Leonardo Daniel Rivera valdez

Fundamentos de R

Vectores

Es una concatenación de datos de un mismo tipo, es un conjunto de datos en esencia. Por ejemplo:

- Un numérico sería 1, 2, 3, 5, 6, 7
- Uno de caracteres (character) podría ser “a”, “b”, “c”, “d”, “e”

#la función **c()** nos sirve para combinar elementos en un solo vector
vector_double <- c(5, 9, 48, 592)
vector_double

#Funciones con **is.algo()** verifican que el elemento sea o no de de dicho tipo

is.numeric(vector_double)
is.integer(vector_double)
is.double(vector_double)

vector_integers <- c(5L, 9L, 50L, 60L)

is.numeric(vector_integers)
is.integer(vector_integers)
is.double(vector_integers)

#Debido a que numérico es una categoría que abarca a los double e integer es que tenemos que ambos son numeric, pero solamente el primero es double y el segundo es integer

vector_character <- c("Hola", "amigo", "cómo estás?")

is.numeric(vector_character)

```
is.character(vector_character)

#En el siguiente vector se ilustra el principio que dice que no se pueden combinar tipos de datos. El 45 se va a almacenar como "".

vectos_dif_tdatos <- c("jejej", "Fodor", 45, "Dennett")



#Función seq() para generar secuencias como 1:10. Genera secuencias de números
seq(1, 10)

#Si se pone un tercer elemento se modifica la secuencia en función
#de ese número. De 2 en 2, de 3 en 3, etc.
seq(1, 10, 2)

vector_secuencia <- seq (1, 20, 5)
vector_secuencia

#Función rep() para repetir un elemento N cantidad de veces. Si yo pongo
#(5, 10) repito el 5 10 veces
rep(5, 10)

vector_rep <- rep(5, 250)
vector_rep

#También podemos repetir datos de tipo character, o incluso vectores
rep(vector_rep, 2)
rep("jeje", 10)
```

Uso de corchetes

```
#Dado el siguiente vector
W <- c("a", "b", "c", "d", "e")
W

#Corchetes me permiten acceder a los elementos del vector por su orden. Si pongo uno
accede al primer elemento, 2 al segundo y así.
W[1]
W[2]
W[3]

#Si coloco números negativos puedo obtener todos los elementos menos el que corresponde a
dicha posición. Si -1, entonces todos menos el primero.
W[-1]
W[-3]
W[-5]

#Esto anterior también es un vector en sí mismo
vector1 <- W[-1]

#Por medio de : podemos delimitar los elementos a los cuales accedemos en sentido
#de un rango
W[1:3]
W[1:2]
W[1:4]

#Si incluímos c() dentro de los [] podemos pedir datos específicos
#en el orden que queramos y la cantidad que queramos
W[c(1,2,5,3)]
W[c(5,4,3,2,1,5,5,5)]
W[c(1,1,1,1,1,1,1,5,5,5,5,5,5,5)]
```

#Al combinar esto con números negativos podemos filtrar los datos

```
W[c(-1,-2)]
W[c(-1,-3,-4,-5)]
W[c(-1,-5)]
W[c(-1:-2)]
W[c(-1:-4)]
W[c(-2:-4)]
W[c(1:4)]
```

Operaciones con vectores

Aritmética en R es más simple que en otros lenguajes ya que solamente necesita el símbolo de la operación para realizar cosas como +, -, /, *, etc. Lo mismo aplica con las operaciones lógicas. En casos donde el vector 1 es más corto que el vector 2, el vector 1 se recicla para poder ajustarse al vector 2, siempre y cuando 2 sea un múltiplo del vector 1.

Por ejemplo: Si el vector 1 es de 5 elementos, y el vector 2 es de 10 elementos y se quiere hacer una suma entre estos. Entonces, si 1 es [1,2,3,4,5] y 2 [1,2,3,4,5,6,7,8,9,10], lo que va a ocurrir es que del 1 al 5 se sumarán con los elementos de 1, y después del 6 al 10 nuevamente con 1 a 5 del vector 1.

En caso de que no sean múltiplos lo va a realizar pero el reciclado va a estar incompleto y el programa te lo hará saber. En nuestro caso anterior esto sería si por ejemplo el vector 1 fuera del 1 al 6, y el vector 2 se mantuviera igual.

