



unesco



British Embassy  
Mexico City

inai



Instituto Nacional de Transparencia, Acceso a la  
Información y Protección de Datos Personales

Sesión 04

# Introducción de R para periodistas

Curso de **Periodismo de datos**

**Juvenal Campos**

21/04/2022

Programa de

**Actualización y Capacitación para Periodistas**  
en México 2022

# Contenido

## TEMAS:

- a)Instalación de R**
- b)R en general**
- c)Cosas que se pueden hacer en R**
- d)Instalación de paquetes**
- e)Práctica introductoria a R**

# ¿Por qué R?



R

R es un entorno y un ***lenguaje de programación*** con un enfoque en el análisis estadístico.

Apareció en 1993, como una evolución del lenguaje “S” por **Ross Ihaka** y **Robert Gentleman**. En 1995 se le dio su nombre.

R y RStudio no son lo mismo. R es el lenguaje y RStudio es la herramienta para programar en este lenguaje de programación.





## ¿Por qué R?

1. ¡Es gratis!, *open source*, y disponible para los principales sistemas operativos.
2. R tiene una comunidad muy grande de usuarios y programadores, generando material de ayuda y de referencia.
3. Existe una gran cantidad de paquetes y librerías.
4. Tiene herramientas de visualización y generación de reportes muy buenas.

Fuente: Wickham, 2019. Advanced R. Introducción.



## ¿Por qué R?

**5. Contamos con el IDE RStudio.**

**6. R se utiliza en desarrollos de Inteligencia Artificial y Machine Learning.**

**7. Entre otras ventajas** (Vectorización, productos de RStudio, Metaprogramación, conexión con C, C++, etc).

Fuente: Wickham, 2019. Advanced R. Introducción.



# Alternativas a R

**Python.** Lenguaje de programación muy utilizado en ciencia de datos, pero también en tareas de programación pura y dura, especialmente para páginas web. Al igual que R, es gratuito, libre y de código abierto.



**Stata.** Paquete de software estadístico muy utilizado en economía, sociología, ciencias políticas y algunas ramas de la medicina, y sirve para hacer estadística y econometría. No es libre, hay que pagar licencia, y no tiene una base de usuarios tan activa.



**Excel.** El programa de hojas de cálculo por excelencia. Nos permite limpiar y analizar bases de datos, así como hacer visualizaciones de datos (igual que R), pero no es libre, ni gratuito, ni permite la replicabilidad ni la automatización de procesos.



**Otros más:** Julia, SPSS, eViews, SAS, etc.

# Alternativas a R

	Libre y Gratis	Procesos replicables	Hojas de Cálculo	Usuarios	Librerías
	Si	Si	No	Muy usado en entornos académicos y de ciencias sociales	Muchas, comunidad muy activa
 <b>Python</b>	Si	Si	No	Muy usado en Modelos de Inteligencia Artificial y por programadores	Muchas, muchos programadores.
 <b>Stata</b>	No	Si	Algunas funciones	Usado mayoritariamente sólo para econometría y para ciencias sociales	Algunas, las elaboran principalmente los empleados de la empresa.
 <b>Excel</b>	No	No	Si	Ampliamente usado por todo mundo	Algunos plugins para hacer cosas avanzadas.

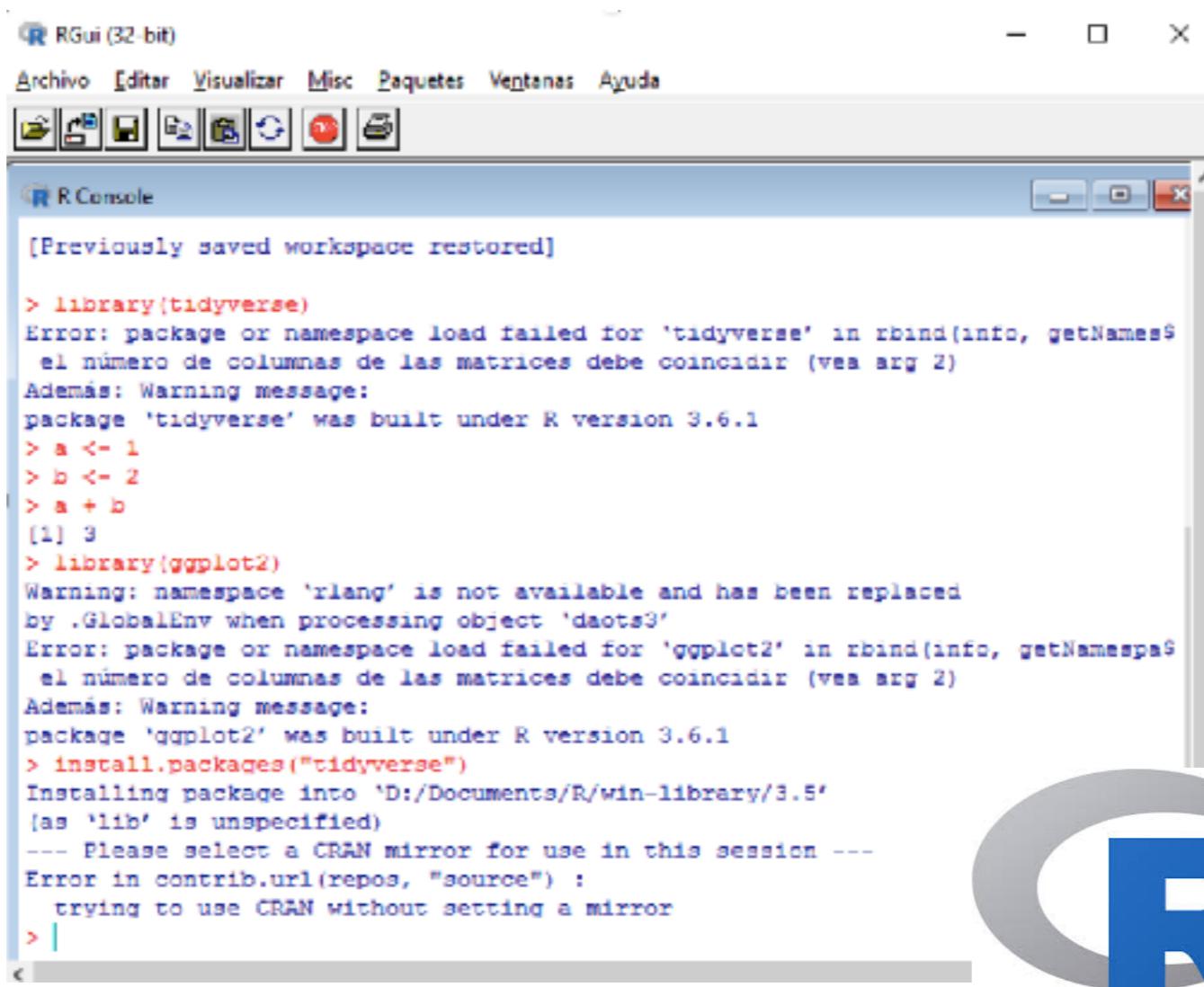
Fuente: Yo

# Alternativas a RStudio



## (Recordemos que R != RStudio)

**Consola de R.** Es la consola default que instalamos al instalar R (no RStudio). No proporciona las facilidades que proporciona RStudio (editor de texto, visualizador web, etc).



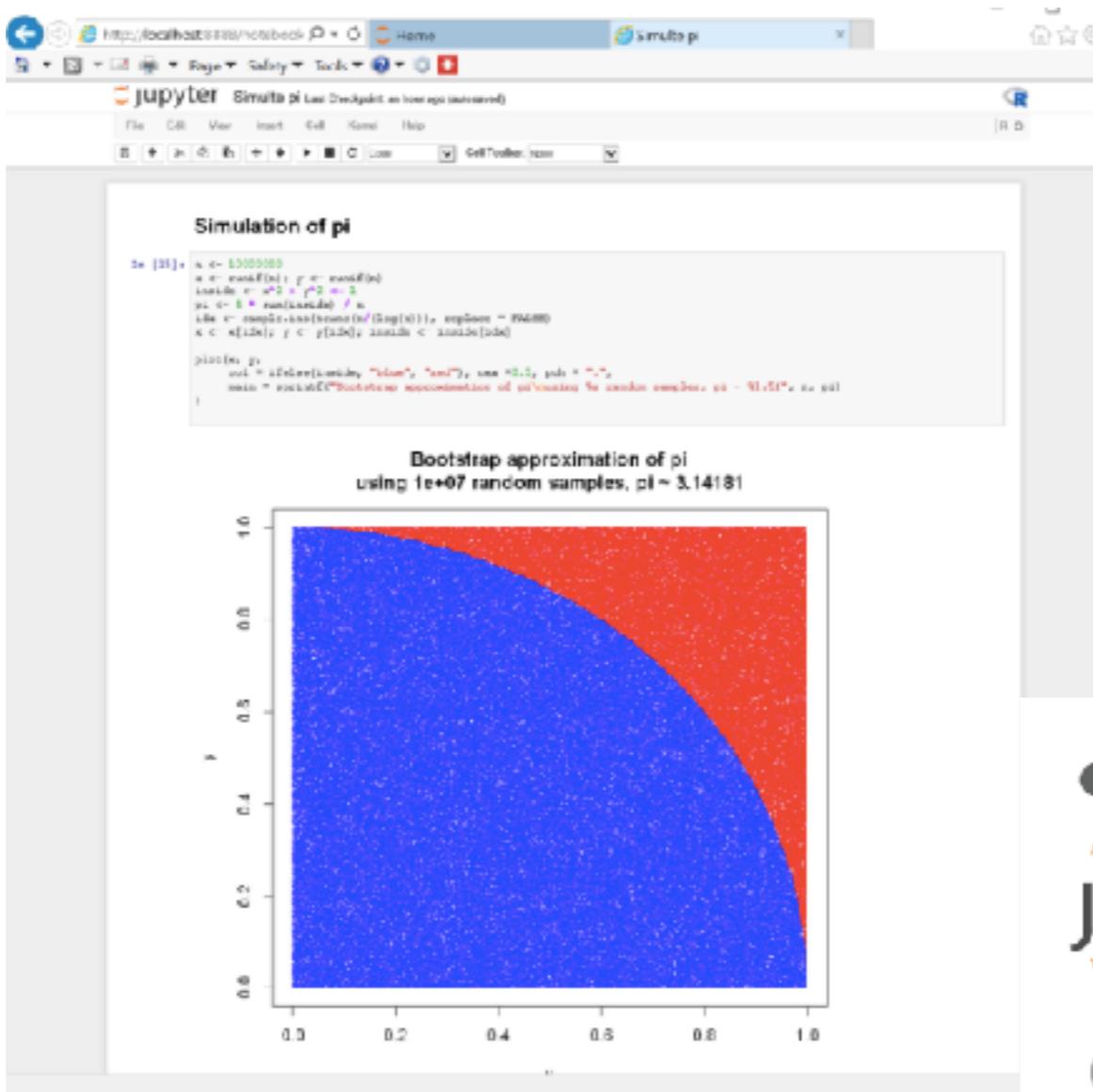
A screenshot of the RGui (32-bit) application window. The title bar says "RGui (32-bit)". The menu bar includes "Archivo", "Editar", "Visualizar", "Misc", "Paquetes", "Ventanas", and "Ayuda". Below the menu is a toolbar with several icons. The main window is titled "R Console" and contains the following text:

```
[Previously saved workspace restored]

> library(tidyverse)
Error: package or namespace load failed for 'tidyverse' in rbind(info, getNamesp...
  el número de columnas de las matrices debe coincidir (vea arg 2)
Además: Warning message:
package 'tidyverse' was built under R version 3.6.1
> a <- 1
> b <- 2
> a + b
[1] 3
> library(ggplot2)
Warning: namespace 'rlang' is not available and has been replaced
by .GlobalEnv when processing object 'daots3'
Error: package or namespace load failed for 'ggplot2' in rbind(info, getNamesp...
  el número de columnas de las matrices debe coincidir (vea arg 2)
Además: Warning message:
package 'ggplot2' was built under R version 3.6.1
> install.packages("tidyverse")
Installing package into 'D:/Documents/R/win-library/3.5'
(as 'lib' is unspecified)
--- Please select a CRAN mirror for use in this session ---
Error in contrib.url(repos, "source") :
  trying to use CRAN without setting a mirror
> |
```



**Jupyter Notebook.** Aplicación web *Open Source* que permite crear y compartir documentos que contienen código, ecuaciones. Visualizaciones y texto narrativo. Tiene varias de las funciones que tiene RStudio, pero la instalación es mucho más complicada, al igual que el arranque.



