REDES Revista Hispana para el Análisis de Redes Sociales

Vol.31,#2, (2020), 157-175

http://revistes.uab.es/redes https://doi.org/10.5565/rev/redes.882

Análisis de redes egocéntricas con R (I). Introducción a R

Raffaele Vacca[1]

University of Florida

RESUMEN

Este texto es el primero de una serie de cuatro que conjuntamente constituyen un taller sobre análisis de ego-redes (y/o redes personales) con R. El texto está acompañado por ficheros de datos y los scripts de lenguaje R necesarios para realizar las actividades propuestas.

Palabras clave: *Ego-redes - Redes personales – R.*

ABSTRACT

This text is the first of a series of four documents that together constitute a workshop on analysis of ego-networks (and / or personal networks) using R. The text is accompanied by data files and the R scripts necessary to carry out the suggested activities.

Key words: *Ego-networks – Personal networks – R.*

[1] *Contacto con los autores: Raffaele Vacca (r.vacca@ufl.edu). Traducción de José Luis Molina*

INTRODUCCIÓN

Los materiales para este taller incluyen 4 archivos *script* con código R (archivos con la extensión ".R"). Se puede acceder y ejecutar el código R de un script abriendo ese archivo en una GUI para R (por ejemplo, RStudio). Cada articulo está acompañado de su correspondiente script, con comentarios y explicaciones sobre el código:

- "R01_basics.R" provee de una introducción al lenguaje R (este texto).
- "R02_one_egonet.R" explica cómo archivar datos de redes egocéntricas en R y cómo analizar una ego-red.
- "R03_many_egonets.R" aplica el mismo procedimiento de análisis en múltiples ego-redes simultáneamente.
- "R04_appendix.R" se ocupa de temas complementarios.

El script "R01_basics.R" cubre algunos de los conceptos básicos del lenguaje R. Obviamente este texto no pretende ser una introducción exhaustiva y comprehensiva, sino que se propone mostrar algunas nociones y herramientas esenciales de R usadas habitualmente en el análisis de datos de las ciencias sociales, incluyendo el análisis de redes egocéntricas.

El script abarca los siguientes temas:

- Inicio y ayuda en R.
- Crear y guardr objetos R.
- Vectores y matrices; marcos de datos (*dataframes*) y *tibbies* (dataframes amigables).
- Operaciones aritméticas y de comparación.
- Indexación de vectores, matrices y dataframes.

Datos

Este taller utilizada materiales tomados de una investigación realizada en 2012 que recogió mediante una encuesta las redes personales de 107 emigrantes de Sri Lanka en Milan, Italia. De los 107 participantes, 102 reportaron su red personal. Los ficheros de datos incluyen información a nivel de ego (sexo, edad, nivel educativo, etc. para cada participante), atributos de los alteri (nacionalidad de los alteri, país de residencia,

cercanía emocional con ego, etc.), e información sobre los lazos alter-alter. Cada red personal tiene un número fijo de 45 alteri. La información sobre las variables y categorías está disponible en "./Data/codebook.xlsx".

Todos los datos están guardados como objetos R en el fichero de datos R "data.rda". Los objetos son los siguientes:

- "ego.df": un dataframe con atributos al nivel de ego para todos los participantes (egos).
- "alter.attr.all": un dataframe con atributos a nivel der alter para todos los alteri de todos los participantes.
- "gr.list": una lista. Cada elemento de la lista es una ego-red guardada como un objeto "igraph".
- "alter.attr.28": un dataframe con atributos al nivel de alter solamente con los alteri nominados por el ego ID 28.
- "gr.28": La ego-red de ego ID 28 guardada como un objeto "igraph".

Los objetos R mencionados pueden ser importados desde ficheros csv en bruto. El documento "R04_appendix" muestra el código para crear objetos de datos R a partir de ficheros csv. Todos los ficheros csv están el subdirectorio "./Data/raw_data/":

- "ego_data.csv": un fichero "csv" con datos a nivel de ego para todos los egos.
- "alter_attributes.csv": un único fichero "csv" que incluye los atributos de todos los alteri nominados por todos los egos.
- "alter_ties_028.csv": El listado de aristas (*edgelist* en lo sucesivo) de la red egocéntrica del ego ID 28.
- "alter_attributes_028.csv": Los atributos de los alteri de la red egocéntrica del ego ID 28.
- "adj_028.csv": la matriz de adjacencia de la red egocéntrica del ego ID 28.
- "alter_ties.csv": Un fichero único "csv" con el edgelist de todos los alteri de todos los egos.

NOTA: Antes de ejecutar los scripts R, es necesario asegurarse que el subdirectorio "./Data/" es el directorio de trabajo activo. Para saber qué directorio está en uso se puede usar la instrucción "getwd()". Este subdirectorio debería ser el correcto si simplemente se abre el fichero "Rproj". También es necesario asegurarse que se ejecuta el código de los scripts línea por línea ya que si nos saltamos una o más se pueden producir errores.

INICIAR R Y CARGAR LOS PAQUETES

* Comprobar el *directorio de trabajo*. Por defecto, R buscará los ficheros y guardará los nuevos en este directorio.

- Ejecutar "getwd()" para comprobar el directorio de trabajo.
- Si se abre el fichero proyecto de R (".Rproj") de este taller, RStudio fijará correctamente el directorio de trabajo en el mismo lugar en el que se halle el fichero ".Rproj".
- Se puede usar "setwd()" para cambiar manualmente el directorio de trabajo a cualquier dirección, pero normalmente es más cómodo trabajar con proyectos R y sus directorios por defecto.

* Carga todos los paquetes necesarios para la sesión en curso. Hay dos pasos para usar un paquete en R:

1. *Instalar el paquete*. Esto paso se realiza una sola vez. Es necesario usar "install.packages("nombre_del_paquete")" o el menú apropiado en el GUI de R utilizado (e.g. en RStudio: *Tools* > *Install packages in RStudio*). Una vez el paquete está instalado, los ficheros están en el subdirectorio de sistema de R y éste podrá encontrarlos allí.

2. *Cargar el paquete* en la sesión en curso. El comando es "library("package_name")". Es necesario instalar el paquete en cada session R, es decir, cada vez que se arranca R y se necesita el paquete. El comando "library(package_name)" es escribe sin las comillas en el nombre del paquete.

* Un paquete de R es simplemente una colección de funciones. Por tanto, es posible usar una función R si está incluida en el paquete cargado en la sesión en curso.

* En ocasiones dos funciones diferentes de dos paquetes diferentes tienen el mismo nombre. Por ejemplo, tanto el paquete "igraph" como el paquete "sna" tienen una función llamada "degree()". Si ambos paquetes están cargados, escribir simplemente "degree" puede conllevar resultados inesperados porque R tomará una de las dos funciones (la del paquete cargado más recientemente), la cual podría no ser la función requerida. Para evitar este problema se puede usar la notación "package::function()": "igraph::degree()" activará la función "degree()" del paquete "igraph", mientras que "sna::degree()" activará la función "degree()" del paquete "sna".

