Introducción

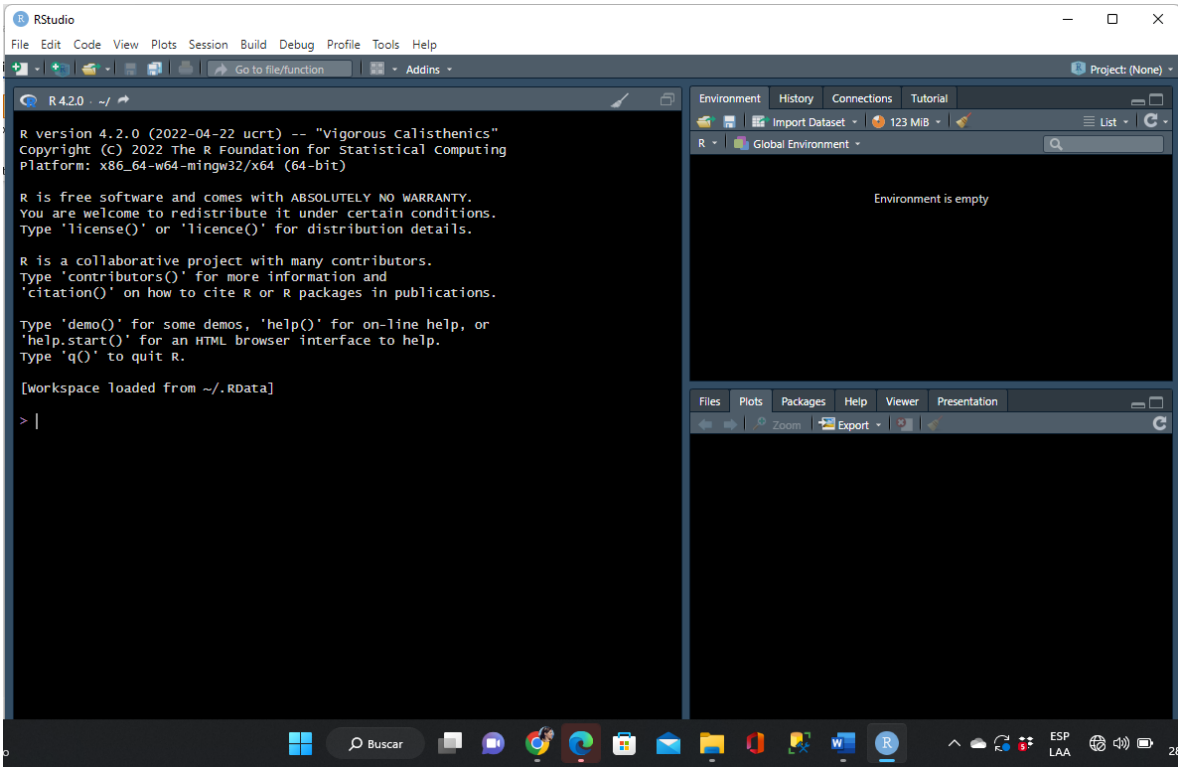
En el siguiente material abordaremos:

- *Primeros pasos: comandos, objetos, funciones, script.
- *Clases de Objetos: numéricos, alfanuméricos, vectores, matrices, listas, Data Frames
- *Instalación de paquetes y librerías.

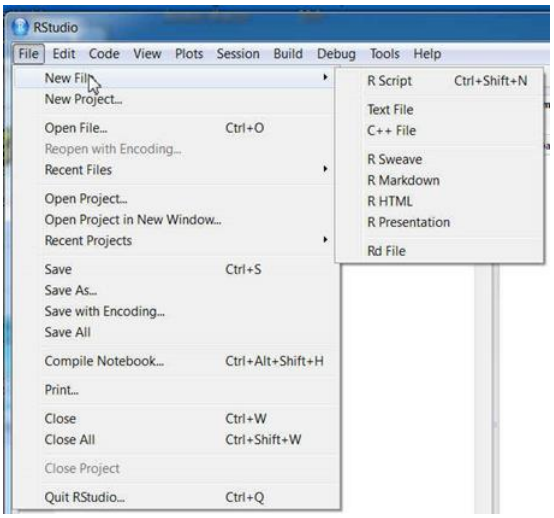
01. Primera sesión en R

Para iniciar una sesión en R solo tenemos que abrir RStudio, por ejemplo, desde *Inicio > Todos los programas > RStudio >* ***RStudio.***

La primera vez que abramos RStudio veremos solo 3 ventanas:
Si bien podemos trabajar directamente en la consola, donde aparece el cursor >, es mejor trabajar en la ventana de edición de texto, en un script.



