



Capacitación en R y herramientas de productividad

Proyecto Estratégico Servicios Compartidos para la Producción Estadística

Introducción y herramientas de exploración de datos

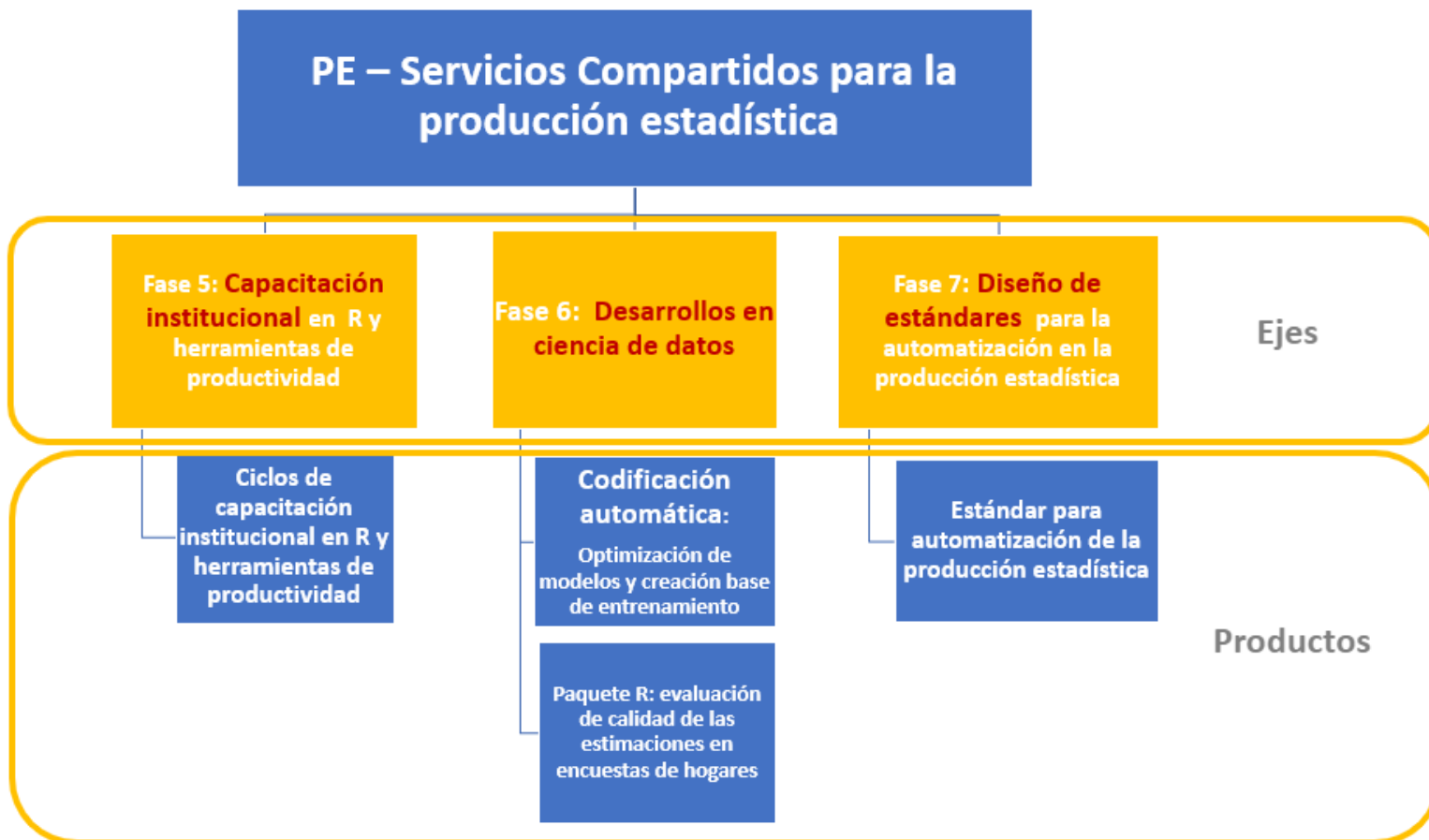
Agosto 2020

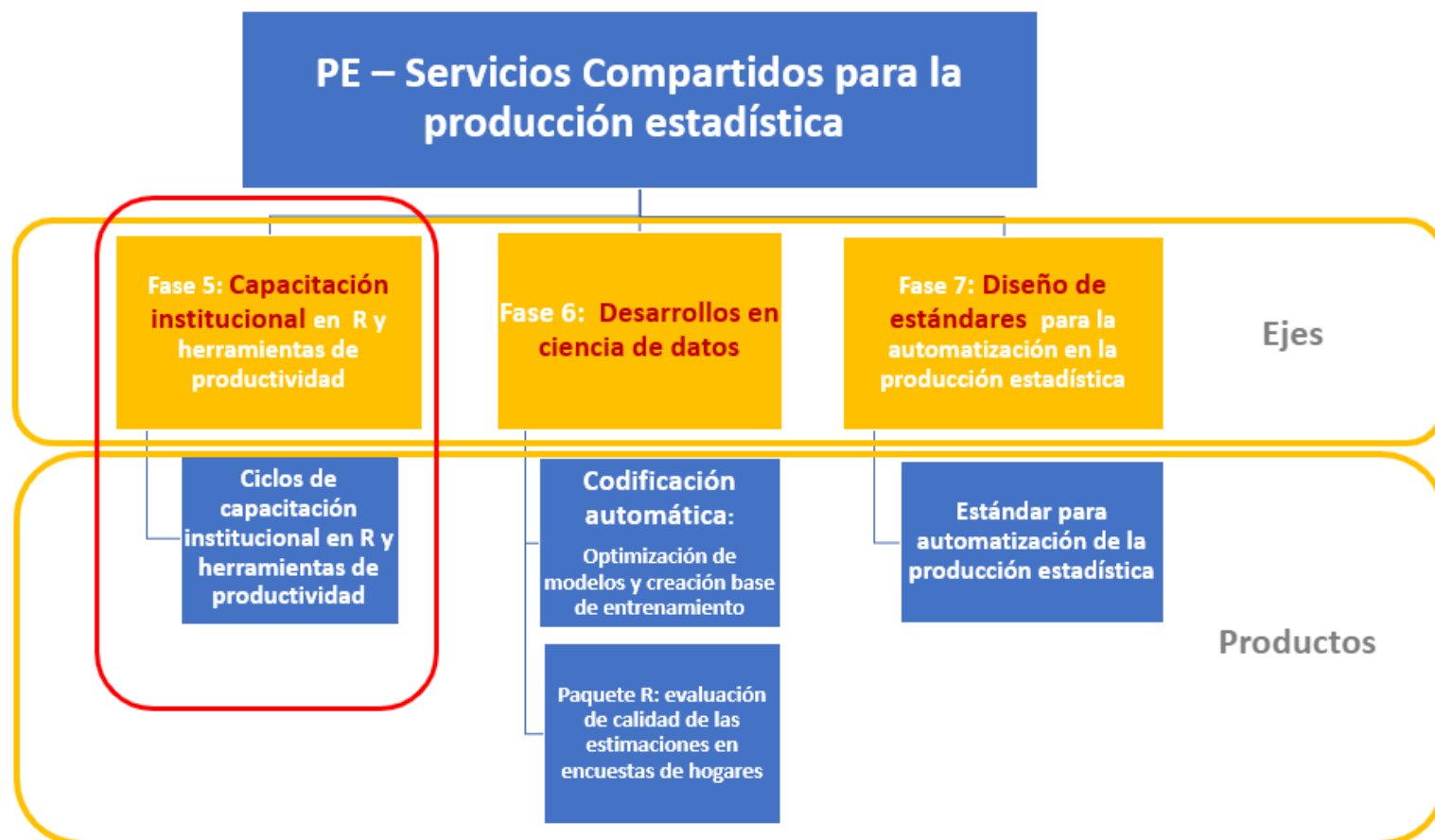
Esta actividad de capacitación se enmarca en el **PE SSCC**.

El PE SSCC es uno de los 4 proyectos estratégicos actualmente en funcionamiento en el INE (2018-2022).

Este proyecto busca:

*"Proveer a la institución de estándares y desarrollos que permitan **automatizar**, **estandarizar**, **ordenar** e **innovar** en la producción estadística, permitiendo reducir **tiempos** y **costos** del procesamiento y análisis de los diferentes productos del INE y minimizando la probabilidad de **errores** en la publicación de resultados".*





Al finalizar la capacitación se espera que los participantes:

- **Estén familiarizados con el lenguaje de programación R** y cuenten con las habilidades necesarias para **profundizar aspectos de su interés** en este lenguaje.
- Sean capaces de **explorar** y **transformar** objetos en R.
- Manejen herramientas de **visualización** en R.
- Aprendan **buenas prácticas de escritura de código** en R.
- Conozcan la **herramienta git** para el control de versiones en el desarrollo de proyectos de datos.
- Conozcan **recomendaciones** que les permitan hacer **reproducibles** y **trazables** sus rutinas en R.
- Conozcan herramientas para la **automatización de reportes** y de **tareas de procesamiento y análisis de datos**.

Organización de la capacitación

- La capacitación se compone de **10 sesiones**.
- Las sesiones se realizarán **semanalmente**, los días **miércoles entre 10:00 y 13:00** (a excepción de esta).
- Haremos una pausa de 10 minutos durante la sesión, en un momento a convenir.
