un vector, si indexas filas o combinaciones de celdas obtienes un DF)

df[5, 2] # Cómo indexar una celda.

df[1:5, 1:2] # Cómo indexar varias celdas.

df[1:2, c("col3", "col6")] <- 0 # Cómo asignar valores a celdas.

df[-nrow(df), ] # Selecciono todas las filas menos la última

df[(df$col5 > 150 & df$col5 < 200), ] # Obtengo as filas que, en la columna 5, son mayores de 150 y menores de 200.

df$hp # Extraemos una columna como un vector

df[, "hp"] o df[, 4] # Extraemos una columna como un vector.

df["hp"] o df[4] # Extraemos una columna como un DF.

indice\_booleano <- df$col > mean(df$col) # Obtenemos un vector de T y F.

indice\_numerico <- which(indice\_booleano) # Al usar el índice booleano obtenemos un vector con los valores de True.

**Ordenar el Data Frame**

positions <- order(df$col5, decreasing = TRUE) # Devuelve el índice con las posiciones. Ojo, no ordena el Df.

largest <- df[positions, ] # Al usar el índice, obtenemos el Df ordenado.

**Merge** (Unión de DFs mediante una key)

join <- merge(df.x, df.y, by = c("col1")) # Crea un DF con los elementos comunes de X e Y en la col1.

left.join <- merge(df.x, df.y, by = c("col1"), all.x = T) # Crea un DF con todos los elementos de X y solo los coincidentes de Y.

right.join <- merge(df.x, df.y, by = c("col1"), all.y = T) # Crea un DF con todos los elementos de Y y solo los coincidentes de X.

full.join <- merge(df.x, df.y, by = c("col1"), all = T) #Fusiona todos los elementos de X e Y. En los no coincidentes pone NA.

**Ejemplos Ejercicios**

|  |  |
| --- | --- |
| **Creación de un Data Frame**  empty <- data.frame() # Creamos uno vacío.  c1 <- 1:10 # vector de enteros  c2 <- letters[1:10] # vector de strings  df <- data.frame(col1 = c1, col2 = c2) # Creamos uno a partir de 2 vec.  df <- read.csv("filename.csv", header = T) # Lectura desde fichero  **Análisis exploratorio de un Data Frame**  Análisis exploratorio de la base de datos mtcars.  head(mtcars) # Devuelve las primeras observaciones.  head(mtcars, 10)  head(mtcars, -10) # Todas menos las 10 primeras.  tail(mtcars) # Devuelve las últimas observaciones.  tail(mtcars, 10)  tail(mtcars, -10) # Todas menos las 10 últimas.  str(mtcars) # Resumen de variables, tipo y ejemplo de datos.  summary(mtcars) # Análisis estadístico de cada variable.  **Manipulación de Data Frames**  Añade una fila al final del DF  df <- rbind(mtcars, data.frame(mpg = 22, cyl = 5, disp = 202, hp = 100, drat = 2.56, wt = 3.1, qsec = 15, vs = 1, am = 0, gear =5, carb = 4, row.names=c("seat"))) # La nueva fila debe tener las mismas variables.  Añadir columnas al final del DF  df$newcolumn <- rep(1, nrow(df)) # Creamos un vector de “unos”  df[, 'copyofhp'] <- df$hp # Creamos una copia de la columna hp  df$hp.gear <- df$hp / df$gear # Nueva colum rdo de una operación.  v <- 1:nrow(df) # Vector que va desde 1 hasta el nº de filas del DF  df <- cbind(df, v)  **Indexación de Data Frames**  Indexando celdas  df <- data.frame(mtcars)  df[5, 2] # Obtenemos una única celda  df[1:5, 1:2] # Varias celdas, filas 1 a 5, columnas 1 a 2.  df[1:2, c("gear", "am")] # filas 1 a 2, columnas por nombre.  df[1:2, c("gear", "am")] <- 0 # Asignamos un valor a las celdas  Indexando filas (devuelve data frames)  df[1, ] # Primera fila.  df[-nrow(df), ] # Todas menos la última.  df[1:5, ] # filas 1 a 5.  df[(df$hp > 150 & df$hp < 200), ] # Devuelve las filas que cumplan.  subset(df, hp > 150 & hp < 200) # Otra manera de hacerlo.  Podemos convertir el resultado de la indexación en vector.  vrow <- as.numeric(as.vector(df[1, ]))  Indexando columnas (distintas manera de hacer lo mismo).  df$hp # Devuelve una columna como un vector  df[, "hp"] # Devuelve un vector  df[, 4] # Devuelve un vector  df["hp"] # Devuelve un data frame con una columna  df[4] # Devuelve un data frame con una columna  df[["hp"]] # Devuelve un vector  df[ , c(4, 6)] # Devuelve un data frame  df[ , c("hp", "wt")] # Devuelve un data frame  **Unión de Data frames mediante una key: merge**  c1 <- 1:10; c2 <- letters[1:10]; c3 <- 5:20; c4 <- letters[5:20]  df.x <- data.frame(col1 = c1, col2 = c2)  df.y <- data.frame(col1 = c3, col2 = c4)  join <- merge(df.x, df.y, by = c("col1")) #join  left.join <- merge(df.x, df.y, by = c("col1"), all.x = T) #left join  right.join <- merge(df.x, df.y, by = c("col1"), all.y = T) #right join  full.join <- merge(df.x, df.y, by = c("col1"), all = T) #full join | **Crea a partir de los siguientes vectores un Data Frame**  planets <- c("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune")  type <- c("Terrestrial planet", "Terrestrial planet", "Terrestrial planet", "Terrestrial planet", "Gas giant", "Gas giant", "Gas giant", "Gas giant")  diameter <- c(0.382, 0.949, 1, 0.532, 11.209, 9.449, 4.007, 3.883)  rotation <- c(58.64, -243.02, 1, 1.03, 0.41, 0.43, -0.72, 0.67)  rings <- c(FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, TRUE)  planets\_df <- data.frame(planets, type, diameter, rotation, rings)  **Comprueba el contenido del Data Frame (nº variables y tipo).**  str(planets\_df)  **Selecciona la información de los tres primeros planetas.**  closest\_planets\_df <- planets\_df[1:3, ]  **Selecciona la información de los últimos tres planetas.**  furthest\_planets\_df <- planets\_df[6:8, ]  **Selecciona la columna diameter de los últimos seis planetas.**  furthest\_planets\_diameter <- planets\_df[3:8, "diameter"]  **Selecciona sólo los planetas que tienen anillos.**  planets\_with\_rings\_df <- planets\_df[planets\_df$rings==TRUE, ]  **Selecciona los planetas que tienen un diámetro inferior al de la tierra** (aquellos que tienen diametro < 1, la variable es relativa a la tierra)  indice<-planets\_df$diameter<1  small\_planets\_df <- planets\_df[indice,]  **Ordena el Data Frame según el diámetro de los planetas, ascendentemente, usando la función order.**  positions <- order(planets\_df$diameter, decreasing = FALSE)  largest\_first\_df <- planets\_df[positions, ] |

# Manipulación de datos

Editor de datos (similar a una hoja de cálculo): data.entry(x) abre un editor donde se puede modificar directamente.

Tipos de datos básicos en R: Logical, numeric, character y date.

Tipos de objetos en R: Vector, matrix, factor, data.frame y list.

¿Cómo se puede analizar la naturaleza de un objeto?

Class(x)

Attributes(x)

Str(x) # imprime un resumen.

Dim(x) o length(x) # Indica las dimensiones del data frame o longitud del vector.

