Módulo 2: R base

Matías Poullain

25/3/2022

Introducción

Bienvenidos al primer módulo práctico del curso **Introducción al lenguaje de programación R**. A partir de ahora, todos los módulos serán prácticos. Cada módulo viene con un archivo tipo RMarkdown (.Rmd) modificable. Estos son archivos de texto con intercalaciones de texto y código que deberán modificar a lo largo de la práctica. Además, hay un archivo PDF que no es modificable que se generó a partir del archivo RMarkdown original que funciona como apunte teórico.

Estas clases están pensadas para ser seguidas desde el archivo RMarkdown dentro de RStudio, por lo que si usted no se encuentra en esta situación, realice un doble click sobre tal archivo y se abrirá en RStudio.

En primer lugar realizaremos una presentación del RMarkdown. Son archivos de texto modificables con partes de texto y otras de código. La parte de texto sirve para explicar lo que se crea necesario sobre la parte de código. Mantener un registro de qué está realizando el código es un paso muy importante ya que leer código sin una explicación previa puede ser muy dificultoso para terceros o bien para uno mismo, tiempo después de haberlo escrito.

Cada código se encuentra en una zona sombreada llamada **chunck**. Para crear un chunck puede clickear en la **+C** verde que se encuentra en la parte superior derecha de la sub-ventana del script o bien presionando **Ctrl+Alt+I**. Si presiona el botón **+C** podrá notar que puede elegir varios lenguajes, en esta práctica nos limitaremos a R. Cree un chunck a continuación y dentro de él escriba una operación matemática sencilla. Una vez escrita presione el botón con el símbolo de **Reproducir** verde en la parte superior derecha del chunck para que se realice (corra) la operación. Otra forma de correr el chunck es presionando **Ctrl+Shift+Enter** cuando se encuentre situado sobre él. Para realizar varias operaciones seguidas, sepárelas con un salto a la línea. Es importante entender que R no realiza operaciones simultáneas sino que son secuenciales: una vez realizada una operación sigue con la de la línea de abajo ya así sucesivamente hasta que encuentre un error o no hayan más líneas.

Inserte el chunck aquí

Los resultados de la corrida se muestran en dos sitios: en la **consola** y justo por debajo del chunck. Esas salidas son idénticas.

Operaciones matemáticas y búsqueda en internet

A continuación se presentan algunas operaciones matemáticas básicas que se pueden realizar. Al lado de algunas de ellas verá un `#` seguido de texto. El `#` indica que lo que le sigue es un **comentario**, es decir que R va a ignorar lo que se encuentre después de él en esa misma línea. Los comentarios son útiles para realizar pequeñas anotaciones línea a línea.

```r
1 + 5
```

```
## [1] 6
```

```r
5^2 # 5 al cuadrado
```

```
## [1] 25
```

```r
5**2 #Otra forma de escribir 5 al cuadradp
```

```
## [1] 25
```

```r
10/2
```

```
## [1] 5
```

```r
10/3
```

```
## [1] 3.333333
```

```r
(25+3)/15 ** 0.1
```

```
## [1] 21.35743
```

```r
sqrt(25) # Raíz cuadrada (square root en inglés) de 25
```

```
## [1] 5
```

Observe los resultados con decimales. Acá se presenta una de las diferencias claves con Excel, en R los decimales se separan con un **punto** y NO con una **coma** (que más adelante veremos para qué se usan).

Existen muchas más operaciones matemáticas. Si necesita aplicar alguna que conozca pero que no sabe cómo es su sintaxis en R siempre puede buscarla en Internet. Estas búsquedas son moneda corriente en la programación en general. Siendo que los comandos son tantos, tan específicos y tan variados es imposible acordarse de todos, por eso es importante aprender a encontrar lo que se esta buscando en Internet.