- Si van surgiendo dudas, pueden consultar durante la clase.
- Compartiremos la presentación antes del inicio de la sesión. Si existen correcciones detectadas, actualizaremos el repositorio durante la semana posterior.
- Al finalizar cada sesión (salvo la primera) se les presentará un desafío para resolver durante la semana 🏠.

Parte I: Herramientas básicas

Sesión 1: Introducción y herramientas de exploración de datos

- ¿Qué es R y por qué usarlo?
- La interfaz de R Studio
- Tipos de datos y operaciones básicas
- Exploración y manipulación básica de un data frame

Sesión 2: Procesamiento de bases de datos (1)

- Cargar datos desde distintos formatos a R
- Encadenamiento de instrucciones: manejo de pipes (`%>%`)
- Transformación de datos (creación y edición de variables)
- Agregación de unidades con `group_by`

Sesión 3: Procesamiento de bases de datos (2)

- Unión de bases de datos (`joins`, `binds`)
- Manejo de variables de tiempo (fechas)
- Manejo de variables factor

Sesión 4: Visualización de datos usando el paquete ggplot2

- Visualizar para comunicar efectivamente
- La gramática de los gráficos: visualizar datos con ggplot2

Sesión 5: Funciones y estilo de código

- ¿Qué es una función?
- ¿Por qué son importantes las funciones?
- ¿Cómo crear funciones en R?
- Creando mi primera función para mi trabajo diario
- Buenas prácticas para la escritura de código

Parte II: Aplicaciones para el negocio

Sesión 6: Validación de datos

- Introducción al paquete validate
- Creación de reglas de validación
- Reporte de validación (usando funciones de validate)

Sesión 7: Más sobre edición de datos

- Editando variables con dplyr
 - Case_when
 - Varias columnas a la vez (Mutate_*)
- Edición de variables de texto
 - Grep, grepl y gsub
 - Introducción a las expresiones regulares