Es posible convertir datos u objetos a otros tipos:

as.numeric(), as.logical(), as.character(),

as.factor(), as.vector(), as.matrix(), as.list(), as.data.frame(),

# Funciones muy útiles

sort(x) # ordena el vector.

order(x) # devuelve un índice, pero no devuelve el vector ordenado.

str\_rev(x) # Necesita library("miscset")

gsub("in", "adios", vector) # busca los in en el vector y los reemplaza por adiós.

paste("a", "b", "c", sep = ";") # concatena

strsplit(vector, ' ') # separa

grepl('in', vector) # Devuelve si la subcadena existe o no dentro de la cadena

any(x %in% c(1, 3, 5)) # ¿Alguno de estos valores están en x?

all(x %in% c(1, 3, 5)) # ¿Estan todos estos valores en el vector?

which(c(48, 50, 1, 20, -9) %in% x) # busca valores en X y devuelve sus índices.

match(26, x) # Si existe el 26 en el vector te devuelve la posición, sino, NA

# Trabajar con fechas

today <- Sys.Date()

now <- Sys.time()

d <- as.Date("2016-03-17")

d <- as.Date("17-03-2016", format = "%d-%m-%Y")

d + 1 # Añade un día

d2 <- as.Date("2015-03-17")

d - d2 # Diferencia en días

**Ejemplos Ejemplos**

|  |  |
| --- | --- |
| **Funciones muy útiles**  rep(NA, 10)  append(1:20, c(1, 2, 3))  seq(from = 5, to = 100, by = 5)  x <- c(15, 26, 5, 9, 1, -9)  sort(x)  order(x)  str\_rev(x) # Necesita library("miscset")  any(x %in% c(1, 3, 5))  all(x %in% c(1, 3, 5))  all(c(15, 9, 1) %in% x)  which(c(48, 50, 1, 20, -9) %in% x)  match(26, x)  **Funciones matemáticas**  x <- seq(from = 5, to = 100, by = 5)  is.na(x) # vector de T o F  is.finite(x) # Vector de T  is.infinite(x) # Vector de F  abs(x)  sqrt(x)  log(x)  log10(x)  exp(x)  ceiling(log(x)) # Redondeo al más cercano.  floor(log(x)) # Redondeo hacia abajo.  round(log(x), digits = 2) # Redondeo con dos dígitos.  sin(x)  cos(x)  tan(x)  sum(x)  prod(x)  cumsum(x)  cumprod(x) | **Funciones sobre cadenas**  s <- "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur."  nchar(s) # Número de caracteres.  toupper(s) # Todo mayúsculas.  tolower(s) # Todo minúsculas.  gsub("in", "adios", s) # Busca los in y los cambia por adios.  substr(s, 7, 11) # Devuelve una subcadena, inicio – fin.  substr(s, 7, 11) <- "IPSUM" # Modifica una subcadena.  paste("a", "b", "c", sep = ";")  strsplit(s, ' ') # Separa la cadena por espacios en blanco.  grepl('dolor', s) # Devuelve si la subcadena existe o no dentro de la cadena  library(stringr)  str\_locate\_all(pattern ='dolor', s) # Devuelve posiciones de la subcadena  **Funciones sobre fechas**  today <- Sys.Date()  class(today)  now <- Sys.time()  class(now)  d <- as.Date("2016-03-17")  d <- as.Date("17-03-2016") # No da error pero lo hace mal  d <- as.Date("17-03-2016", format = "%d-%m-%Y")  t <- as.POSIXct("2016-03-17 22:32:00")  t <- as.POSIXct("17-03-2016 22:32:00", format = "%d-%m-%Y %H:%M:%S")  **Cálculos con fechas**  d + 1 # Añade un día  d2 <- as.Date("2015-03-17")  d - d2 #Diferencia en días.  t + 1 #Añade un segundo  t2 <- as.POSIXct("2015-03-17 22:32:00")  t - t2 #Diferencia en días.  unclass(d) # Da un entero: número de días desde el 1 Enero 1970.  unclass(t) # Da un entero: número de segundos desde el 1 Enero 1970. |

# Cómo importar y guardar datos

**Importar datos**

Parámetro enconding: utf-8 (para que lea bien los acentos y la “ñ”).

Utils es la librería estándar de R, pero existen otras librerías que nos permiten importar ficheros.

|  |  |  |
| --- | --- | --- |
|  | **Utils** | **xlsx** |
| txt | read.table() |  |
| csv | read.csv() |  |
| tsv | read.delim() |  |
| Excel |  | read.xlsx |

Para poder importar, lo primero que tenemos que hacer es establecer el directorio de trabajo. Establece, por ejemplo, el escritorio.

|  |  |
| --- | --- |
|  |  |

mun\_csv\_1 <- read.table("dat/municipios1.csv",

header = TRUE,

sep = ",", # sep = "\t", #El separador es el tabulador

stringsAsFactors = FALSE, # No deseamos convertir el resultado en un factor (poner siempre esto).

dec = ",", #el separador decimal es la coma

quote = "") #Especificamos las cadenas de caracteres

mun\_csv\_2 <- read.csv("dat/municipios1.csv", stringsAsFactors = FALSE)

mun\_tsv\_2 <- read.delim("dat/municipios2.tsv", stringsAsFactors = FALSE) # Lee TSV separados por tab.

library(xlsx)

read.xlsx(paste(getwd(),"/warrants.xls", sep = ""), sheetName="Warrants", startRow=6, endRow=NULL, , as.data.frame=TRUE, header=TRUE, keepFormulas=FALSE, encoding="UTF-8")

**Guardar datos**

Write.table guarda el obj en un archivo (suele ser un data.frame, pero puede ser de cualquier tipo).

write.table(x, file=” “). x es el nombre del objeto a guardar (vector, matriz…). Append True anexa los datos sin borrar los existentes.

write.table(MyData, file = "MyData.csv", row.names=FALSE, na="", col.names=FALSE, sep=",")

write.csv(MyData, file = "MyData.csv",row.names=FALSE, na="")

write.xlsx(DF\_a\_guardar, "nombre\_fichero.xlsx") # Guardamos un excel.

Save(x, y, z, file =”xyz.RData”) # Guardamos las variables X, Y y Z. Se pueden cargar con load(“xyz.RData”).

|  |  |  |
| --- | --- | --- |
| **CSV** | **TSV** | **Excel** |
|  |  |  |

# Limpieza de datos

**Exploración de los datos**

vinos <- read.csv("wine.csv",header = T)

|  |  |  |  |
| --- | --- | --- | --- |
| class(vinos)  dim(vinos)  names(vinos)  str(vinos)  summary(vinos) | head(vinos)  tail(vinos)  print(vinos) | hist(vinos$alcohol)  plot(vinos$alcohol, vinos$proline) | **CSV** |

**Preparación**

library(stringr)

str\_trim(" this is a test ") # Elimina los espacios en blanco al principio y al final de la frase.

str\_pad("244493", width = 7, side = "left", pad = "0") # Lo transforma en un nº de 7 dígitos añadiendo un 0 a la izq.

is.na(df) # Busca los na del dataset y devuelve matriz de T y F.

any(is.na(df)) # si hay alguno devuelve T

sum(is.na(df)) # Cuenta los na

df[complete.cases(df), ] # Elimina las filas que tienen Na. Otra manera de hacerlo es na.omit(df)