* Consejo: para saber de qué paquete viene una función simplemente hay que visitar la

página de la función incluida en el manual. El paquete está indicado en la primera línea de la página. E.g., teclea "?degree" para ver de dónde viene la función "degree()".

* En el caso de que ningún paquete cargado en ese momento tenga una función llamada "degree", teclear "?degree" devolverá un aviso del sistema ("No documentation for "degree"").

* Si múltiples paquetes cargados en ese momento tienen una función llamada "degree", entonces teclear "?degree" llevará a una página con la lista de los paquetes en cuestión.

* Este taller usa los siguientes paquetes:

- "egor".
- "igraph".
- "ggraph".
- "summarytools".
- "tidygraph".
- "tidyverse". Ésta es una colección de diferentes paquetes que comparten un lenguaje común y un conjunto de principios: "dplyr", "forcats", "ggplot2", "purrr", "readr", "stringr", "tibble", "tidyr". Ver tidyverse.org y Wickham & Grolemund (2017) para más información.

Consola versus scripts

as medidas descritas previamente se recogieron * Cuando se abre una GUI R normalmente se ven dos ventanas separadas: el editor de scripts y la consola. Se puede escribir código R en cualquiera de ellas.

* *Consola*. Aquí se escribe el código R línea a línea. Una vez se teclea una línea, se presiona "ENTER" para ejecutarla. Presionando la tecla con la fleche hacia arriba es posible ir atrás, a la última línea ejecutada. Si la tecla se mantiene apretada es posible navegar a través de todas las líneas de código previamente ejecutadas. Esto se llama "histórico de commandos" (todas las líneas ejecutadas en la sesión en curso). Todo este código se perderá (todo el histórico) cuando se cierre R, a menos que explícitamente se salve el histórico en un fichero.

**Editor de scripts*. Aquí se escriben los scripts. Éste es el modo más común de trabajar con R. Un script es simplemente un fichero de texto plano donde todo el código R está almacenado. Si tu tranbajo está en un script, éste es *reproducible*.

* Tanto la GUI estándard de R como RStudio tienen un editor de scripts con algunas funciones. En particular la que permite ejecutar un script mientras lo estás escribiendo. Presionando las teclas "CTRL+R" (Windows) or "CMD+ENTER" (Mac) la línea de script debajo del cursor se ejecuta (o la porción de script seleccionada).

* Obsérvese que con RStudio es posible ejecutar la línea en concreto en la que está el cursor, una porción seleccionada de código, la porción de código correspondiente al inicio del script hasta donde está el cursor, así como la porción de código desde el cursor hasta el final del script. Ver el menu *Código* y sus atajos de teclado.

* El editor de scripts también permite guardar el script. E.g. RStudio: *File* > *Save*. Los ficheros de script R normalmente tienen la extensión ".R" (e.g. ""miscript.R""). Pero obsérvese que un fichero script es solamente un fichero de texto (como cualquier fichero ".txt"), el cual se puede abrir y editar con cualquier editor de textos, o en Microsoft Word y similares.

* También es possible ejecutar un script complete de una vez – a esto se llama *sourcing* un script. Tecleando "source("miscript.R")", se adquiere el fichero fuente "miscript.R". En RStudio: ver *Code* > *Source* y su atajo de teclado.

* Tanto en la consola como en el editor de scripts cualquier línea que comienza con "#" es llamada un *comentario*. R no tiene en cuenta los comentarios –solamente los imprime tal cual están en la consola (no los identifica y ejecuta como código de programación). Es importante usar comentarios para documentar lo que el hace el código (lo cual es bueno para ti y para otros).

* Observa que se puede navegar a través de las cabeceras de los scripts en "W1_Intro_to_R.R" en RSstudio (parte inferior izquierda del editor de scripts). Cualquier línea que comience con "#" y termine con "####", "----", o "====" es identificada como título o cabecera por RStudio.

Obtener ayuda

*La actividad más común mientras se usa R es buscar ayuda. En tanto que principiante es necesario obtener ayuda constantemente (e.g. leer páginas del manual) sobre funciones R. Pero también los usuarios expertos usan frecuentemente las páginas del manual para consultar funciones específicas. Nadie, sin distinción por su nivel de experiencia, puede

usar R sin comprobar de formar regular las páginas del manual u otra documentación de ayuda.

* A continuación se presentan unas cuantas funcionalidades de ayuda disponibles en R:

- "help(...)" o "?..." son las formas más comunes de obtener ayuda: remiten a la página del manual de R para una función específica. E.g. "help(sum)" o "?sum" (son equivalentes).
- "help.start ()" (o RStudio: *Ayuda> R Ayuda*) ofrece páginas de ayuda generales en HTML (introducción a R, referencias a todas las funciones en todos los paquetes instalados, etc.).
- "demo()"ofrece demostraciones sobre temas específicos. Ejecutar "demo()" para ver todos los temas disponibles.
- "ejemplo()" proporciona un código de ejemplo en funciones específicas, e.g. "example(suma" para la función "sum".
- "help.search (...)" o "??..." busca una cadena específica en las páginas del manual, e.g. "??histogram".

* Hay que tener en cuenta que, además de las funcionalidades de ayuda integradas dentro de R, hay muchas maneras de obtener **ayuda de R en línea**. Algunos paquetes R populares tienen su propio sitio web, e.g. ggplot2, igraph y statnet. Otros sitios web para obtener ayuda general sobre R incluyen rdocumentation.org y stackoverflow.com.

```
# What's the current working directory?
# getwd()
# Un-comment to check your actual working directory.
# Change the working directory.
# setwd("/my/working/directory")
# (Delete the leading "#" and type in your actual working directory's path
# instead of "/my/working/directory")
# You should use R projects (.Rproj) to point to a working directory instead of
# manually changing it.
# Suppose that we want to use the package "igraph" in the following code.
library(igraph)

##
## Attaching package: 'igraph'
## The following objects are masked from 'package:stats':
##
## decompose, spectrum
## The following object is masked from 'package:base':
##
## union
# NOTE that we can only load a package if we have it installed. In this case, I
# have igraph already installed. Had this not been the case, I would have
# needed to install it:
# install.packages("igraph").
# (Packages can also be installed through a GUI menu item).
# Let's load another suite of packages we'll use in the rest of this script.
library(tidyverse)
## Registered S3 methods overwritten by 'ggplot2':
## method from
## [.quosures rlang
## c.quosures rlang
## print.quosures rlang
## -- Attaching packages ------------------- tidyverse 1.2.1 --
## v ggplot2 3.1.1 v purrr 0.3.2
## v tibble 2.1.3 v dplyr 0.8.1
## v tidyr 0.8.3 v stringr 1.4.0
## v readr 1.3.1 v forcats 0.4.0
## -- Conflicts ---------------------- tidyverse_conflicts() --
## x dplyr::as_data_frame() masks tibble::as_data_frame(), igraph::as_data_frame()
## x purrr::compose() masks igraph::compose()
## x tidyr::crossing() masks igraph::crossing()
```

```
## x dplyr::filter() masks stats::filter()
## x dplyr::groups() masks igraph::groups()
## x dplyr::lag() masks stats::lag()
## x purrr::simplify() masks igraph::simplify()
# Note that whatever is typed after the "#" sign has no effect but to be printed
# as is in the console: it is a comment.
```