Sesión 8: Trabajando con varias tablas simultáneamente

- Ciclo for
- Purrr (map, map_df)
- Leer un conjunto de archivos desde una carpeta (usando funcionales)
- Editar un conjunto de tablas de manera eficiente
- Generar tabulados para varias tablas
- Generar reporte gráfico para varias tablas

Sesión 9: Data.table para el manejo de grandes volúmenes de datos

- Es hora de abandonar dplyr (dplyr vs data.table)
- Creación de un objeto data.table
- Sintaxis básica de data.table
- Consultas a la base de datos
- Transformación de datos (edición y creación de variables)

Sesión 10: En camino a la reproducibilidad

- ¿Por qué es importante el trabajo reproducible?
- Introducción a Rmarkdown
 - Instalación
 - ¿Qué es RMarkdown?
 - Creando mi primer reporte con RMarkdown
- Introducción a git y github
 - ¿Qué es el control de versiones?
 - ¿Qué es git?
 - ¿Qué es github?
 - Combinando R con git

Antes de comenzar
¿Qué saben de R?



¿Qué es R y por qué usarlo?

R es un lenguaje de programación creado por dos profesores de estadística de la Universidad de Auckland en 1993 (**Robert Gentleman y Ross Ihaka**).

R es gratis, su código es abierto y **se encuentra disponible en la mayoría de las plataformas** (Windows, Mac OS, Linux).


¡Es muy versátil! Hoy es utilizado para abordar problemas de **distintas disciplinas**, sobre todo de estadística.

Posiblemente las **técnicas o metodologías más recientes tengan alguna aplicación en R**. Los investigadores muchas veces publican un paquete del programa en compañía de sus artículos.

Su lenguaje esta hecho para el análisis de datos. Esto incluye características como valores perdidos (*missing values*), manipulación de *data frames* y selección de casos.

¿Qué es R y por qué usarlo?

¡Su comunidad de usuarios es muy activa!

Se puede contactar con una comunidad local y global de usuarios a través de **grupos de usuarios**, twitter . Además se puede obtener ayuda de expertos utilizando [stackoverflow](#).

En Chile la comunidad de R se ha organizado en torno al [Grupo de Usuarios de R en Chile](#) y R-Ladies con capítulos en [Santiago](#), [Valparaíso](#), [Concepción](#) y [Talca](#).

Anualmente se realizan conferencias de a nivel mundial [UseR!](#) y a nivel latinoamericano [Latin-R](#).

Cuenta con **poderosas herramientas para comunicar resultados**. Varios paquetes en **R** facilitan la creación de reportes en pdf o en html, así como también para la creación de web interactivas.

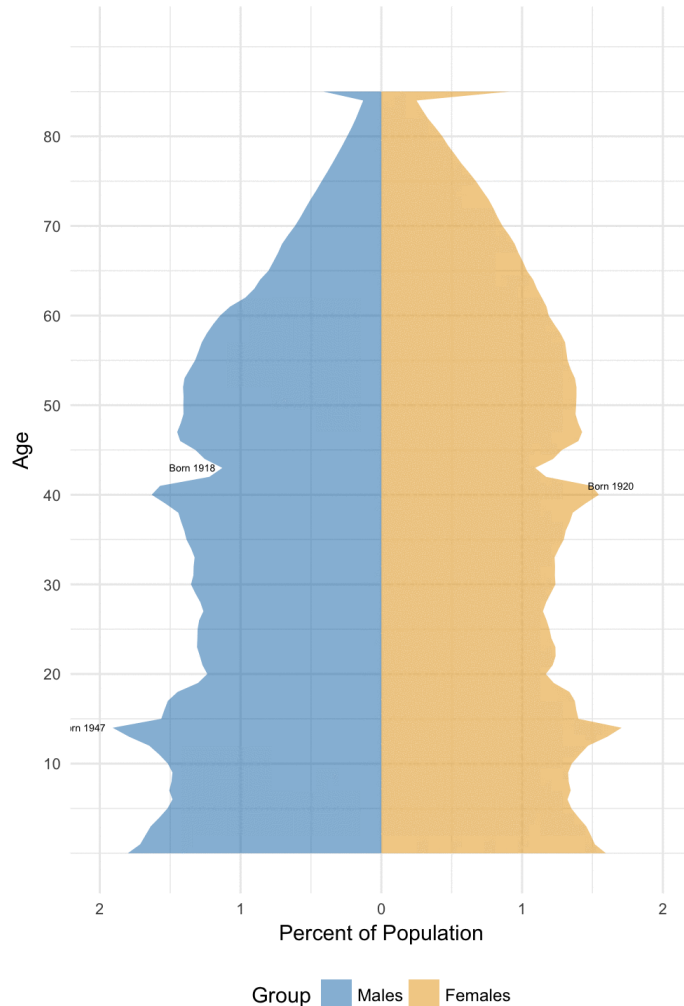
Cuenta con un **entorno de desarrollo interactivo** diseñado especialmente para el análisis de datos y la programación estadística (**RStudio**).

Varias de estas razones se encuentran en: [Wickham, 2014](#)

¿Qué cosas podemos hacer en R?