Realice el siguiente ejercicio: Busque y encuentre la forma de implementar un truncamiento de un número. El truncamiento es quedarse solo con la parte entera del número (borrar la parte decimal), es decir que truncar 5.49 devuelve 5. Es muy recomendable agregar a la búsqueda la frase "en R" (o "in R" en inglés) para filtrar mejor los resultados. Cree un nuevo chunck y trunque los números **15.222** y **-2.3**.

Inserte el chunck aquí

Generalmente, los lenguajes de programación están desarrollados y mantenidos en el idioma inglés. Por esta razón suele ser conveniente realizar las búsquedas en ese idioma. Sin embargo, hay una gran comunidad de usuarios de R hispanohablantes y una gran cantidad de respuestas se encuentran en español.

Comandos especiales y lógica

En R existen ciertos comandos especiales escritos como texto pero que son reconocidos de forma distinta. Los más usuales son `NA`, `NULL`, `TRUE` y `FALSE`. Los comandos especiales son resaltados automáticamente en el código para que sean identificables más fácilmente.

NA

`NA` significa "valor vacío", es decir que es la representación de un número desconocido y esto hace que tenga propiedades especiales:

```
NA
```

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

```
NA + 1
```

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

```
NA * 5
```

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

```
0
```

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

```
0 + 1
```

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

```
0 * 5
```

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

Las operaciones matemáticas que involucren un `NA` devolverán un `NA` a modo de aviso que esa operación no puede ser realizada correctamente. Es muy importante distinguirlo del valor `0` el cuál explicita que el valor es conocido y que ese valor es 0.

El `NA` es especialmente útil en tablas, haciendo explícita la falta de un valor en ese casillero volviendo inconfundible una celda vacía de una celda con puros espacios. En el próximo chunck se muestra un ejemplo. No se detenga demasiado en la sintaxis de creación de una tabla, eso lo veremos más adelante, observe detalladamente el resultado:

```
tabla <- data.frame(ID_persona = c(1, 5, NA, 20, 3),
                    Estado_cuenta = c("Alta", "Alta", " ", "Baja", NA)
                    )

tabla
```

```
##   ID_persona Estado_cuenta
## 1          1          Alta
## 2          5          Alta
## 3         NA
## 4         20          Baja
## 5          3          <NA>
```

En la línea 3 hay una celda con `NA` y otra con espacios. La diferencia es clara. Si bien la de espacios pareciera estar vacía, contiene caracteres " " por lo que no tiene un NA visible. También observará que la última celda tiene `<NA>` con <>, esto es debido a que la columna Estado_cuenta es de tipo *caracter* y no numérica (cómo si lo es ID_persona), diferenciando entre una celda numérica vacía con una de caracteres vacía.

NULL

`NULL` (nulo en inglés), tiene la misma idea que `NA` pero para objetos como una tabla. Por ejemplo, para representar una tabla que no existe, se usa `NULL`, para representar un número que no está se usa `NA`.

Operadores lógicos TRUE y FALSE

Una de las herramientas más importantes en R es la posibilidad de realizar operaciones lógicas y para ellas se usan los comandos especiales `TRUE` y `FALSE` (VERDADERO y FALSO respectivamente en inglés). Estos pueden ser combinados de distintas formas con los operadores `&` y `|` que se refieren a "y" y "o" respectivamente. Algunas operaciones lógicas básicas se muestran a continuación:

```
TRUE
```

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

```
FALSE
```

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

```
TRUE & TRUE #Verdadero y verdadero
```

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

```
TRUE & FALSE #Verdadero y falso
```

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

```
FALSE | TRUE #Falso o verdadero
```

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

```
FALSE | FALSE #Falso o Falso
```

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

Además, se pueden hacer realizar operaciones relacionales con números o textos que devuelvan resultados lógicos:

```
3 < 1 #mas chico que...
```

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

```
4 >= 10 #mas grande o igual que...
```

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

```r
5 <= -1 #mas chico o igual que
```

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

```r
3 == 3 #igual a...
```

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

```r
3 != 3 #diferente a...
```

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

```r
"texto" == "TEXTO"
```

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

```r
"texto" > "letras"
```

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

Al utilizar estos operadores siempre se espera que el resultado sea `TRUE` o `FALSE`. Algo que suele llamar la atención con estos operadores es el `==` que realiza una prueba de igualdad. Intuitivamente, uno quisiera utilizar el comando `=`, sin embargo este tiene otro propósito crucial que se verá más adelante, por lo que se creó el comando `==` para que no hayan ambigüedades.