# Alternativas a RStudio

## Visual Studio Code.

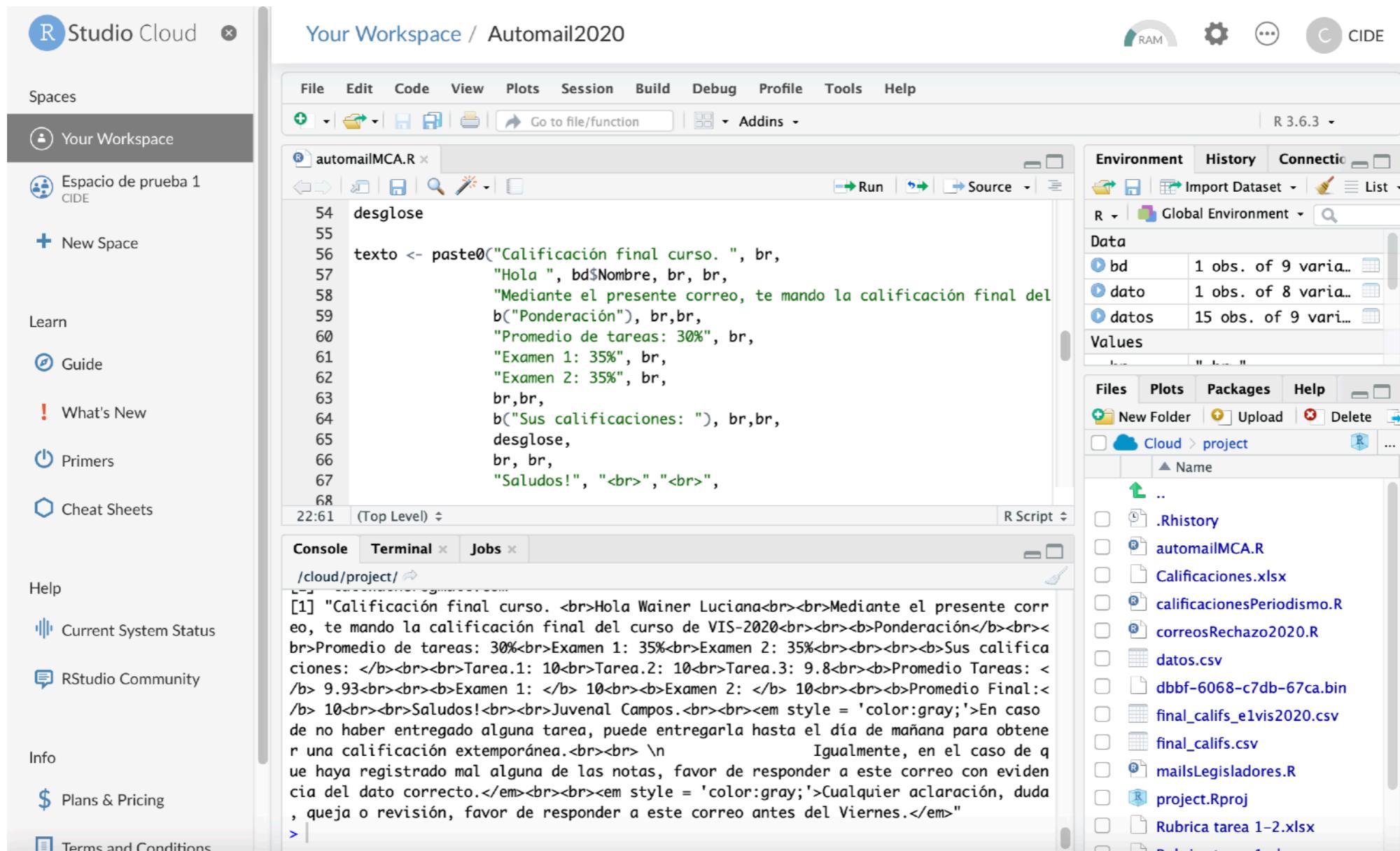
Visual Studio code es una aplicación para programar creada por Microsoft y ampliamente utilizada en el mundo de la programación. Si se quiere programar todo en la vida con un solo programa, VSCode es la solución.

The screenshot shows the Visual Studio Code interface with the following components:

- Left Panel (Terminal):** Displays R code and its output. The code generates a scatterplot titled "Scatterplot" showing "Area Vs Population". The plot includes a blue loess smoothing line. The R console output shows the code run and some warning messages about removed rows.
- Middle Panel (Editor):** Shows an R script with a while loop for calculating a sum of digits. A code completion tooltip for the "floor" function is displayed, providing its documentation.
- Right Panel (Search Results):** Shows the documentation for the "floor" function, which takes a single numeric argument and returns the largest integer not greater than the argument.

# RStudio Cloud

Tenemos la posibilidad también de ejecutar códigos de R desde el servicio de RStudio en línea.



# Instalación de R y RStudio

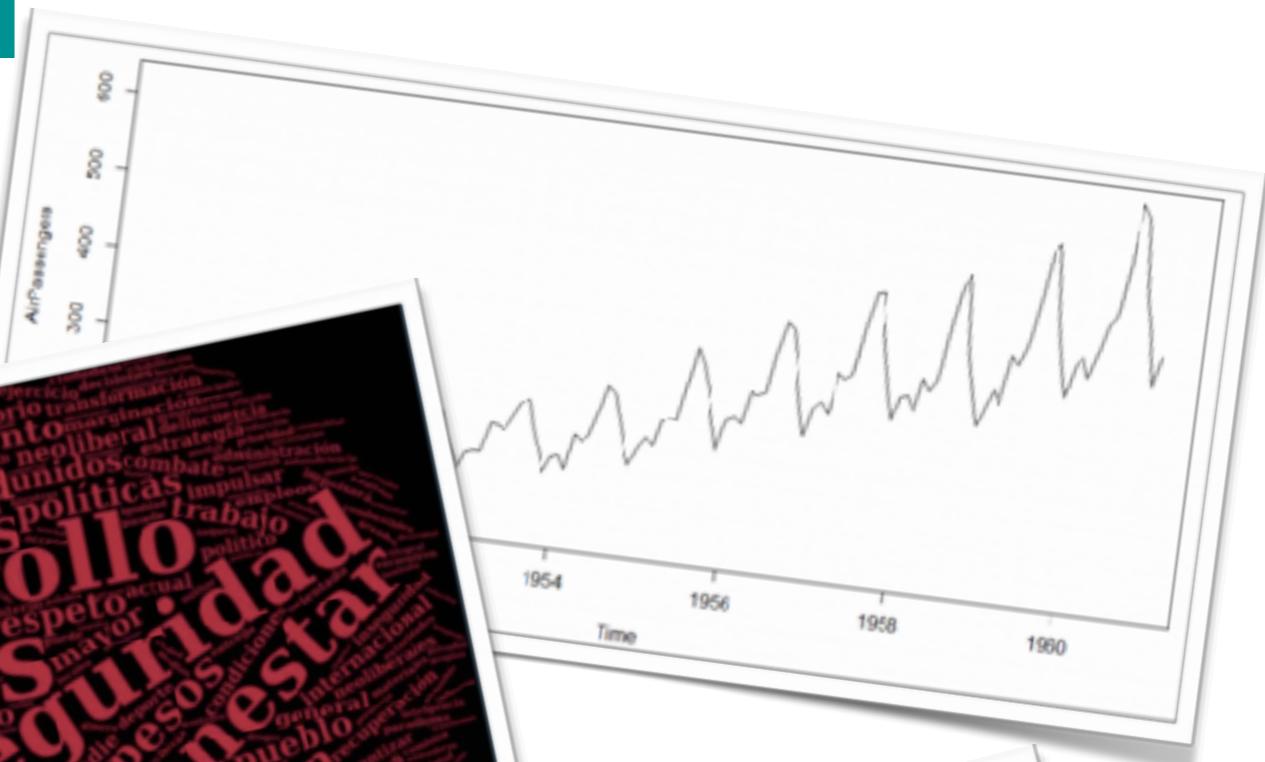
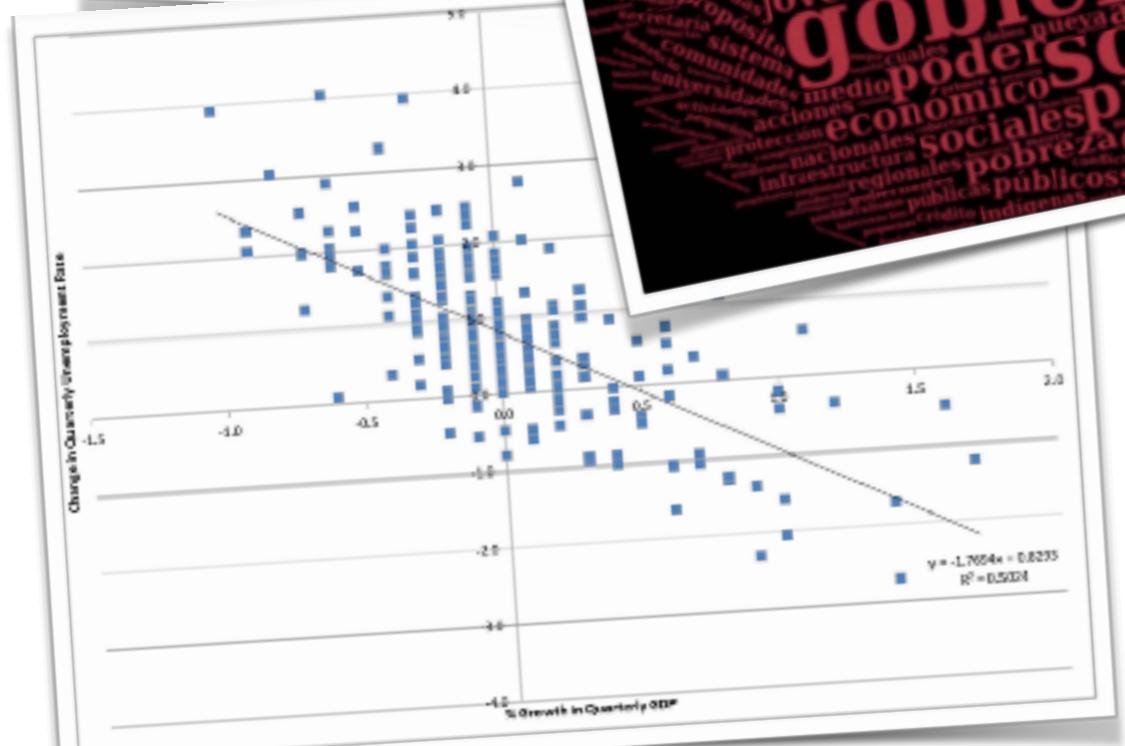
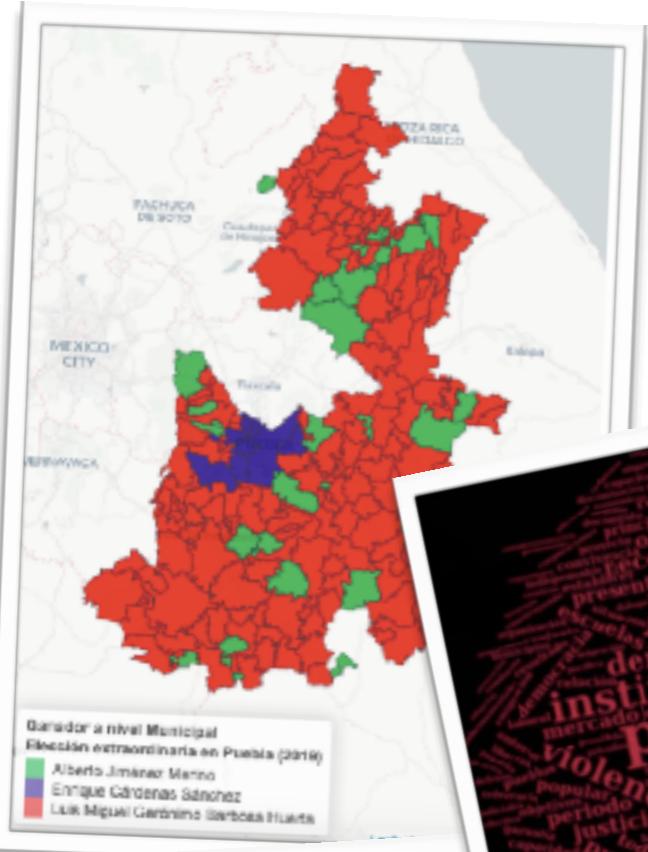


1. Podemos instalar R desde el mirror del ITAM: <https://cran.itam.mx> seleccionando el sistema operativo que nos corresponda.
2. Posteriormente, instalamos como cualquier programa normal.
3. Para actualizar R, volvemos a descargar R y volvemos a instalar. Esta instalación sobreescribe la versión anterior.



