# Laboratorio 1: Estadística Inferencial

C. Torres - J. Ugarte

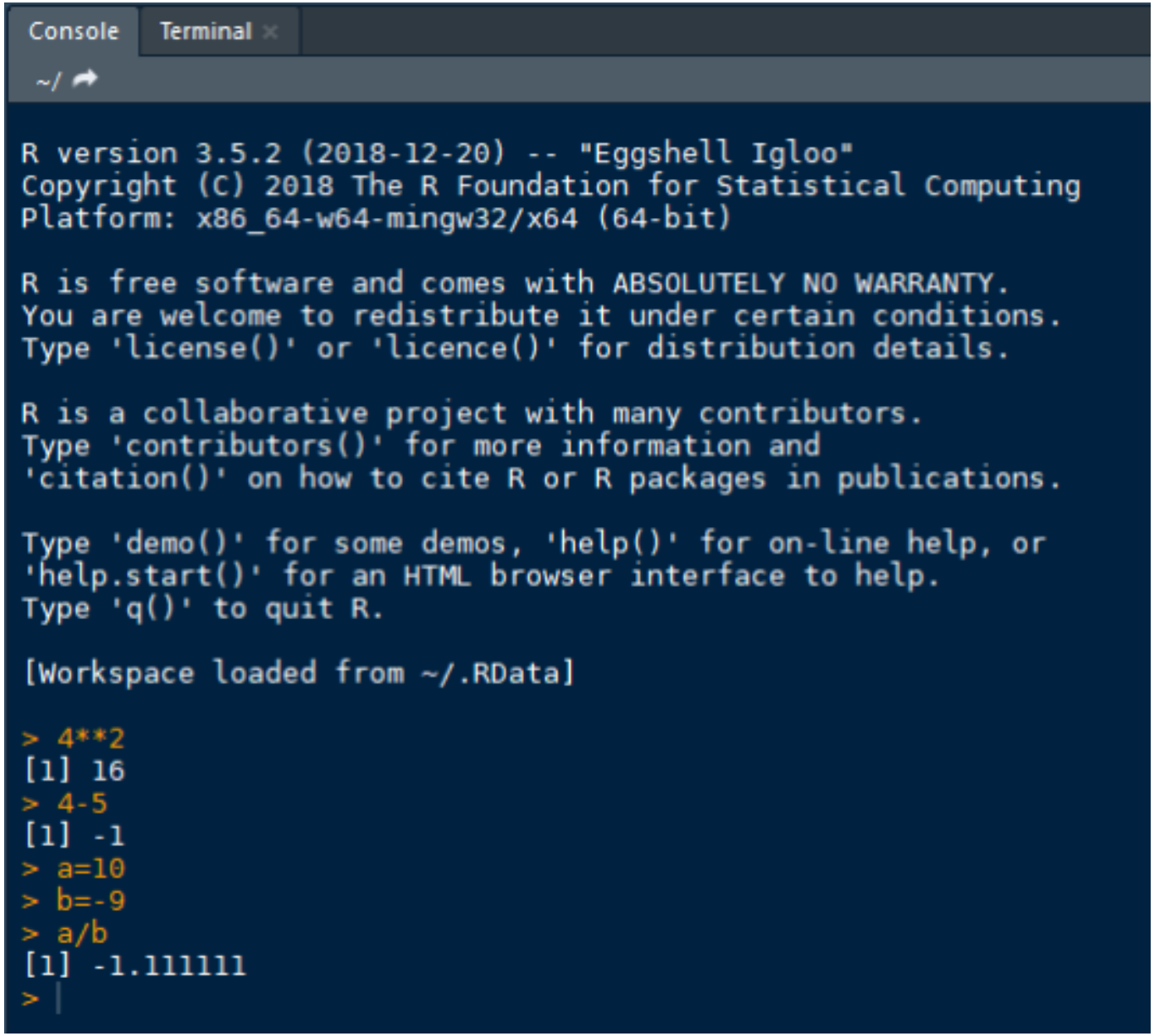
Facultad de Ciencias (UNI)

22 de agosto de 2019

# 1. R

1. Instalación de R.
   - 1.1 https://cloud.r-project.org
   - 1.2 Seleccione Download R for Windows
   - 1.3 Seleccione install R for the first time.
2. Descargue RStudio del siguiente enlace
   - 2.1 Enlace: https://www.rstudio.com/products/rstudio/download
   - 2.2 Seleccione RStudio FREE DOWNLOAD
   - 2.3 Seleccione el sistema operativo y la versión de su PC
3. Instale R
4. Instale RStudio
5. Para culminar la instalación, ejecute en la consola de RStudio las siguientes operaciones básicas para garantizar que la instalación es correcta.

# 1.1. Instalación de R



# 1.2. Introducción:¿Qué es R y para qué es usado?

- R es un lenguaje de programación y entorno computacional dedicado a la estadística.
- Decimos que es un lenguaje de programación porque nos permite dar instrucciones, usando código, a nuestros equipos de cómputo para que realicen tareas específicas.
- R es diferente a otros lenguajes de programación que por lo general están diseñados para realizar muchas tareas diferentes; esto es porque fue creado con el único propósito de hacer estadística.
- Esta característica es la razón de que R sea un lenguaje de programación peculiar, que puede resultar absurdo en algunos sentidos para personas con experiencia en otros lenguajes, pero también es la razón por la que R es una herramienta muy poderosa para el trabajo en estadística

# 1.3. RStudio un IDE para R

- Aunque podemos usar R directamente, es recomendable instalar y usar un entorno integrado de desarrollo (IDE, por sus siglas en inglés).
- Un IDE nos proporciona herramientas para escribir y revisar nuestro código, administrar los archivos que estamos usando, gestionar nuestro entorno de trabajo y algunas otras herramientas de productividad. Tareas que serían difíciles o tediosas de realizar de otro modo, son fáciles a través de un IDE.
- Hay varias opciones de IDE para R, y entre ellas mi preferido es RStudio. Este entorno, además de incorporar las funciones esenciales de una IDE, es desarrollado por un equipo que ha contribuido de manera significativa para lograr que R sea lenguaje de programación más accesible, con un énfasis en la colaboración y la reproducción de los análisis.