El comando `!=` equivale a un singo de inequidad (representa el igual tachado), pero como en el teclado no está ese símbolo se usa el `!=`.

Observe lo que ocurrió con `"texto" == "TEXTO"`, entregó el resultado `FALSE`. Si bien son la misma palabra, los caracteres no lo son, ya que uno está en minúsculas y el otro en mayúsculas, por lo que R detecta que son distintos. No importa la cantidad de diferencias, con que haya una sola, esos dos textos son distintos. Ocurre lo mismo con las tildes.

En el caso de `"texto" > "letras"` se devolvió `TRUE`. Cuando ocurren $<$ o $>$ entre caracteres, R estudia su orden alfabético. Como "texto" viene después que "letras" alfabéticamente, "texto" $>$ "letras" es verdadero.

Clases de datos

Los datos suelen ser de 5 clases básicas:

- **numeric** (numérico): Número en su sentido más amplio. Este tipo agrupa todos los números enteros, decimales, racionales, negativos e infinitos
- **integer** (enteros): Cómo **numeric** pero solo avocado a enteros
- **logical** (lógicos): TRUE o FALSE
- **character** (caracteres): Texto. Se lo explicita escribiendo el texto entre `"` o `'`. R reconoce el texto como una secuencia de caracteres ordenados. No tiene restricciones en cuanto a lo que se puede escribir.
- **factor**: Similar a **character** pero con niveles definidos: la luz de un semáforo puede ser roja, amarilla o verde, no hay otras opciones.

Usualmente, cómo se devuelve un resultado deja implicito la clase de dato. Sin embargo se puede utilizar la función `class()` para saberlo con certeza. Esta función devuelve la clase del dato entre paréntesis.

```r
10
```

```
## [1] 10
```

```r
10L #La L explicita que se trata de un integer y no de un numeric
```

```
## [1] 10
```

```r
TRUE
```

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

```r
"texto"
```

```
## [1] "texto"
```

```r
as.factor("texto")
```

```
## [1] texto
## Levels: texto
```

```r
class(10)
```

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

```r
class(10L)
```

```
## [1] "integer"
```

```r
class(TRUE)
```

```
## [1] "logical"
```

```r
class("texto")
```

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

```r
class(as.factor("texto"))
```

```
## [1] "factor"
```

Objetos

Una facilidad clave que tiene R es el manejo de objetos. Para crearlos se utilizan los operadores `<-` (como una flecha) o `=` que funcionan indistintamente aunque, por convención y a modo de diferenciación con otros lenguajes, la flecha es la más utilizada. Asignar un objeto es similar a asignarle un nombre a algo a modo de poder utilizarlo fácilmente. Observe el código siguiente e intente entender que está ocurriendo:

```
objeto.1 <- 50
objeto.1
```

```
## [1] 50
```

```
objeto.1 + 50
```

```
## [1] 100
```

```
objeto.2 <- objeto.1 * 3
objeto.2
```

```
## [1] 150
```

```
objeto.1 < objeto.2
```

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

Al nombre `objeto.1` (aunque podría ser cualquier otro nombre que no contenga comandos especiales) se le asignó el número 50 por lo que, a partir de ese momento, llamar a `objeto.1` es equivalente a usar el número 50 y se le pueden realizar operaciones matemáticas y lógicas sobre él como ya se vio anteriormente. Además, se pueden usar para crear nuevos objetos, en este caso se creó `objeto.2` a partir de `objeto.1`.