1. Podemos instalar RStudio en <https://www.rstudio.com/products/rstudio/download/>. Con la versión gratuita es suficiente.
2. Instalamos como cualquier programa normal, una vez que tengamos R instalado.
3. Actualizamos igual que R.

# ¿Qué se puede hacer con R?

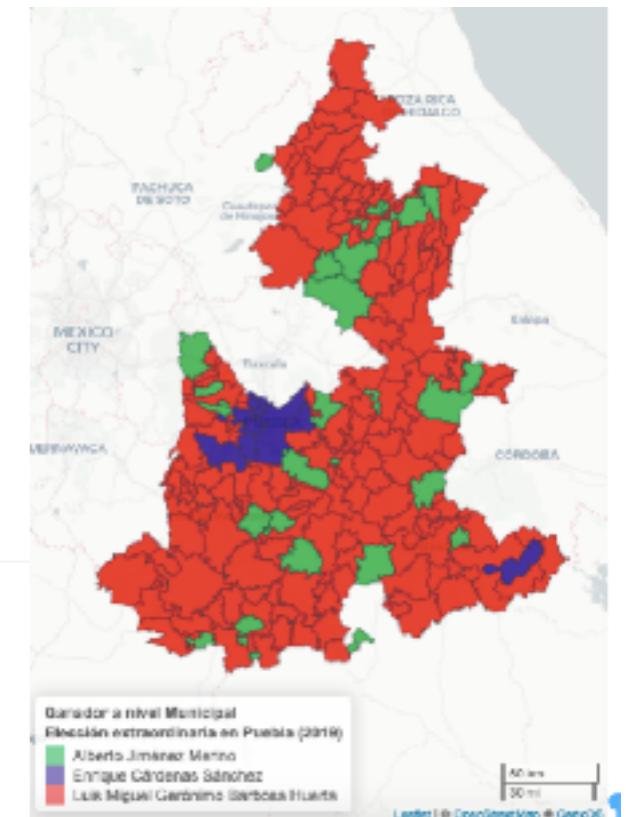


# ¿Qué se puede hacer con R?

## 1. Manejo y manipulación de datos en R.

```
library(tidyverse)
```

```
72   datos <- prep %>%  
73     select(ECS, AJM, LMGBH, TOTAL_VOTOS, LISTA_NOMINAL, MUNICIPIO, DISTRITO) %>%  
74     filter(!is.na(MUNICIPIO)) %>%  
75     group_by(MUNICIPIO) %>%  
76     summarise(ECS = sum(ECS, na.rm = TRUE),  
77                 AJM = sum(AJM, na.rm = TRUE),  
78                 LMGBH = sum(LMGBH, na.rm = TRUE),  
79                 Total_Votos = sum(TOTAL_VOTOS, na.rm = TRUE),  
80                 ListaNominal = sum(LISTA_NOMINAL, na.rm = TRUE)  
81   )
```



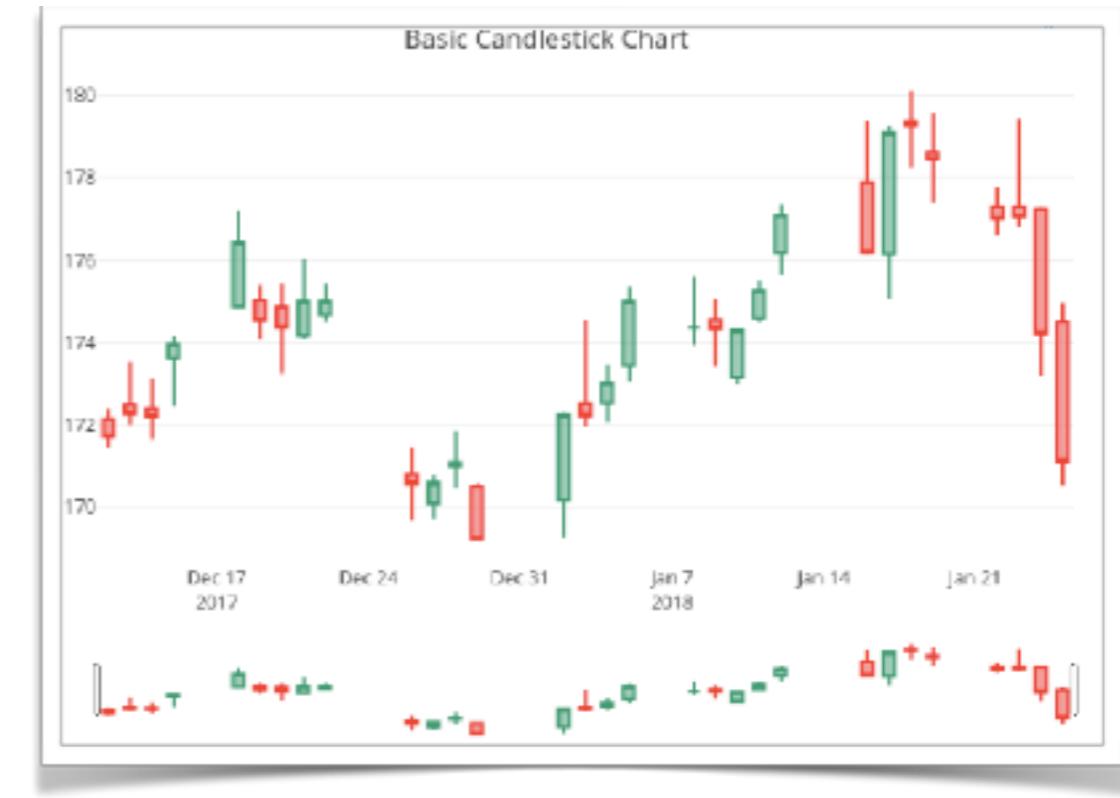
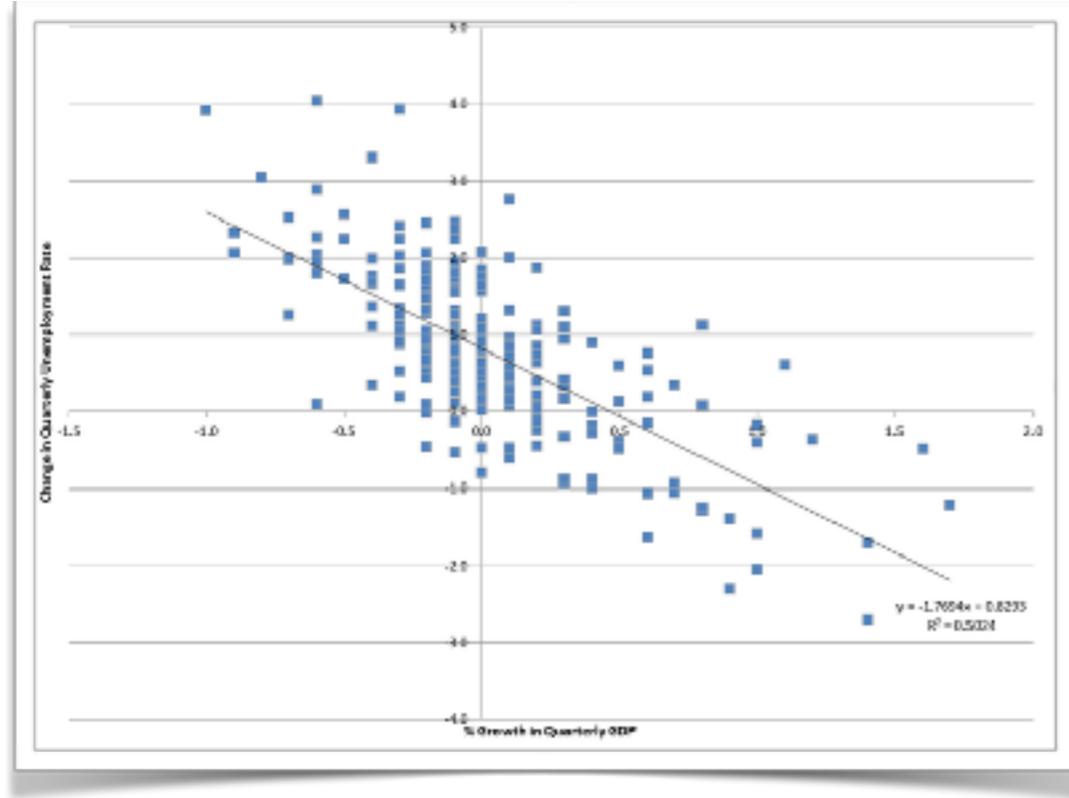
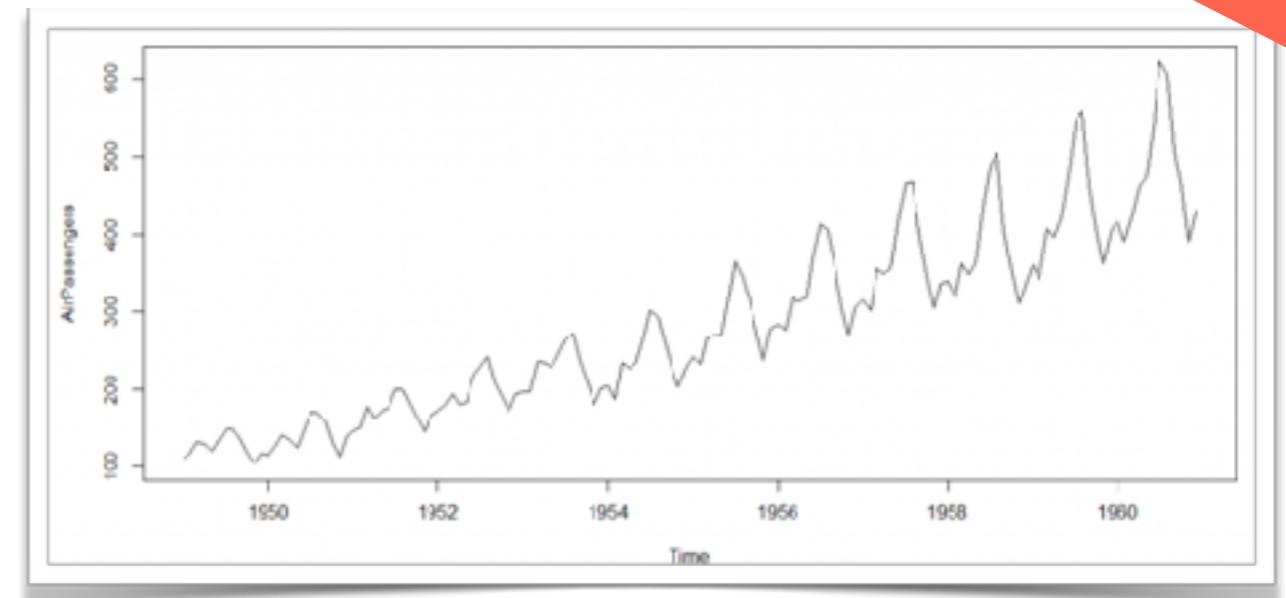
# ¿Qué se puede hacer con R?