OBJETOS EN R

Creando objetos: conceptos básicos

> En R, todo lo que **existe** es un objeto. Todo lo que **sucede** es una llamada de función.
>
> John Chambers

* R es un lenguaje de programación orientado a objetos. Los datos, las herramientas de análisis y los resultados están contenidos en **objetos**. Todo está contenido en un objeto: conjuntos de datos, matrices de adyacencia, redes, funciones, resultados de regresión, estadísticas descriptivas, etc.

* Los objetos tienen *un nombre y un valor*. Es posible crear un objeto **asignando** un valor a un nombre. Se asigna con "<-" o con "=".

* Cada vez que ejecuta una operación o se ejecuta una función en R, se debe asignar el resultado a un objeto si se desea guardarlo y reutilizarlo más tarde. **O se asigna o se pierde**. Cualquier cosa que no esté asignada a un objeto se imprime en la consola y se pierde.

* Los objetos tienen un tamaño (bytes, megabytes, etc.) y un **tipo** (técnicamente, una *clase*, un *tipo* y un *modo*) -- más sobre esto más adelante (# tipos-y-clases-de- objetos).

* La **función** A es un tipo particular de objeto. Las funciones toman otros objetos como argumentos (entrada) y devuelven más objetos como resultado (salida). Las funciones R son lo que otros programas de análisis de datos llaman "comandos". Consultar "Análisis de redes egocéntricas con R (II)" para obtener más información sobre las funciones.

* Durante la sesión R, los objetos (datos, resultados) se encuentran en la memoria principal del ordenador. Forman su **espacio de trabajo**. Éstos desaparecerán en cuanto se salga de R, a menos que los guarde en archivos en el disco.

* Hay que tener en cuenta que R es sensible a **mayúsculas y minúsculas**: el objeto "mydata" es diferente del objeto "Mydata".

El espacio de trabajo

* El espacio de trabajo es el conjunto de **todos los objetos** disponibles en la memoria.

* Cuando se sale, R pregunta si se desea guardar el espacio de trabajo actual. Este espacio lo guarda por defecto en un archivo llamado ".RData" en el directorio activo.

* Cuando se inicia R, si existe un archivo llamado ".RData" en el directorio activo, el espacio de trabajo se carga automáticamente en la sesión actual.

* La función "ls()"muestra el contenido del espacio de trabajo activo.

* RStudio dispone de un panel de entorno con detalles sobre el espacio de trabajo (consultar *RStudio> Preferences ...> Pane Layout*).

Guardando y eliminando objetos

* Existen dos funciones principales para **guardar** objetos R en archivos: "save()" (guarda objetos específicos, sus argumentos); "save.image()" (guarda todo el espacio de trabajo actual).

* A menos que especifique una ruta diferente, todos los archivos que guarde de R se colocarán en su directorio de trabajo activo.

* Las extensiones de archivo más comunes para los archivos que almacenan objetos R son ".rda" y ".RData".

* Si se dispone de un archivo con objetos R, digamos "objects.rda", éste se puede **cargar** en la sesión de R actual utilizando la función "load()": "load(file="objects.rda") (y suponiendo que "objects.rda" está en el directorio de trabajo activo).

* La función "rm()" **elimina** objetos específicos del espacio de trabajo. Puede usarse para borrar el espacio de trabajo de todos los objetos existentes escribiendo "rm(list= ls())" (recuérdese que "ls()" devuelve un vector de caracteres con los nombres de todos los objetos en el espacio de trabajo actual).

```
# Create the object a: assign the value 50 to the name "a"
a <- 50
# Display ("print") the object a.
a
## [1] 50
# Let's create another object.
b <- "Mark"
# Display it.
b
6
## [1] "Mark"
# Create and display object at the same time.
(obj <- 10)
## [1] 10
# Let's do a simple operation.
a + 3
## [1] 53
# What if we wanted to save this result?
result <- a + 3
# All objects in the workspace
ls()
## [1] "a" "b" "obj" "result"
# Now we can view that result whenever we need it.
result
## [1] 53
# ...and re-use it
result*2
## [1] 106
# Note that R is case-sensitive, "result" is different from "reSult".
reSult
## Error in eval(expr, envir, enclos): object 'reSult' not found
# Let's clear the workspace before proceeding.
rm(list=ls())
# The workspace is now empty.
ls()
## character(0)
```

Objetos vectores y matrices

* **Los vectores** son los objetos más básicos que utiliza R. Los vectores pueden ser numéricos (datos numéricos), lógicos (datos VERDADERO / FALSO) y caracteres (datos de cadena).

* La función básica para crear un vector es "c()" (**concatenate**).

* Otras funciones útiles para crear vectores: "rep()" y "seq()". También téngase en cuenta que el acceso directo ":": "c(1, 2, 3, 4)" es lo mismo que "1:4".

* La **longitud** (número de elementos) es una propiedad básica de los vectores ("longitud()").

* Cuando indicamos "print()" a los vectores, los números entre corchetes indican las posiciones de los elementos del vector.

* Funciones útiles para crear vectores: "rep()" y "seq()". Una función que usaremos para crear vectores más adelante en el taller es "seq_along()". "Seq_along(x)" crea un vector que consiste en una secuencia de enteros de 1 a "length (x)" en pasos de 1.

* Para crear una matriz: "matriz()". Sus argumentos principales son: los valores de las celdas (dentro de "c()"), el número de filas ("nrow") y el número de columnas ("ncol"). Los valores se organizan en una matriz "nrow" x "ncol" *por columna*. Ver "matriz".

* Cuando indicamos "print()" matrices, los números entre corchetes indican los números de fila y columna.

3.5. Data frames

* Dataframes ("marco de datos") es el nombre de R para el conjunto de datos o *datasets*. Un conjunto de datos es una colección de casos (filas) y variables (columnas) que se miden en esos casos.

* Cuando indicamos "print" en R los dataframes parecen matrices. Sin embargo, a diferencia de las columnas de matriz, las columnas del dataframe pueden ser de diferentes tipos, e.g. una variable numérica y una variable de caracteres.