# 1.4. La consola de R

- Lo primero que nos encontramos al ejecutar R es una pantalla que nos muestra la versión de este lenguaje que estamos ejecutando y un prompt:

  $>$ comand

- Esta es la consola de R y corresponde al entorno computacional de este lenguaje. Es aquí donde nuestro código es interpretado.
- Si estás usando RStudio, te encontrarás la consola de R en uno de los páneles de este programa.

# 1.5. Objetos

- En R, todo es un objeto. Todos los datos y estructuras de datos son objetos. Además, todos los objetos tienen un nombre para identificarlos.
- La explicación de esto es un tanto compleja y se sale del alcance de este curso. Se relaciona con el paradigma de programación orientada a objetos y ese es todo un tema en sí mismo.
- Lo importante es que recuerdes que al hablar de un objeto, estamos hablando de cualquier cosa que existe en R y que tiene un nombre.

# 1.6. Constantes y variables

- De manera análoga al uso de estos términos en lenguaje matemático, una constante es un objeto cuyo valor no podemos cambiar, en contraste, una variable es un objeto que puede cambiar de valor.
- Las constantes y variables en R tienen nombres que nos permiten hacer referencia a ellas en operaciones.
- En R usamos $<-$ o $=$ para hacer asignaciones. De este modo, podemos asignar el valor 3 a la variable radio

```
# Este es un comentario
# asignar el valor 3 a la variable radio
> radio <- 3
> radio = 3
> radio
[1] 3
```

# 1.7. Funciones (introducción básica)

- Una función es una serie de operaciones a la que les hemos asignados un nombre. Las funciones aceptan argumentos, es decir, especificaciones sobre cómo deben funcionar.
- En R reconocemos a una función usando la notación: *nombre_de_la_fu*
  Por ejemplo:

  **mean**( ) , **quantile** ( ) , **summary** ( ) , **density** ( ) , **c** ( )

# 1.8. Documentación

- Las funciones de R base y aquellas que forman parte de paquete tienen un archivo de documentación.
- Este archivo describe qué hace la función, sus argumentos, detalles sobre las operaciones que realiza,los resultados que devuelve y ejemplos de uso.
- Para obtener la documentación de una función, escribimos el ? antes de su nombre y lo ejecutamos. También podemos usar la función *help*(), con el nombre de la función.

```
> ?mean()
> help("mean")
```

- También podemos obtener la documentación de un paquete, si damos el argumento *package* a la función *help*(), con el nombre de un paquete.

```
> help(package = "stats")
```

# 1.9. Directorio de trabajo

- El directorio o carpeta de trabajo es el lugar en nuestra computadora en el que se encuentran los archivos con los que estamos trabajando en R. Este es el lugar donde R buscara archivos para importarlos y al que serán exportados, a menos que indiquemos otra cosa.
- Puedes encontrar cuál es tu directorio de trabajo con la función *getwd*().

```
> getwd()
[1] "C:/Users/User/Documents"
```

- Puedes cambiar el directorio de trabajo usando la función setwd(), dando como argumento la ruta del directorio que quieres usar.

```
setwd("C:\otro_directorio")
```

# 1.10. Directorio de trabajo

- Si deseas conocer el contenido de tu directorio de trabajo, puedes ejecutar. la función *list.files*(), sin argumentos, que devolverá una lista con el nombre de los archivos de tu directorio de trabajo.
- La función *list.dirs*(), también sin argumentos te dará una lista de los directorios dentro del directorio de trabajo.

```
# Ver archivos
list.files()
# Ver directorios
>list.dirs()
```

# 1.11. Sesión

- Los objetos y funciones de R son almacenados en la memoria *RAM* de nuestra computadora.
- Cuando ejecutamos R, ya sea directamente o a través de RStudio, estamos creando una instancia del entorno del entorno computacional de este lenguaje de programación, **cada instancia es una sesión**.
- Todos los objetos y funciones creadas en una sesión, permanecen sólo en ella, no son compartidos entre sesiones, sin embargo una sesión puede tener el mismo directorio de trabajo que otra sesión.
- Para conocer los objetos y funciones que contiene nuestra sesión, usamos la función *ls*(), que nos devolverá una lista con los nombres de todo lo guardado en la sesión.

```
ls()
```

# 1.12. Paquetes

- R puede ser expandido con **paquetes**. Cada paquete es una colección de funciones diseñadas para atender una tarea específica. Por ejemplo, hay paquetes para trabajo visualización geoespacial, análisis psicométricos, mineria de datos, interacción con servicios de internet y muchas otras cosas más.
- Estos paquetes se encuentran alojados en *CRAN*, así que pasan por un control riguroso antes de estar disponibles para su uso generalizado.
- Podemos instalar paquetes usando la función *install.packages*(), dando como argumento el nombre del paquete que deseamos instalar, entre comillas.

```
install.packages("readr")
```