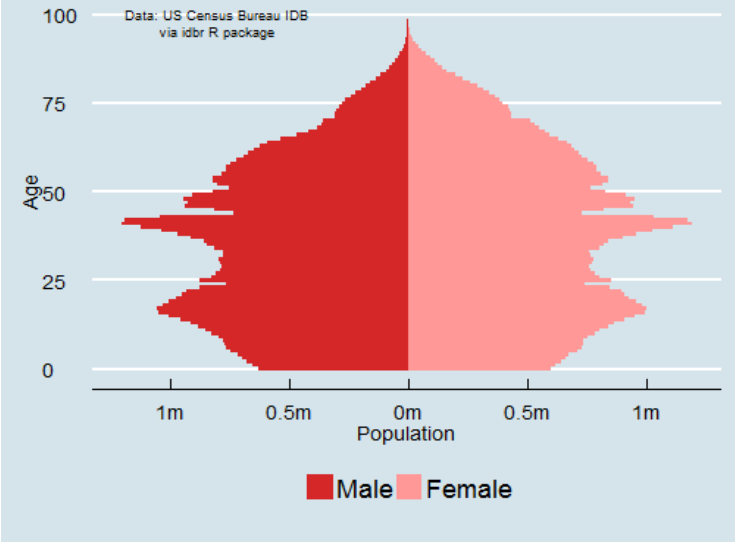
Age Distribution of the Population of England and Wales: 1961

Age is top-coded at 85 before 1971 and 90 thereafter.



Kieran Healy / kieranhealy.org / Data: UK ONS.

Population structure of Japan, 1990



¿Qué cosas podemos hacer en R?

Boletín índice de Precios al Productor (IPP):

¿Qué cosas podemos hacer en R?

Boletines en Direcciones Regionales:

Boletín EMAT de la Dirección Regional de Ñuble

Documentos metodológicos EPF:

- Cálculo de medidas de precisión para medianas de ingreso y gasto de la VIII EPF
- Reclasificación de la no respuesta: distinción entre la no respuesta al ítem y la no respuesta a l unidad
- Métodos de Imputación VIII EPF: Gastos diarios e ingresos de la actividad laboral y jubilaciones

**Esta presentación está desarrollada
completamente en R** 🤖 🤖

Ahora abramos RStudio



Interactuando con R

La interfaz de RStudio está distribuida de la siguiente manera

The screenshot shows the RStudio interface with the following components highlighted:

- 1 La Consola:** The bottom-left pane showing the R console output, including the R version (4.0.2) and the R Foundation copyright notice.
- 2 El Editor:** The top-left pane showing the R script editor with a file named 'Untitled1'.
- 3 El entorno de variables:** The top-right pane showing the Environment tab, which is currently empty.
- 4 Las utilidades:** The bottom-right pane showing the Packages tab, which lists installed and available packages in the User Library.

Name	Description	Version
AsioHeaders	'Asio' C++ Header Files	1.16.1-1
askpass	Safe Password Entry for R, Git, and SSH	1.1
assertthat	Easy Pre and Post Assertions	0.2.1
backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.7
base64enc	Tools for base64 encoding	0.1-3
bit	R Base C++ Header Files	72.0-3
blob	A Simple S3 Class for Representing Vectors of Binary Data ('BLOBS')	1.2.1
bookdown	Authoring Books and Technical Documents with R Markdown	0.20
brew	Templating Framework for Report Generation	1.0-6
broom	Convert Statistical Objects into Tidy Tibbles	0.7.0
callr	Call R from R	3.4.3

Existen dos formas principales de interactuar con **R**:

- A través de la consola.
- Usando archivos *script* (texto plano que contiene nuestro código).

Consola

En **RStudio** corresponde al panel ubicado abajo a la izquierda.

El símbolo **>** significa que **R** está listo para que le entreguemos una instrucción a ejecutar.

En esta ventana también aparecen los **resultados** de las instrucciones.

Los comandos que se escriban aquí **no serán grabados** para un nuevo inicio de sesión.

El código escrito en consola se ejecuta con la tecla **Enter**.

Sin embargo, una práctica habitual y recomendada es guardar rutinas completas en archivos *script*.

Script

Es común que una tarea la queramos **repetir** más de una vez.

Esta tarea muchas veces involucra un conjunto de tareas, por lo que es **poco eficiente utilizar la consola** para realizarlas.

Resulta conveniente guardar múltiples instrucciones en un archivo de texto (*script*) que luego nos permita **volver a ejecutar** estas tareas.

Además resulta mucho **más fácil compartir** nuestro trabajo de esta manera.

Para ejecutar el código en *script* puedes utilizar el *shortcut* **Ctrl + Enter**.

👁 Para comentar en el *script* se utiliza el signo gato (**#**)

El signo **#** también sirve para **silenciar** código sin tener que borrarlo.

👁👁 Con el *shortcut* **Ctrl + Shift + c** puedes **silenciar** varias líneas a la vez.

Primeras operaciones en R

R puede ser usado como una calculadora.

Pruébenlo ustedes mismos en su consola o script.

```
4 + 5
```

```
## [1] 9
```

```
6 * 8
```

```
## [1] 48
```

```
(4 + 5) + (6 * 8)
```

```
## [1] 57
```

Se pueden hacer cosas mucho más interesantes. Para eso es útil **asignar valores a objetos**

Para esto se utiliza el operador "<-" (shortcut "**Alt + -**")

```
x <- 5
```

Primeras operaciones en R

Entonces, podríamos hacer cosas como esta.

```
a <- 20  
b <- 5  
  
a + b
```

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

El signo "^" representa al exponencial. También se puede usar "**"

```
a + b ^ 2
```

```
## [1] 45
```

Entonces, podemos hacer operaciones sobre los valores almacenados, a través de estos objetos abstractos (a y b).

Mini- ejercicio 1

- 1- ¿Qué sucede si cambiamos el valor de "a" por 45 y luego realizamos la última operación?
- 2- ¿Es posible asignar el resultado a una nueva variable "c"? **¿Cómo lo harían?**

La mayor parte del tiempo, en cualquier *software*, usamos **funciones** integradas.