**Ejemplos**

|  |  |
| --- | --- |
| vinos <- read.csv("dat/wine.csv", header = T) # Cargamos el dataset  **Exploración de la estructura**  class(vinos)  dim(vinos)  names(vinos)  str(vinos)  summary(vinos)  **Exploración de los datos**  head(vinos)  tail(vinos)  print(vinos)  **Visualizando los datos**  hist(vinos$alcohol)  plot(vinos$alcohol, vinos$proline)  **Limpieza: preparación**  library(lubridate) # Trabajando con formatos de fecha.  ymd("2015-08-25")  ymd("2015 August 25")  mdy("August 25, 2015")  hms("14:17:07")  ymd\_hms("2015/08/25 13.33.09")  library(stringr) # Trabajando con string  str\_trim(" this is a test ")  str\_pad("244493", width = 7, side = "left", pad = "0")  names <- c("Sarah", "Tom", "Alice")  str\_detect(names, "Alice")  str\_replace(names, "Alice", "David") | **NAs**  df <- data.frame(A = c(1, NA, 8), B = c(3, NA, 88), C = c(2, 45, 3))  Detección  is.na(df) #Da una matriz de T o F si tiene na.  any(is.na(df))  sum(is.na(df))  summary(df)  Eliminación  df[complete.cases(df), ] # elimina las filas con na.  na.omit(df) # elimina las filas con na.  **Outliers**  set.seed(10) # Inicializamos la semilla para generar valores aleatorios  x <- c(rnorm(30, mean = 15, sd = 5), -5, 28, 35)  # Generamos 30 aleatorios y añade -5, 28…  boxplot(x, horizontal = T) # Lo graficamos para ver qué pinta tiene.  # Muestra una “caja” con el 1er y 3er cuartil, y la mediana  # Las mechas son 1,5 veces Q3 – Q1  # Los atípicos están fuera de las mechas  df2 <- data.frame(A = rnorm(100, 50, 100), B = c(rnorm(99, 50, 10), 500), C = c(rnorm(99, 50, 10), -1))  hist(df2$B, 20)  boxplot(df2) |

# Dplyr

Es un paquete de manipulación de datos, creado por un usuario al que no le convencían las funciones de Dataframe.

Crea un nuevo objeto al que llama tabla (similar a un Data Frame): Permite una mejor visualización del contenido.

El paquete permite trabajar con los objetos Data Frame y tabla.

Para poder trabajar con Dplyr, primero debes instalarlo (packages/install) y luego invocarlo: library(dplyr)

tabla <- tbl\_df(datos)# Convierte los datos a una tabla.

glimpse(tabla) # El equivalente a str.

**Selección**

hflights.select <- select(tabla, col1, col2, col7, col32)#1er parametro es tabla y el resto son las columnas seleccionadas.

select(hflights.tbl, colum6:colum12) # Selecciona desde la columna 6 a la 12.

select(hflights.tbl, -colum3:-colum5) # Selecciono todas las variables menos aquellas entre las columnas 3 y 5.

select(hflights.tbl, starts\_with ("Delay")) # Selecciono las variables que empiezan con "Delay".

select(hflights.tbl, ends\_with("Time"), ends\_with("Delay")) # Selecciono las variables que terminan en "Time" o "Delay"

**Creación de columnas**

seleccion <- mutate(hflights.select, loss = ArrDelay - DepDelay)# creamos una columna usando los datos de otras dos.

m1 <- mutate(hflights.tbl, loss = ArrDelay - DepDelay, loss\_percent = loss / DepDelay \* 100) # Creamos dos columnas.

hflights.tbl$UniqueCarrier <- recode.carrier[hflights.tbl$UniqueCarrier] #Estamos sustituyendo los datos de la columna UniqueCarrier por los datos un de vector llamado recode.carrier.

**Filtramos**

hflights.filter <- filter(hflights.select, Cancelled == 1)#Seleccionamos las filas donde la columna Cancelled sean = 1

filter(hflights.tbl, Distance >= 1000) # Filtra los vuelos cuya distancia recorrida es mayor o igual a 1000.

filter(hflights.tbl, UniqueCarrier %in% c("JetBlue", "Southwest", "Delta")) # Filtra la columna UniqueCarrier, buscando los valores que sean JetBlue, Southwest o Delta.

filter(hflights.tbl, DepTime < 500 | ArrTime > 2200) # Filtra los vuelos que salieron antes de las 5am o llegaron después de las 10pm.

filter(hflights.tbl, Cancelled == 1 & DayOfWeek %in% c(6, 7)) # Filtra los vueltos cancelados en fin de semana.

**Ordenar**

hflights.arrange <- arrange(hflights.select, DepDelay) # ordenamos por la columna DepDelay.

hflights.arrange <- arrange(hflights.select, DepDelay, ArrDelay)# ordenamos por dos columnas (de izq a der).

dtc <- filter(hflights.tbl, Cancelled == 1, !is.na(DepDelay)) ; arrange(dtc, DepDelay)

# Ordenamos por las columnas dtc y depdelay. Filtramos las filas que son = 1 de la columna Cancelled y quitamos las filas que contienen Na de la columna DepDelay.

**Summarise** (permite usar funciones de agregación de R)

summarise(hflights.select, min = min(DepDelay), max = max(DepDelay), mean = mean(DepDelay), median = median(DepDelay)) # Obtengo los estadísticos usando la selección anterior.

summarise(filter(hflights.tbl, Diverted == 1), max\_div = max(Distance)) # En una sola instrucción, filtramos los vuelos desviados y calculamos la distancia máxima recorrida.

Summarise permite usar las siguientes funciones: min(x), max(x), mean(x), median(x), quantile(x), sd(x) desviación estándar, var(x), first(x), last(x), nth(x,n) es el enésimo elemento del vector x, n() nº de filas del DF o tabla, n\_distinct(x) nº de valores únicos de x.

**Agrupar filas Group by**

hflights.group <- group\_by(hflights.tbl, UniqueCarrier) # Agrupamos valores (filas) por la columna UniqueCarrier.

**Pipes**

Encadena llamadas a funciones, enviando la salida a la siguiente línea.

Ahorra espacio al no ser necesario declarar variables intermedias y facilita la lectura del código.

|  |  |
| --- | --- |
| Con programación normal  Summarise(  Mutate(  Filter(  Select(tabla, x, y, z),  x > y),  Q = x + y + z),  All = sum(Q)) | Con pipes  Tabla %>% # Indico con qué tabla trabajamos.  Select(x, y, z) %>% # Selecciono las columnas.  Filter (x > y) %>% # Filtro los datos que me interesan.  Mutate(Q = x + y + z) %>% # Creo una nueva variable.  Summarise (all = sum(Q)) # Extraigo la suma. |

hflights.tbl %>% # Selecciono la tabla.

filter(!is.na(DepDelay)) %>% # Elimino los datos con NA (valor no disponible).

summarise(min = min(DepDelay), max = max(DepDelay), mean = mean(DepDelay), median = median(DepDelay))

hflights.tbl %>% # Selecciono la tabla.

group\_by(UniqueCarrier) %>% # Agrupo la información por compañías.

summarise(n\_flights = n(), # Obtendo el nº de vuelos.

n\_canc = sum(Cancelled), # Calculo el nº de vuelos cancelados.

p\_canc = mean(Cancelled) \* 100, # Calculo el % de vuelos cancelados.

avg\_delay = mean(ArrDelay, na.rm = TRUE)) %>% # Calculo el retraso medio de llegada quitando los NA.

arrange(avg\_delay, p\_canc) # Ordena los resultados por el retraso medio y el porcentaje de vuelos cancelados

hflights.tbl %>%

filter(!(is.na(ArrDelay))) %>% # Elimino los NA

group\_by(UniqueCarrier) %>% # Agrupo por compañía

summarise(p\_delay = mean(ArrDelay > 0)) %>% #Calculo el retraso medio (pero solo de los vuelos con retraso).

mutate(rank = rank(p\_delay)) %>% # creo una variable nueva que es un ranking de los vuelos con retraso.

arrange(rank) # ordeno el ranking.