Esto no lo vamos a ver en la clase 😊

## 2. Análisis estadístico y econometría.

**library(base)**

**library(MASS)**



# ¿Qué se puede hacer con R?

Esto no lo vamos a ver en la clase 😊

## 3. Machine Learning y Deep Learning.

**library(e1071)**

**library(tensorflow)**

**library(caret)**

**library(rpart)**

etc.

**Classification**



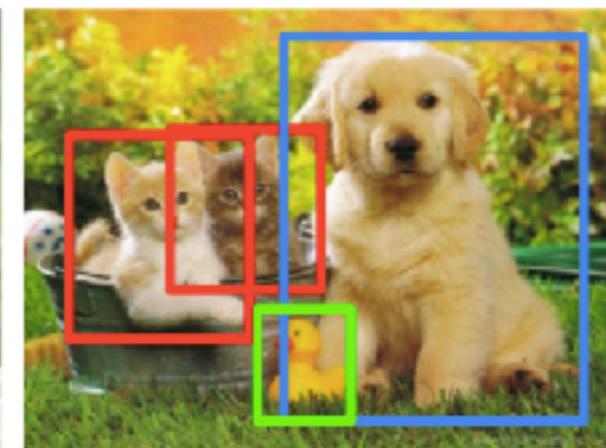
CAT

**Classification + Localization**



CAT

**Object Detection**



CAT, DOG, DUCK

**Instance Segmentation**



CAT, DOG, D

Single object

Multiple objects

Fuente: <https://www.mocomakers.com/cats-versus-dogs/>

# ¿Qué se puede hacer con R?

*Esto lo vamos a  
ver en la clase*

## **4. Análisis de texto.**

# library(tm)

```
library(stringr)
```



## Nube de palabras.

## **Solicitudes de Acceso a información realizadas en el Estado de Morelos.**



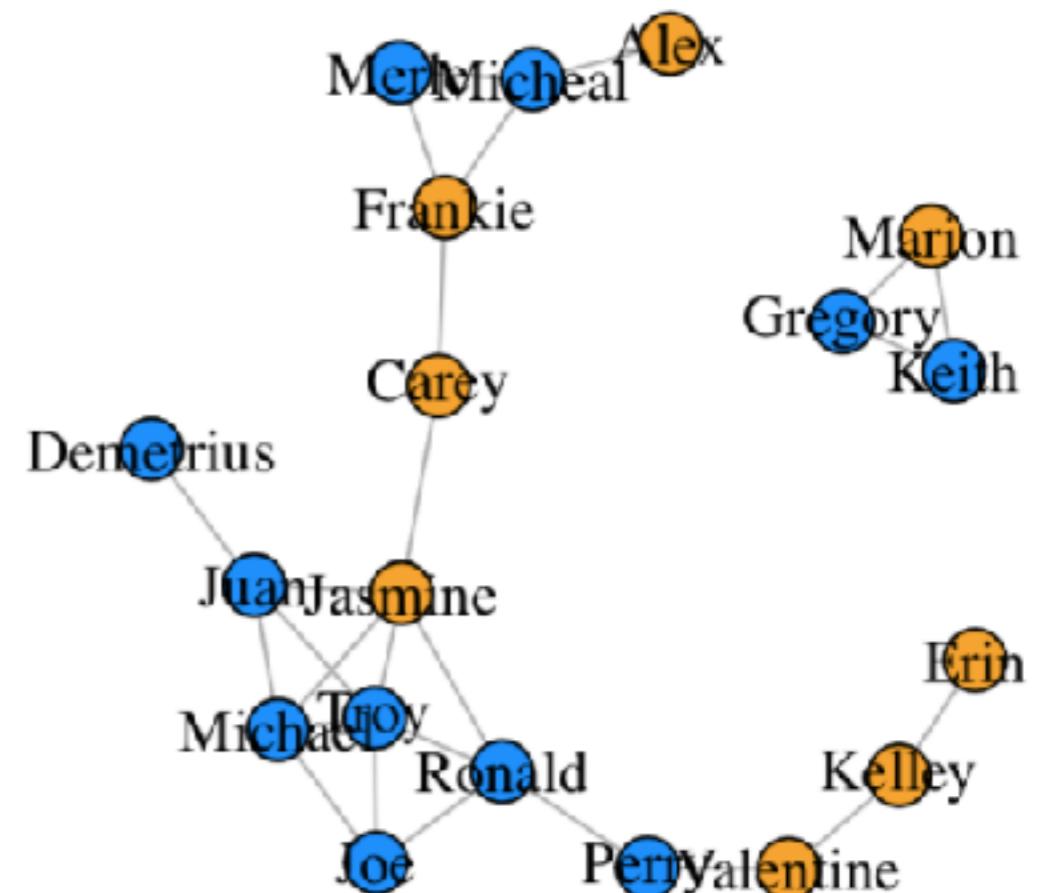
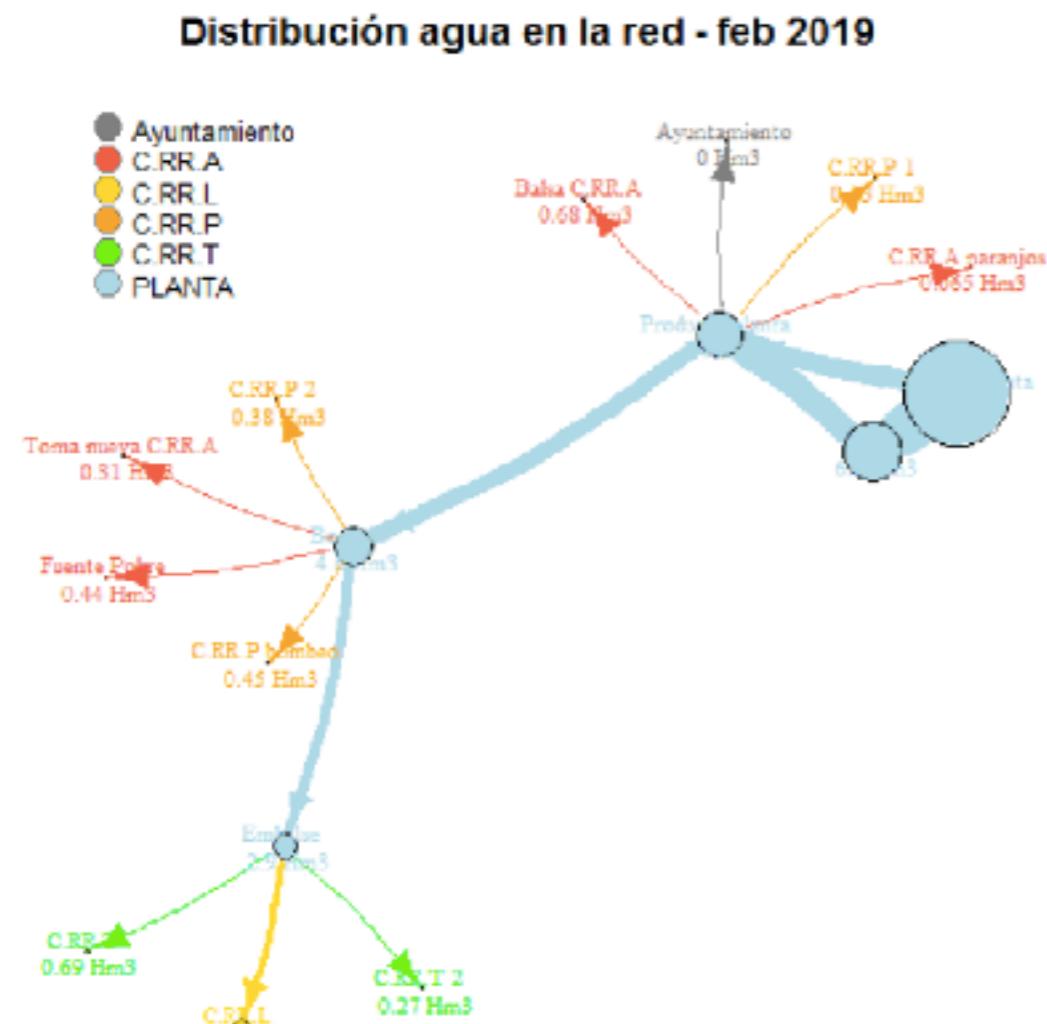
## Nube de palabras.