Desafío 1: Crear y guardar un script en R
Haga clic en el botón **"File"** del menú y, a continuación, **"New File"**
Seleccione "**R Script**"



Haga clic en el ícono  y guarde el archivo como "Clase_1"

01.01 Directorio de trabajo

El **Directorio de Trabajo** es por *default* **el lugar de donde se leen y guardan los archivos que utilicemos** en la sesión.

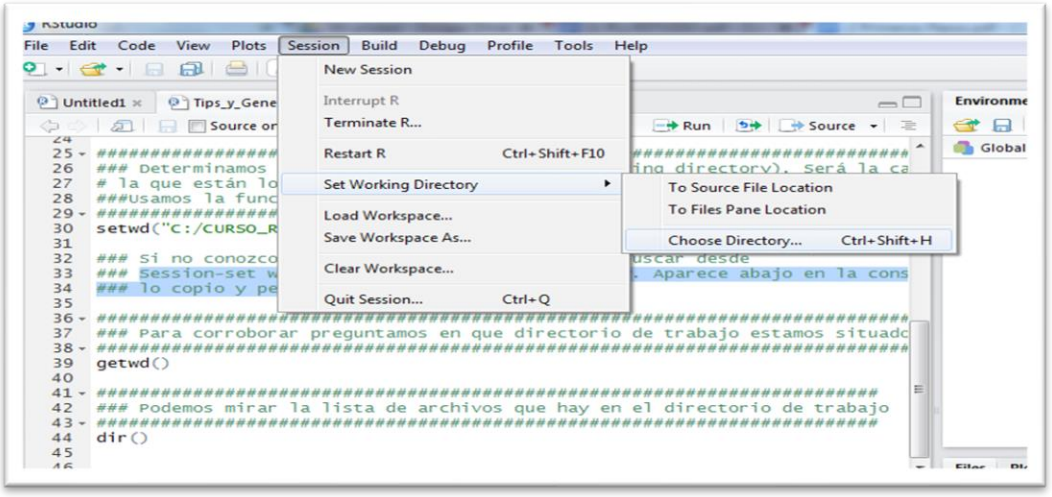
El directorio de trabajo actual se muestra dentro de la barra de título del bloque Consola. También podemos obtenerlo con el comando **getwd()** del inglés, *get working directory*.

```
> getwd()
```

Para cambiar el directorio de trabajo, se utiliza la función **setwd()**, utilizando como argumento la dirección que queremos como nuevo directorio de trabajo.

```
> setwd("C://cursoR")
```

Otra forma de designar el Directorio de Trabajo es desde la barra de trabajo, apretando el botón "**Session" > "Set working directory" > "Choose Directory"**. Ahí podemos elegir la carpeta que queremos definir como el directorio de trabajo. La confirmación de esta selección, la veremos en la consola.



¿Qué puede salir mal?

Escribir el directorio entre comillas. Algunas veces se presentan diferencias en la encodificación de los textos. Por tanto, conviene asegurarse que sean las comillas correctas: ¡las que se encuentran sobre el número 2!

Seleccionar una carpeta que exista y escribir correctamente el nombre. RStudio tiene problemas con nombres de carpetas que utilicen espacios, del tipo "Mis Documentos" o "Curso de R". Evítenlas por vuestra salud mental.

Directorios cortos e intuitivos. Háganse la vida más fácil, trabajen con directorios con nombres cortos.

Las barras. Si están trabajando en Windows, R suele preferir el uso de // en lugar de una simple / para separar los directorios.

01.02 R como calculadora

La cosa más simple que podemos hacer en R, es realizar cálculos aritméticos. Si ingresamos en la consola:

```
> 5 + 2
```

R nos devolverá la respuesta, precedida de un "[1]". No te preocupes por esto por ahora, te lo explicaremos más tarde. Por ahora piensa en ello como una salida indicadora.

```
[1] 7
```

Si escribes un comando incompleto, R esperará a que lo completes:

```
> 5 +
```

```
+
```

Cada vez que pulses **Enter** y la sesión R muestre un "+" en lugar de un ">", significa que está esperando a que completes el comando escrito. Si deseas cancelarlo, simplemente pulse **"Esc"** y RStudio le devolverá la indicación ">".