**Ejemplos Ejercicios**

|  |  |
| --- | --- |
| **dplyr**  Instala el paquete hflights y dplyr  library(dplyr) # Invocamos a la librería ya instalada.  library(hflights) # Información del aeropuerto de Houston.  hflights.tbl <- tbl\_df(hflights) # Convertimos los datos en una table.  hflights.tbl # Muestra la información de una manera amigable.  glimpse(hflights.tbl) # Muestra un resumen de la info.  **Select (selección de datos)**  hflights.select <- select(hflights.tbl, ActualElapsedTime, AirTime, ArrDelay, DepDelay) # El primer parámetro es la tabla y el resto las columnas seleccionadas.  **Mutate (creación de variables)**  hflights.mutate <- mutate(hflights.select, loss = ArrDelay - DepDelay) # Usando el resultado del select, creamos una nueva variable “loss”, restando dos columnas (ArrDelay y DepDelay)  **Filter (selección de datos)**  hflights.select <- select(hflights.tbl, starts\_with("Cancel"), DepDelay) # Seleccionamos las columas cuyo nombre empiece por “Cancel” y la columna DepDelay.  hflights.filter <- filter(hflights.select, Cancelled == 1) # Seleccionamos las filas que sean = 1 de la columna Cancelled.  **Arrange (ordenación de datos)**  hflights.select <- select(hflights.tbl, TailNum, contains("Delay")) # Seleccionamos las filas que contengan “Delay” de la columna TailNum.  hflights.arrange <- arrange(hflights.select, DepDelay) # Ordena la selección por la columna DepDelay.  hflights.arrange <- arrange(hflights.select, DepDelay, ArrDelay) # Ordena por dos columnas (izq 1º).  **Summarise (estadísticos)**  hflights.summarise <- summarise(hflights.select, min = min(DepDelay), max = max(DepDelay), mean = mean(DepDelay), median = median(DepDelay)) # Obtengo los estadísticos de la selección.  **Group by (agrupar filas)**  hflights.group <- group\_by(hflights.tbl, UniqueCarrier) # Agrupa las filas por la columna UniqueCarrier.  hflights.summarise.group <- summarise(hflights.group, avgDep = mean(DepDelay, na.rm = T), avgArr = mean(ArrDelay, na.rm = T)) # Sacamos los estadísticos de la agrupación, quitando los “NA”.  **Pipes (encadena llamadas a funciones)**  Pasa el resultado a la siguiente fila / función.  hflights.tbl %>%  filter(!is.na(DepDelay)) %>%  summarise(min = min(DepDelay), max = max(DepDelay), mean = mean(DepDelay), median = median(DepDelay))  Indicamos la tabla con la que trabajamos.  Realizamos un filtro eliminando los Na.  Sacamos los estadísticos. | **Carga la librería de vuelos del aeropuerto de Houston, conviérte la información en una tabla y guárdala en una variable.**  library(hflights)  library(dplyr)  hflights.tbl <- tbl\_df(hflights)  **Recodifica la variable UniqueCarrier a partir del vector recode.carrier**  recode.carrier <- c("AA" = "American", "AS" = "Alaska", "B6" = "JetBlue", "CO" = "Continental", "DL" = "Delta", "OO" = "SkyWest", "UA" = "United", "US" = "US\_Airways", "WN" = "Southwest", "EV" = "Atlantic\_Southeast", "F9" = "Frontier", "FL" = "AirTran", "MQ" = "American\_Eagle", "XE" = "ExpressJet", "YV" = "Mesa")  hflights.tbl$UniqueCarrier <- recode.carrier[hflights.tbl$UniqueCarrier]  **Recodifica la variable CancellationCode a partir del vector recode.cancellation**  hflights.tbl[hflights.tbl$CancellationCode == "", "CancellationCode"] <- "E"  recode.cancellation <- c("A" = "carrier", "B" = "weather", "C" = "FFA", "D" = "security", "E" = "not cancelled")  hflights.tbl$CancellationCode <- recode.cancellation[hflights.tbl$CancellationCode]  **Selecciona las variables: ActualElapsedTime, AirTime, ArrDelay, DepDelay**  select(hflights.tbl, ActualElapsedTime, AirTime, ArrDelay, DepDelay)  **Selecciona las variables desde Origin a Cancelled**  select(hflights.tbl, Origin:Cancelled)  **Selecciona todas las variables menos aquellas entre DepTime y AirTime**  select(hflights.tbl, -DepTime:-AirTime)  **Selecciona todas las variables que terminan en "Delay"**  select(hflights.tbl, ends\_with("Delay"))  **Selecciona las variables que terminan en "Num" o empiezan por "Cancell"**  select(hflights.tbl, ends\_with("Num"), starts\_with("Cancell"))  **Selecciona las que terminan en "Num" y la variable UniqueCarrier**  select(hflights.tbl, UniqueCarrier, ends\_with("Num"))  **Crea la variable ActualGroundTime como la resta de ActualElapsedTime y AirTime. Guarda el resultado en g1.**  g1 <- mutate(hflights.tbl, ActualGroundTime = ActualElapsedTime - AirTime)  **Añade a g1 una nueva variable GroundTime como la suma TaxiIn y TaxiOut**  g2 <- mutate(g1, GroundTime = TaxiIn + TaxiOut)  **Crea dos variables a la vez y guardalas en m1:**  loss como la resta de ArrDelay y DepDelay  loss\_percent como el ratio de ArrDelay - DepDelay entre DepDelay  m1 <- mutate(hflights.tbl, loss = ArrDelay - DepDelay, loss\_percent = (ArrDelay - DepDelay) / DepDelay \* 100)  **Filtra los vuelos con distancia recorrida mayor o igual que 3000**  filter(hflights.tbl, Distance >= 3000)  **Filtra los vuelos de JetBlue, Southwest, o Delta**  filter(hflights.tbl, UniqueCarrier %in% c("JetBlue", "Southwest", "Delta"))  **Filtra los vuelos cuyo tiempo en Taxi fue mayor que el tiempo de vuelo.**  filter(hflights.tbl, TaxiOut + TaxiIn > AirTime)  **Filtra los vuelos que salieron con retraso y llegaron a tiempo**  filter(hflights.tbl, DepDelay > 0 & ArrDelay < 0)  **Filtra los vueltos cancelados después de haber sido retrasados**  filter(hflights.tbl, Cancelled == 1, DepDelay > 0)  **Ordena la tabla dtc según el retraso de salida**  dtc <- filter(hflights.tbl, Cancelled == 1, !is.na(DepDelay))  arrange(dtc, DepDelay)  **Ordena los vuelos según compañía y retraso de salida descendentemente**  arrange(hflights.tbl, UniqueCarrier, desc(DepDelay))  **Ordena los vuelos según el retraso total**  arrange(hflights.tbl, ArrDelay + DepDelay)  **Filtra los vuelos con destino DFW y hora de salida anterior a las 8am, ordenalos según el tiempo en vuelo descendentemente.**  arrange(filter(hflights.tbl, (Dest == "DFW") & (DepTime < 800)), desc(AirTime))  **Calcula la distancia máxima de los vuelos desviados**  summarise(filter(hflights.tbl, Diverted == 1), max\_div = max(Distance))  **Crea en una variable con los vuelos que no tienen NA en TaxiIn y TaxiOut Calcula un estadístico (max\_taxi\_diff) que contenga la mayor diferencia en valor absoluto entre TaxiIn y TaxiOut**  temp2 <- filter(hflights.tbl, (!is.na(TaxiIn)) & (!is.na(TaxiOut)))  summarise(temp2, max\_taxi\_diff = max(abs(TaxiIn - TaxiOut)))  **Usando pipes a partir de este punto, realiza:**  **1. Crea una variable diff como la diferencia de TaxiIn y TaxiOut**  **2. Filtra aquellas filas para las que diff no es NA**  **3. Calcula la media**  hflights.tbl %>%  mutate(diff = TaxiOut - TaxiIn) %>%  filter(!is.na(diff)) %>%  summarise(avg = mean(diff))  **Usando los pipes calcula cuantos vuelos nocturos hay. Los vuelos nocturnos son aquellos para los que la hora de llegada es menor que la de salida.**  hflights.tbl %>%  filter(!is.na(DepTime) & !is.na(ArrTime) & ArrTime < DepTime) %>%  summarise(n = n())  **Usando los pipes calcula, para cada compañía los siguientes estadísticos:**  **n\_flights: número de vuelos**  **n\_canc: número de vuelos cancelados**  **p\_canc: porcentaje de vuelos cancelados**  **avg\_delay: retraso medio de llegada (cuidado con los NAs)**  **Ordena los resultados por el retraso medio y % de vuelos cancelados**  hflights.tbl %>%  group\_by(UniqueCarrier) %>%  summarise(n\_flights = n(),  n\_canc = sum(Cancelled),  p\_canc = mean(Cancelled) \* 100,  avg\_delay = mean(ArrDelay, na.rm = TRUE)) %>%  arrange(avg\_delay, p\_canc)  **Calcula qué avión (TailNum) voló más veces. ¿Cuántas?**  hflights.tbl %>%  group\_by(TailNum) %>%  summarise(n = n()) %>%  filter(n == max(n))  **¿Cuántos aviones volaron a un único destino?**  hflights.tbl %>%  group\_by(TailNum) %>%  summarise(ndest = n\_distinct(Dest)) %>%  filter(ndest == 1) %>%  summarise(nplanes = n())  **¿Cuál es el destino más visitado de cada compañía?**  hflights.tbl %>%  group\_by(UniqueCarrier, Dest) %>%  summarise(n = n()) %>%  mutate(rank = rank(desc(n))) %>%  filter(rank == 1)  **¿Qué compañías viaja más a cada destino?**  hflights.tbl %>%  group\_by(Dest, UniqueCarrier) %>%  summarise(n = n()) %>%  mutate(rank = rank(desc(n))) %>%  filter(rank == 1) |
|  |

