Introducción a R

Programación para el análisis de datos

Departamento de Ciencias Sociales, UCU - Martín Opertti

¿Qué es y qué se puede hacer con R?

R y R Studio

- R es un software y un lenguaje de programación gratuito enfocado en el análisis estadístico y la visualización de datos.
- R cuenta con gran potencia y flexibilidad, así como una númerosa -y creciente- comunidad de usuarios tanto académicos como profesionales.

- R Studio es un entorno de desarrollo integrado (IDE) . O en otras palabras... es una interfaz un poco (bastante) más amigable que usar R directamente.

¿Por qué usar R?

- Es un software libre y gratuito
- Generar nuevas funciones es fácil, por lo que las está en constante desarrollo
- Tiene muchos usuarios de diversas disciplinas lo que genera una comunidad (particularmente mediante foros) que es de gran utilidad para la resolución de problemas de código
- Es uno de los programas más utilizados para técnicas innovadaras en estadística y visualización de datos
- Trabaja muy bien con otros programas/lenguajes (Excel, Latex, HTML, etc.)
- Es cada vez más usado tanto en el ámbito académico como profesional

¿Qué se puede hacer con R?

¿Qué se puede hacer con R?

- Ingresar datos
- Importar y exportar datos (de Excel, Stata, documento de texto, APIs, etc)
- Manipular datos (recodificación, cambios de estructura)
- Estadística descrpitiva (media, rango, etc.) e inferencial (prueba de hipótesis, regresión, etc.)
- Visualización de datos (gráficos de alta calidad)
- Técnicas estadísticas y de visualización de datos innovadoras
- Crear tus propias funciones y paquetes
- Escribir artículos y presentaciones integrando código
- Escribir libros
- Webscrapping, trabajar con Big data y machine learning
- Aplicaciones intercativas para visualización de datos (shiny apps)
- Hasta jugar videojuegos!

Algunos ejemplos: visualización de datos

Algunos ejemplos: visualización de datos

Algunos ejemplos: visualización de datos

Algunos ejemplos: visualización de datos

Algunos ejemplos: visualización de datos

Algunos ejemplos: visualización de datos

Algunos ejemplos: Webscrapping (APIs)

```
tweets_eim <- get_timeline("EscuelaMetodos", n = 2, token=twitter_token)
tweets_eim <- select(tweets_eim, text, retweet_count, favorite_count)
```

text	retweet_count	favorite_count
[#curso] ltimos das para inscribirse a nuestro curso de Mypes y digitalizacin, con foco en el anlisis y diseo de polticas. Consult por nuestro plan de becas en secretariacursos@oitcinterfor.org; y conoc ms sobre la propuesta ac: https://t.co/2VdhxGyylT https://t.co/qv8wW0eFLk	4	0
Nuestra profesora @RosselMCecilia estar hablando sobre transparencia y corrupcin en prximo 22/08. Ms informacin https://t.co/IRIL98Dsu7 https://t.co/UzKcr6foxx	6	0

Aplicaciones (shiny apps), paquetes y presentaciones

- Mirador DESCA
- OPUY
- Esta misma presentación, ver Xaringan
- r4ds

Aprendizaje

¿Qué tan difícil es aprender R?

¿Qué tan difícil es aprender R?

- Al comienzo puede ser más difícil que la sintaxis básica de otros programas, particularmente porque es un lenguaje bastante distinto al resto. Sin embargo, una vez que se logra cierto entendimiento y autonomía, las posibilidades son infinitas.
- Muchas de las dificultades para aprender R se deben a sus principales ventajas: flexibilidad y potencia.

Recursos

Ningún recurso es en si mismo suficiente para aprender R. Cada análisis de datos es particular en su manera y las soluciones no siempre estarán en el contenido de un curso o libro específico. Hay muchos recursos para aprender R de forma general y para obtener ayuda puntual.