Al usar R como calculadora, el orden de las operaciones es el mismo que habrías aprendido en la escuela. De la más alta a la más baja precedencia:

Paréntesis: ( )
Exponentes: ^ o **
Divide: /
Multiplica: *
Sumar: +
Restar: -

Veamos algunos ejemplos:

División:

```
> 2 / 3
```

```
[1] 0.6666667
```

Exponente:

```
> 5 ^ 2
```

```
[1] 25
```

Probemos algo un poquito más difícil:

```
> ((10-3)*(4 + 8)) ^ 3
```

```
[1] 592704
```

Estas son las operaciones aritméticas, pero también tenemos otros tipos de operaciones, como las operaciones relacionales y las operaciones lógicas. Esto lo veremos en otro documento. ¡¡¡No te lo pierdas!!!

En lugar de escribir las funciones en la consola, podemos hacerlo en un script que luego guardaremos.

Código de colores de RStudio
En la consola:
El código que escribimos aparece en azul
Los resultados aparecen en **negro.**
En la zona del Editor de Código:
Los comentarios y cadenas de caracteres se ven de color verde
En **negro** se visualizan las funciones y los operadores.
En azul todo lo que sea numérico.

01.03. Script

Un script es un archivo que contiene un listado secuencial de líneas de código. Trabajar sobre un script nos permite guardar lo que vamos creando, volver a utilizarlo y compartirlo con otras personas.

Ya vimos cómo crear un script en el Desafío 1, ahora **comenzaremos a trabajar en él**.

Las líneas del script están numeradas a la izquierda para facilitar el trabajo. Para comenzar una nueva línea debemos presionar la tecla **Enter**. Al hacerlo veremos cómo se crean nuevas líneas numeradas.

Si queremos ejecutar el script que tenemos escrito, presionamos la tecla 

, ejecutando solo la línea donde está el cursor. También, podemos seleccionar varias líneas y apretar la combinación de teclas **ctrl+Enter** y obtendremos el mismo resultado.

Tras esto, **observaremos la salida en la consola** y también los cambios que se dan en la ventana **Environment**, donde aparecerán los objetos que vayamos creando y manipulando.

Desde el editor de código se puede modificar o incorporar alguna línea al script que estamos trabajando, recordando que al hacer un cambio debemos volver a ejecutar a partir de esa o esas líneas todo el código para que se realice la modificación.

01.04. # Comentar

A la hora de programar es muy importante la documentación, ya que permite entender lo que se hace en el script, ya sea para la reutilización del código como para compartir. Un comentario es un texto libre, en el idioma que se quiera, con anotaciones importantes que ayudan a la comprensión del script.

Es aconsejable que cuando comencemos un script en R, **dejemos el detalle de la autoría, la fecha y los objetivos de ese script**. A su vez, a medida que vayamos avanzando en el desarrollo de nuestro trabajo, conviene incorporar títulos o divisiones y comentarios que ayuden a entender lo escrito en lenguaje R.

Los **comentarios se escriben en R empezando la línea de código con el símbolo #**.

Todo lo escrito luego de un # es entendido como un comentario y no será tenido en cuenta cuando se ejecute el código.

```
> # esto es un comentario que ocupa toda la línea
> getwd()       # este un comentario dentro de la misma línea
```

Desafío 3: Introduce tus datos en el script
Abra el script **clase_1.R** creado en el Desafío 1
Escriba en la primera línea: # Certificación en Data Science
Presiona **Enter**
Escriba en la segunda línea: # Nombre: ...
Presiona **Enter**
Escriba en la tercera línea: # Fecha: ...
Presiona **Enter**

Haga click en el ícono  para guardar los cambios

01.05. Clases de Objetos I

Si queremos realizar cosas complejas, ya no simples cálculos aritméticos, será necesario crear objetos. **Los objetos son una forma de guardar información en la memoria activa**. Podemos, por ejemplo, guardar un resultado que nos interese para continuar utilizándolo mientras dure la sesión.
¡Veamos cómo funciona!
Si escribimos en la consola:

```
> numero <- 24
```

y luego apretamos **Enter**
Veremos que en la pestaña **Environment** aparece una variable llamada **numero**.
Si escribimos en la consola:

```
> numero
```

R nos devolverá:

```
[1] 24
```

Es decir, R guardó el número 24 en una variable llamada **numero**.

La combinación del signo menor con el guión medio (<-) es llamado el **símbolo de asignación** y es característico de R. Cuando usamos este símbolo es porque asignamos lo que viene tras la flecha a lo que se sitúa delante, tal como indica su dirección. Y se lee: al objeto *numero* le asignamos el número *24*.