Puede cambiar el valor de un objeto reasignándole otro valor:

```
objeto.1 <- 2
objeto.1
```

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

Ahora bien, `objeto.1` es 2. Pero ¿Qué ocurrió con `objeto.2`? Este fue definido a partir de `objeto.1` cuando este era 50, entonces ¿mantuvo el valor de 150 o se convirtió automáticamente en 6? Puede checkearlo escribiendo su nombre en la consola (subventana de abajo) para hacer un checkeo rápido sin necesidad de crear un chunck.

`objeto.2` devuelve 150. Esto demuestra que `objeto.2` (una vez creado) es independiente a los cambios que le ocurran a `objeto.1`

Otra forma de checkear rápidamente los valores de un objeto es mirando la subventana de Ambiente (Environment, arriba a la derecha). Aunque para objetos más grandes (tablas por ejemplo), no se los ve directamente.

Los objetos más comunes que se pueden crear son los vectores y las tablas:

Vectores

Los vectores no son más que secuencias ordenadas de datos de una misma clase. No puede contener datos de distinta clase. Los vectores se crean con la función `c()` (que hace referencia a la palabra "concatenar") y cada elemento está separado por comas:

```r
numeros_impares <- c(1, 3, 5, 7, 9, 11)
numeros_impares
```

```
## [1]  1  3  5  7  9 11
```

```r
class(numeros_impares) #Tipo de datos del vector
```

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

```r
numeros_impares[4] #Valor del elemento en la posición 4 del vector
```

```
## [1] 7
```

Aquí queda en evidencia el uso de las comas, sirven para separar elementos y NO para delimitar decimales.

Los corchetes `[]` que preceden al vector permiten acceder a un elemento específico del vector según su posición. Por ejemplo `numeros_impares[4]` devuelve el cuarto elemento del vector, que es 7.

De forma similar a lo visto anteriormente, se pueden realizar distintos tipos de operaciones sobre los vectores. Las operaciones son las mismas que para un valor único pero repetidos para toda la secuencia.

Cree un nuevo chunck y multiplique todos los números del vector `numeros_impares` por 5 y haga la prueba de cuales de esos son más chicos que 10 en una sola línea. Luego, obtenga el número máximo y, por último, el mínimo del vector `numeros_impares` (puede realizar una búsqueda en Internet para aprender cómo se hace).

Inserte chunck aquí

```r
numeros_impares*5 < 10
```

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

```r
max(numeros_impares)
```

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

```r
min(numeros_impares)
```

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

¿Qué ocurre si el vector contiene un `NA`? Cree un nuevo chunck copiando y pegando el contenido de los dos últimos y modifique la creación de `numeros_impares` de forma tal que contenga un `NA`, no importa en cuál posición. Observe los resultados.

Inserte chunck aquí

```r
numeros_impares <- c(1, 3, 5, NA, 9, 11)
numeros_impares
```

```
## [1]  1  3  5 NA  9 11
```

```r
class(numeros_impares)
```

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

```r
numeros_impares*5 < 10
```

```
## [1]  TRUE FALSE FALSE    NA FALSE FALSE
```

```r
max(numeros_impares)
```

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

```r
min(numeros_impares)
```

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

Es muy común que aplicar funciones a vectores con `NA` devuelva `NA`. Si bien es útil que R haga explícito que dentro de ese vector hay un `NA` y no pudiendo realizar los cálculos correctamente, se puede forzar a que el cálculo se realice de todas formas agregando `, na.rm = TRUE` dentro de la función. `na.rm` es una abreviación de **NA remove** (Remover NAs) que hace que los `NA` sean ignorados:

```r
max(numeros_impares, na.rm = TRUE)
```

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

```r
min(numeros_impares, na.rm = TRUE)
```