# Apply

La familia apply realiza una operación a todos los elementos de un vector o lista. Es como un for que aplica una función.

Recorre los elementos de un vector o lista, les aplica una función y devuelve el resultado en un vector.

apply(matriz, 1, mean) # Hace la media por filas.

apply(matriz, 2, mean) # Hace la media por columnas.

apply(matriz, 1:2, function(x) x/2) # Divide entre 2 cada valor (usando las filas y columnas).

# Lapply

Recorre los elementos de un vector o lista, les aplica una función y devuelve el resultado en una lista.

|  |  |
| --- | --- |
| **Usando un bucle**  for (elemento in vector) {  print(class(elemento))  }  num\_chars <- c() #vector en blanco.  for (i in 1:length(cities)) { #recorre los elementos  num\_chars[i] <- nchar(cities[i]) #nº caract del elemento  } | **Usando lapply**  lapply(vector, class) #hace lo mismo que el bucle de la izq, pero más eficiente.  lapply(cities, nchar) # Hace lo mismo que el de la izq. Siempre devuelve una lista.  unlist(lapply(cities, nchar)) #Con unlist, obtenemos un vector. |

**Ejemplos Ejercicios**

|  |  |  |  |
| --- | --- | --- | --- |
| nyc <- list(pop = 8405837, boroughs = c("Manhattan", "Bronx", "Brooklyn", "Queens", "Staten Island"), capital = FALSE)   |  |  | | --- | --- | | Usando bucle  for (info in nyc) {  print(class(info))  } | Usando lapply  lapply(nyc, class) |   cities <- c("New York", "Paris", "London", "Tokyo", "Rio de Janeiro", "Cape Town")  num\_chars <- c()  for (i in 1:length(cities)) {  num\_chars[i] <- nchar(cities[i])  }  lapply(cities, nchar) #Ojo, siempre devuelve una lista  unlist(lapply(cities, nchar)) #Con unlist, pasamos a vector  **Con funciones propias**  oil\_prices <- list(2.37, 2.49, 2.18, 2.22, 2.47, 2.32)  triple <- function(x) {  x \* 3  }  unlist(lapply(oil\_prices, triple))  **Con funciones propias pasando argumentos**  oil\_prices <- list(2.37, 2.49, 2.18, 2.22, 2.47, 2.32)  multiply <- function(x, factor) {  x \* factor  }  unlist(lapply(oil\_prices, multiply, factor = 3)) | Vector con listado de nombres de matemáticos y sus años de nacimiento.  pioneers <- c("GAUSS:1777", "BAYES:1702", "PASCAL:1623", "PEARSON:1857")  **Separa los nombres y años utilizando la función strsplit.**  split\_math <- strsplit(pioneers, ":")  **Aplica la función tolower a todos los elementos de split\_math.**  split\_low <- lapply(split\_math, tolower)  **Escribe una función que devuelva el primer elemento de un vector.**  select\_first <- function(x) {  x[1]  }  **Aplica la función select\_first a split\_low**  names <- lapply(split\_low, select\_first)  **Escribe una función que devuelva el segundo elemento de un vector.**  select\_second <- function(x) {  x[2]  }  **Aplica la función select\_second a split\_low**  years <- lapply(split\_low, select\_second) |

# Sapply

Devuelve un vector o matriz con los rdos (muy parecido al Apply original).

**Ejemplos Ejercicios**

|  |  |
| --- | --- |
| cities <- c("New York", "Paris", "London", "Tokyo", "Rio de Janeiro", "Cape Town")  sapply(cities, nchar) # Obtenemos un vector  first\_and\_last <- function(name) {  name <- gsub(" ", "", name) # Elimina los espacios en blanco  letters <- strsplit(name, split = "")[[1]] #Separa ciudades en vector.  c(first = min(letters), last = max(letters))  } #Devuelve la letra “menor” y la “mayor” de la ciudad.  sapply(cities, first\_and\_last) # Obtenemos una matriz | Muestras de temperatura de cada día de la semana.  temp <- list(monday = c(3, 7, 9, 6, -1), tuesday = c(6, 9, 12, 13, 5), wednesday = c(4, 8, 3, -1, -3), thursday = c(1, 4, 7, 2, -2), friday = c(5, 7, 9, 4, 2), saturday = c(-3, 5, 8, 9, 4), sunday = c(3, 6, 9, 4, 1))  **Utiliza lapply para encontrar la temperatura mínima de cada día.**  lapply(temp, min)  **Utiliza sapply para encontrar la temperatura mínima de cada día.**  sapply(temp, min)  **Crea una función que calcule la media entre el día mínimo y el máximo.**  extremes\_avg <- function(x) {  mean(c(max(x), min(x)))  }  **Aplica la nueva función utilizando sapply.**  sapply(temp, extremes\_avg)  **Crea una función extremes que devuelva una matriz con:**  **1. La temperatura mínima.**  **2. La temperatura máxima.**  extremes <- function(x) {  c(min = min(x), max = max(x))  }  **Aplica la nueva función utilizando sapply.**  sapply(temp, extremes)  **Crea una función que devuelva las muestras menores que cero.**  below\_zero <- function(x) {  x[x < 0]  }  **Aplica la nueva función utilizando sapply y guardarla en freezing\_s**  freezing\_s <- sapply(temp, below\_zero) |

# Vapply

Controla el tipo de dato de retorno (el resto es igual que Sapply).