Por último, como ya se ha mostrado anteriormente, los vectores de R también pueden ser indicados como una función de las 2 manera siguientes:

- f(x)
- x <- f(y)

```
#Primero definamos unos vectores
x <- c(seq(1:100))
y <- c(seq(100, 200))

#suma y resta de vectores. Nota, cuando no son múltiplos se necesita almacenarlos en una variable para después ejecutarla

z <- x + y
z
z+ y
```

```
z2 <- x-y
z

#división de vectores
a <- z/c(rep(10,100))
a
a1 <- a/x
a1

#multiplicación de vectores
b <- z * a
b
b1 <- b*z
b1

#sqrt() de vectores
sqrt(z+a)
sqrt(z-a)
sqrt(b+a)
sqrt(y+a)
```

#Algunas formas de acceder a los valores de los vectores por los medios convencionales de la programación

```
x1 <- rnorm(5)
x2 <- rnorm(100, mean = 50, sd = 5 )

for (i in x1){
  print(i)
}

for(i in 1:5){
  print(x1[i])
}

for(j in x2){
  print(j)
}

for(j in 1: 50){
  print(x2[j])
}
```

#Esta siguiente por medio de print es posible pero ineficiente porque no es un ciclo

```
print(x1[1])
print(x1[2])
print(x1[1:3])
```

#Operaciones con y sin vectores. Más sencillo con vectores, y más rápido, debido a que son menos pasos como se ve a continuación con estas multiplicaciones.

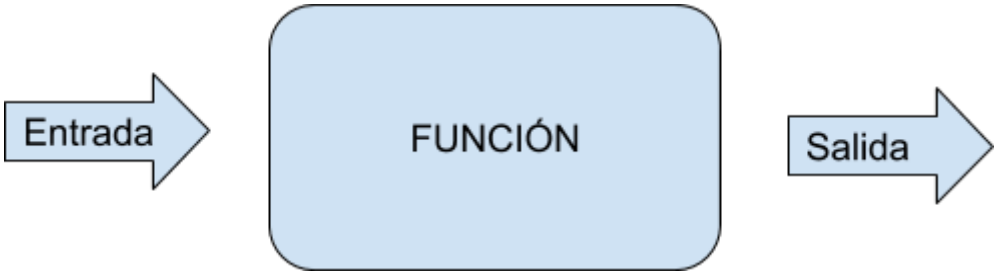
#Dadas las siguientes definiciones

```
N <- 100
a <- rnorm(N)
```

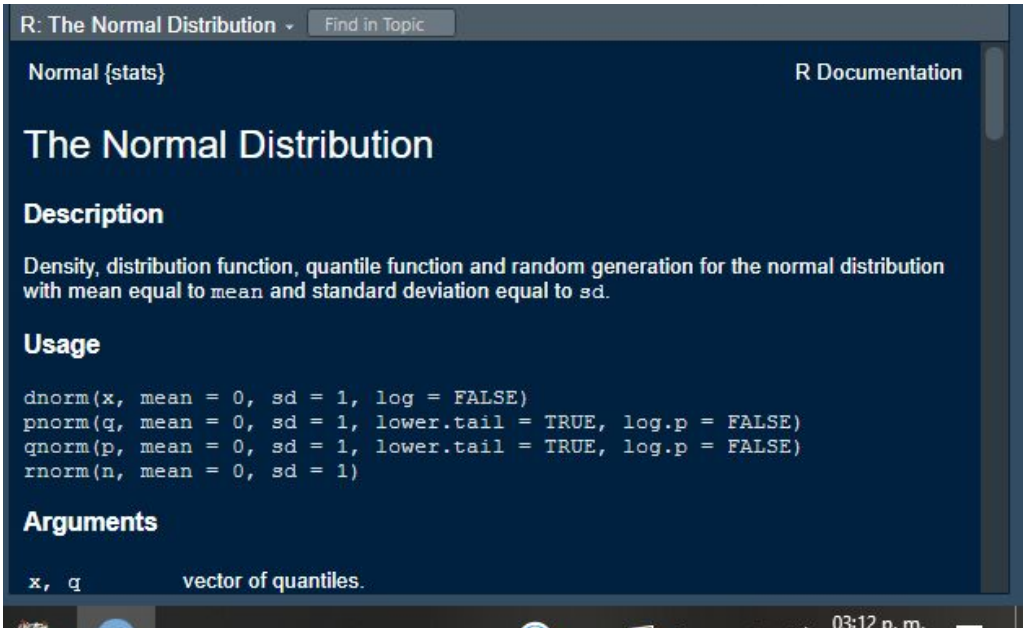
```
b <- rnorm(N)
d <- rep(NA, N)

#Vectores

c <- a * b

#Convencionales
#A diferencia del anterior, aquí se va pasando por cada uno de los elementos del vector y se van multiplicando y almacenado en otro vector. Es por ello que es más lento, sobretodo cuando más elemento tienen los vectores.

for (i in 1:N){
  d[i] <- a[i] * b[i]
}
```

