



Finanzas en R

Notebook 01

Sebastián Egaña Santibáñez 

Nicolás Leiva Díaz 

Enlaces del profesor

-  <https://segana.netlify.app>
 -  <https://github.com/sebaegana>
 -  <https://www.linkedin.com/in/sebastian-egana-santibanez/>
-

Clase 01

Hello World

Un ejercicio clásico, es programar un saludo dentro de cada lenguaje de programación.

¿Cómo podemos hacer esto en R?

```
print("Hola")
```

Para hacerlo en C, deberíamos implementar lo siguiente:

```
#include <stdio.h>
int main() {
    // printf() displays the string inside quotation
    printf("Hello, World!");
    return 0;
}
```

```
# Voy a declarar un print
print("Hola")
```

```
[1] "Hola"
```

Variables

Cuando se le atribuye un nombre a un elemento dentro de un entorno de programación, a esto se le denomina **variable**. En Python tenemos lo siguiente:

```
x = 2
y = 5
xy = 'Hey'
```

Consultemos ahora la variable “xy”

```
xy = 'Hey'
```

Realicemos un print que combine lo anterior

```
print(x, y, xy)
```

```
2 5 Hey
```

Lo que en R se genera de la siguiente manera:

```
# Inserte código acá

x = 2
y = 5
xy = 'Hey'
```

```
print(paste(x,y,xy))
```

```
[1] "2 5 Hey"
```

Editando el separador

```
print(paste(x,y,xy, sep = ""))
```

```
[1] "25Hey"
```

Librerías en R

Dentro de R existen librerías, las que pueden ser definidas como colección de módulos; estos, no son más que bloques de códigos generados para un fin particular.

Ejemplos de librerías

Algunas de las librerías más conocidas son:

Dplyr

Librería orientada a la manipulación de datos.

Cheat Sheet: <https://nyu-cdsc.github.io/learningr/assets/data-transformation.pdf>

Ggplot2

Corresponde a la principal librería para graficar.

Cheat Sheet: <https://www.maths.usyd.edu.au/u/UG/SM/STAT3022/r/current/Misc/data-visualization-2.1.pdf>

Tidyverse

Biblioteca de librerías orientada a la manipulación de datos.

Dichas librerías son tres de una gran variedad de librerías que pueden ser utilizadas en R. Lo relevante es que para ser utilizadas se debe generar un proceso de instalación y llamado previo de dichas librerías.

Otra librerías se pueden encontrar en el siguiente enlace: <https://posit.co/resources/cheatsheets/>

Instalación y Llamado

Para la utilización de dichas librerías debemos seguir dos pasos:

1. Instalación.
2. Llamado

Para la instalación de librerías en R en modalidad local (en el computador) se debe utilizar el siguiente código:

```
install.packages("nombre del paquete")
```

Asumiendo que ya tenemos instalada la librería, cada vez que utilicemos un código de la librería debemos realizar el llamado. Esto se realiza de la siguiente manera:

```
install.packages("dplyr")
library(dplyr)

datos = iris

select(datos, Species)
```

En este caso, estamos haciendo un llamado a la librería **dplyr** para utilizar una de sus funciones que corresponde a **select**, la que permite seleccionar una columna de un set de datos llamado **iris**.

En el caso de utilizar una función de una librería no cargada, generará un error del siguiente tipo: **could not find function**. Por ejemplo:

```
vtree(FakeData,"Severity Sex", sameline=T)
```

```
install.packages("dplyr")
```

Realizamos el llamado

```
library(dplyr)
```

Utilizamos la librería

```
# iris corresponde a un set de datos precargado en R  
  
datos = iris  
select(datos, Species)
```

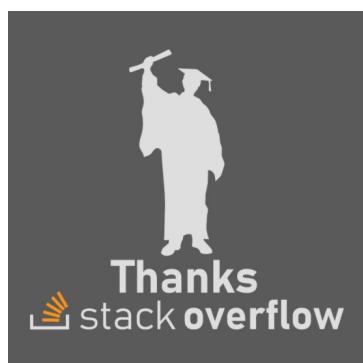
Por ejemplo, si no tenemos cargada una librería:

```
library(dplyr)  
  
detach("package:dplyr", unload = TRUE)  
  
datos = iris  
select(datos, Species)
```