* Por otro lado, al igual que las columnas de matriz, las columnas de las dataframes (variables) deben tener la misma "longitud" (número de casos). No se pueden agrupar variables (vectores) de diferente longitud en el mismo dataframe.

* Aunque los dataframes parecen matrices, desde el punto de vista de R son un tipo específico de *list* (más sobre listas en el texto III). De hecho, la "clase" de un dataframe es "data.frame", pero el "tipo" de un marco de datos es "list". Los elementos de la lista para un dataframe son sus variables (columnas).

* **Tibbles**. Los paquetes tidyverse, que usamos en este taller, se basan en una forma más eficiente de dataframe, llamada **tibble**.

- Un tibble tiene la clase "tbl_df" **y** "data.frame". Esto significa que, para R, un tibble es **también** un dataframe, y las funciones que funcionan en dataframes normalmente también funcionan en los tibbles.
- Un tibble tiene una serie de ventajas sobre un dataframe tradicional, algunas de las cuales veremos en este taller.
- Una de estas ventajas es la forma más clara e informativa en que se imprimen los tibbles. Cuando imprimimos un dataframe tibble, podemos ver inmediatamente el número de filas, el número de columnas, los nombres de las variables y el tipo de cada variable (numérico, entero, carácter, etc.).
- Para convertir un dataframe existente en tibble: "as_tibble()". Para crear un tibble desde cero (similar a la función "data.frame()" en la base R): "tibble()".

* Los dataframes se pueden crear manualmente en R (con las funciones "data.frame()" en la base R y "tibble()" en "tidyverse"), pero los datos se importan más comúnmente en R desde fuentes externas, como un " csv "o archivo" txt ".

* Importaremos datos de archivos csv usando la función "read_csv()" de "tidyverse".

- "Read_csv()" lee archivos csv (valores separados por "," o ";"). "Read_delim()" lee archivos en los que los valores están separados por cualquier delimitador.
- Estas funciones tienen muchos argumentos que las hacen muy flexibles y permiten a los usuarios importar básicamente cualquier tipo de tabla almacenada en un archivo de texto. Echa un vistazo a "?Read_delim".
- En la base R, las funciones correspondientes son "read.csv()" y "read.table()".

* Los datos también se pueden importar a R desde la mayoría de los formatos de archivo externos (SAS, SPSS, Stata, Excel, etc.) utilizando los paquetes tidyverse "readxl" y "haven", o el paquete "foreign" en R estándar.

* Téngase en cuenta que puede hacer clic en el nombre de un dataframe en el panel Entorno de RStudio. Eso abrirá el dataframe en una ventana, similar a la vista de datos de SPSS.

```
# Let's create a simple vector.
(x <- c(1, 2, 3, 4))
## [1] 1 2 3 4
# Shortcut for the same thing.
(y <- 1:4)
## [1] 1 2 3 4
# What's the length of x?
length(x)
## [1] 4
# Note that when we print vectors, numbers in square brackets indicate positions
# of the vector elements.
# Create a simple matrix.
adj <- matrix(c(0,1,0, 1,0,0, 1,1,0), nrow= 3, ncol=3)
8
```

```
# This is what our matrix looks like:
adj
## [,1] [,2] [,3]
## [1,] 0 1 1
## [2,] 1 0 1
## [3,] 0 0 0
# Notice the row and column numbers in square brackets.
# Normally we create data frames by importing data from external files, for
# example csv files.
ego.df <- read_csv("./Data/raw_data/ego_data.csv")
## Parsed with column specification:
## cols(
## ego_ID = col_double(),
## ego.sex = col_character(),
## ego.age = col_double(),
## ego.arr = col_double(),
## ego.edu = col_character(),
## ego.inc = col_double(),
## empl = col_double(),
## ego.empl.bin = col_character(),
## ego.age.cat = col_double()
## )
# View the result.
ego.df
## # A tibble: 102 x 9
## ego_ID ego.sex ego.age ego.arr ego.edu ego.inc empl ego.empl.bin
## <dbl> <chr> <dbl> <dbl> <chr> <dbl> <dbl> <chr>
## 1 28 Male 61 2008 Second~ 350 3 Yes
## 2 29 Male 38 2000 Primary 900 4 Yes
## 3 33 Male 30 2010 Primary 200 3 Yes
## 4 35 Male 25 2009 Second~ 1000 3 Yes
## 5 39 Male 29 2007 Primary 0 1 No
## 6 40 Male 56 2008 Second~ 950 4 Yes
## 7 45 Male 52 1975 Primary 1600 3 Yes
## 8 46 Male 35 2002 Second~ 1200 4 Yes
## 9 47 Male 22 2010 Second~ 700 4 Yes
## 10 48 Male 51 2007 Primary 950 4 Yes
## # ... with 92 more rows, and 1 more variable: ego.age.cat <dbl>
# Note the many pieces of information that are displayed when printing a tibble
# data frame.
```

OPERACIONES ARITMÉTICAS, ESTADÍSTICAS Y COMPARATIVAS

Operaciones aritméticas

* R puede funcionar como una calculadora normal.

  * Adición/sustracción: "7+3"

  * Multiplicación: "7*3"

  * Negativo: "-7"

  * División: "7/3"

  * División de enteros: "7%/%3"

  * Resto entero: "7%%3"

  * Potenciación: "7^3"

* Muchas operaciones que involucran vectores en R se realizan por **elementos**, es decir, por separado en cada elemento del vector (ver ejemplos a continuación).

* La mayoría de las operaciones en vectores usan la regla **reciclaje**: si un vector es demasiado corto, sus valores se reutilizan la cantidad de veces que se necesite para que coincida con la longitud deseada (ver ejemplos a continuación).

+ Ejemplos de operaciones de vectores y reciclaje:

  - "[1 2 3 4] + [1 2 3 4] = [1 + 1 2 + 2 3 + 3 4 + 4]" (suma de elementos).

- "[1 2 3 4] + 1 = [1 + 1 2 + 1 3 + 1 4 + 1]" ("1" se recicla 3 veces para que coincida con la "longitud" del primer vector).
- "[1 2 3 4] + [1 2] = [1 + 1 2 + 2 3 + 1 4 + 2]" ("[1 2]" se recicla una vez).
- "[1 2 3 4] + [1 2 3] = [1 + 1 2 + 2 3 + 3 4 + 1]" ("[1 2 3]" se recicla un tercio de las veces: R advertirá que la longitud del vector más largo no es un múltiplo de la longitud del vector más corto).

Operaciones de comparación y vectores lógicos

* **Operadores** de comparación: ">", "<", "<=", "> =". Igual es "==" (NO "="). *No* igual es "!=".

* Nota: igual es "==", mientras que "=" tiene un significado diferente. "=" Se utiliza para asignar argumentos de función (por ejemplo, "matriz(x, nrow = 3, ncol = 4)"), o para asignar objetos ("x <- 2" es lo mismo que "x=2").