## **Plan Nacional de Desarrollo. 2019.**

# ¿Qué se puede hacer con R?

## 5. Análisis de redes.

`library(igraph)`



# ¿Qué se puede hacer con R?

## 6. Visualización de datos.

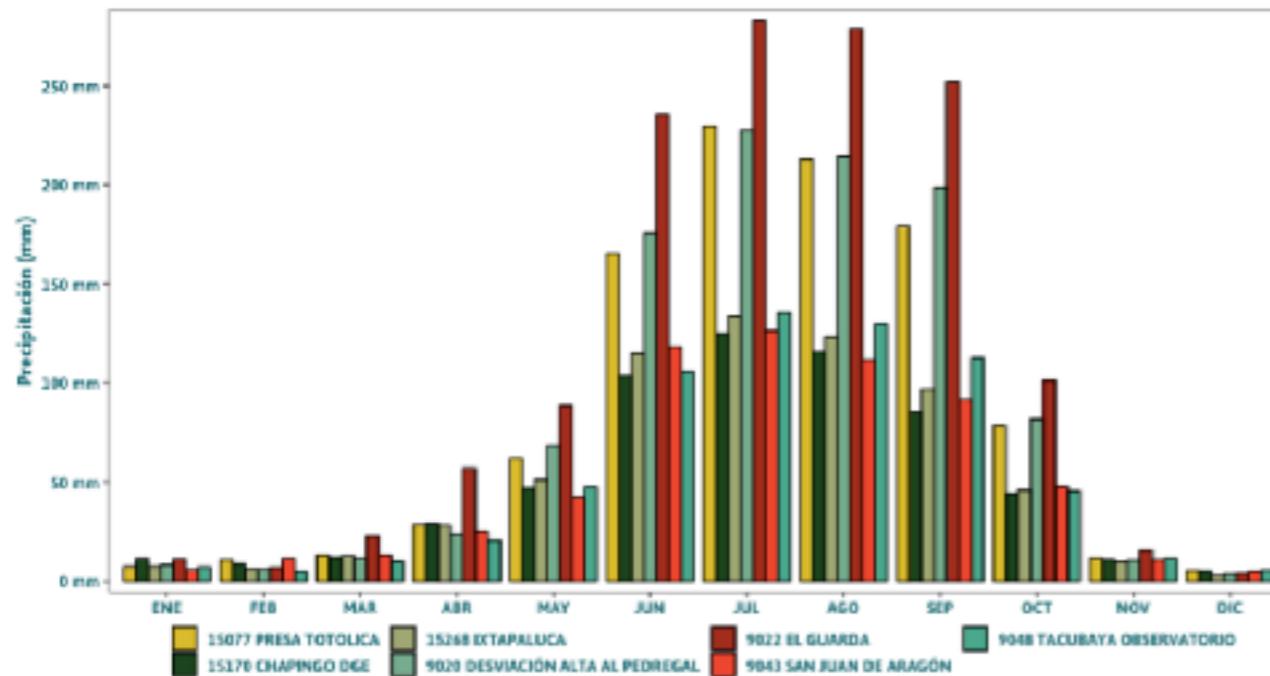
**library(ggplot2)**

**library(plotly)**

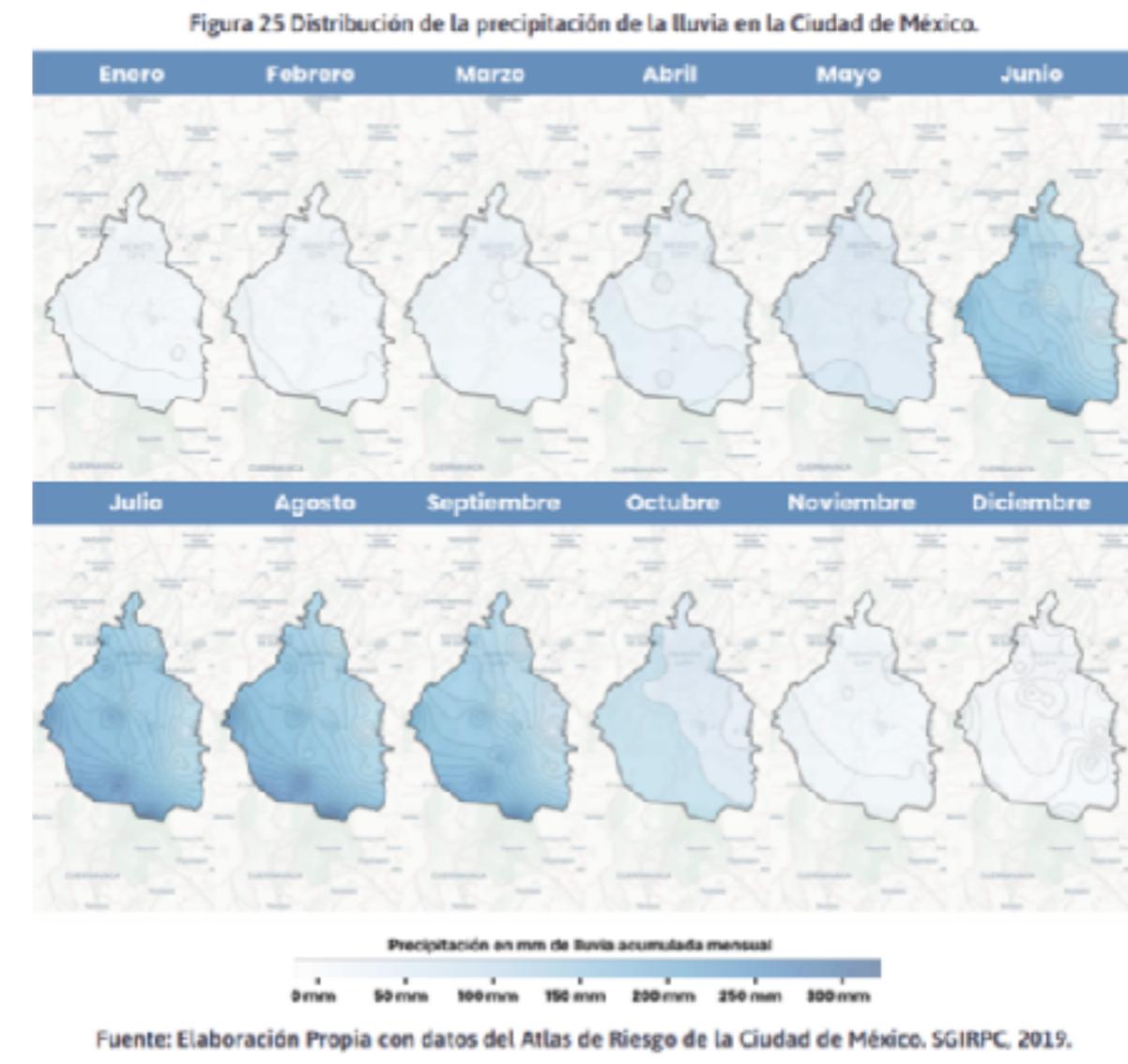
**library(leaflet)**

**library(htmlwidgets)**

Figura 24 Precipitación mensual promedio en 7 estaciones seleccionadas de la ZMVM.



Gráfica de barras de la distribución de la lluvia en la CDMX, en el tiempo.



Mapas de la distribución de la lluvia en la CDMX, en el espacio y tiempo.

# ¿Qué se puede hacer con R?

Esto no lo vamos a ver en la clase 😊

## 7. Recolección automática de información (Web Scrapping, Data Crawling).

**library(rvest)**

**library(xml)**

**Ejemplo: Extraer precios e información de vuelos desde Google Flights**

Flight Details	Duration	Stops	Price
06:05 – 14:35 <sup>1</sup> United - ANA	18 h 30 m MEX-NRT	1 stop 2 h 35 m SFO	MX\$20,089 round trip
07:30 – 15:20 <sup>1</sup> United, ANA	17 h 50 m MEX-NRT	1 stop 1 h 35 m IAH	MX\$20,089 round trip
02:20 – 06:45 <sup>1</sup> ANA	14 h 25 m MEX-NRT	Non-stop	MX\$21,689 round trip
01:30 – 06:20 <sup>1</sup> Aeroméxico - UA	14 h 50 m MEX-NRT	Non-stop	MX\$31,471 round trip
17:30 – 14:20 <sup>2</sup> United, ANA	30 h 50 m MEX-NRT	2 stops: ▲ IAH, ORB	MX\$20,089 round trip
17:30 – 14:20 <sup>2</sup> United, ANA	30 h 50 m MEX-NRT	2 stops: ▲ IAH, ORB	MX\$20,089 round trip

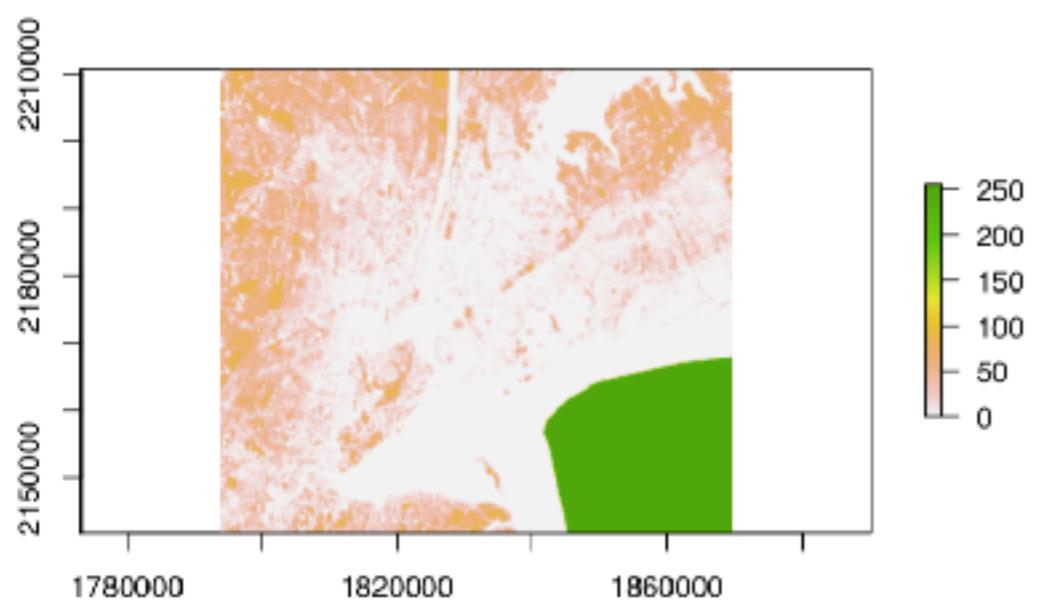
[https://www.google.com/flights?lite=0#flt=MEX.NRT.2019-10-21\\*NRT.MEX.2019-11-05;c:MXN;e:1;sd:1;t:f](https://www.google.com/flights?lite=0#flt=MEX.NRT.2019-10-21*NRT.MEX.2019-11-05;c:MXN;e:1;sd:1;t:f)

# ¿Qué se puede hacer con R?

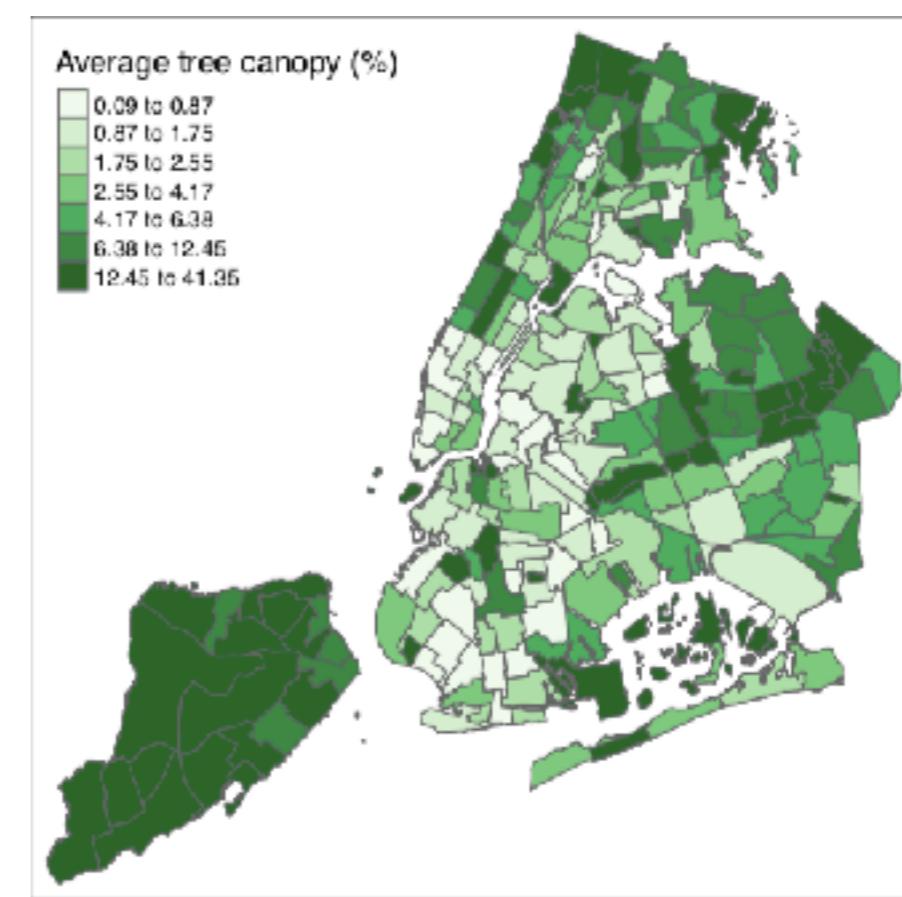
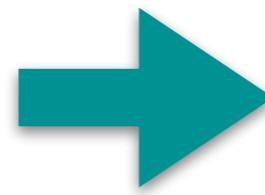
## 8. Análisis Geoespacial.

`library(sf)`

Abrir información geográfica, modificarla y visualizarla, así como realizar análisis a partir de esta.



