R_curso_R para_big_data_introductorio

Rdouglas

1/2/2022

R Markdown

crear un vector

Los vectores vectores son arreglos ordenados en los cuales se puede almacenar

#información de tipo numérico (variable cuantitativa), alfanumérico # (variable cualitativa) o lógico (TRUE o FALSE), pero no mezclas de éstos

edad <- c(15, 19, 13, NA, 20)

edad

class(edad)

crear una lista

#Las listas son otro tipo de objeto muy usado para almacenar objetos de diferente #tipo. La instrucción para crear una lista es list( )

x <- c(45, 12, 56, 14, 16) y <- c("Coche", "Bicicleta") z <- matrix(1:12, ncol = 4)

milista <- list(x, y, z)

milista

class(milista) class(x) class(y) class(z)

crear un data frame

mimarco <- data.frame(edad = c(15, 19, 13, NA, 20), deporte = c(TRUE, TRUE, NA, FALSE, TRUE), comic_fav = c(NA, 'Superman', 'Batman', NA, 'Batman'))

mimarco

class(mimarco)

library(tidyverse) ??read_csv

cargar el dataset

df<- as.data.frame(read_csv("2008.csv", n_max = 1000))

visualizar el dataset

View(df)

seleccion de columnas

df$ArrDelay df[“ArrDelay”] df[1:5,c(“ArrDelay”,“DepDelay”)]

Programacion Vectorial

delay <- df$ArrDelay delay <- delay[!is.na(delay)]

sumar los valores de taxin y taxout

df$TaxiIn + df$TaxiOut

con esto se puede ver cuando tax in es mayor a Taxout

df$TaxiIn > df$TaxiOut

a <- delay[delay > 0]*2 b <- delay[delay > 0]+ delay[delay > 0]

print(a) print(b)

a == b

all(a == b)

sum(delay)

delay %*% rep(1,length(delay))

df2 <- df[,c(“CarrierDelay”, “WeatherDelay”, “NASDelay”, “SecurityDelay”, “LateAircraftDelay”)]

df2 <- df2 [complete.cases(df2),]

df2 <- as.matrix(df2)

df2 %*% rep(1,ncol(df2))

resultado <- rep(1,nrow(df2)) %*% df2 resultado[5] == sum(df2[,5]) view(df2)

Planificar la creación de Variables

df<- as.data.frame(read_csv(“2008.csv”, n_max = 100000))

df2 <- df[,c(“CarrierDelay”, “WeatherDelay”, “NASDelay”, “SecurityDelay”, “LateAircraftDelay”)]

df2 <- df2 [complete.cases(df2),]

matriz <- matrix(rep(0, nrow(df2) * 5), nrow = nrow(df2)) colnames(matriz) <- c(“Minimo”, “Mediana”, “Maximo”, “Media”, “Std. Dev”)

esta funcion permite crear una matrix que computa las medidas de tendencia central por medio de un bucle

head(matriz)

for (i in 1:nrow(df2)) { val <- as.numeric(df2[i,]) matriz[i,] <- c(min(val), median(val), max(val), mean(val), sd(val)) }

head(matriz)

Apply . una alternativa a un Loops

start <- Sys.time()

resultado3 <- apply(X = df2 , MARGIN = 1, FUN = mean, na.rm= TRUE) ## x = al data frame ## margin 1 (significa filas) margin 2 (significa que trabaja con columnas) ## FUN corresponde a la funcion que se desea aplicar

resultado3

print(Sys.time()- start)

Funciones Basicas con Apply

df<- as.data.frame(read_csv("2008.csv", n_max = 100000))

df2 <- df[,c("CarrierDelay", "WeatherDelay", "NASDelay", "SecurityDelay", "LateAircraftDelay")]

head(df2) # se comprubea que existen valores Na o NaN

seleccionar solo los valores que están en el datset (omitir los vlaores nulos)

df_clean <- df2[complete.cases(df2),] head(df_clean)