Una vez creado el objeto **numero**, si queremos hacer procedimientos con este solo es necesario escribir el nombre de la variable.

Ejemplo:

```
> numero * 234
```

```
[1] 5616
```

Cuando los datos numéricos son muy grandes se pueden expresar en notación exponencial:

```
> N <- 6.02e23
> N
```

```
[1] 6.02e23
```

A los objetos podemos aplicar las operaciones aritméticas, relacionales y lógicas.

¿Cómo construir el nombre de los objetos?

Para construir el nombre de los objetos hay muchas libertades y algunas restricciones:
Se pueden usar combinaciones de letras, números y algunos símbolos, como el punto . , el guión medio - y el guión bajo _ .
Los nombres no pueden comenzar con un número.
R distingue minúsculas de mayúsculas. Por ejemplo: Casa y casa pueden ser 2 variables distintas
R tiene una lista de nombres y símbolos reservados que no pueden utilizarse para crear nuevos objetos. Por ejemplo: "function", "if", !

01.06. Vectores

Parece tonto guardar un número de 2 cifras dentro de una variable.
Así que probemos con una clase más compleja, como por ejemplo un **vector** de números.
Para eso, debemos escribir:

```
> vector <- c(2, 5, 6, 7)
```

Cada vez que queremos escribir un vector, debemos escribir los elementos entre paréntesis, separados con comas, y anteponiendo la letra *c* de concatenar.
Con un vector de números podemos también hacer operaciones aritméticas:

```
> vector * 2
```

```
[1]  4 10 12 14
```

Así como tenemos vectores de números, también podemos tener vectores de caracteres:

```
> vector_2 <- c("Ana P.", "Lucas J.", "Pablo E.")
```

Entonces, **un vector es un conjunto de valores** (números o símbolos), **todos del mismo tipo ordenados** de la forma (elemento 1, elemento 2, … , elemento *n*), siendo *n* la longitud o tamaño del vector.

Para acceder a los elementos de un vector escribimos entre corchetes [ ] el lugar del elemento que nos interesa. Por ejemplo,

```
> vector_2[3]  #nos devuelve el objeto en el lugar 3 del vector
```

```
[1] “Pablo E.”
```

01.07. Atributos de los objetos

Cada objeto pertenece a una **clase** y eso determina el tipo de **atributos** que éste tiene y las operaciones que podemos realizar con él.

Hay cuatro tipos principales de clases:

- numeric - números reales.
- character - caracteres
- integer - enteros
- complex - número complejo. Ej: z <- 1 + 2i
- logical - lógico o booleanos, del tipo binario (FALSE [Falso] or TRUE [Verdadero])

Para conocer **la clase de los elementos** que se encuentran almacenados en el objeto.

Para conocer la clase de objeto se usa la función **class()**,

```
> class(vector)
```

```
[1] "numeric"
```

Si en cambio realizamos:

```
> class(vector_2)
```

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

01.08. Secuencias y repeticiones

Para crear un vector donde todos los valores sean 0 existe la función **numeric()**. Sólo debemos detallar cuál es el largo del vector que queremos construir.

```
> x <- numeric(5)
> x
```

```
[1] 0 0 0 0 0
```

Para escribir una secuencia de números tenemos la función **seq()**. Debemos detallar desde y hasta dónde (from = , to =) queremos construir la secuencia de números y con cuál es el intervalo (by =)

```
> x <- seq( from = 1, to = 6, by = 2)
> x
```

```
[1] 1 3 5
```

Esta función también permite generar un vector donde el intervalo numérico que especifiquemos sea dividido en segmentos iguales:

```
> x <- seq( from = 1, to = 6, length = 6)
> x
```

```
[1] 1 2 3 4 5 6
```

Otra función interesante es **rep()** que nos permite crear un vector del largo que queramos repitiendo el número o caracter que le indiquemos:

```
> x <- rep( “casa” , time = 4)
> x
```

```
[1] "casa" "casa" "casa" "casa"
```

Algo que antes no aclaramos pero que es importante notar, es que las variables pueden sobreescribirse. Es decir, podemos asignar nuevos valores a variables ya existentes. El valor que asignamos la primera vez se pierde, y la variable adquiere uno nuevo.

Desafío 4: Vectores y series
Evalúa qué sucede cuando en la consola ejecutamos: 1:5
¿Y si en su lugar hacemos: 5:1?
Genera un vector llamado **desafio_3** que contenga la secuencia de números entre el 4 y el 13
Ejecuta las sentencias:
> desafio_3[1]
> desafio_3[1:3]
> desafio_3[-1]

¿Qué sucedió en cada caso?