# Paquetes

- Una vez concluida la instalación de un paquete, podrás usar sus funciones con la función *library*().

```
library(readr)
```

- En caso de escribir en *install.packages*() el nombre de un paquete no disponible en *CRAN*, se nos mostrará una advertencia y no se instalará nada.
- Los paquetes que hemos importado en nuestra sesión actual aparecen al llamar *sessionInfo*().

```
sessionInfo()
```

- También podemos ver qué paquetes tenemos ya instalados ejecutando la función *installed.packages*() sin ningún argumento.

```
installed.packages()
```

# 1.13. Scripts

- Los scripts son documentos de texto con la extensión de archivo *.R*, por ejemplo *mi_script.R*.
- Estos archivos son iguales a cualquier documentos de texto, pero R los puede leer y ejecutar el código que contienen.
- Aunque R permite el uso interactivo, es recomendable que guardes tu código en un archivo .R, de esta manera puedes usarlo después y compartirlo con otras personas. En realidad, en proyectos complejos, es posible que sean necesarios mútiples scripts para distintos fines.
- Podemos abrir y ejecutar scripts en R usando la función *source*(), dandole como argumento la ruta del archivo .R en nuestra computadora, entre comillas.

```
source("C:/Mis_scripts/mi_script.R")
```

# 2. Tipos de datos

- En R los datos pueden ser de diferentes tipos. Cada tipo tiene características particulares que lo distinguen de los demás. Entre otras cosas algunas operaciones sólo pueden realizarse con tipos de datos específicos.
- Los tipos de datos de uso más común en R son los siguientes.

| Tipo | Ejemplo | Nombre en inglés |
|---|---|---|
| Entero | 1 | integer |
| Numérico | 1,3 | numeric |
| Cadena de texto | "uno" | character |
| Factor | uno | factor |
| Lógico | TRUE | logical |
| Perdido | NA | NA |
| Vacio | NULL | null |

# 2.1. Entero y numérico

- Como su nombre lo indica, los datos enteros representan números enteros, sin una parte decimal o fraccionaria, que pueden ser usados en operaciones matemáticas.
- Los datos numéricos también son llamados doble o float (flotantes). Este nombre se debe a que, en realidad, son números de doble precisión, pues tienen una parte entera y una fraccionaria decimal, y son llamados float debido a que se usa un punto flotante para su representación computacional.

```
> y=1.5
> mode(y)
[1] "numeric"
> typeof(y)
[1] "double"
> class(y)
[1] "numeric"
```

# 2.2. Cadena de texto

- El tipo character representa texto y es fácil reconocerlo porque un dato siempre esta rodeado de comillas, simples o dobles. De manera convencional, nos referimos a este tipo de datos como cadenas de texto, es decir, secuencias de caracteres.
- Este es el tipo de datos más flexible de R, pues una cadena de texto puede contener letras, números, espacios, signos de puntuación y símbolos especiales.

```
> a='hola mundo'
> mode(a)
[1] "character"
> typeof(a)
[1] "character"
> class(a)
[1] "character"
```

# 2.3. Factor

- Un factor es un tipo de datos específico a R. Puede ser descrito como un dato numérico representado por una etiqueta.
- Supongamos que tenemos un conjunto de datos que representan el sexo de personas encuestadas por teléfono, pero estos se encuentran capturados con los números 1 y 2. El número 1 corresponde a femenino y el 2 a masculino.
- cada una de las etiquetas o valores que puedes asumir un factor se conoce como nivel. En nuestro ejemplo con femenino y masculino, tendríamos dos niveles.

```
> x <- factor(c(1, 2, 2, 1, 1, 2, 1, 2, 1))
> x
[1] 1 2 2 1 1 2 1 2 1
Levels: 1 2
> y <- as.factor(x)
> y
> levels(y)
> summary(y)
```

# 2.4. Lógico

- Los datos de tipo lógico sólo tienen dos valores posibles: verdadero **(TRUE)** y falso **(FALSE)**. Representan si una condición o estado se cumple, es verdadero, o no, es falso.
- Este tipo de dato es, generalmente, el resultado de operaciones relacionales y lógicas, son esenciales para trabajar con **álgebra Booleana**
- Como este tipo de dato sólo admite dos valores específicos, es el más restrictivo de R.

```
> Z=TRUE
> mode(Z)
[1] "logical"
```

# 2.5. NA y NULL

- En R, usamos **NA** para representar datos perdidos, mientras que **NULL** representa la ausencia de datos.
- La diferencia entre las dos es que un dato **NULL** aparece sólo cuando R intenta recuperar un dato y no encuentra nada, mientras que **NA** es usado para representar explícitamente datos perdidos, omitidos o que por alguna razón son faltantes.
- Por ejemplo, si tratamos de recuperar la edad de una persona encuestada que no existe, obtendríamos un **NULL**, pues no hay ningún dato que corresponda con ello. En cambio, si tratamos de recuperar su estado civil, y la persona encuestada no contestó esta pregunta, obtendríamos un **NA**

```
> x=c(1,NA, 'juan', TRUE,FALSE)
> x=c(1,NA, 'juan', TRUE)
> class(x)
[1] "character"
> as.logical(x)
[1]   NA   NA   NA TRUE
```