Para utilizar sin cargar:

```
dplyr::select(datos, Species)
```

Ayuda en R



Help

Dentro de R, viene considerada la opción de buscar ayuda. Para esto, debemos utilizar el siguiente código:

```
?print
```

Páginas

Por lo general, una parte importante de programar es aprender dónde buscar ejemplos y preguntas relacionadas con la labor que queremos programar. Para esto, existen algunas páginas claves:

W3 Schools : <https://www.w3schools.com/r/default.asp>

R para ciencia de datos: <https://es.r4ds.hadley.nz/>

Curso free de R en Datacamp: <https://www.datacamp.com/courses/free-introduction-to-r>

Google

Una de las mejores opciones es utilizar Google; poner frases como: **Cómo hacer X en R** o **How to do X in R**

Asignación en R

Se tienen 3 maneras:

Utilizando =

```
x = 1  
print(x)
```

```
[1] 1
```

Hacia la izquierda, utilizando <-

```
y <- 2  
print(y)
```

```
[1] 2
```

Hacia la derecha, utilizando ->

```
3 -> z  
print(z)
```

[1] 3

Operadores matemáticos

En este caso, R es muy parecido a una calculadora (pero mucho más potente). Veamos cuáles son los operadores dentro de este lenguaje:

Símbolo	Tarea ejecutada
+	Suma
-	Resta
/	División
%%	Módulo
*	Multiplicación
%/%	División de enteros
//	Función de parte entera - Piso
** ó ^	Potencia

Veamos la siguiente diferencia entre operadores:

```
15 / 6
```

[1] 2.5

```
15 %% 6
```

[1] 3

Ejercicio 1

Defina x como un número cualquiera. Compruebe que x es par.

Solución

```
# De una manera básica

x = 10001

x %% 2

[1] 1

print(ifelse(x %% 2 == 0, "Par", "Impar"))

[1] "Impar"

# Generando una función para esto

es_par <- function(numero) {
  if (numero %% 2 == 0) {
    return(TRUE)
  } else {
    return(FALSE)
  }
}

# Ejemplo de uso
numero1 <- 10
numero2 <- 7

print(es_par(numero1)) # Devuelve TRUE, ya que 10 es par.

[1] TRUE

print(es_par(numero2)) # Devuelve FALSE, ya que 7 no es par.

[1] FALSE
```

Operadores relacionales

En el caso de operaciones de carácter comparativo, existen los siguientes operadores:

Símbolo	Tarea ejecutada
<code>==</code>	igual
<code>!=</code>	no igual
<code><</code>	menor que
<code>></code>	mayor que
<code><=</code>	menor o igual que
<code>>=</code>	mayor o igual que

Veamos la diferencia entre los siguientes operadores

```
z = 1
```

```
z == 2
```

```
[1] FALSE
```

```
z >= 1
```

```
[1] TRUE
```

Ejercicio 2

Considere el siguiente código

```
(z == 2) + (z >= 1)
```

```
[1] 1
```

Explique el resultado del código.

Operadores lógicos

Corresponde a los utilizados para álgebra booleana, es decir para escribir relaciones lógicas:

Símbolo	Tarea ejecutada
$x \mid y$	$x \text{ o } y$ son verdaderos
$x \& y$	$x \text{ y } y$ son verdaderos
$!x$	x no es verdadero
isTRUE(x)	x es verdadero

Input del usuario

Refiere a la posibilidad de generar interacciones con el usuario.

```
readline(prompt="Mi nombre es: ")
```

Otra posibilidad, es guardar ese input en una variable y después imprimirla utilizando print.

```
nombre = readline(prompt="Mi nombre es: ")
```

Para imprimir:

```
print(paste("Hola yo soy", nombre))
```

```
    readline(prompt="Mi nombre es: ")
```

```
Mi nombre es:
```

```
[1] ""
```

Dejandolo dentro de una variable

```
nombre = readline(prompt="Mi nombre es: ")
```

```
Mi nombre es:
```

Para imprimirlo

```
print(paste("Hola yo soy", nombre))
```

```
[1] "Hola yo soy "
```

Pipe operator

En programación, se relaciona con el encadenamiento de funciones. Puede estar representado por los siguientes símbolos |, >> o %>%

El tercer caso corresponde a R, y puede ser utilizado a través de shortcut ctrl + shift + m