Datos Crudos  
(Raw Data)



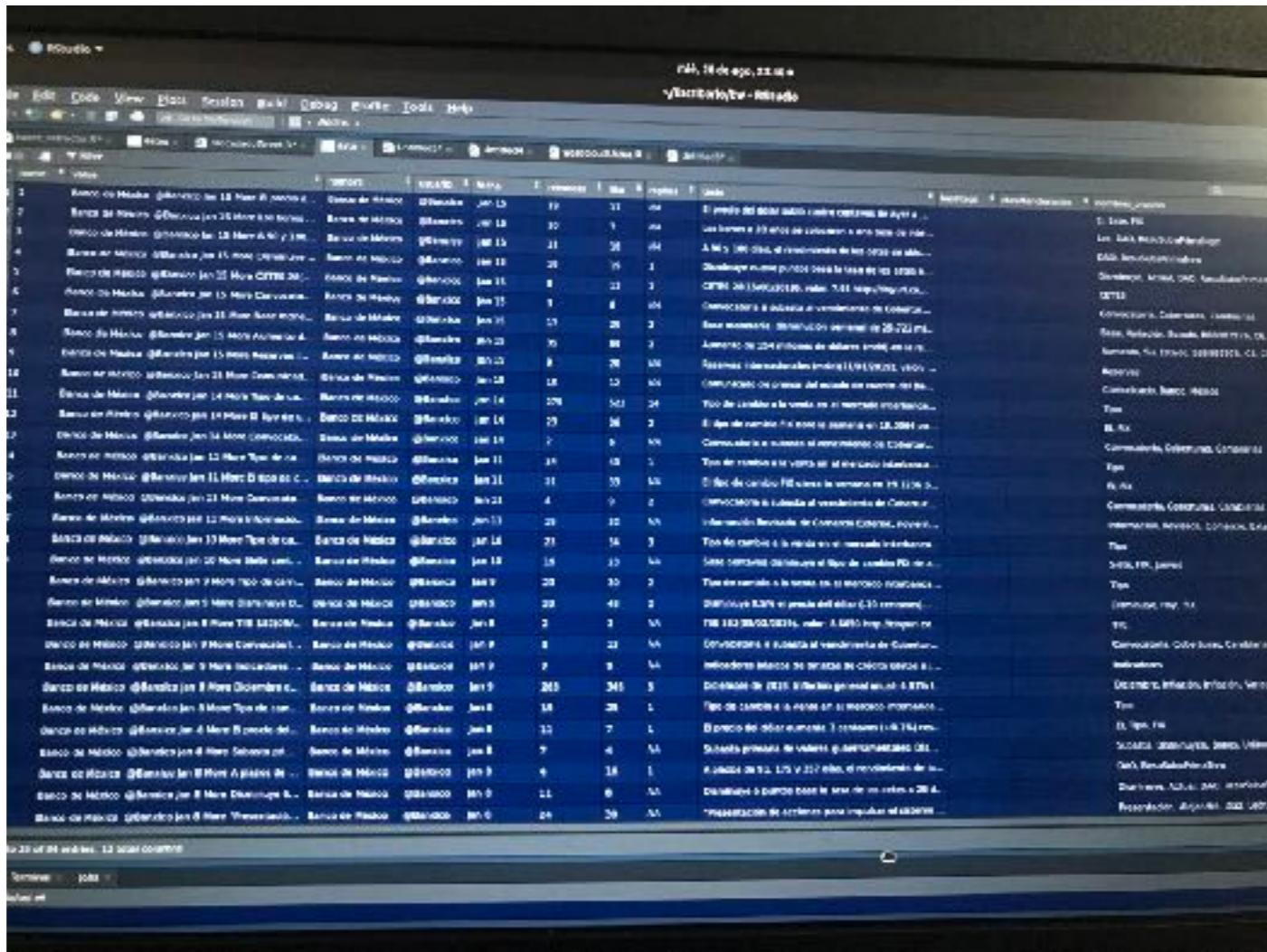
Datos Procesados que permiten llegar a conclusiones

# ¿Qué se puede hacer con R?

## 9. Automatización de tareas.

### library(Rselenium)

La librería RSelenium permite programar el navegador para que replique cosas que nosotros podríamos hacer manualmente (p. ej. Descargar archivos, revisar Twitter, mandar correos, etc.).



# ¿Qué se puede hacer con R?

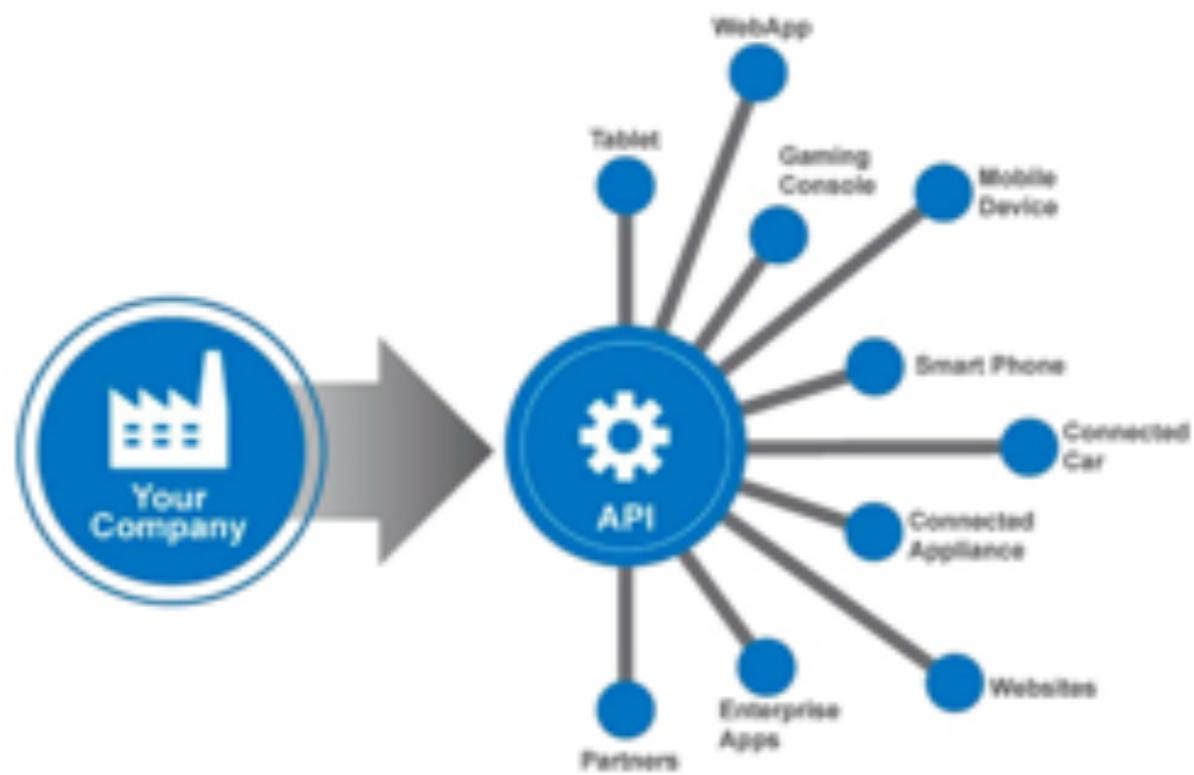
Esto no lo vamos a ver en la clase 😊

## 10. Creación de APIs.

`library(plumber)`

Definición: API (Interfaz de Programación de Aplicaciones).

Un API es un programa que permite la comunicación entre componentes de software (por ejemplo, la App de Facebook del celular se comunica con los servidores mediante la API de FB).

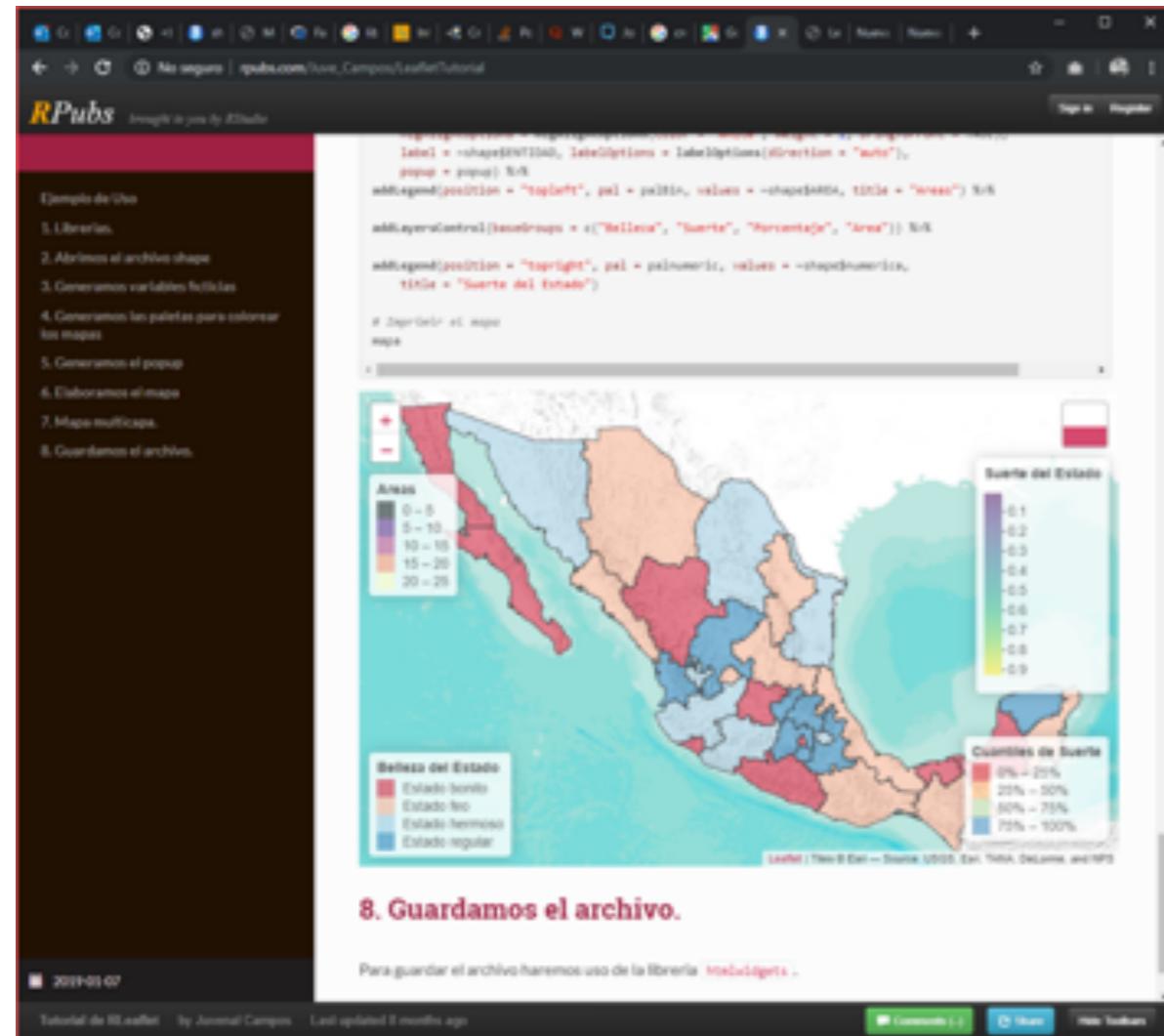


Si un día necesitamos desarrollar una API, podemos hacer esto a partir de la librería *plumber*.

# ¿Qué se puede hacer con R?

## 11. Páginas web y reportes (\*.pdf, \*.doc, diapositivas, tableros estáticos).

### library(markdown)



Unit 2 of the course

Juvenal Campos - Based on DataCamp ML Classification Course

19/1/2018

### Vector and raster coordinate systems

In order to perform any kind of analysis with more than one layer, you have to define the reference system (CRS), and the first step is determining what coordinate reference system your data has. To do this you can make use of the `sf` function `st_crs()` and the `raster` function `crs()`.

When the geographic data was read with `sf` already has a CRS defined both sf and raster will recognize and retain it. When the CRS is not defined you will need to define it yourself using either the EPSG number or the `proj4string`.