# 2.6. Atributos

- Como tabla resumen, se muestran a continuación los atributos que pueden ser de información de los objetos:

| Atributos | Información [Se usa la función] |
|---|---|
| Modo | Cualquier tipo de entidad que maneja R. [mode(objeto)]. |
| Tipo | Tipo de datos de los objetos: entero, carácter, double, etc. [typeof(objeto)]. |
| Nombres | Etiquetas de los elementos individuales de un vector o lista [names(objeto)]. |
| Dimensiones | Dimensiones de los arrays (alguna puede ser cero) [dim(objeto)]. |
| Dimnames | Nombres de las dimensiones de los arrays. [dimnames(objeto)]. |
| Clase | Vector alfanumérico con la lista de las clases del objeto. [class(objeto)]. |
| Otros atributos | Atributos de una serie temporal. |

# Atributos

- Y por último, la función *attributes*(*objeto*) proporciona una lista de todos los atributos no intrínsecos de
nidos para el objeto en ese momento. La función *attr*(*objeto*, *nombre*) puede usarse para seleccionar un atributo específico.
- Estas funciones no se utilizan habitualmente, sino que se reservan para la creación de un nuevo atributo con fines específicos, por ejemplo, para asociar una fecha de creación o un operador con un objeto de R.
- En definitiva, como ya se ha expuesto, casi todo en R son objetos, como ejemplo:

# Atributos

| Objetos | Información |
|---|---|
| Vector | Cadenas unidimensionales (es decir una sola columna o fila) de un tipo único de valores (numéricos, caracteres, etc.) |
| Matrices | Estructura bidimensional donde todos los datos deben ser del mismo tipo. |
| Factores | Este tipo de objeto es fundamental para el análisis estadístico pues es la forma como se tratan las variables categóricas. |
| Listas | Las listas son objetos genéricos que corresponden a colecciones de distintos objetos que pueden ser de tipos distintos. |
| Data Frames | Como una matriz pero pudiendo contener distintos tipos de datos. Es lo que clásicamente se ha llamado base de datos o hoja de datos. |

# 2.7. Coerción

- En R, los datos pueden ser coercionados, es decir, forzados, para transformarlos de un tipo a otro.
- La coerción es muy importante. Cuando pedimos a R ejecutar una operación, intentará coercionar de manera implícita, sin avisarnos, los datos de su tipo original al tipo correcto que permita realizarla.
- Lo anterior ocurre porque no todos los tipos de datos pueden ser transformados a los demás, para ello se sigue una regla general.
  lógico $->$ entero $->$ numérico $->$ cadena de texto
  (logical $->$ integer $->$ numeric $->$ character)
- Las coerciones no pueden ocurrir en orden inverso.

# 2.7.1. Coerción explícita con la familia *as*()

- También podemos hacer coerciones explícitas usando la familia de funciones *as*().

| Función | Tipo al que hace coerción |
|---|---|
| as.integer() | Entero |
| as.numeric() | Numerico |
| as.character() | Cadena de texto |
| as.factor() | Factor |
| as.logical() | Lógico |
| as.null() | NULL |

```
as.character(5)
as.numeric("cinco")
as.factor(5)
# Cadena de texto
as.character(factor_cinco)
# Numerico
as.numeric(factor_cinco)
```

# 2.7.2. Coerción explícita con la familia *as*()

- ```
  as.numeric(TRUE)
  as.numeric(FALSE)
  # Logico
  as.null(FALSE))
  # Numerico
  as.null(457)
  # Cadena de texto
  as.null("palabra")
  ```
  ```

# 2.8. Verificación con la familia de funciones *is*()

- También podemos verificar si un dato es de un tipo específico con la familia de funciones *is*().

| Función | Tipo que verifican |
|---|---|
| is.integer() | Entero |
| is.numeric() | Numerico |
| is.character() | Cadena de texto |
| is.factor() | Factor |
| is.logical() | Lógico |
| is.nan() | NA |
| is.null() | NULL |

- Estas funciones toman como argumento un dato, si este es del tipo que estamos verificando, nos devolverán *TRUE* y en caso contrario devolverán *FALSE*.

```
is.numeric(5)
is.character(5)
```

# 3. Operadores

- Los operadores son los símbolos que le indican a R que debe realizar una tarea. Combinando datos y operadores es que logramos que R haga su trabajo.
- Existen operadores específicos para cada tipo de tarea. Los tipos de operadores principales son los siguientes:
  - Aritméticos
  - Relacionales
  - Lógicos
  - De asignación
- Familiarizarnos con los operadores nos permitirá manipular y transformar datos de distintos tipos.

# 3.1. Operadores aritméticos

- En R tenemos los siguientes operadores aritméticos:

| Operador | Operación | Ejemplo | Resultado |
|---|---|---|---|
| $+$ | Suma | $5 + 3$ | 8 |
| $-$ | Resta | $5 - 3$ | 2 |
| $*$ | Multiplicación | $5 * 3$ | 15 |
| / | División | 5 /3 | 1.666667 |
| ^ | Potencia | 5 ^ 3 | 125 |
| %% | División entera | 5 %% 3 | 2 |

- Es posible realizar operaciones aritméticas con datos de tipo entero y numérico.

```
> 4 + "tres"
> 21 * NA
> NA ^ 13
```

# 3.2. Operadores relacionales

- Los operadores lógicos son usados para hacer comparaciones y siempre devuelven como resultado **TRUE** o **FALSE** (verdadero o falso, respectivamente).

| Operador | Comparación | Ejemplo | Resultado |
|---|---|---|---|
| $<$ | Menor que | $5 < 3$ | FALSE |
| $<=$ | Menor o igual que | $5 <= 3$ | FALSE |
| $>$ | Mayor que | $5 > 3$ | TRUE |
| $>=$ | Mayor o igual que | $5 >= 3$ | TRUE |
| $==$ | Exactamente igual que | $5 == 3$ | FALSE |
| $!=$ | No es igual que | $5! = 3$ | TRUE |

- Es posible comparar cualquier tipo de dato sin que resulte en un error.
- Sin embargo, al usar los operadores $>$, $>=$, $<$ y $<=$ con cadenas de texto, estos tienen un comportamiento especial.