Se implementa a través de la librería magrittr. Veamos un ejemplo clásico:

```
# Example of using the pipe operator in R
library(magrittr)

# Traditional approach without pipe operator
result <- sqrt(mean(log10(abs(c(-2, -10, 3, 4, 8)))))

# Using the pipe operator for the same computation
result <- c(-2, -10, 3, 4, 8) %>% abs() %>% log10() %>% mean() %>% sqrt()
```

Otra implementación:

```
quote(1:3 |> sum())
```

```
sum(1:3)
```

```
# sum(1:3)
```

```
quote(100 |> rnorm(n = 5))
```

```
rnorm(100, n = 5)
```

```
# rnorm(100, n = 5)
```

```
quote(split(x = iris[-5], f = iris$Species) |>
      lapply(min) |>
      do.call(what = rbind))
```

```
do.call(lapply(split(x = iris[-5], f = iris$Species), min), what = rbind)
```

Por lo general esta implementación es más restrictiva, por lo mismo, menos usada.

Ver Notebook ejercicios 01

Clase 02

Special constants en R: NA, NULL, Inf, -Inf, NaN

NA indica datos perdidos o indefinidos

```
5 + NA
```

```
[1] NA
```

```
is.na(5+NA)
```

```
[1] TRUE
```

Aplicado con la función promedio:

```
mean(c(1, 2, NA, 4, 5))
```

```
[1] NA
```

```
mean(c(1, 2, NA, 4, 5), na.rm = TRUE)
```

```
[1] 3
```

NULL indica un objeto vacío

```
10 + NULL
```

```
numeric(0)
```

```
is.null(NULL)
```

```
[1] TRUE
```

Inf y -Inf representan infinitos tanto positivo como negativo. Pueden ser producto de operaciones matemáticas

```
5/0
```

```
[1] Inf
```

```
is.finite(5/0)
```

```
[1] FALSE
```

```
is.infinite(5/0)
```

```
[1] TRUE
```

NaN (Not a Number) - el resultado no puede ser expresado o definido

```
0/0
```

```
[1] NaN
```

```
is.nan(0/0)
```

```
[1] TRUE
```

Tipos de objetos

Vectores

```
v1 <- c(1, 5, 11, 33)      # Numeric vector, length 4  
v1
```

```
[1] 1 5 11 33
```

```
v2 <- c("hello", "world")    # Character vector, length 2 (a vector of strings)  
v2
```

```
[1] "hello" "world"
```

```
v3 <- c(TRUE, TRUE, FALSE) # Logical vector, same as c(T, T, F)
v3
```

```
[1] TRUE TRUE FALSE
```

Combinar distintos elementos en un vector, genera coerción hacia el elemento más restrictivo

```
v4 <- c(v1,v2,v3,"boo")      # All elements turn into strings
v4
```

```
[1] "1"      "5"      "11"     "33"     "hello"   "world"   "TRUE"    "TRUE"    "FALSE"
[10] "boo"
```

```
class(v4)
```

```
[1] "character"
```

Los vectores operan por defecto de manera iterativa en R

```
v1 + v3
```

Warning in v1 + v3: longitud de objeto mayor no es múltiplo de la longitud de uno menor

```
[1] 2 6 11 34
```

```
v1 + 1
```

```
[1] 2 6 12 34
```

```
v1 * 2
```

```
[1] 2 10 22 66
```

```
v1 + c(1,7)
```

```
[1] 2 12 12 40
```

Operaciones matemáticas para vectores

```
sum(v1)
```

```
[1] 50
```

```
mean(v1)
```

```
[1] 12.5
```

```
sd(v1)
```

```
[1] 14.27118
```

```
cor(v1,v1*5)
```

```
[1] 1
```

Operaciones lógicas con vectores

```
v1 > 2
```

```
[1] FALSE TRUE TRUE TRUE
```

```
v1==v2
```

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

```
v1 !=v2
```

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

```
(v1>2) | (v2>0)
```

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

```
(v1>2) & (v2>0)
```

```
[1] FALSE TRUE TRUE TRUE
```

Selección de elementos en un vector

Seleccionando elementos de un vector:

```
v1[3]
```

```
[1] 11
```

```
v1[2:4]
```

```
[1] 5 11 33
```

```
v1[c(1,3)]
```

```
[1] 1 11
```

```
v1[c(T,T,F,F,F)]
```

```
[1] 1 5
```

```
v1[v1>3]
```

```
[1] 5 11 33
```

Cabe destacar, que la indexación se hace desde 1; esto no es un comportamiento clásico en lenguajes de programación.