**Ejemplos Ejercicios**

|  |  |
| --- | --- |
| cities <- c("New York", "Paris", "London", "Tokyo", "Rio de Janeiro", "Cape Town")  vapply(cities, nchar, numeric(1)) # El dato de retorno es numérico.  first\_and\_last <- function(name) {  name <- gsub(" ", "", name)  letters <- strsplit(name, split = "")[[1]]  c(first = min(letters), last = max(letters))  }  vapply(cities, first\_and\_last, character(2)) # # El 2 es xq devuelve una matriz de 2 filas.  vapply(cities, first\_and\_last, character(1)) # Esto daría un error, ya que estamos devolviendo un vector de longitud 2. | Muestras de temperatura de cada día de la semana.  temp <- list(monday = c(3, 7, 9, 6, -1), tuesday = c(6, 9, 12, 13, 5), wednesday = c(4, 8, 3, -1, -3), thursday = c(1, 4, 7, 2, -2), friday = c(5, 7, 9, 4, 2), saturday = c(-3, 5, 8, 9, 4), sunday = c(3, 6, 9, 4, 1))  **Crea la función basics que devuelve un vector con nombres con:**  **1. min: temperatura mínima**  **2. mean: temperatura media**  **3. max: temperatura máxima**  basics <- function(x) {  c(min = min(x), mean = mean(x), max = max(x))  }  **Usando vapply aplica la función basics.**  vapply(temp, basics, numeric(3))  **Con esta nueva definición de la función basics.**  basics <- function(x) {  c(min = min(x), mean = mean(x), median = median(x), max = max(x))  }  **Arregla el error:**  vapply(temp, basics, numeric(3))  vapply(temp, basics, numeric(4))  **Convierte este sapply en un vapply.**  sapply(temp, max)  vapply(temp, max, numeric(1))  **Convierte este sapply en un vapply.**  sapply(temp, function(x, y) { mean(x) > y }, y = 5)  vapply(temp, function(x, y) { mean(x) > y }, y = 5, logical(1))  **Con esta función get\_info, convierte el sapply en un vapply.**  get\_info <- function(x, y) {  if (mean(x) > y) {  return("Not too cold!")  } else {  return("Pretty cold!")  }  }  sapply(temp, get\_info, y = 5)  vapply(temp, get\_info, y = 5, character(1)) |

# Data table

Paquete de manipulación de datos. Es una versión mejorada de los data frames.

Su objetivo es reducir el tiempo de programación y procesamiento.

Para poder trabajar con DT, primero debes instalarlo (packages/install) y luego invocarlo: library(data.table)

iris.dt <- as.data.table(iris) # Convierto un data frame en un data table. Es a la vez un DF y DT (acepta ambas instrucciones).

Funcionamiento DT[selección de filas, selección de columnas, by agrupación de filas] en SQL [where, select, group by].

**Conversión a Data Table**

DT <- data.table(x = c("a", "b", "c", "d", "e"), y = c(1, 2, 3, 4, 5)) # Creación de un data.table

DT <- as.data.table(DF) # Convertimos un Data Frame en un Data Table.

**Selección de filas**

DT[3] # Seleccionamos la tercera fila, devuelve un data.table.

DT[[3]] # Seleccionamos la tercera fila, devuelve un vector.

DT[2:3] o bien DT[2:3,] # Seleccionamos la segunda y tercera fila.

DT[.N - 1] # Selecciona la penúltima fila

DT[c(2, 2, 3)] # Selecciona la segunda fila dos veces y la tercera

iris.dt[base \* altura > 10] # Filtra las filas para las que el área es mayor que 10

iris.dt[Species %in% c("virginica", "versicolor")] # Filtra las filas de la especie virginica y versicolor

**Selección de columnas**

DT[, .(B)] #Devuelve un data.table con la columna B.

DT[, B] #Devuelve un vector

DT[c(1, 3), .(B, C)] # Selección de las filas 1 y 3 y columnas B y C.

ans <- DT[, .(B, val = A \* C)] # Selección de la columna B y creación de una columna val como la multiplicación de A por C.

**Agrupando con By**

DT[, .(MySum = sum(A), MyMean = mean(A)), by = .(B)] # Agrupa por la col B, hace la suma y media de los valores de la col A.

DT[, .(MySum = sum(C)), by = .(Grp = C%%2)] # Hace la suma de la columna C, agrupándola por par e impar.

**Encadenando llamadas**

DT[, sum(B), by = A][order(A)] # Agrupando por la columna A, suma la columna B y ordena los rdos por la columna A.

DT[, .(C = sum(C)), by = .(A, B)][, .(C = tail(C, 2)), by = A]

# Agrupando por las columnas A y B, suma los valores de C; y muestra los dos últimos valores de C ordenados por la col A.

**.SD Extrae subconjunto de datos**

DT[, lapply(.SD, mean), by = A] # Hace la media de todas las columnas agrupando por los valores de la columna A.

DT[, lapply(.SD, sum), .SDcols = 2:4] # Hace la suma de las columnas 2 a 4.

DT[, lapply(.SD, sum), by = x, .SDcols = c("x", "y", "z")] # Agrupando por x, suma las columnas (x, y, z).

DT[, lapply(.SD, cumsum), by = .(by1 = x, by2 = z > 8), .SDcols = c("x", "y")]

# Calcula la suma acumulada de las columnas X e Y, agrupando por los valores de X y los valores de Z > 8.

**:= crea, modifica o elimina el contenido de una columna**

DT[, Total := sum(B) , by = A] # Añade una nueva columna “Total” agregando por los valores de A.

DT[c(2, 4), B := B + 1] # Añade 1 a la columna B en las filas 2 y 4.

DT[, Total := NULL] # Elimina columna Total.

DT[, 2 := NULL] # Borra la segunda columna.

DT[, `:=`(B = B + 1, C = A + B, D = 2)] # Añade 1 a la columna B y crea las columnas C y D.

**Key (crea índices para optimizer filtrados)**

setkey(DT, A, B) # Creación del índice por las columnas A y B.