Recursos

La comunidad de usuarios de R es inmensa y muy abierta. Por esto hay muchísimos recursos para aprender de forma independiente y resolver problemas cuando nos estancamos:

- Libro "R for Data Science". Es muy completo y referencia en la mayoría de los cursos de R, pueden acceder a la versión online original y a una traducción
- Hands On Programming with R es otro libro libre muy útil sobre R
- R Bloggers y rpubs publican miles de tutoriales para temas específicos
- Existen foros -por ej. Stack Overflow- donde responden una infinidad preguntas de programación en R.
- IntRo es un excelente curso de R (con gran contenido teórico) de FCS-UdelaR a cargo de Nicolás Schmidt
- AnalizaR es un libro sobre análisis de datos en R con énfasis en Ciencia Política

Ayuda

Ayuda

- Obtener la ayuda correcta es fundamental al programar en R. Podemos obtener ayuda de todas las funciones que utilizamos con el comando `help()`
- Si no podemos solucionar un error con la documentación de las funciones/paquetes muchas veces sirve buscar en un navegador
- Muchas páginas contienen información relevante para solucionar problemas, entre las que se destaca stackoverflow
- En caso de no encontrar solución se puede consultar en páginas como stackoverflow mediante un ejemplo reproducible o reprex

Ayuda

```
help(mean)
```

Primeros pasos

Abrimos R Studio

Personalizar R Studio (opcional)

R Studio

- **Source (editor):** es donde creamos y editamos los scripts, es decir, donde escribimos y almacenamos el código.
- **Console (consola):** imprime el código que corremos y la mayoría de los resultados. Podemos escribir código directamente aquí también, aunque si queremos guardarlo lo recomendable es hacerlo en el script.
- **Environment (ambiente):** Muestra todos los objetos que creaste en cada sesión.
- **Gráficos (y más):** Imprime los gráficos. En el mismo panel figuran otras pestañas como "Help" que sirve para buscar ayuda.

Scripts

- Es un archivo de texto con el código y anotaciones.
- Se crea arriba a la izquierda "file/New File/R Script" o `ctrl + shift + n`.
- Se guarda con `ctrl + s` y es un documento de texto como cualquier otro (word, txt). Esto nos permite reproducir paso a paso todo lo que hicimos durante nuestro análisis.
- Haciendo click luego en el script guardado se inicia R Studio.
- Para ejecutar una línea de código pueden usar el botón de "Run" arriba a la derecha o -más cómodo- `ctrl + enter`

Workspace

- R nos ofrece guardar el ambiente (objetos, funciones, datos, etc.) luego de terminada cada sesión (lo que llama workspace). Si lo guardamos, la próxima vez que abramos ese script, nos encontraremos todo como lo dejamos (existirán los mismos objetos, funciones y datos).
- Lo recomendado es NO guardar el workspace, y guardar solamente el Script. De esta forma, cuando retomemos nuestro análisis en una nueva sesión de R, podemos correrlo y chequear que efectivamente genere los que querramos.
- Pueden desactivar la pregunta en Tools - Global Options - General - desmarcando la opción "Save workspace into RData" y desmarcando "restore RData into workspace"

Lenguaje básico de R

Anotaciones

- Es importante ser prolijo y cuidadoso con lo que hacemos
- Los scripts nos dan la posibilidad de anotar comentarios, lo que es muy útil:

```
## Esta línea es una anotación.

## R ignora todo lo que está acá adentro (tiene que empezar con #)

## Podemos escribir nombres de funciones u objetos y R no las va a
# interpetar

## Usar anotaciones es clave para poder entender qué fue lo que
# hicimos anteriormente
```

- De esta forma, podemos comentar que fue lo que hicimos para acordarnos nosotros, y que los demás entiendan

R como calculadora

Para empezar, R sirve como calculadora. Se pueden realizar operaciones matemáticas, por ejemplo:

```
# Operaciones sencillas
2 + 2
```

```
## [1] 4
```

```
20 - 10
```

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

```
10 / 2
```

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

```
10 * 10
```

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

Objetos en R

En muchos programas estadísticos (como Stata) solemos solamente "imprimir" resultados (lo que llamamos expresiones). En R podemos utilizar este enfoque:

```
# Una operación sencilla:
43*47 # Se imprime el resultado
```

```
## [1] 2021
```