01.09. Matrices

Una **matriz es un vector bidimensional,** que se visualiza como una tabla conformada por columnas y filas ordenadas y donde **todos los elementos son del mismo tipo**.
Se pueden crear a partir de un vector solamente añadiendo información sobre el número de filas y columnas de la matriz. Entonces, es similar a un vector, pero contiene adicionalmente el atributo de dimensión.

Los datos en las matrices deben ser todos del mismo tipo, ya sean estos numéricos, o de tipo carácter o lógico.

La sintaxis general de la orden para crear una matriz es la siguiente:

matrix(data, nrow, ncol, byrow=F)

donde:

data	datos que forman la matriz
nrow	número de filas de la matriz
ncol	número de columnas de la matriz
byrow	Los datos se colocan por filas o por columnas según se van leyendo. Por defecto se colocan por columnas.

Una matriz se puede crear usando la función ***matrix ()***
La dimensión de la matriz se puede definir pasando el valor apropiado para argumentos *nrow* y *ncol*. No es necesario proporcionar valor para ambas dimensiones. Si se proporciona una de las dimensiones, la otra se deduce de la longitud de los datos.

Podemos nombrar las filas y columnas de la matriz durante la creación agregando una lista de 2 elementos al argumento *dimnames*

Veamos algunos ejemplos:

Si al crear la matriz no especificamos el número de filas y columnas, se entiende que se desea crear un vector columna. Podemos revertir y rellenar en fila pasando TRUE en el argumento *byrow*

```
> matrix(1:6)
```

```
     [,1]
[1,]    1
[2,]    2
[3,]    3
[4,]    4
[5,]    5
[6,]    6
```

Al especificar el número de filas, los números de la secuencia 1:6 se acomodan por columnas.

```
> matrix(1:6, nrow = 2)
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Mientras que al introducir el parámetro ***byrow*** = TRUE se puede leer por filas:

```
> matrix( 1:6, nrow = 2, byrow = TRUE)
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Otra forma de crear matrices es mediante el uso de las funciones **cbind()** y **rbind()**

```
> cbind(c(1,2,3),c(4,5,6))
```

Finalmente, también puede crear una matriz a partir de un vector estableciendo su dimensión usando *dim()*.

```
> x <- c(1,2,3,4,5,6)
> x
[1] 1 2 3 4 5 6
> class(x)
[1] "numeric"
> dim(x) <- c(2,3)
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> class(x)
[1] "matrix"
```

01.10. Indexación en matrices

Para seleccionar los elementos de una matriz, podemos hacerlo introduciendo el número de fila y columna entre [ ].

Veamos esto con un ejemplo. Para ello, creamos una matriz x:

```
> x <- matrix( 1:6, nrow = 3)   # Creamos una matriz 2 x 3
> x
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Podemos obtener un elemento especificando la [fila, columna]

```
> x[1,2]   #Se muestra el elemento de fila 1 columna 2
```

```
[1] 4
```

Si dejamos libre el lugar de filas, R nos devuelve la columna entera:

```
> x[ ,2]   #Se muestra toda la columna 2
```

```
[1] 4 5 6
```

01.11. Algunas funciones sobre matrices

A continuación, les dejamos una lista de las funciones más utilizadas sobre matrices:

dim()	devuelve las dimensiones de una matriz
dimnames()	devuelve el nombre de las dimensiones de una matriz
colnames()	devuelve el nombre de las columnas de una matriz
rownames()	devuelve el nombre de las filas de una matriz
mode()	devuelve el tipo de datos de los elementos de una matriz
length()	devuelve el número total de elementos de una matriz

Para aplicarlas, solo debemos introducir el nombre de la matriz:

```
> mode( x )      # Tipo de datos de la matriz x
```

```
[1] "numeric"
```

01.12. Añadir columnas o filas

Es habitual en la práctica con matrices que queramos añadir filas o columnas. Para eso podemos utilizar las funciones ***cbind()*** y ***rbind()***.

La función ***cbind()*** añade una columna a la matriz en el orden que escribamos la función. En el siguiente ejemplo, se incorpora una columna de ceros a la derecha:

```
> cbind(x,c(0,0,0))     # Se añade una columna de ceros a x
```

```
     [,1] [,2] [,3]
[1,]    1    4    0
[2,]    2    5    0
[3,]    3    6    0
```

Por su parte, la función ***rbind()*** añade una fila a la matriz:

```
> rbind(x, c(0,0))       # Se añade una fila de ceros a x
```

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
[4,]    0    0
```

Es fundamental para estas funciones que coincidan las dimensiones de la matriz con el vector a agregar.