* Las operaciones de comparación dan como resultado vectores **lógicos**: vectores de valores "VERDADERO" / "FALSO".

* Al igual que las operaciones aritméticas, las comparaciones se realizan por elementos en los vectores, y se aplica el reciclaje.

* Operadores lógicos: "&" para AND, "|" para OR.

* Negación (es decir, opuesto) de un vector lógico: "!".

* ¿Está el valor *x* en el vector *y*? "X%in%y".

* R puede convertir vectores lógicos a numéricos ("as.numeric()", "as.integer()"). En esta conversión, "VERDADERO" se convierte en "1" y "FALSO" se convierte en "0". Por el contrario, si se convierte en lógico ("as.logical()"), "1/0" son "VERDADERO / FALSO" ("TRUE/FALSE").

- Por lo tanto, si "x" es un vector lógico, "sum(x)" da el cómputo de "TRUE"s en "x" (suma de los "1"s en el vector).
- "mean(x)" da la *proporción* de "1"s en "x" (media de un vector binario: suma de "1"s en el vector dividido por el número de elementos en el vector).

```
# Just a few arithmetic operations between vectors to demonstrate element-wise
# calculations and the recycling rule.
# [1 2 3 4] + [1 2 3 4]
(v1 <- 1:4)
## [1] 1 2 3 4
(v2 <- 1:4)
## [1] 1 2 3 4
v1 + v2
## [1] 2 4 6 8
# [1 2 3 4] + 1
1:4 + 1
## [1] 2 3 4 5
# [1 2 3 4] + [1 2]
(v1 <- 1:4)
## [1] 1 2 3 4
(v2 <- 1:2)
## [1] 1 2
v1 + v2
## [1] 2 4 4 6
# [1 2 3 4] + [1 2 3]
1:4 + 1:3
## Warning in 1:4 + 1:3: longer object length is not a multiple of shorter
## object length
## [1] 2 4 6 5
# Now on to comparison operations.
# Let's take a single variable from the ego attribute data: ego's age (in years)
# for the first 10 respondetns. Let's put the result in a separate vector. This
# code involves indexing, we'll explain it better below.
age <- ego.df$ego.age[1:10]
age
## [1] 61 38 30 25 29 56 52 35 22 51
```

```
# Note how the following comparisons are performed element-wise, and the value
# to which age is compared (30) is recycled.
# Is age equal to 30?
age==30
## [1] FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
# The resulting logical vector is TRUE for those elements (i.e., respondents)
# who meet the condition.
# Which respondent's age is greater than 40?
age > 40
## [1] TRUE FALSE FALSE FALSE FALSE TRUE TRUE FALSE FALSE TRUE
# Which respondent age values are lower than 40 OR greater than 60?
age < 40 | age > 60
## [1] TRUE TRUE TRUE TRUE TRUE FALSE FALSE TRUE TRUE FALSE
# Which elements of "age" are lower than 40 AND greater than 30?
age < 40 & age > 30
## [1] FALSE TRUE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE
# Notice the difference between OR (|) and AND (&).
# Is 30 in "age"? I.e., is one of the respondents of 30 years of age?
30 %in% age
## [1] TRUE
# Is "age" in c(30, 35)? That is, which values of "age" are either 30 or 35?
age %in% c(30, 35)
## [1] FALSE FALSE TRUE FALSE FALSE FALSE FALSE TRUE FALSE FALSE
# ***** EXERCISE (1):
# Obtain a logical vector indicating which elements of "age" are smaller than 30
# OR greater than 50. Then obtain a logical vector indicating which elements of
# "age" are smaller than 30 AND greater than 50. Why all elements are FALSE in
# the latter vector?
# *****
# ***** EXERCISE (2):
# Create a vector that goes from 1 to 100 in steps of 1. Obtain a logical vector
# that is TRUE for the first 10 elements and the last 10 elements of the vector.
# *****
# A logical vector can be coverted to numeric: TRUE becomes 1 and FALSE becomes
# 0.
age > 45
## [1] TRUE FALSE FALSE FALSE FALSE TRUE TRUE FALSE FALSE TRUE
as.numeric(age > 45)
## [1] 1 0 0 0 0 1 1 0 0 1
# This allows us to use the sum() and the mean() functions to get the count and
# proportion of TRUE's in a logical vector.
# Count of TRUE's: Number of respondents (elements of "age") that are older than
# 40.
sum(age > 45)
## [1] 4
# How many respondents in the vector are older than 30?
sum(age > 30)
## [1] 6
# Proportion of TRUE's: What's the proportion of "age" elements (respondents)
# that are greater than 50.
mean(age > 50)
## [1] 0.4
# ***** EXERCISE (1):
# How many respondents are younger than 50? What percentage of respondents is
# between 30 and 40 years of age, including 30 and 40? Is there any respondent
# who is younger than 20 OR older than 70?
# *****
# ***** EXERCISE (2):
# How many elements of the vector 1:100 are greater than the length of that
# vector divided by 2? Use sum().
# *****
```

Funciones aritméticas y estadísticas básicas

* Las funciones aritméticas *vector* se ejecutan elemento a elementos y devuelven un vector:

- Exponencial: "exp(x)".
- Logaritmo: "log(x)" (base *e*) o "log10 (x)" (base 10).
- Las funciones estadísticas *scalar* se realizan en el conjunto de todos los elementos vectoriales simultáneamente, y devuelven un escalar:
  - "mean(x)" y "median(x)".
  - Desviación estándar y varianza: "sd(x)", "var(x)".
  - Mínimo y máximo: "min(x)", "max(x)".
  - "Sum (x)": suma de todos los elementos en "x".

* "table(x)" es otra función estadística básica, pero no es escalar:

- "table(x)" devuelve un objeto "tabla" con las frecuencias absolutas de valores en "x". Consultar la "tabla" para saber cómo se tratan los valores de NA (perdidos) en función de los argumentos de la función.
- "prop.table(table(x))" devuelve las frecuencias relativas de valores en "x". Téngase en cuenta que "prop.table()" se ejecuta en el objeto "tabla", no en el vector original.

* Si bien las funciones como "sum()", "mean()" y "table()" son esenciales al programar en R (por ejemplo, al escribir sus propias funciones), si solo se necesitan estadísticas descriptivas para los datos, hay herramientas más adecuadas. Una de éstas es el paquete "summarytools":

- "Summarytools::descr()" proporciona estadísticas univariadas para variables continuas.
- "Summarytools::freq()" proporciona tablas de frecuencia para variables categóricas.

Valores perdidos e infinitos

* Infinito puede ser un resultado de operaciones aritméticas: "Inf" e "-Inf" (por ejemplo, "3/0").

* "NaN" (*not a number*) también puede producirse, lo que significa No es un número (por ejemplo, "0/0").

* Los valores perdidos están representados por "NA" (no disponible).