Sin embargo, en R también podemos almacenar los resultados en objetos. Creamos los objetos mediante asignaciones (`<-`). En este caso, guardemos el valor (a diferencia de imprimirlo).

```
year <- 43*47 # Se crea un objeto
```

Si a esto lo ponemos entre paréntesis combinamos ambos enfoques: se guarda el objeto y se imprime el resultado

```
(year <- 43*47) # Se crea un objeto y se imprime
```

```
## [1] 2021
```

Asignaciones

- El símbolo para crear un objeto es `<-` (alt + -) y se llama asignador, también se puede usar `=` pero no es recomendable.
- Las asignaciones se crean de la siguiente manera: nombre_del_objeto `<-` valor.
- Como vimos, una vez que creo un objeto, R (por defecto) no imprime su valor. Este se puede obtener escribiendo simplemente el nombre del objeto o mediante la función `print()`:

```
year <- 43*47 # Se crea un objeto

year # Imprime el objeto year
```

```
## [1] 2021
```

```
print(year) # Imprime el objeto year
```

```
## [1] 2021
```

Algunos comandos básicos

```
ls() # Lista los objetos en el ambiente

rm(year) # Borra objeto del ambiente

rm(list=ls()) # Borra todos los objetos del ambiente

help(ls) # Buscar ayuda sobre una función
```

Objetos

Clases y tipos de objetos

- En R utilizamos constantemente objetos. Cada objeto tiene una clase, tipo y atributos.
- Esto es importante porque las funciones que podemos aplicar a nuestros datos dependen del objeto en el que los definimos.
- El uso de objetos tiene muchos benificios como extraer parte de ellos para determinados usos, duplicarlos o realizar operaciones sin imprimir en la consola.

Tipos de objetos

El tipo de un objeto refiere a cuál es el tipo de los datos dentro del objeto. Los tipos más comunes son:

Nombre	Tipos	Ejemplo
integer	Númerico: valores enteros	10
double	Númerico: valores reales	10.5
character	Texto	Diez
logical	Lógico (TRUE or FALSE)	TRUE

Clases o estructura de datos

Las clases de objetos son formas de representar datos para usarlos de forma eficiente. Se dividen en cuántas dimensiones tienen y si poseen distintos tipos de datos o no. Las clases de datos más comunes en R son:

- `vector` (vectores): es la forma más simple, son unidimensionales y de un solo tipo
- `lists` (listas): son unidimensionales pero no están restringidas a un solo tipo de datos
- `matrix` (matrices): tienen dos dimensiones (filas y columnas) y un solo tipo de datos.
- `dataframes` (marcos de datos): son el tipo de estructura al que más acostumbrado estamos, con dos dimensiones (filas y columnas) y puede incluir distintos tipos de datos (uno por columna). Pueden considerarse como listas de vectores con el mismo tamaño.

En ocasiones podemos transformar objetos de una clase a otra.

Clases y tipos de objetos

R variables and data types: Introduction to R Programming, Sydney-Informatics

Funciones para explorar objetos

R tiene funciones que nos permiten identificar la clase, el tipo, la estructura y los atributos de un objeto.

- `class()` - ¿Qué tipo de objeto es?
- `typeof()` - ¿Qué tipos de data tiene el objeto?
- `length()` - ¿Cuál es su tamaño?
- `attributes()` - ¿Tiene metadatos?

Clases y tipos de objetos

```
  # Creamos algunos objetos distintos
  year <- 2021
  nombre <- "Dos mil veintiuno"

  # Uso la función class para averiguar la clase de objeto
  class(year)
```

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

```
  class(nombre)
```

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

```
  # Uso la función typeof para averiguar el tipo de la data del objeto
  typeof(nombre)
```

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

Clases y tipos de objetos

Todo lo que escribimos entre comillas se interpeta como texto, por más que sean números.

```
  year_2 <- "2021"

  class(year_2)
```

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

```
  vof <- TRUE
  class(vof)
```

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

¿Por qué importan los tipos y clases?