**sf**

```
# Load the packages
library(sf)

# Linking to GDAL 3.6.1, GML 2.1.0, proj.4 4.9.3
library(raster)

# Loading required package: sp
root <- "/Users/juvicampos/Desktop/DataCamp/GeoSpatial-Analyst/RV_raster_sf_and_rasterDatabases"
setwd(root)

# Read in the tree shapefile
tree <- st_read(paste0(root, "/trees/tree.shp"))

# Reading layer 'trees' from data source '/Users/juvicampos/Desktop/DataCamp/GeoSpatial-Analyst/RV_raster_sf_and_rasterDatabases/trees.shp'
# Simple feature collection with 6227 features and 7 fields
# geometry type: POLYGON
# dimension: XY
# bbox: -16.2846 ymin: 40.40939 ymax: -13.71079 ymin: 40.91165
# epsg (SRID): 4326
# proj4string: +proj=longlat +ellps=WGS84 tmerc,dems
# Read in the neighborhood shapefile
neighborhoods <- st_read(paste0(root, "/neighborhoods/neighborhoods.shp"))

# Reading layer 'neighborhoods' from data source '/Users/juvicampos/Desktop/DataCamp/GeoSpatial-Analyst/RV_raster_sf_and_rasterDatabases/neighborhoods.shp'
# Simple feature collection with 128 features and 5 fields
# geometry type: POLYGON
# dimension: XY
# bbox: -16.2846 ymin: 40.40939 ymax: -13.70021 ymin: 40.91054
# epsg (SRID): 4326
# proj4string: +proj=longlat +ellps=WGS84 tmerc,dems
# Read in the tree canopy shapefile
canopy <- raster(paste0(root, "/canopy.tif"))
```

**raster**

[http://rpubs.com/Juve\\_Campos/LeafletTutorial](http://rpubs.com/Juve_Campos/LeafletTutorial)

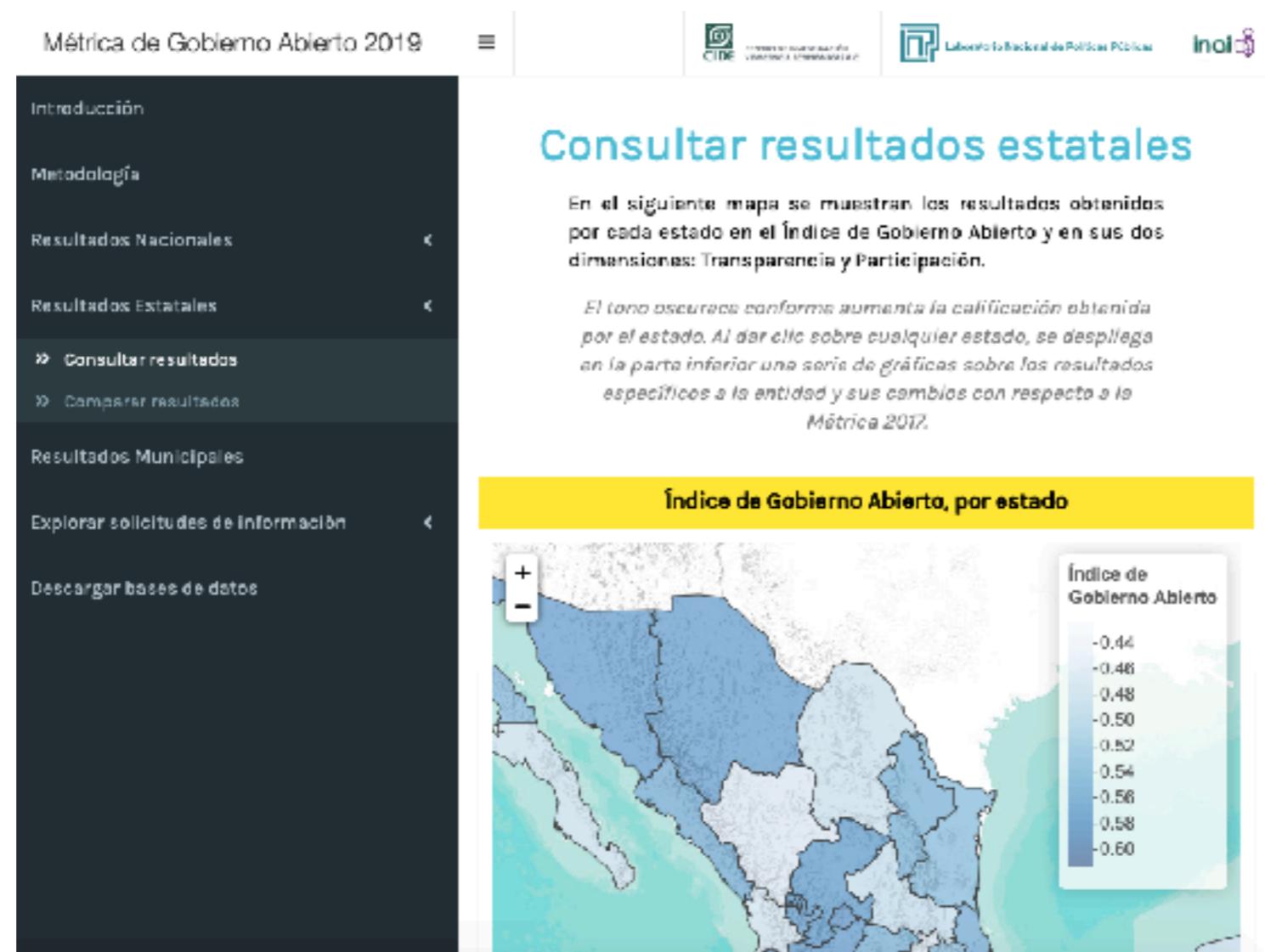
LATEX pdf

# ¿Qué se puede hacer con R?

Esto no lo vamos a ver en la clase 😊

## 12. Web Apps.

### library(shiny)



Página web de resultados de la métrica de gobierno abierto, 2019.

<https://lnppmicrositio.shinyapps.io/metricadegobiernoabierto2019/>

# Sugerencias

## 1. Google y StackOverflow son tus aliados.

Si tienes alguna duda o surge algún error desconocido en tu código, o si quieres aprender a hacer algo muy puntual y específico, **googlear debe ser tu primera opción.** ***La probabilidad de que alguien haya tenido la misma duda que tu es muy alta,*** y es probable que la solución ya se encuentre en línea. Y si está en línea, lo más probable es que se encuentre en StackOverflow.com.

# Sugerencias

## 2. No sufran en silencio.

**Tus compañeros pueden ayudarte, al igual que tus profesores.**

No acumules dudas por pena ni durante mucho tiempo, ya que los temas son acumulativos y una duda no resuelta en una clase puede hacer que no entiendas las clases posteriores.

# Sugerencias



## 3. Haz un perfil de GitHub

GitHub es una plataforma para programadores.

Sirve como:

- almacén de código y archivos (repositorio),
- como control de versiones,
- como plataforma de colaboración y
- como red social para programadores.

Si tu trabajo es perfectamente reproducible, **un usuario ajeno debe poder descargar el repositorio de GitHub y, siguiendo tus instrucciones, poder correrlo en su computadora personal sin mayor problema.**

# Sugerencias

## 4. Dale estructura a tu código.

Trata de llevar un orden y estructura de tu código, de tal forma que facilite su lectura y reproducción. Recuerda incluir los comentarios necesarios para que sepas que hiciste en cada paso de tu trabajo.

### Sugerencia personal:

- Primero:** Configurar el entorno de trabajo (Que se trabaje con caracteres del idioma español, UTF-8, quitar la notación científica, etc.).
- Segundo:** Sección de librerías.
- Tercero:** Sección de funciones propias.
- Cuarto.** Sección para importar las bases de datos.
- Quinto:** Exploración, Limpieza y manejo de las Bases de Datos.
- Sexto:** Análisis y estadísticas.
- Séptimo:** Impresión de resultados y guardado de la información.

Ejemplo: [http://rpubs.com/Juve\\_Campos/estructuraCodigoR](http://rpubs.com/Juve_Campos/estructuraCodigoR)

# Acordeones y cheatsheets

Algunos cheatsheets de interés:

-**Fechas y tiempos:** <https://github.com/rstudio/cheatsheets/raw/master/lubridate.pdf>

-**Manejo de cadenas de texto:**

<https://github.com/rstudio/cheatsheets/raw/master/strings.pdf>

-**Funciones apply**

<https://github.com/rstudio/cheatsheets/raw/master/purrr.pdf>

-**Importación de datos**

<https://github.com/rstudio/cheatsheets/raw/master/data-import.pdf>

# Acordeones y cheatsheets

-Transformación de datos:

<https://github.com/rstudio/cheatsheets/raw/master/data-transformation.pdf>

-RMarkdown

<https://github.com/rstudio/cheatsheets/raw/master/rmarkdown-2.0.pdf>

-Shiny

<https://github.com/rstudio/cheatsheets/raw/master/shiny.pdf>

-Visualización de datos

<https://github.com/rstudio/cheatsheets/raw/master/data-visualization-2.1.pdf>

-Mapas leaflet

<https://github.com/rstudio/cheatsheets/raw/master/leaflet.pdf>

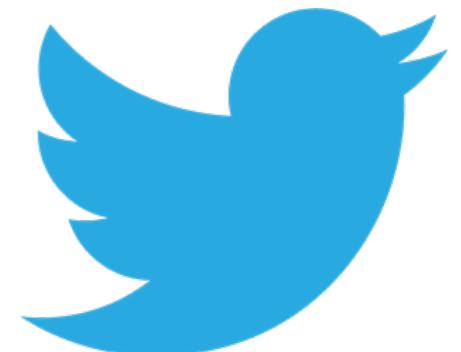
-Paquete sf

<https://github.com/rstudio/cheatsheets/raw/master/sf.pdf>

# A quien seguir en redes sociales:

## -Algunos dateros en Twitter:

- @JuvenalCamposF
- @segasi
- @ jjsantoso
- @datavizero
- @tacosdedatos
- @moaimx
- @jmatoralc
- @nerudista
- @nadiehbremmer
- @dr\_xeo
- @CVWickham
- @HadleyWickham
- @claudidanielpd
- @ioexception
- @rafaelprietoc
- @guzmart\_
- @datacivica
- @haro\_ca\_
- @gdeandajauregui
- @SerendipiaData
- @AlbertoCairo
- @ElUniversalData
- @daniel\_isita
- @allison\_horst



## Otros cursos



Laboratorio Nacional  
de Políticas Públicas

### Cursos de la Escuela de Métodos del CIDE-LNPP

Revisar: <https://www.lnpp.mx/edm/catalogo>

#### Nivel principiante:

- Introducción a R y manejo de datos.
- Visualización de datos en R
- Introducción al análisis cuantitativo
- Estadística descriptiva y probabilidad

#### Más especializado:

- Series de tiempo univariadas
- Series de tiempo multivariadas
- Análisis de componentes principales
- Metodología de encuestas
- Visualización de datos con shiny
- Análisis de conglomerados
- Introducción al Machine Learning
- Análisis de encuestas con R

## Otros cursos

### Cursos de Datacamp:

- Introduction to R
- Intermediate R
- Introduction to the Tidyverse
- Introduction to data visualization with ggplot2
- Data manipulation with dplyr
- Introduction to importing data in R
- Cleaning data in R
- Joining data with dplyr



*Cursos de business-science.io de Matt Dancho*

<https://university.business-science.io/p/5-course-apps-time-series>

- Data Science for Business with R
- Business Analysis with R
- Shiny Dashboards
- Shiny Developer with AWS



unesco



British Embassy  
Mexico City

inai



Instituto Nacional de Transparencia, Acceso a la  
Información y Protección de Datos Personales