```
> "casa" > "barco"
> as.factor("casa") > "barco"
```

# 3.3. Operadores lógicos

| Operador | Comparación | Ejemplo | Resultado |
|---|---|---|---|
| $x\|y$ | x ó y es verdadero | TRUE\|FALSE | TRUE |
| $x\&y$ | x y y son verdaderos | TRUE&FALSE | FALSE |
| $!x$ | x no es verdadero ) (negación) | !TRUE | FALSE |
| $isTRUE(x)$ | x es verdadero (afirmación) | isTRUE(TRUE) | TRUE |

- Los operadores lógicos son usados para operaciones de **álgebra Booleana**, es decir, para describir relaciones lógicas, expresadas como verdadero (**TRUE**) o falso (**FALSO**).
- Estos operadores pueden ser usados con estos con datos de tipo numérico, lógico y complejo. Al igual que con los operadores relacionales, los operadores lógicos siempre devuelven TRUE o FALSE.

# Operadores lógicos

- Para realizar operaciones lógicas, todos los valores numéricos y complejos distintos a 0 son coercionados a TRUE, mientras que 0 siempre es coercionado a FALSE.

```
> 5 | 0
> 5 & 0
> isTRUE(0)
> isTRUE(5)
> !(FALSE | FALSE)
```

# 3.4. Operadores de asignación

- Este es probablemente el operador más importante de todos, pues nos permite asignar datos a variables.

| Operador | Operación |
|---|---|
| $<-$ | Asigna un valor a una variable |
| $=$ | Asigna un valor a una variable |

```
> estatura <- 1.73
> peso <- 83
> peso / estatura ^ 2
> peso <- 76
> peso / estatura ^ 2
> V_i <- 110
> V_f <- 185
> T_i <- 0
> T_f <- 15
> raz_camb <- (V_f - V_i)/(T_f-T_i)
```

# 3.5. Orden de operaciones

- En R, al igual que en matemáticas, las operaciones tienen un orden de evaluación definido.
- El orden de operaciones incluye a las aritméticas, relacionales, lógicas y de asignación.

| Orden | Operadores |
|---|---|
| 1 | ^ |
| 2 | *, / |
| 3 | +, - |
| 4 | <, >, <=, >=, ==, != |
| 5 | ! |
| 6 | & |
| 7 | \| |
| 8 | <- |

- Si deseamos que una operación ocurra antes que otra, rompiendo este orden de evaluación, usamos paréntesis (Podemos tener paréntesis anidados.).

# 4. Estructuras de datos

- Las estructuras de datos son objetos que contienen datos. Cuando trabajamos con R, lo que estamos haciendo es manipular estas estructuras.
- Las estructuras tienen diferentes características. Entre ellas, las que distinguen a una estructura de otra son su número de **dimensiones** y si son **homogeneas** o **hereterogeneas**.

| Dimensiones | Homogeneas | Heterogeneas |
|---|---|---|
| 1 | Vector | Lista |
| 2 | Matriz | Data frame |
| n | Array | |

# 4.1. Vectores

- Un vector es la estructura de datos más sencilla en R. Un vector es una colección de uno o más datos del mismo tipo.
- Todos los vectores tienen tres propiedades:
  - **Tipo**. Un vector tiene el mismo tipo que los datos que contiene (atómicos).
  - **Largo**. Es el número de elementos que contiene un vector (dimensión).
  - **Atributos**. Los vectores pueden tener metadatos de muchos tipos, los cuales describen características de los datos que contienen.

```
> is.vector(3)
> length(3)
> is.vector("tres")
> length("tres")
```

# 4.1.1. Creación de vectores

- Creamos vectores usando la función $c()$ (combinar).

```
# Vector num\'erico
c(1, 2, 3, 5, 8, 13)
# Vector de cadena de texto
c("arbol", "casa", "persona")
# Vector l\'ogico
c(TRUE, TRUE, FALSE, FALSE, TRUE)
# agregar un elemento a un vector ya existente
mi_vector <- c(TRUE, FALSE, TRUE)
mi_vector <- c(mi_vector, FALSE)
# combinaci\'on de vectores
mi_vector_1 <- c(1, 3, 5)
mi_vector_2 <- c(2, 4, 6)
mi_vector_3 <- c(mi_vector_1, mi_vector_2)
```

# Creación de vectores

- Si intentamos combinar datos de diferentes tipos en un mismo vector, R realizará coerción automáticamente. El vector resultante será del tipo más flexible entre los datos que contenga, siguiendo las reglas de **coerción**.

```
mi_vector <- c(1, 2, 3)
class(mi_vector)
# agregar un dato de tipo cadena de texto
mi_vector_nuevo <- c(mi_vector, "a")
class(mi_vector_nuevo)
# otro ejemplo
mi_vector_mezcla <- c(FALSE,2,"tercero",4.00)
class(mi_vector_mezcla)
```

# Creación de vectores

- Podemos crear vectores de secuencias numéricas usando :. De un lado de los dos puntos escribimos el número de inicio de la secuencia y del otro el final.