* Mientras que "NA" puede aparecer en cualquier tipo de objeto, "Inf", "-Inf" y "NaN" solo pueden aparecer en objetos numéricos.

* "Is.na(x)" verifica si cada elemento de "x" es NA y devuelve VERDADERO si ese es el caso y FALSO si es lo contrario. Es una función vectorial (su valor tiene la misma "longitud" que "x").

Indexación

* **Indexar** es crucial en R. La indexación significa agregar un índice a un objeto para extraer uno (o más) de sus elementos. La indexación también se llama *subscripting*.

* La indexación se puede usar para **extraer** (ver, consultar) el elemento de un objeto, o **reemplazar** (asignar un valor diferente a ese elemento).

* La notación básica para indexar en R es "[]": "x[i]" proporciona el elemento *i-avo* del objeto "x".

* **La indexación numérica** utiliza números enteros entre corchetes "[]": e.g."X[3]". Téngase en cuenta que se pueden usar enteros negativos para indexar (seleccionar) todo *excepto* ese elemento: e.g. "X[-3]", "x[c (-2, -4)]".

* **La indexación lógica** utiliza vectores lógicos entre corchetes "[]". Se utiliza para indexar objetos según una condición, e.g. para indexar todos los valores en "x" que son mayores que 3 (véase el código de ejemplo a continuación).

* La **indexación de nombres** usa nombres de elementos. Los elementos en un vector y las filas o columnas en una matriz pueden tener nombres.

- Los nombres se pueden mostrar y asignar utilizando la función "names()".

* Al indexar, se debe tener en cuenta el **número de dimensiones** de un objeto. Por ejemplo, los vectores tienen 1 dimensión, las matrices tienen 2. Las matrices se pueden definir con 3 dimensiones o más (por ejemplo, tablas de tres vías).

- Los corchetes suelen contener una posición para cada dimensión del objeto, separada por una coma:

- "x[i]" indexa el elemento *i-avo* del objeto unidimensional "x";
- "x[i, j]" indexa el elemento *i*, *j-ésimo* del objeto bidimensional "x" (por ejemplo, si "x" es una matriz, *i* se refiere a una fila y *j* se refiere a una columna);
- "x[i, j, k]" indexa el elemento *i*, *j*, *k-ésimo* del objeto tridimensional "x", etc.

- Obsérvese que el espacio de una dimensión puede estar vacío, lo que significa que indexamos todos los elementos en esa dimensión. Por lo tanto, si "x" es una matriz, "x[3,]" indexará toda la tercera fila de la matriz, es decir, "["fila 3, todas las columnas"]".
- Si "x" tiene más de una dimensión (por ejemplo, es una matriz), entonces "x[3]" (sin coma, solo una posición) sigue siendo válido, pero puede dar resultados inesperados.

*Las matrices tienen funciones especiales que se pueden usar para indexar, e.g. "diagonal()", "upper.tri()", "lower.tri()". Estos pueden ser útiles para manipular matrices de adyacencia.

* Se pueden aplicar reglas de indexación particulares a determinadas "clases" de objetos, por ejemplo, listas y dataframes (consultar la sección siguiente ("dataframes de indexación y subconjuntos)).

Indexación y segmentación de data frames

***Notaciones de lista**. Los dataframes son una clase especial de listas. Al igual que cualquier lista, los dataframes se pueden indexar de las siguientes 3 formas:

1. **Notación "[]"**, e.g. "Df[3]" o "df["variable.name"]". Esto devuelve otro dataframe que solo incluye los elementos indexados, e.g. solo el 3^cer^ elemento. Nota:

- Esta notación *conserva el* "data.frame" *class*: el resultado sigue siendo un dataframe.
- Esta notación se puede usar para indexar *múltiples elementos* de un dataframe en un nuevo dataframe, e.g. "df[c(1,3,5)]" o "df[c("sexo ","edad")]"

2. **Notación "[[]]"**, e.g."df[[3]]" o "df[[" variable.name"]]". Esto devuelve el elemento específico (columna) seleccionado, no como un dataframe sino como un vector con su propio tipo y clase, e.g. el vector numérico dentro del elemento 3^cer^ de "df". Téngase en cuenta dos diferencias con la notación "[]":

- "[[]]" *no* preserva la clase "data.frame". El resultado *no* es un dataframe.
- De acuerdo con esto, "[[]]" solo puede usarse para indexar un elemento *single*(*colum*) del dataframe.

3. **La notación "$"**. Si *variable.name* es el nombre de una variable específica (columna) en "df", entonces "df$variable.name" indexa esa variable. Esto es lo mismo que la notación "[[]]": "df$variable.name" es lo mismo que "df [["variable.name"]]", y también es lo mismo que "df[[i]] "(donde "i"es la posición de la variable llamada *variable.name* en el dataframe.

* **Notación de matriz**. Los dataframes también se pueden indexar como una matriz, con la notación "[,]":

- "df[2,3]", "df[2,]", "df[, 3]".
- "df[," age "]", "df[, c ("sex","age")]", "df[5,"age"]"

***Tener en cuenta la diferencia** entre lo siguiente:

- Extraer la variable de un dataframe (columna) como un vector singular (numérico, carácter, etc.) - El paquete de pimienta individual en la figura a continuación (panel C). Esto viene dado por "df[[i]]", "df[["variable.name"]]", "df$variable.name".

- Extracción de otro dataframe de solo una variable (columna) - El paquete de pimienta individual *dentro* del frasco de pimienta en la figura

anterior (panel B). Esto viene dado por "df[i]", "df["variable.name"]".

* **Subconjunto de dataframes.** En la base de R, hay diferentes formas de hacer subconjuntos de filas (casos) y / o columnas (variables) de un dataframe mediante la indexación (algunos ejemplos se encuentran en el código que se presenta a continuación). Sin embargo, usaremos principalmente las funciones de subconjunto introducidas por el paquete "dplyr" en "tidyverse":

