Modulo 1: Herramientas Big Data

Herramientas de Análisis: Programación en R II

Leandro Gutierrez

14/05/2024

EJERCICIO 2

Para este ejercicio utilizaremos los datos los datos `millas` que hay el package `datos`. Estos datos consisten en 238 filas y 11 columnas que describen el consumo de combustible de 38 modelos de coche populares.

Puedes consultar más con la ayuda: `?millas`.

Cargamos datos de nuevo.

```
library(datos)
library(ggplot2)
suppressPackageStartupMessages(library(tidyverse))
```

EJERCICIO 2.1.

Escribe un bucle `for` que guarda en nuevo data frame, la media de las columnas numéricas (de tipo integer o numeric) de `millas`. Presenta mediante `print` el data frame de resumen.

```
numeric_cols <- sapply(millas, is.numeric)

numeric_df <- millas[, numeric_cols]

res_df <- data.frame(NA)

for (i in 1:length(numeric_df)) {
    col_vals <- unlist(numeric_df[i], use.names=FALSE)

    res_df[1, i] <- mean(col_vals, na.rm= TRUE)
}

colnames(res_df) <- colnames(numeric_df)

res_df
```

```
##   cilindrada   anio cilindros   ciudad autopista
## 1   3.471795 2003.5  5.888889 16.85897  23.44017
```

EJERCICIO 2.2.

Haz lo mismo que en 2.1 pero utilizando `sapply()` en vez del bucle for.

```
numeric_cols <- sapply(millas, is.numeric)

numeric_df <- millas[, numeric_cols]

res_df <- sapply(numeric_df, MARGIN=2, mean)

res_df <- data.frame(t(res_df))

colnames(res_df) <- colnames(numeric_df)

res_df
```

```
##   cilindrada   anio cilindros   ciudad autopista
## 1   3.471795 2003.5  5.888889 16.85897  23.44017
```

EJERCICIO 2.3.

Explica la diferencia entre la función `if()` e `ifelse()`. Pon un ejemplo de uso de ambas.

Funcion `if()`

```
analizar <- function (name) {
    name %in% c("John", "Paul", "George", "Ringo")
}

v <- c("John", "Chris", "Pete", "Lucas")

v1 <- if (all(analizar(v))) {
    print("todos son geniales")
} else if (any(analizar(v))) {
    print("algunos son geniales")
} else {
    print("ninguno es genial")
}
```

```
## [1] "algunos son geniales"
```

Funcion `ifelse()`

```
v2 <- ifelse(analizar(v), v, NA)

v2
```

```
## [1] "John" NA     NA     NA
```

Respuesta

La función `if()` está diseñada para tomar como parámetro elementos de longitud 1, si por ejemplo en nuestro ejemplo hubiesemos intentando:

```
v1 <- if (analizar(v)) {
    print("todos son geniales")
} ...
```

Hubiesemos obtenido el siguiente mensaje de error de nuestro interprete:

> Error in if (is.positive(v)) { : the condition has length > 1

Es por ello que en nuestro código necesitamos utilizar las funciones `any()` u `all()` las cuales devuelven un valor escalar respecto a un input vectorial.

Mientras que la función `ifelse()` está preparada para recibir un vector como input y devolver asimismo un vector resultado analizado para cada uno de sus elementos.

EJERCICIO 2.4.

¿Qué parámetros son imprescindibles especificar cuando se leen datos de ancho fijo mediante: `read.fwf()`?

Explica qué significan y pon un ejemplo.

Funcionamiento esperado

```
fwf_sample <- read.fwf("www/fwf-sample.txt", widths=c(20,10,12))

fwf_sample
```

```
##                      V1         V2           V3
## 1 John Smith           WA         418-Y11-4111
## 2 Mary Hartford        CA         319-Z19-4341
## 3 Evan Nolan           IL         219-532-c301
```

Funcionamiento erroneo 1

```
fwf_error1 <- read.fwf("www/fwf-sample.txt", widths=c(2,10,12))

fwf_error1
```

```
##   V1         V2           V3
## 1 Jo hn Smith           WA
## 2 Ma ry Hartfor d       CA
## 3 Ev an Nolan           IL
```

Funcionamiento erroneo 2

```
fwf_error2 <- read.fwf("www/fwf-sample.txt")
```

> Error in read.fwf("www/fwf-sample.txt") : argument "widths" is missing, with no default

Respuesta

Al utilizar la función `read.fwf()` para tomar datos de archivos con columnas de ancho fijo, es imprescindible especificar el parametro `widths` el cual indica el ancho de cada columna a leer. También podemos apreciar que al utilizar anchos que no coincidan con los esperados, se producen lecturas equivocadas, derivando en datasets erroneos.

EJERCICIO 2.5.

Calcula la media de millas/galón en autopista para cada `clase` de coche de `millas`.