```
> 1:10
> 10:1
# N\'umero negativo
-43:-30
# Se conservan los decimales del inicio
-2.48:2
# Se redondean los decimales del final
166:170.05
```

# 4.1.2. Vectorización de operaciones

- Existen algunas operaciones al aplicarlas a un vector, se aplican a cada uno de sus elementos. A este proceso le llamamos **vectorización**.

```
mi_vector <- c(2, 3, 6, 7, 8, 10, 11)
# Operaciones aritm\'eticas
mi_vector + 2
mi_vector * 2
mi_vector %%8
mi_vector > 7
mi_vector < 7
mi_vector == 7
```

- Esta manera de aplicar una operación es muy eficiente. Comparada con otros procedimientos, requiere de menos tiempo de cómputo, lo cual a veces es considerable, en particular cuando trabajamos con un número grande de datos.

# 4.2. Matrices y arrays

- Las matrices y arrays pueden ser descritas como **vectores multidimensionales**. Al igual que un vector, únicamente pueden contener datos de un sólo tipo, pero además de largo, tienen más dimensiones.
- Como las matrices son usadas de manera regular en matemáticas y estadística, es una estructura de datos de uso común en R y en la que nos enfocaremos en este curso.

### Observación

1. *En un sentido estricto, las matrices son una caso especial de un array, que se distingue por tener específicamente dos dimensiones.*
2. *Los arrays, por su parte, pueden tener un número arbitrario de dimensiones. Pueden ser cubos, hipercubos y otras formas. Su uso no es muy común en R, aunque a veces es deseable contar con objetos n-dimensionales para manipular datos.*

# Matrices y arrays

## Observación

3. *Como los arrays tienen la restricción de que todos sus datos deben ser del mismo tipo, no importando en cuántas dimensiones se encuentren, esto limita sus usos prácticos.*
4. *En general, es preferible usar listas en lugar de arrays, una estructura de datos que además tienen ciertas ventajas*

# 4.2.1. Creación de matrices

- Creamos matrices en R con la función *matrix*(). La función *matrix*() acepta dos argumentos, *nrow* y *ncol*.

```
# Un vector numerico del uno al doce
1:12
# matrix()sin especificar renglones ni columnas
matrix(1:12)
# Tres renglones y cuatro columnas
matrix(1:12, nrow = 3, ncol = 4)
# Cuatro columnas y tres columnas
matrix(1:12, nrow = 4, ncol = 3)
# Dos renglones y seis columnas
matrix(1:12, nrow = 2, ncol = 6)
```

- Los datos que intentemos agrupar en una matriz serán acomodados en orden, de arriba a abajo, y de izquierda a derecha, hasta formar un rectángulo.

# Creación de matrices

- Si el número de elementos es mayor al número de celdas, se acomodarán todos los datos que sean posibles y los demás se omitirán.

```
matrix(1:12, nrow = 3, ncol = 3)
```

- Si, por el contrario, el número de celdas es mayor que el número de elementos, estos se reciclaran.

```
matrix(1:12, nrow = 5, ncol = 4)
```

# Creación de matrices

- Otro procedimiento para crear matrices es la unión vectores con las siguientes funciones:
  - *cbind*() para unir vectores, usando cada uno como una columna.
  - *rbind*() para unir vectores, usando cada uno como un renglón.

```
vec_1 <- 1:4
vec_2 <- 5:8
vec_3 <- 9:12
vec_4 <- 13:16
matriz <- rbind(vec_1,vec_2,vec_3,vec_4)
matriz <- cbind(vec_1,vec_2,vec_3,vec_4)
# hacer lo mismo con
vector_1 <- 1:2
vector_2 <- 1:3
vector_3 <- 1:5
```

# Creación de matrices

- Las matrices pueden contener **NAs**.

```
vector_1 <- c(NA, 1, 2)
vector_2 <- c(3, 4, NA)
matriz <- rbind(vector_1, vector_2)
```

# 4.2.2. Propiedades de las matrices

- No obstante que las matrices y arrays son estructuras que sólo pueden contener un tipo de datos, no son atómicas. Su clase es igual a **matriz (matrix)** o **array** segun corresponda.

```
mi_matriz <- matrix(1:10)
# verificamos usando la funcion
class(mi_matriz)
```

- Obtenemos el número de dimensiones de una matriz o array con la función *dim*().

```
mi_matriz <- matrix(1:12, nrow = 4, ncol = 3)
dim(mi_matriz)
mi_vector <- 1:12
dim(mi_vector)
```

# Propiedades de las matrices

- Las operaciones aritméticas también son vectorizadas al aplicarlas a una matriz. La operación es aplicada a cada uno de los elementos de la matriz.

```
mi_matriz <- matrix(1:9, nrow = 3, ncol = 3)
# Suma
mi_matriz + 1
# Multiplicacion
mi_matriz * 2
# Potenciacion
mi_matriz^3
# ejemplo
vector_1 <- c(NA, 2, 3)
vector_2 <- c(4, 5, NA)
matriz <- rbind(vector_1, vector_2)
matriz / 2
matriz <- matrix(1:6, nrow = 3)
matriz_t <- t(matriz) #transpuesta t()
```

# 4.3. Data frames

- Los data frames son estructuras de datos de dos dimensiones (rectangulares) que pueden contener datos de diferentes tipos, por lo tanto, son heterogéneas. Esta estructura de datos es la más usada para realizar análisis de datos.
- Podemos entender a los data frames como una versión más flexible de una matriz. Mientras que en una matriz todas las celdas deben contener datos del mismo tipo, los renglones de un data frame admiten datos de distintos tipos, pero sus columnas conservan la restricción de contener datos de un sólo tipo.
- En términos generales, los renglones en un data frame representan casos, individuos u observaciones, mientras que las columnas representan atributos, rasgos o variables.