Supongamos que creamos un objeto con el valor 10, al que luego le sumaremos otro objeto con el valor 20.

```
obj_1 <- "10"

class(obj_1)
```

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

```
obj_1 + 20 # Da error
```

```
## Error in obj_1 + 20: non-numeric argument to binary operator
```

¿Por qué importan los tipos y clases?

En cambio, si creamos el objeto de tipo númerico:

```
obj_1 <- 10

class(obj_1)
```

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

```
obj_1 + 20 # Funciona
```

```
## [1] 30
```

¿Por qué importan los tipos y clases?

Normalmente no trabajamos con objetos de un solo valor, y reescribirlos no es una opción. Para ellos tenemos coercionadores `as.logical()`, `as.integer()`, `as.double()`, o `as.character()`: funciones que transforman un objeto de un tipo a otro. En este caso:

```
obj_1 <- "10"

class(obj_1)
```

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

```
obj_1 <- as.numeric(obj_1)
class(obj_1)
```

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

```
is.numeric(obj_1) # Podemos verificarlo directamente también
```

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

Vectores

Un vector es una colección de elementos. Los vectores atómicos son los que contienen elementos todos del mismo tipo (que es lo más normal en el análisis de datos). Hay 4 tipos de vectores: lógicos, character, integer y double (estos dos últimos son numéricos). Los elementos determinarán el tipo del objeto. Crear un vector es muy sencillo mediante la función `c()`:

```
mi_primer_vector <- c(1, 3, 5, 7, 143)
print(mi_primer_vector)
```

```
## [1]   1   3   5   7 143
```

```
class(mi_primer_vector)
```

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

```
length(mi_primer_vector)
```

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

```
str(mi_primer_vector)
```

```
##  num [1:5] 1 3 5 7 143
```

Vectores

```
v1 <- c(1:5) # Todos los números de 1 a 5
v1
```

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

```
v2 <- seq(0, 50, 10) # De 0 a 50 de a 10 números
v2
```

```
## [1]  0 10 20 30 40 50
```

```
v3 <- c(v1, v2) # Combino vectores creando un nuevo vector
v3
```

```
##  [1]  1  2  3  4  5  0 10 20 30 40 50
```

```
v4 <- c("rojo", "verde", "blanco") # character
v4
```

```
## [1] "rojo"   "verde"  "blanco"
```

```
v5 <- c(TRUE, TRUE, FALSE, TRUE) # lógico
v5
```

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

Indexación

Cuando queremos referirnos a uno o varios elementos dentro de un vector utilizamos `[]` (indexación).

```
## Indexación:
v2
```

```
## [1]  0 10 20 30 40 50
```

```
v2[1] # El primer elemento dentro del vector
```

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

```
# Nos sirve por ejemplo para extraer partes del vector:
v3 <- v2[1:3] # Creo nuevo vector con los elementos del 1 al 3
v3
```

```
## [1]  0 10 20
```

Operaciones con vectores

También podemos realizar operaciones con los vectores númericos:

```
## Operaciones con vectores:
v3
```

```
## [1]  0 10 20
```

```
v3 + 2 # Se realiza la operación sobre cada elemento del vector
```

```
## [1]  2 12 22
```

Coerción

¿Qué pasa si unimos vectores de distinto tipo?

Si unimos vector de tipo caracter con uno númerico, R convertirá todo el vector a caracter. Si unimos un vector númerico (double o integer) a lógico, R convertirá el vector en númerico (`TRUE = 1`, `FALSE = 0`)

```
## Ejemplo de coerción automática:
v2 <- seq(0, 50, 10) # De 0 a 50 de a 10 números
v4 <- c("rojo", "verde", "blanco") # character
v6 <- c(v2, v4)
v6
```

```
## [1] "0"      "10"     "20"     "30"     "40"     "50"     "rojo"   "verde"
## [9] "blanco"
```

```
class(v6)
```

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

Ejercicio

(1) Crear un vector con la edad de los integrantes de tu hogar

(2) Realizar una operación sobre ese vector para calcular la edad de cada integrante en 2030