  - "dplyr::filter()" se usa para subconjuntar filas (casos) en función de ciertas condiciones (es decir, encuestados con ciertos valores en una o más variables).
  - "dplyr::select()" se utiliza para subconjuntar columnas (variables). Puede usar nombres completos de variables o seleccionar variables de muchas otras maneras. Consultar los ejemplos en la "selección" (página del manual).

```
# Numeric indexing ----
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Let's use our vector of ego ages again.
age

## [1] 61 38 30 25 29 56 52 35 22 51
# Index its 2nd element: The age of the 2nd respondent.
Age[2]
## [1] 38

# Its 2nd, 4th and 5th elements.
Age[c(2,4,5)]

## [1] 38 25 29
# Fifth to sevents elements
age[5:7]

## [1] 29 56 52
# Use indexing to assign (replace) an element.
Age[2] <- 45
# The content of x has now changed.
Age

## [1] 61 45 30 25 29 56 52 35 22 51

# Let's index the adjacency matrix we created before.
Adj

## [,1] [,2] [,3]
## [1,] 0 1 1
## [2,] 1 0 1
## [3,] 0 0 0

# Its 2,3 cell: Edge from node 2 to node 3.
Adj[2,3]

## [1] 1

# Its 2nd column: All edges to node 2.
Adj[,2]
## [1] 1 0 0
# Its 2nd and 3rd row: All edges from nodes 2 and 3.
Adj[2:3,]

## [,1] [,2] [,3]
## [1,] 1 0 1
## [2,] 0 0 0
```

```
# Logical indexing ----
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Which values of "age" are between 40 and 60?
# Let's create a logical index that flags these values.
(ind <- age > 40 & age < 60)

## [1] FALSE TRUE FALSE FALSE FALSE TRUE TRUE FALSE FALSE TRUE
# Use this index to extract these values from vector "age" via logical indexing.
age[ind]
## [1] 45 56 52 51

# We could also have typed directly:
age[age > 40 & age < 60]
## [1] 45 56 52 51
# Indexing data frames: list notations ----
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# We'll use our ego-level data frame.
Ego.df

## # A tibble: 102 x 9
## ego_ID ego.sex ego.age ego.arr ego.edu ego.inc empl ego.empl.bin
## <dbl> <chr> <dbl> <dbl> <chr> <dbl> <dbl> <chr>
## 1 28 Male 61 2008 Second~ 350 3 Yes
## 2 29 Male 38 2000 Primary 900 4 Yes
## 3 33 Male 30 2010 Primary 200 3 Yes
## 4 35 Male 25 2009 Second~ 1000 3 Yes
## 5 39 Male 29 2007 Primary 0 1 No
## 6 40 Male 56 2008 Second~ 950 4 Yes
## 7 45 Male 52 1975 Primary 1600 3 Yes
## 8 46 Male 35 2002 Second~ 1200 4 Yes
## 9 47 Male 22 2010 Second~ 700 4 Yes
## 10 48 Male 51 2007 Primary 950 4 Yes
## # ... with 92 more rows, and 1 more variable: ego.age.cat <dbl>

# The 3rd variable.
ego.df[3]

## # A tibble: 102 x 1
## ego.age
## <dbl>
## 1 61
## 2 38
## 3 30
## 4 25
## 5 29
## 6 56
## 7 52
## 8 35
## 9 22
## 10 51
## # ... with 92 more rows

ego.df["ego.age"]
## # A tibble: 102 x 1
## ego.age
## <dbl>
## 1 61
## 2 38
## 3 30
## 4 25
## 5 29
## 6 56
```

```
## 7 52
## 8 35
## 9 22
## 10 51
## # ... with 92 more rows

# The second and third variables.
Ego.df[2:3]

## # A tibble: 102 x 2
## ego.sex ego.age
## <chr> <dbl>
## 1 Male 61
## 2 Male 38
## 3 Male 30
## 4 Male 25
## 5 Male 29
## 6 Male 56
## 7 Male 52
## 8 Male 35
## 9 Male 22
## 10 Male 51
## # ... with 92 more rows

# The [[ ]] notation extracts the actual column as a vector, while [ ] keeps
# the data frame class.
Ego.df[[“ego.age”]]

## [1] 61 38 30 25 29 56 52 35 22 51 50 45 51 32 57 42 32 55 44 25 28 60 54
## [24] 30 37 41 47 33 37 29 39 31 52 30 57 44 47 42 44 31 31 56 36 47 47 35
## [47] 43 22 55 32 NA 60 51 53 25 33 38 37 27 34 32 26 36 53 42 45 36 54 49
## [70] 35 27 32 58 51 55 59 27 35 24 52 27 42 47 43 53 54 35 50 51 40 49 33
## [93] 33 36 32 35 55 61 34 44 50 28

# The age variable, using the $ notation
ego.df$ego.age

## [1] 61 38 30 25 29 56 52 35 22 51 50 45 51 32 57 42 32 55 44 25 28 60 54
## [24] 30 37 41 47 33 37 29 39 31 52 30 57 44 47 42 44 31 31 56 36 47 47 35
## [47] 43 22 55 32 NA 60 51 53 25 33 38 37 27 34 32 26 36 53 42 45 36 54 49
## [70] 35 27 32 58 51 55 59 27 35 24 52 27 42 47 43 53 54 35 50 51 40 49 33
## [93] 33 36 32 35 55 61 34 44 50 28

# Note that this is the same as ego.df[[3]] or ego.df[["ego.age"]]
identical(ego.df[[3]], ego.df$ego.age)

## [1] TRUE

# Indexing data frames: matrix notation ----
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Fifth row of the data frame
ego.df[5,]

## # A tibble: 1 x 9
## ego_ID ego.sex ego.age ego.arr ego.edu ego.inc empl ego.empl.bin
## <dbl> <chr> <dbl> <dbl> <chr> <dbl> <dbl> <chr>
## 1 39 Male 29 2007 Primary 0 1 No
## # ... with 1 more variable: ego.age.cat <dbl>

# Multiple rows (4th to 10th)
ego.df[4:10,]

## # A tibble: 7 x 9
```

```
## ego_ID ego.sex ego.age ego.arr ego.edu ego.inc empl ego.empl.bin
## <dbl> <chr> <dbl> <dbl> <chr> <dbl> <dbl> <chr>
## 1 35 Male 25 2009 Second~ 1000 3 Yes
## 2 39 Male 29 2007 Primary 0 1 No
## 3 40 Male 56 2008 Second~ 950 4 Yes
## 4 45 Male 52 1975 Primary 1600 3 Yes
## 5 46 Male 35 2002 Second~ 1200 4 Yes
## 6 47 Male 22 2010 Second~ 700 4 Yes
## 7 48 Male 51 2007 Primary 950 4 Yes
## # ... with 1 more variable: ego.age.cat <dbl>

# Fifth and sixth row, fourth variable.
ego.df[5:6,4]

## # A tibble: 2 x 1
## ego.arr
## <dbl>
## 1 2007
## 2 2008

# sex and age variables.
ego.df[, c("ego.sex", "ego.age")]

## # A tibble: 102 x 2
## ego.sex ego.age
## <chr> <dbl>
## 1 Male 61
## 2 Male 38
## 3 Male 30
## 4 Male 25
## 5 Male 29
## 6 Male 56
## 7 Male 52
## 8 Male 35
## 9 Male 22
## 10 Male 51
## # ... with 92 more rows

# Subsetting data frames with dplyr: filter and select ----
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Filter to egos who are older than 40