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

Para estas funciones en particular, `na.rm` es un parámetro de la función. La gran mayoría de las funciones tienen varios parámetros y todos ellos deben tener algún valor aunque, a veces, este valor esté explícito y no es necesario explicitarlo. En el último ejemplo, la función `max()` tiene cómo parámetros a un vector y a `na.rm` y en el último chunck el valor del vector fue `numeros_impares` y el valor de `na.rm` fue `TRUE`. Sin embargo, antes pudimos correr `max(numeros_impares)` sin especificar `na.rm`, esto es porque `na.rm == FALSE` por default a menos que se especifique otra cosa. Cómo ver cuales parámetros están implícitos o no en una función será visto más adelante en el módulo 3.

Data Frames

Es el nombre que se les da a las tablas en R y serán los objetos centrales del trabajo en esta plataforma. Tiene columnas y filas y las columnas deben tener un tipo de dato, tal como en Excel. Existen muchas formas de crear un **Data frame** en R, una de las más sencillas es la siguiente:

- Paso 1: Crear tantos vectores como columnas se quieran. Todos deben tener el mismo largo. Los nombres que se les da a los vectores serán los nombres de las columnas.
- Paso 2: Juntarlos con la función `data.frame()` y asignarle un nombre para guardarlo.

```r
#Paso 1:

Nombre <- c("Martín", "Alejandra", "María", "Tomás", "Lucía")
Calle <- c("Av. Directorio", "Juan de Garay", "Pueyrredón", "Perú", "25 de Mayo")
Altura <- c(1805, 1012, 945, 341, 1240)
Localidad <- c("Capital Federal", "Ciudad de Córdoba", "Villa Mercedes", "San Rafael", "Trelew")
Provincia <- c("Capital Federal","Córdoba", "San Luis", "Mendoza", "Chubut")

#Paso 2:

tabla.domicilios <- data.frame(Nombre, Calle, Altura, Localidad, Provincia)
tabla.domicilios #Esta es sólo para observar al tabla
```

```
##      Nombre          Calle Altura         Localidad       Provincia
## 1    Martín Av. Directorio   1805   Capital Federal Capital Federal
## 2 Alejandra  Juan de Garay   1012 Ciudad de Córdoba         Córdoba
## 3     María     Pueyrredón    945    Villa Mercedes        San Luis
## 4     Tomás           Perú    341        San Rafael         Mendoza
## 5     Lucía     25 de Mayo   1240            Trelew          Chubut
```

Cuando la tabla es muy grande, puede ser difícil observarla, por lo que se puede realizar `View(tabla.domicilios)` para que se abra una pestaña nueva con la visualización.

Para acceder a un elemento de la tabla, tal como ocurría con el vector se usan los corchetes `[]`. A diferencia del vector, la tabla tiene filas y columnas, por lo que dentro del corchete hay que escribir el número de fila y el de columna del elemento al que se quiera acceder. Por ejemplo, el elemento "María" se encuentra en la fila 3 y columna 1 de la tabla `tabla.domicilios`. También, se puede acceder a la columna a partir de su nombre. Estos ejemplos están reflejados en el siguiente chunck:

```r
tabla.domicilios[3, 1]
```

```
## [1] "María"
```

```r
tabla.domicilios[3, "Nombre"]
```

```
## [1] "María"
```

Otro operador importante de las tablas es `$`. Permite acceder a columnas enteras por su nombre devolviendo el vector original:

```r
tabla.domicilios$Nombre
```

```
## [1] "Martín"    "Alejandra" "María"     "Tomás"     "Lucía"
```

Siendo que los data frames son los principales elementos de trabajo en R, tienen una muy gran cantidad de operadores y funciones para su manipulación. En los próximos módulos, trabajaremos casi exclusivamente con data frames, veremos algunas de sus operaciones posibles y entenderemos más a fondo la lógica del trabajo con bases de datos en R.