Pueden ser entendidas como *scripts* que funcionan "tras bambalinas": **un conjunto de instrucciones que permiten automatizar rutinas largas y/o complicadas.**

O también operaciones sencillas. Por ejemplo:

```
3 + 3
```

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

Es lo mismo que utilizar la función `sum()`

```
sum(3,3)
```

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

Pero `sum()` es una función muy versátil y más potente.

```
x <- c(1,3,5,6,8,9,18,20)  
sum(x)
```

```
## [1] 70
```

R contiene variadas funciones matemáticas

```
sqrt(4) # raíz cuadrada
```

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

```
round(9.556789) # aproximar
```

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

```
floor(9.556789) # truncar
```

```
## [1] 9
```

Las funciones tienen argumentos, y podemos consultarlos con la función `args()`

```
args(round)
```

```
## function (x, digits = 0)  
## NULL
```

El argumento "digits" me permite indicarle a `round()` cuántos decimales quiero.
Pero si no le digo asume que es cero.

```
round(9.556789, digits = 2)
```

```
## [1] 9.56
```

En este caso también puedo indicarlo de una manera más sencilla

```
round(9.556789, 2)
```

```
## [1] 9.56
```

Como no hay ambigüedad, R lo entiende.

Para obtener ayuda sobre una función lo hacemos de la siguiente forma:

```
?round  
# o  
help(round) # help también es una función
```

Gran parte de las funciones que utilizamos en R vienen contenidas en **paquetes** (*packages*).

Las funciones que acabamos de conocer pertenecen al paquete *base*, que viene con el *software* R.

Sin embargo, **existen más de 10.000 paquetes en R.**

La mayoría creados por sus usuarios 🧑🧑

Más adelante descargaremos y utilizaremos algunas librerías de R.

Conoceremos algunos de los tipos de datos más utilizados.

Vectores

Es el objeto más básico en R.

Un vector es una forma de **almacenar datos** que permite contener **una serie de valores del mismo tipo**.

Veamos algunos ejemplos.

```
nombres <- c("jacqueline", "miguel", "lorena", "luis")  
a <- c(1, 5, 6, 9:12)  
b <- c(1, 2, 3, "gato")
```

La función "c" (combine) permite unir valores u objetos.

- ¿cuántos elementos contiene el vector "a"?
- ¿De qué tipo son los vectores creados?

(con la función `length()` pueden contar los elementos de un vector)

Tipos de datos en R

Existen 5 tipos de vectores en R:

```
character <- c("gato", "perro")
numeric <- c(8, 15.9) # reales o decimales
integer <- c(2L, 4L) # L indica que son enteros
logical <- c(TRUE, FALSE, TRUE)
complex <- 3 + 4i # complejos
```

Podemos consultar cuál es el tipo de vector con `class()` o `typeof()`

```
class(a)
```

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

```
typeof(b)
```

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

Que los vectores sea atómicos  significa que solo pueden contener un tipo de datos.

Creando vectores

Vimos que es posible crear vectores con la función "c".

```
x <- c(1,2,3,4,5)
```

Una secuencia sencilla también podemos crearla de la siguiente manera.

```
y <- 1:20  
y
```

```
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Creando vectores

Sin embargo, existen funciones más poderosas para crear vectores.

Una de ellas es `seq()` (*sequence generation*).

Recibe 4 argumentos: *from*, *to*, *length* y *by*.

¿Cómo crearían un vector que vaya del 1 al 5 y que avance en intervalos de 0.5?

```
a <- seq(from = ¿?, to = ¿?, by = ¿?)
```

¿Cómo crearían un vector que vaya del 1 al 10 y contenga 100 valores?

```
b <- seq(from = ¿?, to = ¿?, length = ¿?)
```

Sin especificarlo, **R** genera intervalos de igual tamaño.

Operaciones matemáticas

R ofrece los siguientes operadores matemáticos

```
x + y    # suma
x - y    # resta
x * y    # multiplicación
x / y    # división
x ^ y    # exponenciación
x %% y   # modulo división (resto) 10 %% 3 = 1
x %/% y  # división por enteros: 10 %/% 3 = 3
```

Generalmente utilizamos los operadores básicos (+, -, *, /).

Sin embargo, el operador **módulo (%%)** es muy útil para programar.

Indexación

En general, cuando trabajamos con datos ordenados **nos interesa seleccionar uno o algunos elementos en particular.**

```
bandas_cl <- c("Los prisioneros", "Los Bunkers",  
              "Los Ángeles Negros")  
bandas_arg <- c("Pescado rabioso", "Spinetta Jade",  
               "Virus", "Soda Stereo")  
bandas <- c(bandas_cl, bandas_arg) # acá uno los dos vectores  
bandas
```

```
## [1] "Los prisioneros"      "Los Bunkers"          "Los Ángeles Negros"  
## [4] "Pescado rabioso"     "Spinetta Jade"        "Virus"  
## [7] "Soda Stereo"
```

Al imprimir el vector `bandas` vemos que aparece un número entre `[]`

Indexación

Este número indica un **índice o posición** de los elementos al interior de un vector.