Para añadir valores, solo se debe anotar la ubicación y los valores.

```
v1[6:10] <- 6:10  
v1
```

```
[1] 1 5 11 33 NA 6 7 8 9 10
```

Se puede asignar de manera directa una extensión a un vector

```
length(v1) <- 15  
v1
```

```
[1] 1 5 11 33 NA 6 7 8 9 10 NA NA NA NA NA NA
```

Factores

Son utilizados para almacenar datos categóricos.

```
eye.col.v <- c("brown", "green", "brown", "blue", "blue", "blue")  
eye.col.f <- factor(c("brown", "green", "brown", "blue", "blue", "blue"))  
eye.col.v
```

```
[1] "brown" "green" "brown" "blue" "blue" "blue"
```

```
eye.col.f
```

```
[1] brown green brown blue blue blue  
Levels: blue brown green
```

Como característica relevante, los factores si poseen una relación con un valor numérico:

```
levels(eye.col.f)
```

```
[1] "blue" "brown" "green"
```

```
as.numeric(eye.col.f)
```

```
[1] 2 3 2 1 1 1
```

```
as.numeric(eye.col.v)
```

Warning: NAs introducidos por coerción

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

```
as.character(eye.col.f)
```

```
[1] "brown" "green" "brown" "blue"  "blue"  "blue"
```

```
as.character(eye.col.v)
```

```
[1] "brown" "green" "brown" "blue"  "blue"  "blue"
```

Matrices y arreglos

Una matriz, es un vector con n dimensiones:

```
m <- rep(1, 20)
dim(m) <- c(5,4)
m
```

```
 [,1] [,2] [,3] [,4]
[1,]    1    1    1    1
[2,]    1    1    1    1
[3,]    1    1    1    1
[4,]    1    1    1    1
[5,]    1    1    1    1
```

Se crea una matriz usando `matrix()`:

```
m <- matrix(data=1, nrow=5, ncol=4)
m <- matrix(1,5,4)
dim(m)
```

```
[1] 5 4
```

```
m
```

```
 [,1] [,2] [,3] [,4]
[1,]    1    1    1    1
[2,]    1    1    1    1
[3,]    1    1    1    1
[4,]    1    1    1    1
[5,]    1    1    1    1
```

O combinando vectores:

```
m <- cbind(1:5, 5:1, 5:9)
m <- rbind(1:5, 5:1, 5:9)
m <- matrix(1:10,10,10)
m
```

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

Selección de elementos en una matriz:

Para seleccionar elementos dentro de una matriz, se opera de manera similar a con los vectores:

```
m[2,3] # Matrix m, row 2, column 3 - a single cell
```

```
[1] 2
```

```
m[2,] # The whole second row of m as a vector
```

```
[1] 2 2 2 2 2 2 2 2 2 2
```

```
m[,2] # The whole second column of m as a vector
```

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

```
m[1:2,4:6] # submatrix: rows 1 and 2, columns 4, 5 and 6
```

```
[,1] [,2] [,3]  
[1,] 1 1 1  
[2,] 2 2 2
```

```
m[-1,]
```

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

Utilizando operadores condicionais:

```
m[1,] == m[,1]
```

```
[1] TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
m > 3
```

```
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]  
[1,] FALSE  
[2,] FALSE  
[3,] FALSE  
[4,] TRUE  
[5,] TRUE  
[6,] TRUE  
[7,] TRUE  
[8,] TRUE  
[9,] TRUE  
[10,] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
m[m > 3]
```

```
[1] 4 5 6 7 8 9 10 4 5 6 7 8 9 10 4 5 6 7 8 9 10 4 5 6 7
[26] 8 9 10 4 5 6 7 8 9 10 4 5 6 7 8 9 10 4 5 6 7 8 9 10 4
[51] 5 6 7 8 9 10 4 5 6 7 8 9 10 4 5 6 7 8 9 10
```

Otro tipo de manipulaciones de matrices:

`t(m)`

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

`m %*% t(m)`

```
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] 10 20 30 40 50 60 70 80 90 100
[2,] 20 40 60 80 100 120 140 160 180 200
[3,] 30 60 90 120 150 180 210 240 270 300
[4,] 40 80 120 160 200 240 280 320 360 400
[5,] 50 100 150 200 250 300 350 400 450 500
[6,] 60 120 180 240 300 360 420 480 540 600
[7,] 70 140 210 280 350 420 490 560 630 700
[8,] 80 160 240 320 400 480 560 640 720 800
[9,] 90 180 270 360 450 540 630 720 810 900
[10,] 100 200 300 400 500 600 700 800 900 1000
```

`m * m`

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

```
[3,] 9 9 9 9 9 9 9 9 9 9  
[4,] 16 16 16 16 16 16 16 16 16 16  
[5,] 25 25 25 25 25 25 25 25 25 25  
[6,] 36 36 36 36 36 36 36 36 36 36  
[7,] 49 49 49 49 49 49 49 49 49 49  
[8,] 64 64 64 64 64 64 64 64 64 64  
[9,] 81 81 81 81 81 81 81 81 81 81  
[10,] 100 100 100 100 100 100 100 100 100 100
```

Arreglos de dos o más dimensiones

```
a <- array(data=1:18, dim=c(3,3,2))  
a <- array(1:18, c(3,3,2))  
a
```

```
, , 1
```

```
[,1] [,2] [,3]  
[1,] 1 4 7  
[2,] 2 5 8  
[3,] 3 6 9
```

```
, , 2
```

```
[,1] [,2] [,3]  
[1,] 10 13 16  
[2,] 11 14 17  
[3,] 12 15 18
```

Para subseleccionar elementos:

```
a[1,3,2]
```

```
[1] 16
```

Listas

Corresponden a colección de objetos:

```
11 <- list(boo=v1,foo=v2,moo=v3,zoo="Animals!")
12 <- list(v1,v2,v3,"Animals!")

13 <- list()
14 <- NULL
```

Acceder a elementos en una lista:

```
11["boo"]
```

```
$boo
[1] 1 5 11 33 NA 6 7 8 9 10 NA NA NA NA NA NA
```

```
11[["boo"]]
```

```
[1] 1 5 11 33 NA 6 7 8 9 10 NA NA NA NA NA NA
```

```
11[[1]]
```

```
[1] 1 5 11 33 NA 6 7 8 9 10 NA NA NA NA NA NA
```

```
11[[1]][2]
```

```
[1] 5
```

```
11$boo
```

```
[1] 1 5 11 33 NA 6 7 8 9 10 NA NA NA NA NA NA
```

Añadir elementos a una lista:

```
13[[1]] <- 11
14[[3]] <- c(22, 23)
11[[5]] <- "More elements!"
11[[8]] <- 1:11
11$Something <- "A thing"
```

Data Frames

Un dataframe es un caso especial de lista, utilizada para guardar las tablas de datos. Posee la estructura clásica fila columna.

Nota: Mientras en un comienzo se usan `data.frame`, más adelante al trabajar con `tidyverse` se utilizan `tibble` que corresponde a un concepto evolucionado de dataframe; de manera más específica corrige algunos comportamientos propios del dataframe, al no interpretarse los strings como factor por defecto y no existir `rownames`.

Creando un dataframe:

```
dfr1 <- data.frame( ID=1:4,  
                      FirstName=c("Jesper","Jonas","Pernille","Helle"),  
                      Female=c(F,F,T,T),  
                      Age=c(22,33,44,55) )  
  
dfr1$FirstName
```

```
[1] "Jesper"   "Jonas"    "Pernille" "Helle"
```

Para acceder a los elementos de un dataframe se debe tener en claro dos cosas: `dfr[row, column]`, donde las para las filas se puede acceder por número o condición y a las columnas por número o nombre. De manera alternativa para las columnas `dfr$column`

```
dfr1[1,]
```

```
ID FirstName Female Age  
1  1      Jesper  FALSE  22
```

```
dfr1[,1]
```

```
[1] 1 2 3 4
```

```
dfr1$Age
```

```
[1] 22 33 44 55
```

```
dfr1[1:2,3:4]
```

```
Female Age
1 FALSE 22
2 FALSE 33
```

```
dfr1[c(1,3),]
```

```
ID FirstName Female Age
1 1 Jesper FALSE 22
3 3 Pernille TRUE 44
```

Para encontrar los nombres de todos quienes son mayores de 30 años:

```
dfr1[dfr1$Age>30,2]
```

```
[1] "Jonas"     "Pernille"  "Helle"
```

Para obtener la media de las edades de las mujeres:

```
mean(dfr1[dfr1$Female==TRUE,4])
```

```
[1] 49.5
```

Tidyverse

Como parte de los desarrollos de Hadley Wickham, la librería **tidyverse**, corresponde a un ecosistema orientado a las ciencias de datos, con paquetes que comparten la orientación de trabajo en relación a los datos.