Funciones en R

Una función puede entenderse como una caja a la cual uno le añade una entrada y da una salida. Esta caja tiene una serie de instrucciones o pasos que sigue para manipular la entrada de x tipo y dar una salida de y tipo.



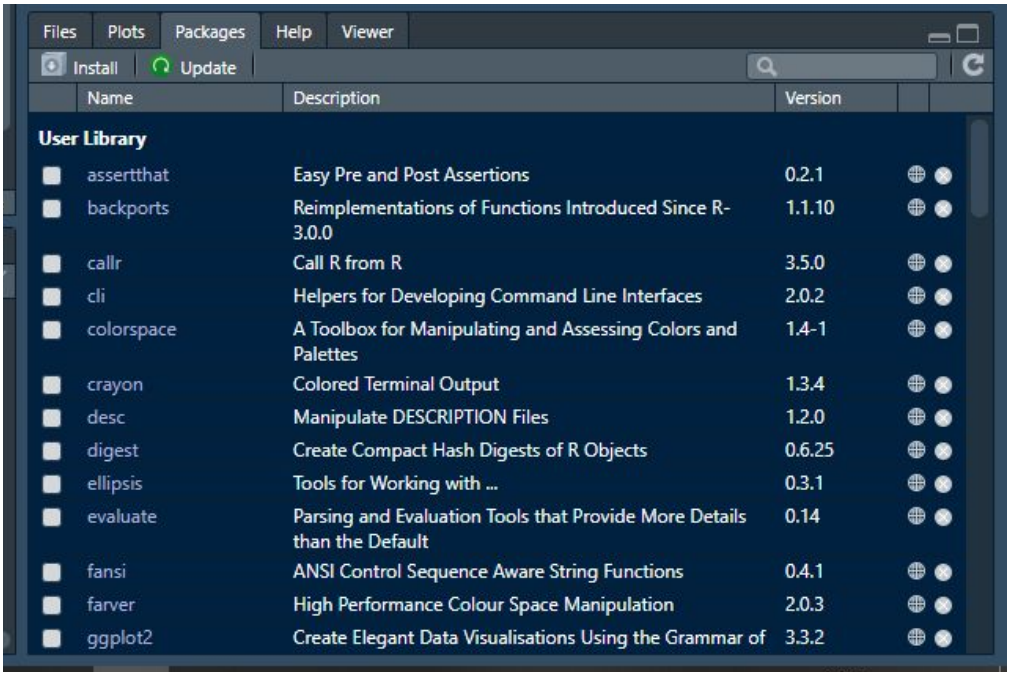
En caso de querer saber qué hace cada función por medio del símbolo **?**. Por ejemplo, si uno quiere saber qué hace rnorm(), uno escribe **?rnorm()** y R nos dice en help lo siguiente:



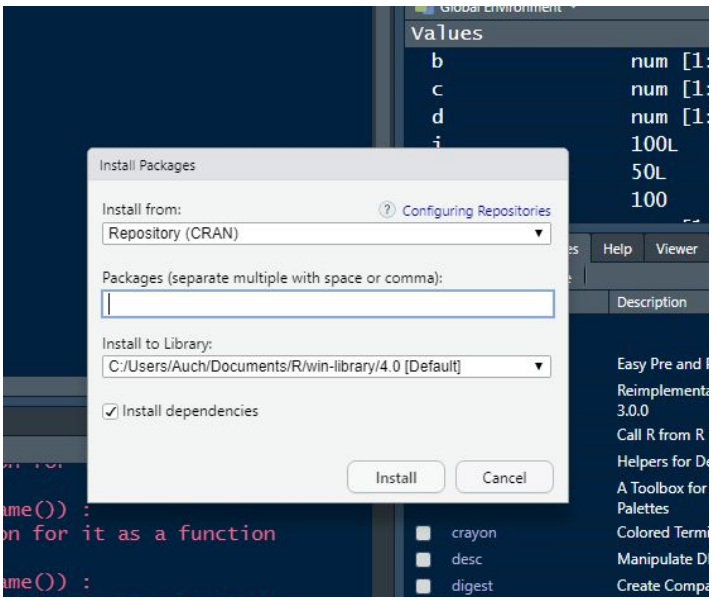
Paquetes de R

Los paquetes de R no son más que una colección de funciones, datos, y código compilados. La librería es donde se encuentran almacenados estos paquetes.

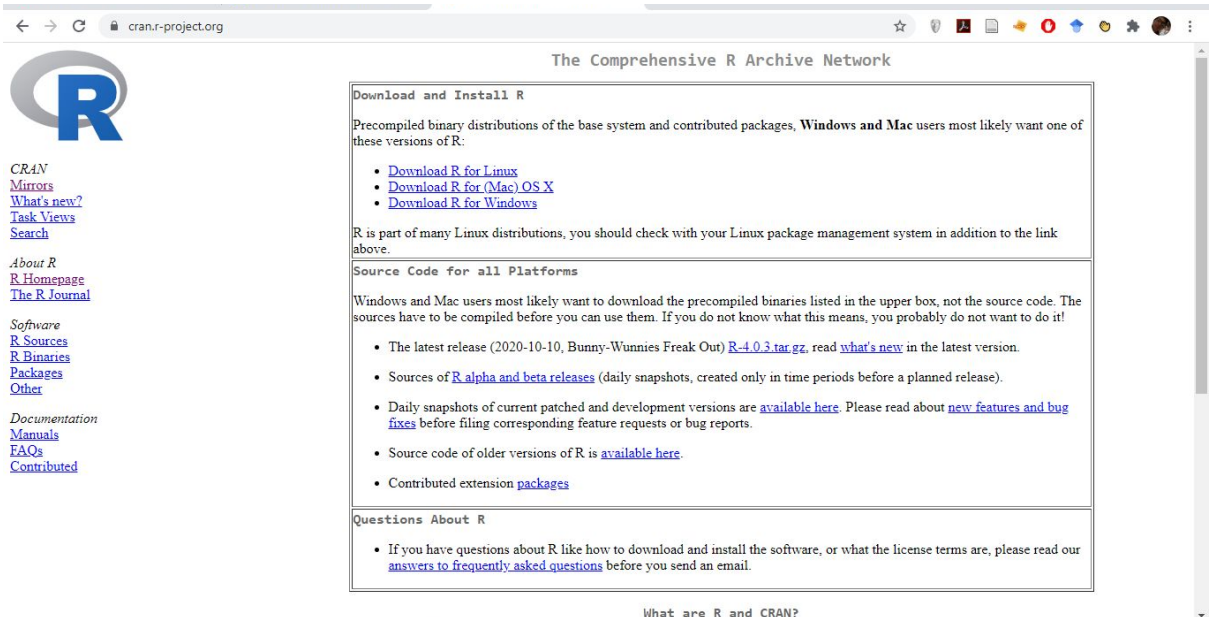
Se encuentran en:



Si uno hace click en install aparece lo siguiente

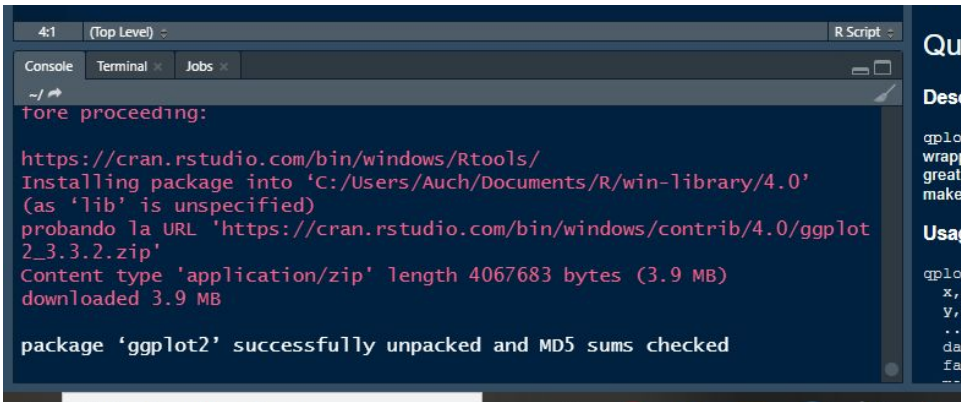


CRAN es the Comprehensive R Archive Network, que es donde se encuentran los paquetes que van generando la comunidad de R. El link https://cran.r-project.org/



Un ejemplo de esto es la siguientes líneas de código que sirven para descargar y activar el paquete ggplot2:

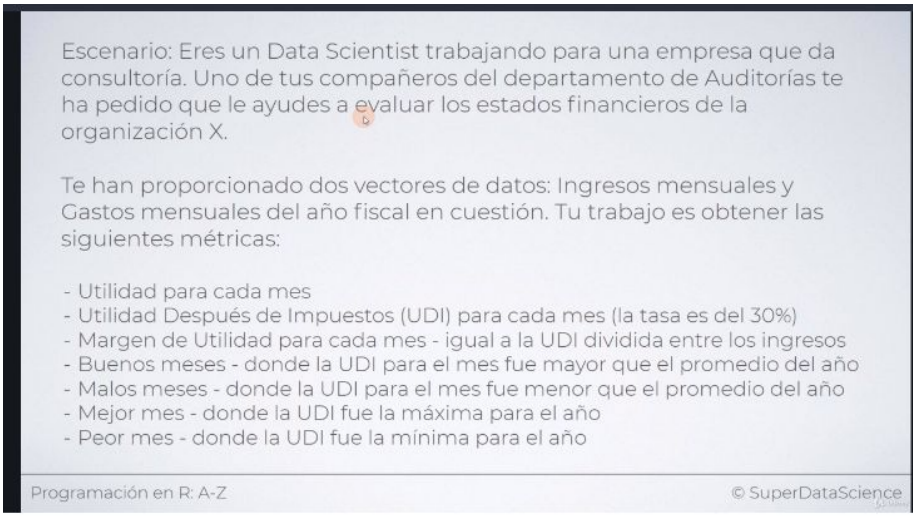
install.packages("ggplot2")
#Genera que ocurra esto en la consola y se descargue el paquete de CRAN

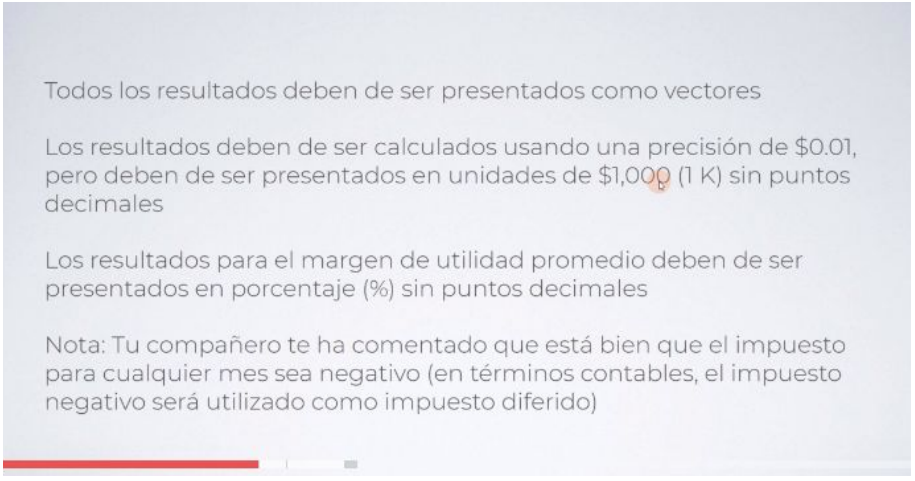


library(ggplot2)