```
bandas[5] # nos entrega el 5to elemento
```

```
## [1] "Spinetta Jade"
```

La **indexación es la selección de subconjuntos de datos de un vector**. `[]` es un operador de selección.

Tenemos **diferentes alternativas para indicar posición**. Veamos ejemplos:

```
# primero creamos vectores
numeros <- 20:40
nombres <- c("jacqueline", "miguel", "lorena", "luis")
```

- Con un operador lógico

```
numeros[numeros>30]
```

```
## [1] 31 32 33 34 35 36 37 38 39 40
```

Indexación

- Con un vector numérico

```
numeros[8]
```

```
## [1] 27
```

```
nombres[c(1,2)]
```

```
## [1] "jacqueline" "miguel"
```

- Con selección negativa (ej. vector[-1])

```
nombres[-2]
```

```
## [1] "jacqueline" "lorena"      "luis"
```

```
nombres[c(-3,-4)]
```

```
## [1] "jacqueline" "miguel"
```

Mini-ejercicio 2

1. Crea un vector numérico del 1 al 30 y asígnalo a un objeto (ponle el nombre que quieras).
2. Selecciona el valor de la quinta posición.
3. Selecciona los valores mayores a 13.
4. Crea un segundo objeto a partir del primero que no contenga el último valor (30).
5. Crea un tercer vector que vaya del 0 al 30, pero **que contenga solo los números pares**.

Recuerda que para crear objetos o asignar valores debes usar <-

Operadores lógicos

Los **operadores lógicos** son **muy** importantes para la programación.

R cuenta con operadores de comparación binaria.

```
x < y      # menor que
x > y      # mayor que
x <= y     # menor o igual que
x >= y     # mayor o igual que
x == y     # igual a
x != y     # distinto a
```

👁👁: Nota que **==** permite comparar si dos valores son iguales. Ten cuidado de **NO** usar **=** que es interpretado como un operador de asignación (es como usar **<-**).

Operadores lógicos

Algunos ejemplos con números:

```
x <- c(1,2,5)
y <- c(4,4,3)
x == y
#> [1] FALSE FALSE FALSE
x != y
#> [1] TRUE TRUE TRUE
x < y
#> [1] TRUE TRUE FALSE
```

Ahora con caracteres:

```
a <- c("izzy", "jazz", "tyler")
b <- c("devon", "vanessa", "hilary")
a < b # orden alfabético
#> [1] FALSE TRUE FALSE
```

Comparaciones en **R** tienen como resultado valores lógicos: **TRUE**, **FALSE** o **NA** (*not available*).

Tal como en los operadores aritméticos, las comparaciones lógicas con un vector son aplicadas **elemento a elemento** y su resultado es un vector de valores de verdad (vector lógico).

Los operadores lógicos son muy usados para **extraer** información que **cumple con ciertas condiciones** de determinado conjunto de datos.

Otros operadores muy importantes son "**|**" (o) y "**&**" (y).

Revisaremos estos operadores más adelante.

Operadores lógicos

También podemos modificar elementos específicos de un vector.

```
# primero creamos un vector  
x <- 1:10
```

Podemos hacerlo indicando una posición

```
x[5] <- 0  
x
```

```
## [1] 1 2 3 4 0 6 7 8 9 10
```

O también podemos seleccionar los elementos a modificar con **operadores lógicos**.

¿Qué creen que hace esta sentencia?

```
x <- 1:10  
x[x > 5] <- x[x > 5] + 5
```

Otros tipos de datos en R

Existen estructuras de datos más complejas que los vectores.

data frames

Es el formato más común al que nos enfrentamos diariamente en hojas de cálculo, programas como Excel, SPSS, Stata, etc.

Podemos entender un data frame como un conjunto de vectores que tienen la misma longitud y que conforman una tabla (i.e. toman forma rectangular).

```
curso <- data.frame(nombre = c("Klaus", "Juan", "Ignacio"),
                     notas = c(7, 7, 1))

curso
```

```
##      nombre notas
## 1    Klaus      7
## 2     Juan      7
## 3  Ignacio      1
```

La mayor parte de este curso se enfocará en cómo trabajar con este tipo de objetos en R.

Matrices

Pueden ser entendidos como una extensión de los vectores, pero con dimensión de filas y columnas.

No pueden convivir tipos de datos diferentes dentro de una matriz.

```
curso <- matrix(c("Klaus", "Juan", "Ignacio", 7, 7, 1),
               nrow = 3,
               ncol = 2)
colnames(curso) <- c("nombre", "nota")

curso
```

```
##      nombre  nota
## [1,] "Klaus"   "7"
## [2,] "Juan"    "7"
## [3,] "Ignacio" "1"
```

Si se fijan, los números fueron forzados a ser *character* ("1").

Listas

Una lista es una estructura que **puede contener diferentes tipos de datos**, incluso listas.

A veces se les llama vectores genéricos, dado que soportan cualquier tipo de formato.

```
a <- c(1,2,3,4,5)
b <- c("rojo", "verde")
c <- data.frame(nom = "Klaus", "Juan",
                nota = 7, 7)

lista1 <- list(a,b,c)
```

Esta estructura es más compleja de entender, pero en la parte II de este ciclo de capacitaciones verán que son muy útiles.



Factores

Son usados para representar **variables categóricas**, ordinales o no.

Parecen vectores de caracteres y a veces se comportan de esa manera (porque **permiten guardar etiquetas**), pero realmente son almacenados como *integers* (enteros, **que son más livianos para almacenar**).