**Ejemplos Ejercicios**

|  |  |
| --- | --- |
| library(data.table)  **Agrupando con by**  DT <- data.table(A = c(1, 2, 3, 4, 5), B = c("a", "b", "c", "a", "b"), C = c(6, 7, 8, 9, 10))  DT[, .(MySum = sum(A), MyMean = mean(A)), by = .(B)]  DT[, .(MySum = sum(C)), by = .(Grp = C%%2)]  **Encadenando llamadas**  DT <- data.table(A = c("c", "b", "a", "c", "b", "a"), B = c(1, 2, 3, 4, 5, 6))  DT[, sum(B), by = A][order(A)] # Suma B agrupando y ordenando A  DT <- data.table(A = c("b", "b", "b", "b", "a", "a", "a", "a"), B = c(1, 1, 2, 2, 3, 3, 4, 4), C = c(3, 8, 4, 5, 1, 7, 2, 6))  DT[, .(C = sum(C)), by = .(A, B)]# Agrupan por columnas A y B y suma C.  **Usando .SD**  DT[, lapply(.SD, mean), by = A] # Media de las columnas.  DT[, lapply(.SD, median), by = A] # Mediana.  DT <- data.table(grp = c(6, 6, 8, 8, 8), Q1 = c(2, 2, 3, 5, 2), Q2 = c(5, 5, 4, 4, 1), Q3 = c(2, 1, 4, 2, 3), H1 = c(3, 4, 5, 2, 4), H2 = c(5, 2, 4, 1, 2))  DT[, lapply(.SD, sum), .SDcols = 2:4] # suma de las columnas Q  DT[, lapply(.SD,sum), .SDcols = paste0("H", 1:2)] # suma de H1 y H2  DT <- data.table(x = c(2, 1, 2, 1, 2, 2, 1), y = c(1, 3, 5, 7, 9, 11, 13), z = c(2, 4, 6, 8, 10, 12, 14))  DT[, lapply(.SD, sum), by = x, .SDcols = c("x", "y", "z")] # Suma las columnas (x, y, z) agrupándolas por x.  DT[, lapply(.SD, cumsum), by = .(by1 = x, by2 = z > 8), .SDcols = c("x", "y")] # Calcula la suma acumulada de x e y agrupando por x y z > 8  **Usando :=**  DT <- data.table(A = c("a", "a", "a", "b", "b"), B = c(1, 2, 3, 4, 5))  DT[, Total := sum(B) , by = A] # nueva columna agregando por A.  DT[c(2, 4), B := B + 1] # Añade 1 a la columna B en las filas 2 y 4  DT[2:4, Total2 := sum(B), by = A] # Nueva columna agregando por A pero solo en las filas 2, 3 y 4.  DT[, Total := NULL] # Elimina la columna  DT <- data.table(A = c(1, 1, 1, 2, 2), B = 1:5)  DT[, `:=`(B = B + 1, C = A + B, D = 2)] # Actualiza B y añade C y D  DT[, 2 := NULL] # Borra la segunda columna  **Usando key**  DT <- data.table(A = letters[c(2, 1, 2, 3, 1, 2, 3)], B = c(5, 4, 1, 9,8 ,8, 6), C = 6:12)  setkey(DT, A, B) # Creación del índice  DT["b"] # Filtra por A="b"  DT[c("b", "c")] # Filtra por A="b" | A="c"  DT[c("b", "c"), mult = "first"] # Selecciona primera fila de cada grupo  DT[c("b", "c"), .SD[c(1, .N)], by = .EACHI] #Selecciona la primera y ultima fila de cada grupo. EACHI (por columnas). | library(data.table)  **Convierte iris en un data.table**  iris.dt <- as.data.table(iris)  **Para cada especie calcula la longitud media del sépalo**  iris.dt[, mean(Sepal.Length), by = Species]  **Agrupando por la primera letra del nombre de cada especie, calcula la longitud media del sépalo.**  iris.dt[, mean(Sepal.Length), by = substr(Species, 1, 1)]  **Filtra las filas de la especie virginica.**  iris.dt[Species == "virginica"]  **Filtra las filas de la especie virginica y versicolor.**  iris.dt[Species %in% c("virginica", "versicolor")]  **Para cada especie calcula la mediana de todas las columnas. Ordena el resultado por el nombre de la especie descendentemente.**  iris.dt[, .(Sepal.Length = median(Sepal.Length),  Sepal.Width = median(Sepal.Width),  Petal.Length = median(Petal.Length),  Petal.Width = median(Petal.Width)), by = Species][order(-Species)]  **Repite el ejercicio anterior usando .SD**  iris.dt[, lapply(.SD, median), by = Species][order(-Species)]  **Elimina el prefijo sepal del nombre de las columnas.**  setnames(iris.dt, gsub("Sepal.", "", names(iris)))  **Elimina las dos columnas que comienzan por "Petal".**  iris.dt[, grep("Petal", names(iris)) := NULL]  **Filtra las filas para las que el área es mayor que 20.**  iris.dt[Width \* Length > 20]  **Añade una columna que indique si la fila tiene un área mayor de 25.**  iris.dt[, is\_large := Width \* Length > 25]  **Filtra aquellas filas que cumplen la condición anterior.**  iris.dt[is\_large == TRUE] |

# Caso a resolver: Análisis de una base de datos

|  |  |
| --- | --- |
| Simulacro de la práctica | En esta hoja encontrarás un​ dataset (​bikes.csv) ​que contiene información agregada por horas de un sistema de alquiler de bicicletas (parecido al servicio BiciMad​).  Las variables del ​ dataset ​  son las siguientes:   * date:​  fecha (en formato yyyy­mm­dd). * season:​  estación del año. Los valores son:   + 1 (winter)   + 2 (spring)   + 3 (summer)   + 4 (fall) * hour:​  la hora del día (0 a 23). * is.holiday:​  1 si es día festivo, 0 en caso contrario. * weekday:​  día de la semana. Los valores son:   + 0 (Sunday)   + 1 (Monday)   + 2 (Tuesday)   + 3 (Wednesday)   + 4 (Thursday)   + 5 (Friday)   + 6 (Saturday) * is.workingday:​  si el día no es ni fin de semana ni vacaciones 1, en caso contrario 0. * weathersit:​  variable categórica con los siguientes valores:   + 1: Clear, Few clouds, Partly cloudy, cloudy   + 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist   + 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds   + 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog * temp:​  temperatura en grados Celsius. * atemp:​  sensación térmica en grados Celsius. * hum:​  humedad. * windspeed:​  velocidad del viento. * casual:​  número de alquileres de usuarios no registrados en el servicio. * registered:​  número de alquileres de usuarios registrados en el servicio. |

## Parte 1 Análisis básico

**Lee el fichero y asígnalo a una variable**

setwd("C:/Users/Guillermo/Desktop")

bikes\_csv<-read.table("bikes.csv",header=T,sep=",",stringsAsFactors = F,dec=".")

**¿De qué clase es el objeto?**

class(bikes\_csv)

**¿Cómo se puede ver el tipo de cada columna y una muestra de ejemplos?**

str(bikes\_csv)

**Muestra los primeros 15 registros del dataset.**

head(bikes\_csv,15)

**Muestra los últimos 20 registros del dataset.**

tail(bikes\_csv,20)

**¿Cuáles son las dimensiones del dataset?**

dim(bikes\_csv)

**¿Cuáles son los nombres de las variables del dataset?**

names(bikes\_csv)

**Las variables season y weekday deberían ser categóricas. Crea un factor con etiquetas para dichas columnas (no hace falta que sea ordenable) y asígnalo a las columnas de nuevo.**

season<-factor(bikes\_csv[,2],levels=c(1,2,3,4)) #lo convierto en un factor

levels(season)[1:4]<-c("winter","spring","summer","fall") #renombramos los niveles

bikes\_csv[,2]<-season #Lo reasigno al data frame.

weekday<-factor(bikes\_csv[,5],levels=c(0,1,2,3,4,5,6)) #lo convierto en un factor

levels(weekday)[1:7]<-c("Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday") #renombramos bikes\_csv[,5]<-weekday #Lo reasigno al data frame.

**Las variables i s.holiday y i s.workingday son variables booleanas. Conviértelas a variable booleana y asígnalas a las columnas de nuevo.**

bikes\_csv[,4]<-as.logical(bikes\_csv[,4])

bikes\_csv$is.holiday<- as.logical(bikes\_csv$is.holiday)#otra manera de hacerlo

bikes\_csv[,6]<-as.logical(bikes\_csv[,6])

**Calcula la suma de la columna casual.**

sum(bikes\_csv[,12])

sum(bikes\_csv$casual)

**Guarda en un vector la suma de las columnas casual y registered.**

total\_por\_fila<-as.vector(bikes\_csv$casual+bikes\_csv$registered)

total\_por\_columna <-c(sum(bikes\_csv$casual), sum(bikes\_csv$registered))

**¿Qué variables son numéricas? PISTA: utiliza sapply junto con la funcion is.numeric.**

sapply(bikes\_csv,is.numeric)

**Utilizando el resultado anterior, selecciona aquellas columnas numéricas y calcula la media de todos los registros.**

seleccion<- bikes\_csv[sapply(bikes\_csv,is.numeric)]

sapply(seleccion,mean)

colMeans(seleccion) # Otra manera de hacerlo.