01.13. Asignando nombre a las filas y columnas de las matrices

Para ello utilizaremos las funciones ***colnames()***, ***rownames()*** *y* ***dimnames()***. Las dos primeras funciones permiten asignar nombres a las columnas y a las filas, respectivamente. Mientras que la función ***dimnames()*** permite pasar una lista con los nombres de las filas y columnas.

Para observar esto, vamos a crear una matriz con tres personas y su edad, altura y peso como variables. Ejecuta el siguiente código en tu consola y observa los cambios que van sucediendo en la matriz datos:

```
> datos <- matrix(c(20, 65, 174, 22, 70, 180, 19, 68, 170),
                  nrow = 3, byrow = T)   #Se crea una matrix 3x3
> datos
> colnames(datos) <- c("edad","peso","altura") # nombres a col
> datos
> rownames(datos) <- c("paco","pepe","kiko") # nombres a filas
> datos
```

Ahora observa el siguiente script:

```
> datos <- matrix(c(20, 65, 174, 22, 70, 180, 19, 68, 170),
                  nrow = 3, byrow = T)   #Se crea una matrix 3x3
> datos
> dimnames(datos) <- list( c("paco", "pepe", "kiko"),
                           c("edad", "peso", "altura"))
```

```
> datos
```

Ahora podemos acceder también a los elementos de la matriz usando los nombres:

```
> datos[,"edad"]          # Edades de todas las personas
```

```
paco pepe kiko
  20   22   19
```

01.14. Dataframe

El término "dataframe" podría traducirse al español como Hoja de datos y es la clase especial que tiene R para el trabajo con datos dispuestos en tablas. Normalmente, cuando se realiza un estudio estadístico, la información se organiza precisamente en un dataframe: una hoja de datos, en los que cada fila corresponde a un sujeto y cada columna a una variable.

La estructura de la clase **data.frame** es muy similar a la de una matriz. La principal diferencia es que una matriz sólo admite valores numéricos, mientras que **en un dataframe podemos incluir otras clases de datos, como alfanuméricos (*character)* o lógicos**.

Se puede pensar un **data.frame** como un grupo de vectores ordenados en forma tabular, donde cada vector conforma una columna del dataframe. De hecho podemos construir dataframes de esta forma:

```
> id <- 1:4
> edad <- c(23, 43, 12, 65)
> sexo <- c("M", "F", "F", "M")
> trabaja <- c(T, T, F, F)
> datos <- data.frame( id, edad, sexo, trabaja)
```

En las primeras cuatro líneas del código creamos 4 vectores con 4 valores cada uno. Los vectores id y edad son numéricos, el vector sexo es de tipo character y trabaja es lógico. Luego de creados utilizamos la función *data.frame()* para construir el dataframe uniendo los vectores en el orden que deseamos.

El resultado de esta operación lo podemos imprimir en pantalla:

```
> datos
```

```
  id edad sexo trabaja
1  1   23    M    TRUE
2  2   43    F    TRUE
3  3   12    F   FALSE
4  4   65    M   FALSE
```

Si queremos ver la tabla en una pestaña dentro de la zona del Editor de código de RStudio podemos aplicar la función *View()*,

```
> View(datos)
```

que es lo mismo que hacer click sobre el nombre de la variable datos en la ventana de *Environment* de RStudio:

id	edad	sexo	trabaja
1	23	M	TRUE
2	43	F	TRUE
3	12	F	FALSE
4	65	M	FALSE

Si usamos la función *class()* nos confirmará que estamos frente a un dataframe.

```
> class(datos)
```

```
[1] "data.frame"
```

Para visualizar cómo se compone el objeto datos podemos usar una función llamada *str()* que devuelve la estructura interna de cualquier objeto en R.

```
> str(datos)
```

```
'data.frame': 4 obs. of 4 variables:
$ id : int 1 2 3 4
$ edad : num 23 43 12 65
$ sexo : Factor w/ 2 levels "F","M": 2 1 1 2
$ trabaja: logi TRUE TRUE FALSE FALSE
```

La función **str()** nos informa que el dataframe datos:

- Posee 4 observaciones y 4 variables.
- La variable id tiene formato int (integer) y los valores 1,2,3,4
- La variable edad es numérica (num) y contiene los valores 23, 43, 12, 65.