```
ocupacion <- factor(x = c(1,2,3,1,2,1,3,4,1,5),  
  labels = c("profesor", "musico", "doctora",  
             "taxista", "pescador"))  
ocupacion
```

```
## [1] profesor musico  doctora  profesor musico  profesor doctora  taxista  
## [9] profesor pescador  
## Levels: profesor musico doctora taxista pescador
```

En las sesiones posteriores veremos que son muy importantes, sobre todo cuando queremos **visualizar** nuestros datos  .

Arrays

Finalmente, este tipo de datos es similar a las matrices, sin embargo, pueden tener más de dos dimensiones.

De hecho, *un array de dos dimensiones es lo mismo que una matriz.*

No las usaremos para nada en el curso (pero pueden consultar más sobre ellas con `help(array)` ☹️)

Recapitulando

Hasta ahora hemos visto:

- Qué es R y por qué usarlo
- Cosas que podemos hacer con R
- Interfaz de RStudio
- Primeras operaciones con R (funciones básicas, asignaciones, ...)
- Tipos de datos en R (vectores, matrices, *data frames*, ...) y...
- Operadores lógicos (>, <, ==, ...)

Volviendo a los *data frames*

Como mencionamos anteriormente, los *data frames* **son la estructura de datos a que nos enfrentamos más a menudo.**

Para explorar y manipular *data frames* utilizaremos el paquete `guaguas` de **Riva Quiroga (chilena).**

README.md

`guaguas`

CRAN 0.1.0 build passing

Datos de nombres de guaguas (bebés) registrados en Chile entre 1920 y 2019, según el Servicio de Registro Civil e Identificación. Incluye todos los nombres con al menos 15 ocurrencias. Este *dataset* permite explorar tendencias en los nombres registrados durante el último siglo y puede utilizarse como fuente de datos de práctica para enseñar/aprender R.



Instalamos el paquete (**nuestro primer paquete** 🤖)

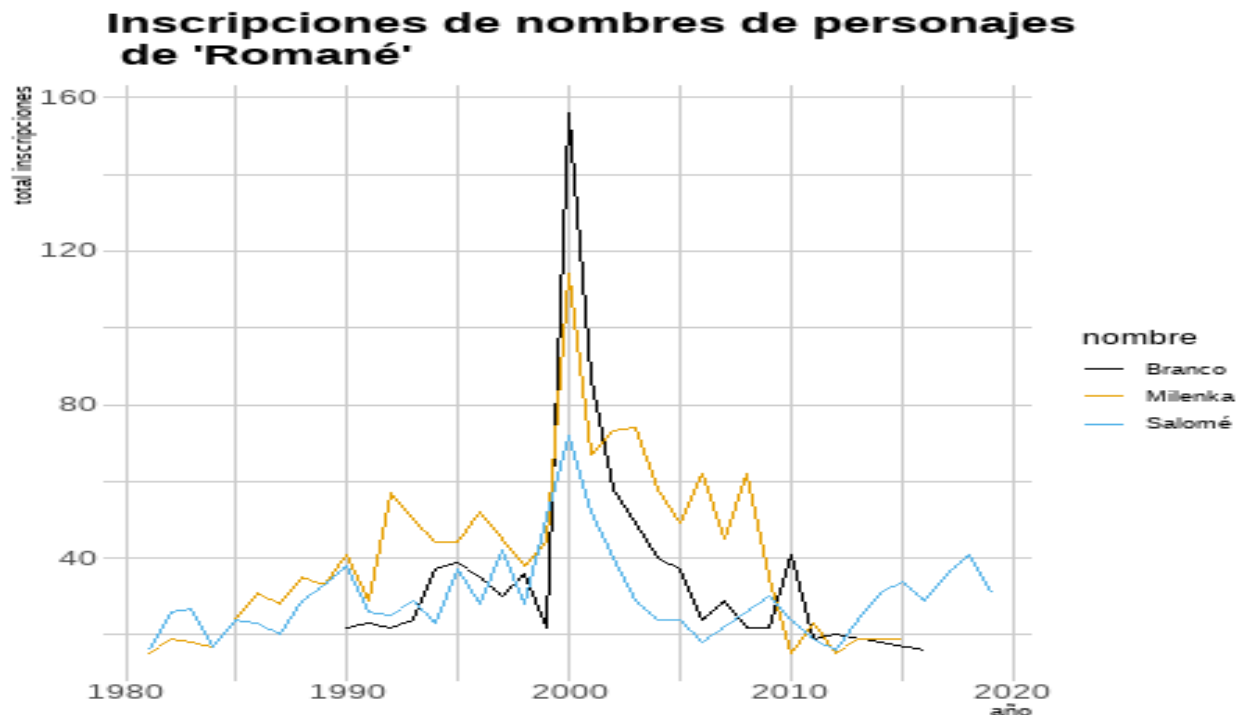
```
install.packages("guaguas")
```


Volviendo a los *data frames*

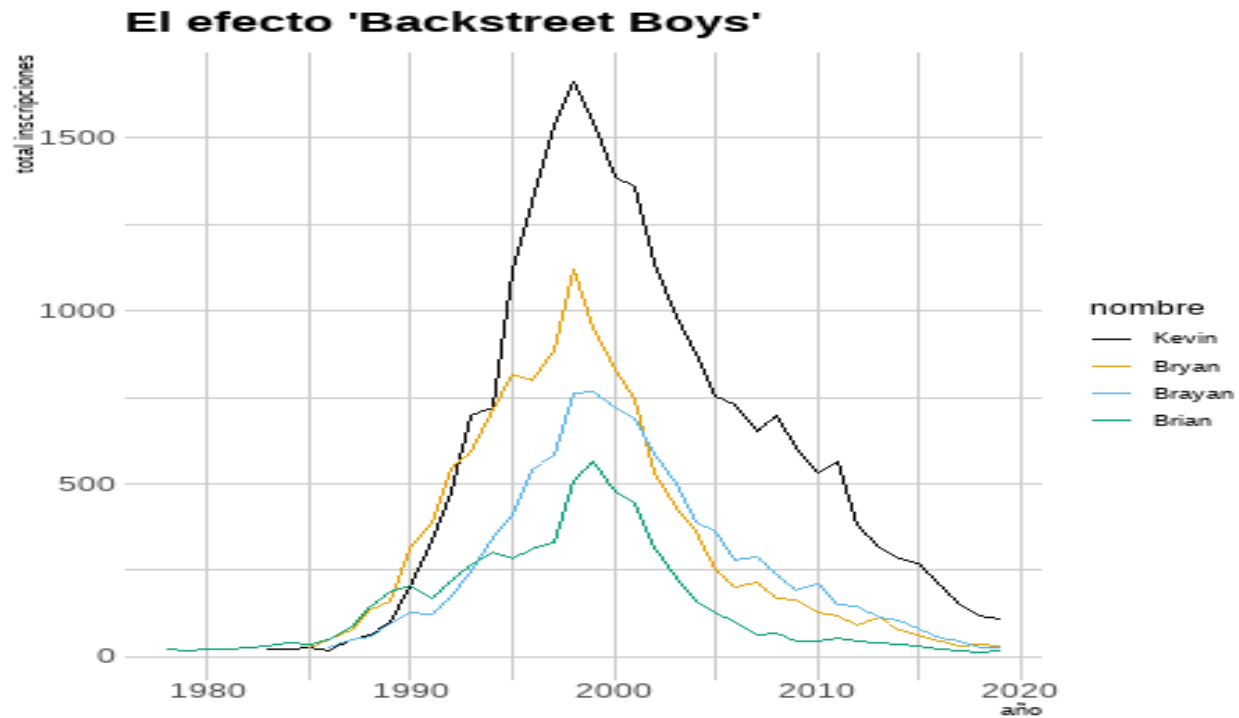
Y luego lo cargamos en nuestra sesión (no basta con instalarlo para usarlo).