Apply function

filas <- apply(X = df_clean, MARGIN = 1, FUN = mean)

columnas <- apply(X = df_clean, MARGIN = 2, FUN = mean)

print(filas)

print(columnas)

lapply function

resultado1 <- sapply(df_clean$CarrierDelay, factorial)

resultado1

Crear funciones especificas para Apply

lapply(1:10, FUN = function(x) x^2)

apply(df_clean, 1, mean,trim = 0.01) apply(df_clean, 1, function(x) mean (x,trim = 0.01))

funcionpropia <- function(fila) { if(all(fila > 0)){ return(mean(fila)) } else { return(max(fila)) }
}

funcionpropia2 <- function(fila) { if(any(is.na(fila > 0))){ return("Contiene NAs") } else { if(all(fila > 0)){
return(list(a = "Media", b= mean(fila))) } else{ return(list(a = "Maximo", b= max(fila)))

```
            }
    }
```

}

resultado <- apply(df2,1,funcionpropia2) head(resultado)

Gestionar el output de las funciones Apply

como crear un TOP 3

top3 <- function(columna){ columna <- columna [!is.na(columna)] columna <- as.numeric(columna[order(-
columna)]) return(list("Primero" = columna [1], "Segundo" = columna[2], "Tercero" = columna[3])) }

resultado <- apply(df2, 2, top3) head(resultado) unlist(resultado) matriz <- matrix(unlist(resultado), byrow
= FALSE, nrow = 3) colnames(matriz) <- names(resultado) rownames(matriz) <- names(resultado$CarrierDelay)

matriz

mat = matrix (c(8,7,9,6,7,9,10,8,9), nrow = 3, ncol = 3, byrow = TRUE)

rownames(mat) <- c("estudiante1","estudiante2","Estudiante3") colnames(mat) <- c("mes1","mes2","mes3")

promedio <- apply(mat, 1, mean)

promedio

mixto <- c(4,3,TRUE)

mixto

Practica de funciones

mysum <- function(a,b){ thesum <- a+b return(thesum) }

mysum(2,8)

?paste

show.datos.vector <- function(mivector, argmodo){ print("Datos del vector. Media, Máximo, mínimo
y ordenado") print(paste("Media:",round(mean(mivector),2))) print(paste("Máximo:",max(mivector)))
print(paste("Mínimo:",min(mivector))) if (argmodo == 'A'){ print(sort(mivector, decreasing = FALSE))
} else { print(sort(mivector, decreasing = TRUE)) } }

show.datos.vector(c(3,4,5,6), 'A')

args(show.datos.vector)

myfun <- function(a,b){ thesum <- a*b return(thesum) }

myfun(2,5)

cuando usar un bucle o un apply

?length

length(delay)

?rep

rep(delay)

delay<- df$ArrDelay delay <- delay[!is.na(delay)] delay <- delay[delay > 0]

variacion <- rep(0, length(delay)-1)

for (i in 1:length(delay)-1) { variacion[i] <- (delay[i+1] - delay[i])/delay[i]*100

} round(as.numeric(variacion))

uso de Aggregate usango groupby en R

esto genera el valor promedio de los retrasos de vuelos para cada dia de semana

aggregate(x = df$ArrDelay, by = list(df$DayOfWeek), FUN= mean, na.rm = TRUE) # el By siempre debe tener una lista como input

aggregate(x = list(df$ArrDelay, df$DepDelay), by= list(df$DayOfWeek), FUN= mean, na.rm = TRUE)

aggregate(x= list("Arr"= df$ArrDelay, "Dep" = df$DepDelay), by= list("Day" = df$DayOfWeek), FUN= mean, na.rm = TRUE)

aggregate(x= list("Arr"= df$ArrDelay, "Dep" = df$DepDelay), by= list("Origin" = df$Origin, "Day" = df$DayOfWeek), FUN= mean, na.rm = TRUE)

esto genera un groupby destino y dia de la semana, permitiendo ver el valor

promedio

aggregate(x= list("Arr"= df$ArrDelay, "Dep" = df$DepDelay), by= list("Origin" = df$Origin, "Day" = df$DayOfWeek), FUN= mean, na.rm = TRUE)

esto genera un groupby destino

aggregate(x= list("Arr"= df$ArrDelay, "Dep" = df$DepDelay), by= list("Origin" = df$Origin, "Day" = df$DayOfWeek), FUN= function(x) return(c(mean(x,na.rm = TRUE), median(x,na.rm = TRUE))))