**Selecciona aquellas observaciones del dataset para las que el día de la semana es Monday, están entre las 15 y 20 horas y la temperatura es superior a 25 grados.**

bikes\_csv[(bikes\_csv$weekday=="Monday" & bikes\_csv$hour>15 & bikes\_csv$hour<20 & bikes\_csv$temp>25),]

**Selecciona las 100 primeras filas y todas las columnas menos las dos últimas (con índices positivos).**

bikes\_csv[1:100,1:11]

**Selecciona las 100 primeras filas y todas las columnas menos las dos últimas (con índices negativos).**

bikes\_csv[1:100,1:(length(bikes\_csv)-2)]

**Obtén los cuantiles de la variable hum.**

quantile(bikes\_csv$hum)

**Obtén los deciles de la variable hum.**

quantile(bikes\_csv$hum,c(0, .1, .2, .3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1))

quantile(bikes\_csv$hum,probs=seq(0,1,0.1)) #una manera más eficiente de hacerlo.

**Obtén los estadísticos básicos de todas las variables en un solo comando.**

summary(bikes\_csv)

**¿Cuantos valores únicos existen para la variable atemp?**

length(unique(bikes\_csv$atemp))

**¿Cuantos registros tienen un valor superior a 100 para la variable casual?**

nrow(bikes\_csv[bikes\_csv$casual>100,])

sum(bikes\_csv$casual>100) #otra manera de hacerlo.

**Ordena de menor a mayor los 100 primeros elementos de la variable registered.**

sort(bikes\_csv[1:100,13])

sort(bikes\_csv$registered[1:100]) #otra manera de hacerlo

**Ordena el dataset por la variable registered de manera ascendente. Inspecciona los primeros resultados.**

indice<-order(bikes\_csv$registered, decreasing = F) #Obtengo el índice por el que realizar la ordenación.

bikes\_csv<-bikes\_csv[indice,]

bikes\_csv<-bikes\_csv[order(bikes\_csv$registered,decreasing = F),] #otra manera de hacerlo

**Obtén los índices de los registros para los que el valor de la variable temp es superior a la media.**

indice\_booleano<- bikes\_csv$temp>mean(bikes\_csv$temp)

indice\_numerico <- which(indice\_booleano)

**Obtén los registros para los que el valor de la variable windspeed es máximo.**

maximo<-max(bikes\_csv[,11])

registros<-bikes\_csv[bikes\_csv$windspeed==maximo,]

**Comprueba si alguna de las variables contiene NAs.**

any(is.na(bikes\_csv))

**Comprueba utilizando el boxplot si la variable windspeed tiene outliers.**

boxplot(bikes\_csv$windspeed) #Sí que los tiene.

**Pinta un histograma de la variable atemp.**

hist(bikes\_csv$atemp)

**Crea una función (cold.heat) que reciba dos parámetros: temperatura y límite; y devuelva cold o heat.**

cold.heat<-function(temp,limit=20){

if (temp>limit){

return ("Heat")

} else if (temp<limit){

return ("Cold")

} else {

return ("ideal")

}

}

cold.heat(5)

cold.heat(20)

cold.heat(30)

**Aplica la función a la columna temp y almacena el resultado en una nueva columna cold.heat del data.frame.**

cold.heat<-sapply(bikes\_csv$temp,cold.heat)

bikes\_csv<-cbind(bikes\_csv,cold.heat)

bikes\_csv$cold.heat<- sapply(bikes\_csv$temp,cold.heat)#en una sola línea.

## Parte 2: Utiliza Data frame, dplyr o data.table (según prefieras)

library(data.table)

**Calcula el número medio de usuarios registrados por día de la semana**

bikes<-data.table(bikes) #Convierto el data frame en un data table.

bikes[,.(media\_usuarios=mean(registered)),by=.(weekday)]

bikes.group.semana <- group\_by(bikes,weekday) # Otra manera de hacerlo.

summarise(bikes.group,media\_usuario = mean(registered))

**Ordena de menor a mayor las temperaturas máximas de cada estación**

bikes[,.(temperaturas\_maximas=max(temp)),by=.(season)][order(temperaturas\_maximas)]

bikes.group.estacion <- group\_by(bikes,season) # Otra manera de hacerlo.

arrange(summarise(bikes.group.estacion,temp\_max = max(temp)),desc(temp\_max))

**Calcula el número total de usuarios (registered + casual) en días festivos y no festivos.**

bikes[,.(num\_total\_usuarios=sum(casual+registered)),by=.(is.holiday)]

bikes.group.festivos<-group\_by(bikes,is.holiday) # Otra manera de hacerlo.

bikes.group.festivos$usuarios = bikes.group.festivos$casual + bikes.group.festivos$registered

summarise(bikes.group.festivos,suma =sum(usuarios))

bikes %>%

group\_by(is.holiday) %>%

mutate(usuarios = casual + registered) %>%

summarise(suma = sum(usuarios))

**Calcula el día que se produjeron el mayor número de alquileres casuales.**

bikes[, .(max=sum(casual)), by = date][order(-max)] # El primer registro tienen el día con el máximo de alquileres casuales.

library(dplyr) # Otra manera de hacerlo.

bikes %>%

group\_by(date) %>%

select(date,casual) %>%

summarise(suma\_dia = sum(casual)) %>%

filter(suma\_dia==max(suma\_dia))

bikes[casual==bikes[,max(casual)]] # MAL, no obtenemos el día en que más se produjeron, sino el día con máximo por hora.

**Calcula el ratio de alquileres registrados/casuales por hora en verano.**

alquileres\_en\_verano<-bikes[season==3]

registrados<-alquileres\_en\_verano[,.(suma\_registrados=sum(registered)),by=.(hour)]

casuales<-alquileres\_en\_verano[,.(suma\_casuales=sum(casual)),by=.(hour)]

rdo<-data.table(registrados,casuales[,suma\_casuales])

rdo[,ratio:=suma\_registrados/V2]

library(dplyr) #otra manera de hacerlo.

bikes %>%

filter(season==3) %>%

group\_by(hour) %>%

summarise(suma\_reg = sum(registered),suma\_cas=sum(casual))%>%

mutate(proporcion = suma\_reg/suma\_cas)

# La manera más sencilla de hacerlo.

data.table: bikes[season == 3, .(registered.casual.ratio = sum(registered) / sum(casual)), by = .(hour)]

**Para todos los lunes festivos calcular el número total de alquileres registrados, temperatura media, humedad máxima y velocidad del viento mínima.**

lunes\_festivos<-bikes[weekday==1 & is.holiday==1]

lunes\_festivos[,.(total\_alquileres\_registrados=sum(registered),temperatura\_media=mean(temp),humedad\_maxima=max(hum),velocidad\_viento\_minimo=min(windspeed))]

bikes %>% # Otra manera de hacerlo.

filter(weekday==1,is.holiday==1) %>% summarise(total\_alquileres\_registrados=sum(registered),temperatura\_media=mean(temp),hum\_max=max(hum),min\_vel=min(windspeed))