```
library(guaguas)
```

Es una base de **alto interés sociológico**. 😊



Volviendo a los *data frames*



Exploración *data frames*

(fuente de paquete y gráficos: [GitHub de Riva Quiroga](#))

R cuenta con algunas funciones *base* para la **exploración**.

Primero carguemos la base en nuestro **ambiente de trabajo**.

```
data <- guaguas
```

Dado que la base viene con el paquete no es necesario cargarla, pero es mejor acostumbrarse de esta manera.

¿Qué información nos aportan estas funciones sobre nuestra base?

```
class(data)
is.data.frame(data)
dim(data)
str(data)
head(data)
names(data)
```

Exploración *data frames*

También usando lenguaje *base* podemos pedir tablas de resumen.

Para eso debemos utilizar la función `table()` y el operador `"$"`, que vincula a una variable con un objeto.

```
table(data$anio)
table(data$nombre)
```

También podemos pedir la proporción, aplicando aplicando `prop.table()` sobre `table()`.

```
prop.table(table(data$sexo))
```

```
##
##           F           M
## 0.5873115 0.4126885
```

Seleccionar filas y columnas

Es similar a la selección de elementos en un vector.

Seleccionar columnas

Podemos seleccionar columnas por su posición.

```
data_2 <- data[,1]  
dim(data_2)
```

```
## [1] 84565      1
```

```
data_3 <- data[,c(2:5)]  
dim(data_3)
```

```
## [1] 84565      4
```

También podemos seleccionar por nombre de columna.

```
data_4 <- data[,c("anio", "nombre", "n")]  
names(data_4)
```

```
## [1] "anio"  "nombre" "n"
```

Seleccionar filas

Se pueden seleccionar filas de distintas maneras.

```
data_5 <- data[data$anio==2019,]  
table(data_5$anio)
```

```
##  
## 2019  
## 948
```

```
data_6 <- data[data$anio > 2010,]  
table(data_6$anio)
```

```
##  
## 2011 2012 2013 2014 2015 2016 2017 2018 2019  
## 998 982 981 983 985 955 924 955 948
```

Por supuesto, también se pueden seleccionar filas y columnas a la vez 🤖.

¿Cómo lo harían? Hagamos algunos ejercicios para finalizar.

Ordenar vectores y *data frames*

Antes vamos a instalar una librería más.

Esta librería la vamos a utilizar **muchísimo** en las próximas sesiones.

```
install.packages("tidyverse")
```

Cargamos la librería.

```
library(tidyverse)
```

En esta sesión vamos a conocer una función llamada `arrange()` que está contenida en `tidyverse`.

`arrange()` cumple la función de **ordenar** vectores o *data frames* **en función de una o más variables**.

Ordenar vectores y *data frames*

Funciona así:

Así podemos ordenar el *data frame* por **anio**.

```
arrange(data, anio)
```

Por defecto lo hace de forma **ascendente**.

Le podemos pedir que lo haga en forma **descendente**.

Además **puede ordenarse por más de una variable**.

```
arrange(data, desc(anio, n))
```


Mini- ejercicio 3 ~~(y final)~~

Con las herramientas para **seleccionar, filtrar y ordenar**, puedes hacerlo de **distintas** maneras.

1. ¿Cuántas personas **con tu nombre** nacieron **el mismo año que tú**?

pista: R es *case sensitive*, y los nombres tienen su primera letra mayúscula (p.e: "Camila").

1. ¿Cuál fue el nombre más usado el año que tú naciste?

1. **bonus:** ¿Cuál es el nombre **más usado** en Chile en todos los tiempos?



Nada de esto sería posible sin:

- [R for Data Science](#), de Hadley Wickham
- [Advanced R](#), de Hadley Wickham
- [Data wrangling, exploration, and analysis with R](#), de Jenny Bryan
- [Introduction to R](#), de Data Carpentry
- [Xaringan: Presentation Ninja](#), de Yihui Xie. Para generar esta presentación con la plantilla ninja ✂

R for Data Science tiene una traducción al español realizada por la comunidad hispana de R:

- [R para ciencia de datos](#), de Hadley Wickham



Capacitación en R y herramientas de productividad

Proyecto Estratégico Servicios Compartidos para la Producción Estadística

Introducción y herramientas de exploración de datos

Agosto 2020