filter(ego.df, ego.age > 40)

## # A tibble: 51 x 9
## ego_ID ego.sex ego.age ego.arr ego.edu ego.inc empl ego.empl.bin
## <dbl> <chr> <dbl> <dbl> <chr> <dbl> <dbl> <chr>
## 1 28 Male 61 2008 Second~ 350 3 Yes
## 2 40 Male 56 2008 Second~ 950 4 Yes
## 3 45 Male 52 1975 Primary 1600 3 Yes
## 4 48 Male 51 2007 Primary 950 4 Yes
## 5 49 Male 50 2001 Second~ 1300 4 Yes
## 6 51 Male 45 2011 Second~ 480 3 Yes
## 7 52 Male 51 2002 Univer~ 1200 4 Yes
## 8 55 Male 57 1979 Second~ 470 7 No
## 9 56 Male 42 1992 Primary 1100 4 Yes
## 10 58 Male 55 1990 Second~ 1450 4 Yes
## # ... with 41 more rows, and 1 more variable: ego.age.cat <dbl>

# Filter to egos with Primary education
filter(ego.df, ego.edu == "Primary")

## # A tibble: 42 x 9
## ego_ID ego.sex ego.age ego.arr ego.edu ego.inc empl ego.empl.bin
```

```
## <dbl> <chr> <dbl> <dbl> <chr> <dbl> <dbl> <chr>
## 1 29 Male 38 2000 Primary 900 4 Yes
## 2 33 Male 30 2010 Primary 200 3 Yes
## 3 39 Male 29 2007 Primary 0 1 No
## 4 45 Male 52 1975 Primary 1600 3 Yes
## 5 48 Male 51 2007 Primary 950 4 Yes
## 6 53 Male 32 2003 Primary 1600 1 No
## 7 56 Male 42 1992 Primary 1100 4 Yes
## 8 57 Male 32 2000 Primary 1200 4 Yes
## 9 60 Male 25 2011 Primary 0 1 No
## 10 64 Male 54 1981 Primary 300 3 Yes
## # ... with 32 more rows, and 1 more variable: ego.age.cat <dbl>

# Combine the two conditions: Intersection.
filter(ego.df, ego.age > 40 & ego.edu == "Primary")

## # A tibble: 21 x 9
## ego_ID ego.sex ego.age ego.arr ego.edu ego.inc empl ego.empl.bin
## <dbl> <chr> <dbl> <dbl> <chr> <dbl> <dbl> <chr>
## 1 45 Male 52 1975 Primary 1600 3 Yes
## 2 48 Male 51 2007 Primary 950 4 Yes
## 3 56 Male 42 1992 Primary 1100 4 Yes
## 4 64 Male 54 1981 Primary 300 3 Yes
## 5 68 Male 41 2008 Primary 600 4 Yes
## 6 82 Male 57 1982 Primary 1190 4 Yes
## 7 85 Male 42 2008 Primary 400 2 No
## 8 90 Male 56 2004 Primary 450 4 Yes
## 9 93 Male 47 2010 Primary 880 4 Yes
## 10 95 Male 43 1997 Primary 800 4 Yes
## # ... with 11 more rows, and 1 more variable: ego.age.cat <dbl>

# Combine the two conditions: Union.
Filter(ego.df, ego.age > 40 | ego.edu == “Primary”)

## # A tibble: 72 x 9
## ego_ID ego.sex ego.age ego.arr ego.edu ego.inc empl ego.empl.bin
## <dbl> <chr> <dbl> <dbl> <chr> <dbl> <dbl> <chr>
## 1 28 Male 61 2008 Second~ 350 3 Yes
## 2 29 Male 38 2000 Primary 900 4 Yes
## 3 33 Male 30 2010 Primary 200 3 Yes
## 4 39 Male 29 2007 Primary 0 1 No
## 5 40 Male 56 2008 Second~ 950 4 Yes
## 6 45 Male 52 1975 Primary 1600 3 Yes
## 7 48 Male 51 2007 Primary 950 4 Yes
## 8 49 Male 50 2001 Second~ 1300 4 Yes
## 9 51 Male 45 2011 Second~ 480 3 Yes
## 10 52 Male 51 2002 Univer~ 1200 4 Yes
## # ... with 62 more rows, and 1 more variable: ego.age.cat <dbl>

# Note that the object ego.df hasn’t changed. To re-use any of the filtered data
# frames above, we have to assign them to an object.
Ego.df.40 <- filter(ego.df, ego.age > 40)
# Select specific variables
dplyr::select(ego.df, ego.sex, ego.age)

## # A tibble: 102 x 2
## ego.sex ego.age
## <chr> <dbl>
## 1 Male 61
## 2 Male 38
## 3 Male 30
## 4 Male 25
## 5 Male 29
```

```
## 6 Male 56
## 7 Male 52
## 8 Male 35
## 9 Male 22
## 10 Male 51
## # ... with 92 more rows

# ***** EXERCISE (1):
# What's the mean age of respondents whose education is NOT "Primary"? Hint:
# Remember that the "different from" comparison operator is !=.
# *****
# ***** EXERCISE (2):
# What is the modal education level (i.e., the most common education level) of
# respondents who are older than 40? Use table() or freq().
# *****
# ***** EXERCISE (3):
# Create the fictitious variable var <- c(1:30, rep(NA, 3), 34:50). Use is.na()
# to index all the NA values in the variable. Then use is.na() to index all
# values that are *not* NA. Hint: Remember the operator use to negate a logical
# vector. Finally, use this indexing to remove all NA values from var.
# *****
# ***** EXERCISE (4):
# Use the $ notation to extract the "ego.arr" variable in ego.df and recode all
# values equal to 2008 as 99.
# *****
# ***** EXERCISE (5):
# Use the [ , ] notation to extract the education level and income of egos who
# are younger than 30. Then do the same thing with the dplyr::filter() function.
# *****
```

TUBERÍAS (PIPES) Y EL OPERADOR "%>%"

* El operador pipe %>% fue introducido por el paquete magrittr y es adoptado por igraph y tidyverse, que usamos en este taller. El operador se está volviendo cada vez más popular entre los desarrolladores de R.

*La idea detrás de las tuberías es, en esencia, muy simple:

- mean(table(x)) se convierte en x%>% table%>% mean.
- En general, f(g(x)) se convierte en x%>% g%>% f.
- Por lo tanto,%>% canaliza la salida de la función anterior (por ejemplo, *table*) en la entrada de la siguiente función (por ejemplo, *mean*). Esto convierte el código de adentro hacia afuera en código de izquierda a derecha. Dado que la mayoría de nosotros estamos aconstumbrados a esa dirección, las tuberías hacen que el código R sea menos engorroso y más legible.

*También es posible ver *pipes* concatenando múltiples líneas de código. Eso es posible y un estilo de codificación común. En vez de x%>% table%>% mean se puede escribir

x%>%

table%>%

mean

REFERENCIAS

Wickham, Hadley & Garrett Grolemund. (2017). *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. Sebastopol, CA: O'Reilly Media.

Remitido: 25-05-2020

Aceptado: 12-06-2020