parApply

require(parallel) nc1<- detectCores() nc1

c1 <- makeCluster(16) df2

results2 <- apply(df2,1,mean,na.rm =TRUE)

results2

results2 <- parapply(df2,1,mean,na.rm =TRUE)

system.time()

stopCluster(c1)

df_clean

head(df_clean)

listabloques <- list()

numfilas <- 50

for (i in 0:999) { listabloques[[i+1]] <- df_clean[(i*50+1):((i+1)*50),]
}

resultado4 <-sapply(listabloques, function(bloque){ summary(lm(formula = CarrierDelay ~ . , data= bloque))$r.squared })

resultado4

Introduccion a Estructuras de datos e Introduccion a dplyr

datatable

install.packages(data.table)

install.packages(data.table)

require(data.table)

data.frame(1:12,1:2)

data.table(1:12, 1:2)

dt <- fread("2008.csv")

head(dt)

opciones de filtrar con data table

dt[dt$Month == 2]

dt[Month==2,ArrDelay]

dt[Month ==2,.(ArrDelay,DepDelay)]

dt[Month ==2 |DayOfWeek == 1,.(ArrDelay,DepDelay)]

dt[Month == 1, .("Llegada" = ArrDelay, "Salida" = DepDelay)]

Group by con la modalidad de data table

#agrupado por meses dt[,.N, by= .(Month)]

dt[Origin == “ATL”,.N, by= .(Month)]

dt[ArrDelay > 60,.N, by= .(Origin, Dest)]

dt[ArrDelay > 60,.(“Retraso Salida Promedio” =mean(DepDelay)), by= .(Origin, Dest)]

dt[Origin == “ATL”,.N, by= .(Month, “>1h Retraso” = ArrDelay > 60)]

dt[Origin == “ATL” & !is.na(ArrDelay),.N, by= .(Month, “>1h Retraso” = ArrDelay > 60)]

Dplyr R con estilo SQL

require(data.table)

require(dplyr)

dt <- fread(“2008.csv”) df3 <- dt[sample(x = 1:nrow(dt), size = 1e5)]

summary(df3) head(df3)

filter(df, Month == 1 & DayOfWeek ==1)

arrange(df, ArrDelay)

?arrange arrange(df, ArrDelay, AirTime)

select(df,ArrDelay, AirTime)

mutate() sirve para crear nuevas variables/columnas a partir de la existentes.

mutate(df, Retraso = ArrDelay > 0)

mutate(df, Velocidad = Distance / AirTime)

transmute(df, Velocidad = Distance / AirTime)

summarise(df, “Dist. mediana” = median(Distance, na.rm = TRUE))

Usar dplyr para problemas complejos

origenes <- group_by(df, Origin)

retraso <- summarise(origenes, recuento = n(), distancia = mean(Distance, na.rm = TRUE), retraso_llegada = mean(ArrDelay, na.rm = TRUE))

retraso

retraso <- filter(retraso, retraso_llegada > 15, distancia > 500)

retraso

agrupaciones dt <- fread(“2008.csv”) df3 <- dt[sample(x = 1:nrow(dt), size = 1e6)]

summary(df3) head(df3)

nrow(df3)

diario <- group_by(df3, Year, Month, DayofMonth) (dias <- summarise(diario, vuelos = n ())) (meses <- summarise(dias, vuelos = sum(vuelos))) (años <- summarise(meses, vuelos = sum(vuelos)))

paquete Bigmmemory

Create, store, access, and manipulate massive matrices. Matrices are allocated to shared memory and may use memory-mapped files. Packages 'biganalytics', 'bigtabulate', 'synchronicity', and 'bigalgebra' provide advanced functionality.

permite aprovechar la memoria en la manipulacion de grandes datos

install.packages("bigmemory") library(bigmemory)

foreach and doParallel

install.packages("foreach") install.packages("doParallel") library(foreach) library(doParallel)

df3 <- df3 [complete.cases(df3),]

ncl <- detectCores() cl <-makeClusters()

stopCluster(cl)

paquete FF

require(ff) require(readr) require(ffbase)

df_ff <- read.csv.ffd(file = "2008.csv")

class(df3)