- La variable sexo es de tipo factor con 2 niveles (“F” y “M”) y los valores 2,1,1,2
- La variable trabaja es de tipo lógica con valores TRUE, TRUE, FALSE, FALSE

Observamos que al vector sexo, que era de tipo character, lo transformó en factor y automáticamente definió los dos niveles encontrados. Esto lo realiza la función *data.frame()* por default, y podemos cambiarlo mediante el argumento stringsAsFactors = FALSE.

Todos los vectores que constituyen la hoja de datos deben tener la misma longitud.

Si queremos construir un dataframe con vectores de distintos tamaños, el proceso arroja un error.

```
> data.frame( nombres = c("julia", "jorge", "javier"),
            notas = c(4, 7, 9, 5))
```

```
Error in data.frame(nombres = c("julia", "jorge", "javier"),
notas = c(4,  :
arguments imply differing number of rows: 3, 4
```

Otras funciones que podemos aplicar en los dataframe son la función *dim()* que nos devuelve un vector con las dimensiones del dataframe, en número de filas y columnas, y las funciones *ncol() y nrow()* que nos ofrece la misma información en forma separada, columnas y filas respectivamente.

```
> dim(datos)
```

```
[1] 4 4
```

Si queremos ver la cabecera del dataframe (las primeras 6 líneas), podemos ejecutar la función *head()*.

Para este ejemplo, utilizaremos un dataframe de prueba que viene instalado en R:

```
> data(women)
> head(women)
```

```
  height weight
1     58    115
2     59    117
```

```
3    60   120
4    61   123
5    62   126
6    63   129
```

Y si queremos ver "cola" o parte final de la hoja de datos, utilizamos la función *tail()*.

```
> tail(women)
```

```
   height weight
10     67    142
11     68    146
12     69    150
13     70    154
14     71    159
15     72    164
```

01.15. Selección de variables

Cuando necesitemos llamar al contenido de alguna columna o variable del dataframe podemos utilizar la notación:

<nombre del objeto dataframe>$<nombre de la variable>

Por ejemplo, si queremos mostrar el contenido de la variable *sexo* del objeto datos hacemos:

```
> datos$sexo
```

```
[1] M F F M
Levels: F M
```

R nos muestra los 4 valores que tiene la variable *sexo* en datos, y como se trata de un factor, veremos aparecer sus niveles.

En cuanto al sistema de indexación se comporta de manera similar a las matrices. Podemos acceder a elementos internos del dataframe de las siguientes formas:

```
> datos[,3]
```

```
[1] M F F M
```

```
Levels: F M
```

Al igual que el ejemplo anterior, usamos la notación con el signo $ para llamar a toda la columna 3 (sexo) del objeto *datos*.

01.16. Combinación de dataframes

Si tenemos dos dataframes y queremos unirlos lo podemos hacer muy fácilmente con las funciones ***rbind()*** *y* ***cbind()***. Estas funciones permiten unir dataframes uno al lado del otro o uno debajo del otro, respectivamente.

Por ejemplo, queremos agregar un nuevo ensayo a nuestra tabla de datos. Primero, recordemos cómo hicimos nuestra tabla **datos**:

```
> id <- 1:4
> edad <- c(23, 43, 12, 65)
> sexo <- c("M", "F", "F", "M")
> trabaja <- c(T, T, F, F)
> datos <- data.frame( id, edad, sexo, trabaja)
```

Y luego, generemos otra con los datos nuevos:

```
> id <- 5:9
> edad <- c(37, 45, 52, 25, 32)
> sexo <- c("F", "F", "F", "M", "M")
> trabaja <- c(T, F, T, F, F)
> datos_nuevos <- data.frame( id, edad, sexo, trabaja)
```

Ahora, la variable **datos_nuevos** tiene los resultados de nuestro nuevo ensayo. Dado que este dataframe tiene el mismo número de columnas que **datos**, podemos unirlo mediante la función ***rbind()***

```
> datos_todos <- rbind(datos, datos_nuevos)
> datos_todos
```

```
  id edad sexo trabaja
1  1   23    M    TRUE
2  2   43    F    TRUE
3  3   12    F   FALSE
4  4   65    M   FALSE
5  5   37    F    TRUE
6  6   45    F   FALSE
```

```
7 7  52   F   TRUE
8 8  25   M   FALSE
9 9  32   M   FALSE
```

En esta función es muy importante que coincidan el número y el nombre de las columnas, si no R desconoce cómo hacer la unión de los dataframe y nos arroja un mensaje de error como el siguiente:

```
Error in match.names(clabs, names(xi)) :
 names do not match previous names
```

En caso de que queramos incorporar una nueva columna podemos hacerlo con la función ***cbind()***.