```
> iris
> head(iris)
> mtcars
> head(mtcars)
```

# Data frames

- Para crear un data frame usamos la función *data.frame*(). Esta función nos pedirá un número de vectores igual al número de columnas que deseemos. Todos los vectores que proporcionemos deben tener el mismo largo.
- Esto es muy importante: **Un data frame está compuesto por vectores.**

```
mi_df <- data.frame(
"entero" = 1:4,
"factor" = c("a", "b", "c", "d"),
"numero" = c(1.2, 3.4, 4.5, 5.6),
"cadena" = as.character(c("a", "b", "c", "d"))
)
# Podemos usar dim() en un data frame
dim(mi_df)
# numero de columnas
length(mi_df)
```

# Data frames

```
# nombres de las columnas
names(mi_df)
# La clase de un data frame es data.frame
class(data.frame)
class(mi_df)
```

- Si los vectores que usamos para construir el data frame no son del mismo largo, los datos no se reciclaran. Se nos devolverá un error.

```
data.frame(
"entero" = 1:3,
"factor" = c("a", "b", "c", "d"),
"numero" = c(1.2, 3.4, 4.5, 5.6),
"cadena" = as.character(c("a", "b", "c", "d"))
)
```

# Data frames

- También podemos coercionar esta matriz a un data frame.

```
matriz <- matrix(1:12, ncol = 4)
#Usamos as.data.frame() para coercionar
#una matriz a un data frame.
df <- as.data.frame(matriz)
class(df)
```

# 4.3.1. Propiedades de un data frame

- Al igual que con una matriz, si aplicamos una operación aritmética a un data frame, esta se vectorizará.
- Los resultados que obtendremos dependerán del tipo de datos de cada columna. R nos devolverá todas las advertencias que ocurran como resultado de las operaciones realizadas, por ejemplo, aquellas que hayan requerido una coerción.

```
mi_df <- data.frame(
"entero" = 1:4,
"factor" = c("a", "b", "c", "d"),
"numero" = c(1.2, 3.4, 4.5, 5.6),
"cadena" = as.character(c("a", "b", "c", "d"))
)
mi_df * 2
```

# 4.4. Listas

- Las listas, al igual que los vectores, son estructuras de datos unidimensionales, sólo tienen largo, pero a diferencia de los vectores cada uno de sus elementos puede ser de diferente tipo o incluso de diferente clase, por lo que son estructuras heterogéneas.
- Podemos tener listas que contengan datos atómicos, vectores, matrices, arrays, data frames u otras listas. Esta última característica es la razón por la que una lista puede ser considerada un vector recursivo, pues es un objeto que puede contener objetos de su misma clase.
- Para crear una lista usamos la función *list*(), que nos pedirá los elementos que deseamos incluir en nuestra lista. Para esta estructura, no importan las dimensiones o largo de los elementos que queramos incluir en ella.
- Al igual que con un data frame, tenemos la opción de poner nombre a cada elemento de una lista

# Listas

- Por último, no es posible vectorizar operaciones aritméticas usando una lista, se nos devuelve un error como resultado.

```
mi_vector <- 1:10
mi_matriz <- matrix(1:4, nrow = 2)
mi_df     <- data.frame("num" = 1:3,+
 "let" = c("a", "b", "c"))
mi_lista <- list("un_vector" = mi_vector, +
"una_matriz" = mi_matriz, "un_df" = mi_df)

#Creamos una lista que contiene otras listas
lista_recursiva <- list("lista1" = mi_lista,+
 "lista2" = mi_lista)
```

# 4.4.1. Propiedades de una lista

- Una lista es unidimensional, sólo tiene largo. El largo de una lista es igual al número de elementos que contiene, sin importar de qué tipo o clase sean.

```
length(lista_recursiva)
dim(lista_recursiva)
class(lista_recursiva)
mi_lista / 2
```

# 4.5. Coerción

- Al igual que con los datos, cuando intentamos hacer operaciones con una estructura de datos, R intenta coercionarla al tipo apropiado para poder llevarlas a cabo con éxito.
- También podemos usar alguna de las funciones de la familia *as*() coercionar de un tipo de estructura de datos.

| Función | Coerciona a | Coerciona exitosamente a |
|---|---|---|
| *as.vector*() | Vector | Matrices |
| *as.matrix*() | Matrices | Vectores, Data frames |
| *as.data.frame*() | Data frame | Vectores, Matrices |
| *as.list*() | Lista | Vectores, Matrices, Data frames |

- Como podrás ver, las estructuras de datos más sencillas, (unidimensionales, homogeneas) pueden ser coercionadas a otras más complejas (multidimensionales, heterogeneas), pero la operación inversa casi nunca es posible.

# Coerción

```
#Creamos un vector, una matriz, un data
#frame y una lista.
mi_vector <- c("a", "b", "c")
mi_matriz <- matrix(1:4, nrow = 2)
mi_df <- data.frame("a"=1:2,"b"=c("a", "b"))
mi_lista <- list("a"=mi_vector,"b"=mi_matriz,+
"c"=mi_df)
#Intentemos coercionar a vector
#con as.vector()
as.vector(mi_matriz)
as.vector(mi_df)
as.vector(mi_lista)
```

- La coerción que intentamos sólo tuvo éxito para una matriz. Para data frame y lista, nos devolvió los mismos objetos.

# Coerción

## Observación

*Nota que as.vector() no devolvió un error o una advertencia a pesar de que no tuvo éxito al coercionar, en este caso un data frame o una lista. Esto es importante, pues no puedes confiar que as.vector() tuvo éxito porque corrió sin errores, es necesaria una verificación adicional.*