Presenta la tabla obtenida.

```
agg1 <- aggregate(millas[,c("autopista")],
                  list(clase=millas$clase), mean, na.action = na.omit)

agg1
```

```
##         clase autopista
## 1   2asientos  24.80000
## 2    compacto  28.29787
## 3     mediano  27.29268
## 4     minivan  22.36364
## 5      pickup  16.87879
## 6 subcompacto  28.14286
## 7         suv  18.12903
```

EJERCICIO 2.6.

Incorpora la media calculada en 2.5. en el data frame `millas` como una nueva columna llamada "autopista_clase".

> Utiliza la funcion `merge()` para juntar el objeto obtenido en 2.5 con `millas`.

Presenta el summary() de la nueva columna.

```
colnames(agg1) <- c("clase", "autopista_clase")

millas <- merge(millas, agg1, by="clase")

summary(millas["autopista_clase"])
```

```
##  autopista_clase
##  Min.   :16.88
##  1st Qu.:18.13
##  Median :27.29
##  Mean   :23.44
##  3rd Qu.:28.14
##  Max.   :28.30
```

EJERCICIO 2.7.

Utiliza las funciones del package dplyr: `group_by()` y `mutate()` para realizar el mismo calculo que en 2.5. y 2.6. sin necesidad de utilizar `merge()`. Llama a la nueva columna "autopista_clase_dplyr"

> Truco: Utiliza el siguiente ejemplo: `datos %>% group_by(var_seg) %>% mutate(nueva_variable=mean(variable))`

Haz un `summary()` para verificar que el resultado es el mismo que en 2.6.

```
millas <- millas %>%
    group_by(clase) %>%
    mutate(autopista_clase_dplyr=mean(autopista))

summary(millas["autopista_clase_dplyr"])
```

```
##  autopista_clase_dplyr
##  Min.   :16.88
##  1st Qu.:18.13
##  Median :27.29
##  Mean   :23.44
##  3rd Qu.:28.14
##  Max.   :28.30
```

EJERCICIO 2.8.

Analiza si `millas` tiene registros duplicados y en caso afirmativo crea un nuevo data frame que contenga una única copia de cada fila.

```
any(duplicated(millas))
```

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

```
millas_dedup <- millas %>% distinct()

any(duplicated(millas_dedup))
```

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

EJERCICIO 2.9.

Crea una función que tenga como input la fecha de tu nacimiento (en formato date) y devuelva tu edad en años.

```r
calcular_edad <- function (nacimiento){
    hoy <- Sys.Date()
    length(seq(from=nacimiento, to=hoy, by='year')) - 1
}

edad <- calcular_edad(as.Date("1991-08-14"))

edad
```

```
## [1] 32
```

EJERCICIO 2.10.

Explica porqué el resultado de fechahora_1 y fechahora_2 son distintos en la siguiente expresión:

```r
library(lubridate)
Sys.setlocale(locale="es_ES.UTF-8")
```

```
## [1] "es_ES.UTF-8/es_ES.UTF-8/es_ES.UTF-8/C/es_ES.UTF-8/en_US.UTF-8"
```

```r
fechahora <- ymd_hms("2020-03-28 15:11:23", tz = "Europe/Madrid")
fechahora_1 <- fechahora + dhours(24)
fechahora_2 <- fechahora + hours(24)

print(fechahora_1)
```

```
## [1] "2020-03-29 16:11:23 CEST"
```

```r
print(fechahora_2)
```

```
## [1] "2020-03-29 15:11:23 CEST"
```

Respuesta

Para entender mejor situación anterior, pondrémos un ejemplo extra:

```r
library(lubridate)
Sys.setlocale(locale="es_ES.UTF-8")
```

```
## [1] "es_ES.UTF-8/es_ES.UTF-8/es_ES.UTF-8/C/es_ES.UTF-8/en_US.UTF-8"
```

```r
fechahora <- ymd_hms("2024-03-30 02:00:00", tz = "Europe/Madrid")
fechahora_1 <- fechahora + dhours(24)
fechahora_2 <- fechahora + hours(24)

print(fechahora_1)
```

```
## [1] "2024-03-31 03:00:00 CEST"
```

```r
print(fechahora_2)
```

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

```r
class(hours(24))
```

```
## [1] "Period"
## attr(,"package")
## [1] "lubridate"
```

```r
class(dhours(24))
```

```
## [1] "Duration"
## attr(,"package")
## [1] "lubridate"
```

Sucede que el 31 de Marzo de este año, como cada último domingo de marzo, se realizó en España el cambio de huso horario, para pasar a utilizar GMT+1 como horario de verano. Hecho que produce un salto en el reloj de pared a las 2 AM del dia 31, pasando a ser las 3 AM del dia mencionado.

Ahora, la función `dhours()`, la cual devuelve un objeto de clase `Duration`, en combinación con la adición al datetime original, tiene en cuenta este cambio de huso horario programado y resuelve corretamente el salto de horas. Mientras que la función `hours()` nos instancia un objeto de clase `Period`, el cual en combinación con una adición sobre un caso limite (cambio de huso horario) puede no responder como es esperado. Ambas altarnativas tienen su propia lógica específica y dependerá del objetivo a conseguir cual opción utilizar.