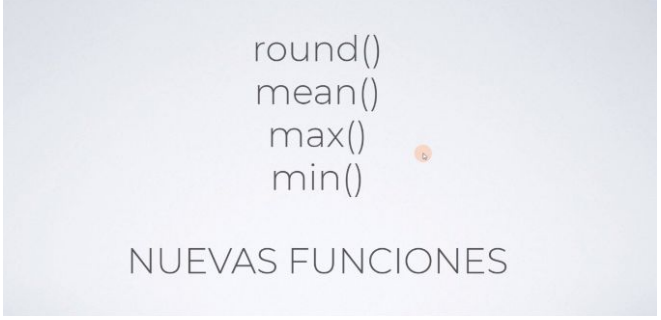
Análisis de estado financiero

Caso a resolver





Funciones extra a usar, revisar con ?



Resolución del ejercicio

```
#Datos del ejercicio
ingresos <- c(14574.49, 7606.46, 8611.41, 9175.41, 8058.65, 8105.44, 11496.28, 9766.09, 10305.32, 14379.96, 10713.97, 15433.50)
gastos <- c(12051.82, 5695.07, 12319.20, 12089.72, 8658.57, 840.20, 3285.73, 5821.12, 6976.93, 16618.61, 10054.37, 3803.96)

?round()
?min()
?max()
?mean()

#Calcular Utilidad como la Diferencia de Ingreso y Gasto
utilidad <- ingresos - gastos
utilidad

#Calcular el Impuesto como el 30% de la Utilidad y Redondear a 2 Puntos Decimales
impuesto <- round(0.30 * utilidad, 2)
impuesto

#Calcular la Utilidad Después de Impuestos o UDI
utilidad.despues.de.impuestos <- utilidad - impuesto
utilidad.despues.de.impuestos

#Calcular el Margen de Utilidad como Utilidad Después de Impuestos sobre Ingresos
#Redondear a 2 Puntos Decimales, Luego Multiplicar por 100 para obtener el %
margen.de.utilidad <- round(utilidad.despues.de.impuestos/ ingresos, 2) * 100
margen.de.utilidad

#Calcular el promedio para los 12 meses de la Utilidad Después de Impuestos
promedio.utilidad.despues.de.impuestos <- mean(utilidad.despues.de.impuestos)
promedio.utilidad.despues.de.impuestos
```

```
#Encuentra los Meses Utilidad Después de Impuestos por Encima de la Media
meses.buenos <- utilidad.despues.de.impuestos > promedio.utilidad.despues.de.impuestos
meses.buenos

#Los Meses Malos son el Opuesto a los Meses Buenos !
meses.malos <- !meses.buenos
meses.malos

#El Mejor Mes es Donde la Utilidad Después de Impuestos es Igual al Máximo
#Aquí compara capada valor de utilidad.después de impuestos con el máximo de ese vector
#Esto es gracias al reciclado de valores, el valor máximo se repite las 12 veces
mejor.mes <- utilidad.despues.de.impuestos == max(utilidad.despues.de.impuestos)
mejor.mes

#El Peor Mes es Donde la Utilidad Después de Impuestos es Igual al Mínimo
#Mismos principios que en el paso anterior
peor.mes <- utilidad.despues.de.impuestos == min(utilidad.despues.de.impuestos)
peor.mes

#Convierte Todos los Cálculos a Unidades de Mil Dólares
ingresos.1000 <- round(ingresos / 1000, 0)
gastos.1000 <- round(gastos / 1000, 0)
utilidad.1000 <- round(utilidad / 1000, 0)
utilidad.despues.de.impuestos.1000 <- round(utilidad.despues.de.impuestos/1000, 0)

#Imprime los Resultados
print(ingresos.1000)
print(gastos.1000)
print(utilidad.1000)
print(utilidad.despues.de.impuestos.1000)
print(margen.de.utilidad)
print(meses.buenos)
print(meses.malos)
print(mejor.mes)
```

```
print(peor.mes)

#Si se agrega el siguiente código se obtiene una tabla o matriz de los datos
M <- rbind(
  ingresos.1000,
  gastos.1000,
  utilidad.1000,
  utilidad.despues.de.impuestos.1000,
  margen.de.utilidad,
  meses.buenos,
  meses.malos,
  mejor.mes,
  peor.mes
)

M
```