Por ejemplo, permiten la utilización del pipe operator.

Tibbles

Corresponde a la versión de dataframe en tidyverse. Se trabaja igual, pero con ciertas mejoras. Se pueden crear de tres maneras:

1. Crear usando `tibble()`
2. Usando explicitamente `as_tibble()` en una tabla.
3. Cuando se aplica una función de `dplyr` en un dataframe, se convierte a tibble.

Vemos la tabla precargada `iris`

```
| iris
```

Convertirlo en un elemento tibble

```
| head(as_tibble(iris))
```

```
# A tibble: 6 x 5
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
    <dbl>       <dbl>       <dbl>       <dbl>   <fct>
1       5.1       3.5       1.4       0.2   setosa
2       4.9       3.0       1.4       0.2   setosa
3       4.7       3.2       1.3       0.2   setosa
4       4.6       3.1       1.5       0.2   setosa
5       5.0       3.6       1.4       0.2   setosa
6       5.4       3.9       1.7       0.4   setosa
```

Comparamos con la creación de un data frame:

```
| tibble(x = 1:5, y = 6:10)
```

```
# A tibble: 5 x 2
  x     y
  <int> <int>
1 1     6
2 2     7
3 3     8
4 4     9
5 5    10
```

```
| data.frame(x = 1:5, y = 6:10)
```

```
x  y
1  6
2  7
3  8
4  9
5 10
```

Importación y exportación de datos

Importación

A pesar de que existen múltiples orígenes para la obtención de archivos, podemos pensar en un Excel como el típico archivo. Para esto, utilizamos `readxl` que viene con `tidyverse`. Podemos ver la librería en la siguiente [página](#)

Exportación

Exportar en Excel

Para esto existen las siguientes librerías:

- `xlsx`
- `openxlsx`
- `writexl`

En donde las dos primeras pueden generar cierto conflicto si se intenta utilizar de manera conjunta, generalmente asociado a Java, y la tercera no tiene requerimiento de Java.

Dependiendo de la que se utilice cambia la forma de generar esto.

```
library("xlsx")
# Write the first data set in a new workbook
write.xlsx(USArrests, file = "myworkbook.xlsx",
           sheetName = "USA-ARRESTS", append = FALSE)
```

Exportar en Latex

Otra forma de exportar puede estar relacionada con tablas en formato Latex:

```
library(stargazer)  
stargazer(attitude)
```

Por ejemplo en base a tres regresiones (dos lineales y una probit):

```
linear.1 <- lm(rating ~ complaints + privileges + learning + raises + critical,  
data=attitude)  
  
linear.2 <- lm(rating ~ complaints + privileges + learning, data=attitude)  
  
attitude$high.rating <- (attitude$rating > 70)  
  
probit.model <- glm(high.rating ~ learning + critical + advance, data=attitude,  
family = binomial(link = "probit"))
```

A pesar de esto, desde mi opinión stargazer no es una buena opción para exportar tablas en latex. Veamos una solución a esto:

```
library(gtsummary)  
library(kableExtra)  
  
t1 <-tbl_regression(linear.1)  
  
as_kable_extra(t1, format = "latex") %>%  
  save_kable("modelo.tex")
```

Combinando los tres modelos:

```
t1 <-tbl_regression(linear.1)  
  
t2 <-tbl_regression(linear.2)  
  
t3 <-tbl_regression(probit.model)  
  
tbl_merge(  
  bls = list(t1, t2, t3),  
  tab_spanner = c("**Regresion 1**", "**Regresion 2**", "**Probit**")  
)  
  
as_kable_extra(t1, format = "latex") %>%
```

```
| save_kable("modelo_3.tex")
```