Como requisito, esta función requiere que el número de filas sea el mismo para realizar la unión:

Creemos una nueva variable llamada **estudia** y la unimos a **datos_todos** mediante la función ***cbind()***. Con la función ***head()*** observamos la cabecera de la tabla para ver si realmente se cumplió la unión:

```
> estudia <- c(T, F, T, F, F, T, T, F, F)
> datos_estudia <- cbind(datos_todos, estudia)
> head(datos_estudia)
```

```
  id edad sexo trabaja estudia
1 1  23   M    TRUE    TRUE
2 2  43   F    TRUE   FALSE
3 3  12   F   FALSE    TRUE
4 4  65   M   FALSE   FALSE
5 1  37   F    TRUE   FALSE
            6    2  45   F   FALSE    TRUE
```

01.17. Listas

Una **lista** contiene una colección ordenada de otros objetos, llamados **componentes** de la lista. A diferencia de los vectores, una lista puede guardar elementos de distintas

clases. La lista nos permite combinar vectores, matrices, caracteres, etc, e incluso otras listas dentro de un único objeto lista.

Para crear una lista usamos la función **list()**:

```
> milista <- list(numeros = 1:5,
                  ciudades = c("Buenos Aires",
                               "Rosario", "Neuquén"))
```

Notar que en las listas podemos asignar nombres a los componentes:

```
> familia <- list(padre = "Juan", madre = "Maria",
                  numero.hijos = 3,
                  nombre.hijos = c("Luis", "Carlos", "Paola"),
                  edades.hijos = c(7, 5, 3),
                  ciudad = "La Plata")
```

Así, cuando usemos la función **names()**, obtendremos un vector con los nombres de los componentes.

Si hacemos:

```
> names(familia)
```

R nos devuelve:

```
[1] "padre" "madre" "numero.hijos" "nombre.hijos" "edades.hijos" "ciudad"
```

01.18. Longitud

Un atributo muy útil para conocer cuando trabajamos con vectores o listas es la **longitud.** Esto es, el **número de elementos** que se encuentran en el objeto. Para esto usamos la función **length()**.

```
> length(vector)
```

```
[1] 4
```

01.19. Funciones

Una función es un script que toma datos de entrada y los usa para computar un resultado/producto. El producto de una función puede ser cualquier tipo de objeto, incluyendo imágenes. Esto dependerá de la función que estemos utilizando.

Por tanto, las **funciones** tienen un nombre que utilizaremos para llamarla y entre paréntesis escribiremos los argumentos o parámetros que la función necesita separados por comas.

En el caso de funciones que no precisan argumentos, debemos igualmente escribir los paréntesis ().

¿Qué funciones recuerdas que hayamos visto? ¿Recuerdas alguna función que no lleve argumento?

01.20. Paquetes

Un aspecto muy interesante del trabajo con R, es la posibilidad de ampliar sus funciones a partir de la descarga de paquetes que podemos activar o desactivar en cualquier momento del análisis.

Un paquete es una colección de funciones R, datos y código compilado. La ubicación donde se almacenan los paquetes se llama *library* (biblioteca). Habitualmente se agrupan por tema o similitud de funciones.

Al momento de escribir este artículo, hay más de 12.000 paquetes disponibles en CRAN.

R y RStudio tienen funcionalidad para administrar paquetes:

- Puede ver qué paquetes están instalados escribiendo **installed.packages()**
- Puede instalar paquetes escribiendo **install.packages("package_name")**, donde *package_name* es el nombre del paquete, entre comillas.
- Puede actualizar los paquetes instalados escribiendo **update.packages()**
- Puede eliminar un paquete con **remove.packages()**
- Puede hacer que un paquete esté disponible para su uso con **library()**

Desafío 5: Instalación del paquete ggplot2

Como último desafío te proponemos instalar un paquete que utilizaremos en el encuentro sobre visualización de datos. Antes de comenzar, asegúrate de contar con conexión a internet.

1. Escribe y ejecuta en consola: install.packages("ggplot2")

Este paso puede tardar unos minutos

2. Llama a la librería instalada: library(ggplot2)
3. ¡Veámosla en acción! Escribe las siguientes líneas en la consola

```
> data("BOD") #trae una base de datos de R
> ggplot(data= BOD) + geom_line(aes( x = Time, y = demand))
```

¿Funcionó? En el encuentro sobre visualización de datos veremos más sobre *ggplot2*..