```
as.matrix(mi_vector)
as.matrix(mi_df)
as.matrix(mi_lista)
as.data.frame(mi_vector)
as.data.frame(mi_matriz)
as.data.frame(mi_lista)
as.list(mi_vector)
as.list(mi_matriz)
as.list(mi_df)
```

# Coerción

- Conocer cómo ocurre la coerción de estructuras de datos te ayudará a entender mejor algunos resultados devueltos por funciones de R, además de que te facilitará la manipulación y procesamiento de datos.

# 5. Subconjuntos

- En R, podemos obtener subconjuntos de nuestras estructuras de datos. Es decir, podemos extraer partes de una estructura de datos (nuestro conjunto).
- Hacemos esto para seleccionar datos que tienen características específicas, por ejemplo, todos los valores mayores a cierto número o aquellos que coinciden exactamente con un valor de nuestro interés.
- Para realizar esta operación haremos uso de índices, operadores lógicos y álgebra Booleana. Algunos procedimientos para obtener subconjuntos pueden usarse con cualquier estructura de datos, mientras que otras sólo funcionan con algunas de ellas.

# 5.1. Índices

- Usar índices para obtener subconjuntos es el procedimiento más universal en R, pues funciona para todas las estructuras de datos.
- Un índice en R representa una posición. Cuando usamos índices le pedimos a R que extraiga de una estructura los datos que se encuentran en una o varias posiciones específicas dentro de ella.
- A diferencia de la mayoría de los lenguajes de programación, los índices en R empiezan en 1, no en 0. El índice del primer elemento de una estructura de datos siempre es 1, el segundo 2, y así sucesivamente.

# Índices

- Un aspecto muy importante de este procedimiento es que, para data frames y listas, **cuando extraemos un subconjunto de un objeto usando corchetes, obtenemos como resultado un objeto de la misma clase que el objeto original**. Si extraemos un subconjunto de un data frame, obtenemos un vector; y si extraemos de una lista, obtenemos una lista.
- El uso de índices tiene además otras características particulares para las distintas estructuras de datos

# 5.1.1. Vectores

- Empecemos creando un vector que contiene los nombres de distintos niveles educativos.

```
nivel<-c("Preescolar","Primaria","Secundaria",+
 "Educaci\'on Media Superior",+
 "Educaci\'on Superior")
 nivel
 length(nivel)
```

- ¿Cómo obtendríamos el tercer elemento de este vector usando índices? ¿O del primer al cuarto elemento? ¿O el segundo y quinto elemento?

```
nivel[3]
nivel[1:4]
nivel[2, 5]
nivel[c(2, 5)]
```

# 5.1.2. Data frames

```
mi_df <- data.frame("nombre" = c("Armando", "Elsa",
"Ignacio", "Olga"),"edad" = c(20, 24, 22, 30),
"sexo" = c("H", "M", "M", "H"),
"grupo" = c(0, 1, 1, 0))
dim(mi_df)
mi_df[1]
mi_df[c(1, 3)]
mi_df[3,]
mi_df[ ,1]
mi_df[3, 3]
mi_df[2:3, 3]
mi_df[4, 3:4]
mi_df[1:2, c(2, 4)]
sub_df <- mi_df[1:2, c(2, 4)]
class(sub_df)
mi_df[7]
mi_df[7, ]
```

# 5.1.3. Matrices

- El procedimiento anterior para data frames funciona de la misma manera para las matrices, con una excepción. **Si usamos como índice un sólo número, entonces obtendremos el valor que se encuentre en esa posición, contando celdas de arriba a abajo y de izquierda a derecha.**

```
mi_matriz <- matrix(1:8, nrow = 4)
mi_matriz[5]
mi_matriz[8]
# Tercer renglon
mi_matriz[3, ]
# Segunda columna
mi_matriz[ ,2]
# Tercer renglon y segunda columna
mi_matriz[3, 2]
```

# 5.1.4. Arrays

- Para objetos de tres o más dimensiones se siguen las mismas reglas que con las matrices, aunque ya no es tan fácil hablar de renglones y columnas.

```
#Creamos un array de cuatro dimensiones.
mi_array<-array(data=1:16,dim=c(2, 2, 2, 2))
mi_array
# Comprobamos el número de dimensiones
#de nuestro objeto
dim(mi_array)
#Intentemos extraer varios subconjuntos
mi_array[1, , , ]
mi_array[1, 2, , ]
mi_array[1, 2, 1, ]
mi_array[1, 2, 1, 2]
```

# 5.2. Nombres

# 5.3. Subconjuntos por índice y nombre

# 5.4. El signo de dolar $ y los corchetes dobles [[]]

# 5.5. Condicionales

# 6. Funciones

# 6.1. ¿Por qué necesitamos crear nuestrar propias funciones?

# 6.2. Funciones definidas por el usuario

# 6.3. Nuestra primera función

# 6.4. Definiendo la función *crear_histograma*()

# 7. Estructuras de control

# 7.1. if, else

# 7.2. for

# 7.3. while

# 7.4. break y next

# 7.5. repeat

# 8. La familia apply

# 9. Importar y exportar datos

# 10. Gráficas