Sesión 04

# RStudio, Estructura y tipos de datos

Curso de **Periodismo de datos**

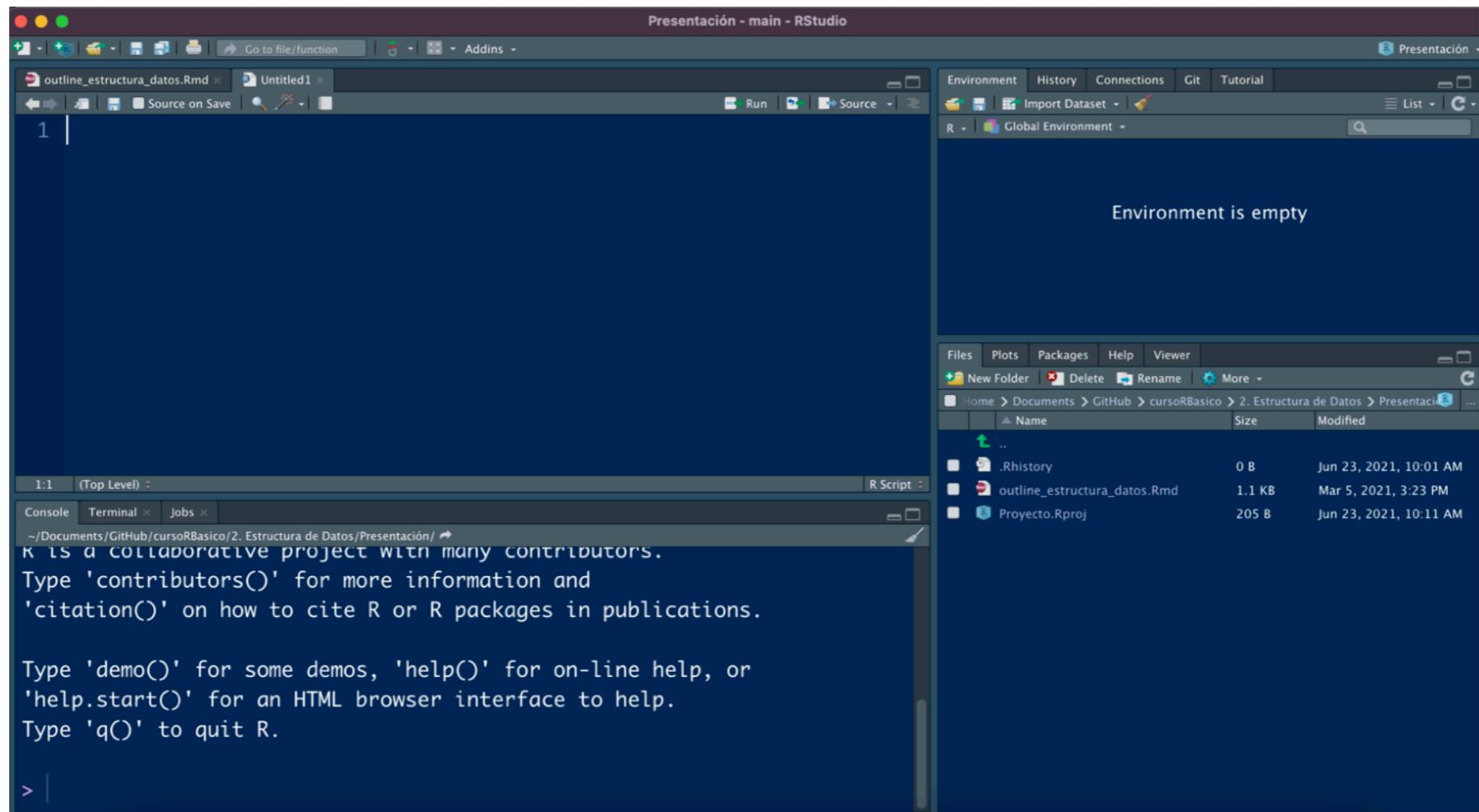
**Juvenal Campos**

21/04/2022

Programa de

**Actualización y Capacitación para Periodistas**  
en México 2022

RStudio es un programa que provee un entorno de desarrollo (IDE) que nos da las herramientas necesarias para poder programar en R.



# Ventanas



The screenshot shows the R Studio interface with several windows open:

- Editor de texto**: A yellow-highlighted window containing R code for data manipulation. It includes lines 2 through 10 of the script, which loads the tidyverse library and defines variables for names, ages, schools, sex, and knowledge.
- Consola**: A green-highlighted window showing the output of the R code run in the Editor. It displays the definition of the 'sex' variable as a factor with levels M and F, and the creation of the 'sabe\_r' variable.
- Visualizador**: A red-highlighted window showing the "Ambiente" (Environment) tab. It lists variables and their types and values:

Variables	Type	Values
años	num	[1:2] 29...
pers...	chr	[1:2] "J...
sabe...	logi	[1:2] T...
- Presentación - main - RStudio**: The main window title bar.
- Toolbar**: Standard R Studio toolbar with file operations like Open, Save, Print, and Addins.
- Bottom Status Bar**: Shows the current working directory as ~/Documents/GitHub/cursoRBasico/2. Estructura de Datos/Presentación/

# Ventanas



## Editor de texto

Sección del programa en la cual registramos las instrucciones que se van a correr en R. Estas instrucciones se guardan en scripts para volver a ellos más adelante.

Acá se pueden escribir códigos de R, HTML, Python, CSS, Markdown, etc.

The screenshot illustrates the RStudio interface with several colored boxes highlighting different components:

- Editor de texto**: The code editor pane, highlighted in yellow, containing R code for data manipulation and analysis.
- Consola**: The console pane, highlighted in green, showing the output of the R code run in the editor.
- Ambiente**: The environment pane, highlighted in pink, displaying the current global variables and their values.
- Visualizador**: The file browser pane, highlighted in red, showing the project structure and files.

The code in the Editor pane:

```
1 # Librerias ----  
2 library(tidyverse)  
3  
4 # Bases de datos ----  
5 personas <- c("Juvenal", "María")  
6 años <- c(29, 30)  
7 escuelas <- c("Colmex", "UNAM")  
8 sexo <- factor(c("M", "F"),  
9                   levels = c("M", "F"))  
10 sabe_r <- c(TRUE, FALSE)
```

The output in the Consola pane:

```
+ library(tidyverse)  
+ personas  
[1] Juvenal  María  
+ años  
[1] 29 30  
+ escuelas  
[1] Colmex  UNAM  
+ sexo  
[1] M F  
Levels: M F  
+ sabe_r <- c(TRUE, FALSE)
```

# Ventanas



## Consola

Sección en la cual se ejecuta el código que vamos a escribir en el editor de texto.

Igualmente, podemos correr acá código de R que no requerimos guardar para más adelante.

The screenshot shows the RStudio interface with several windows open:

- Editor de texto**: A yellow box highlights the code editor window containing R code for creating variables (personas, años, escuelas, sexo, sabe\_r) and setting levels for the sexo factor.
- Consola**: A green box highlights the console window showing the execution of the R code and its output.
- Environment**: A pink box highlights the environment pane showing the global variables: años, pers..., sabe... and their corresponding types and values.
- History**: A pink box highlights the history pane showing the commandLevels = c("M", "F") entered in the console.
- Connections**: A pink box highlights the connections pane.
- Global Environment**: A pink box highlights the global environment pane.
- Ambiente**: A pink box highlights the ambiente pane.
- Visualizador**: A red box highlights the visualizer pane showing the project structure: .., .Rhistory, outline\_estructura\_datos.Rmd, and Proyecto.Rproj.

# Ventanas



## Ambiente

Sección en la cual se pueden explorar y visualizar los objetos generados en nuestro proceso de trabajo en R.

The screenshot illustrates the RStudio interface with several colored boxes highlighting different windows:

- Editor de texto**: The code editor window containing R code for data manipulation.
- Consola**: The console window showing the execution of the R code and its output.
- Visualizador**: The bottom right pane, which is the Environment browser.
- Ambiente**: A pink box highlighting the Environment tab in the top right corner of the RStudio interface.

**Code Editor (Editor de texto):**

```
1 # Librerias ----  
2 library(tidyverse)  
3  
4 # Bases de datos ----  
5 personas <- c("Juvenal", "María")  
6 años <- c(29, 30)  
7 escuelas <- c("Colmex", "UNAM")  
8 sexo <- factor(c("M", "F"),  
9                         levels = c("M", "F"))  
10 sabe_r <- c(TRUE, FALSE)
```

**Console (Consola):**

```
+ Levels = c("M", "F"))  
> sexo  
[1] M F  
Levels: M F  
> sabe_r <- c(TRUE, FALSE)
```

**Environment (Ambiente):**

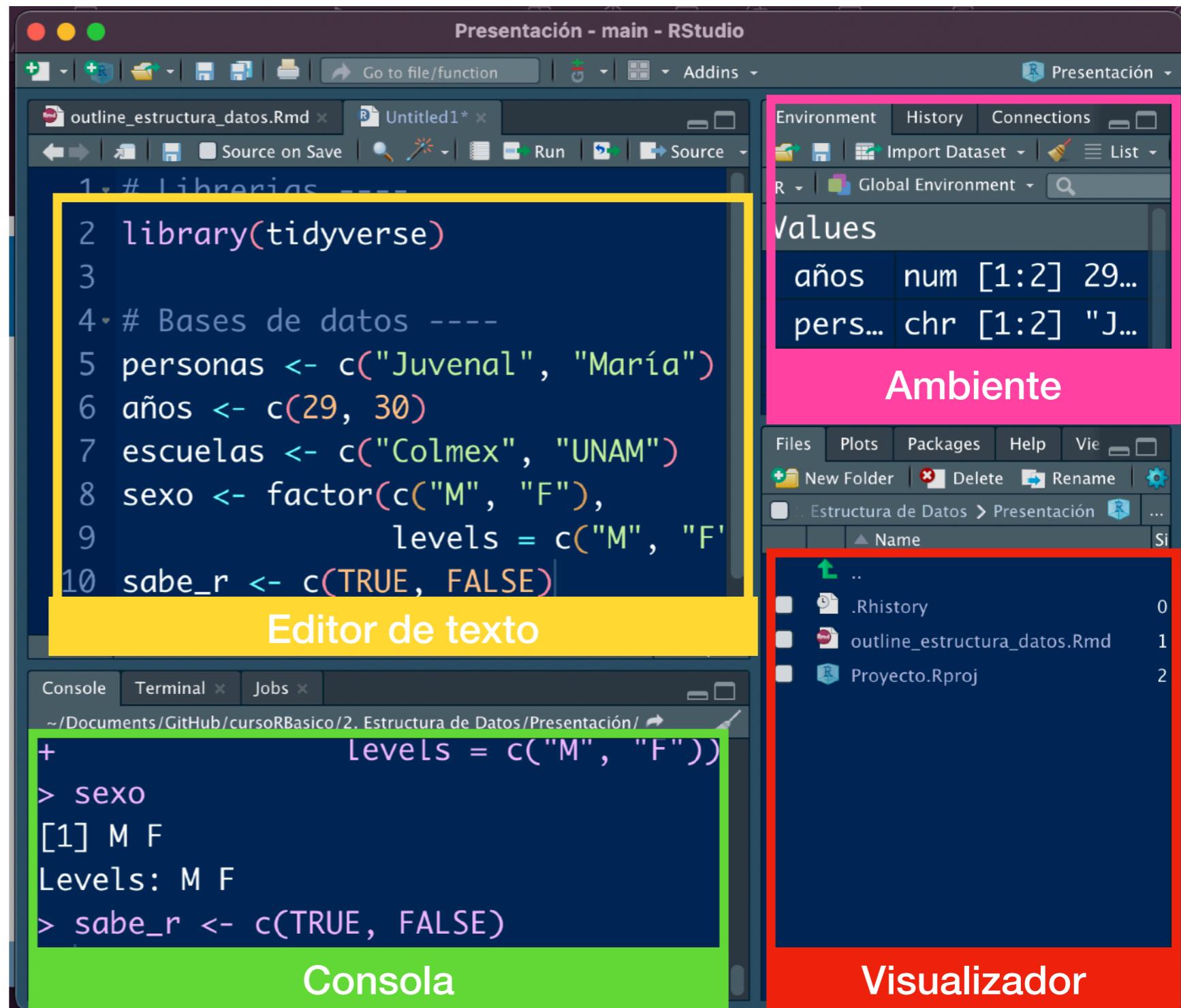
Values	Type	Length	Content
años	num	[1:2]	29...
pers...	chr	[1:2]	"J...
sabe...	logi	[1:2]	T...

# Ventanas

## Visualizador

Sección para visualizar cosas:

- 1) Archivos ubicados en nuestro directorio de trabajo
- 2) Gráficas estáticas generadas con ggplot2 o RBase.
- 3) Las librerías instaladas en nuestro RStudio.
- 4) Las visualizaciones web generadas con R.



# Personalización



Para personalizar RStudio  
vamos a  
Tools > Global Options

Podemos configurar:

- como se visualiza el código,
- los colores de las ventanas,
- el tamaño y fuente de las letras,
- el espacio a ocupar del código,
- las cuentas para publicar resultados, etc.

The screenshot shows the 'Global Options' dialog box in R Studio. The left sidebar lists various categories: General (selected), Code, Console, Appearance, Pane Layout, Packages, R Markdown, Sweave, Spelling, Git/SVN, Publishing, Terminal, Accessibility, and Python. The main panel has tabs for Basic, Graphics, and Advanced, with Basic selected. Under the R Sessions section, the default working directory is set to ~/Library/Mobile Documents/com~apple~CloudDocs, with a 'Browse...' button. Under Workspace, there are checkboxes for restoring .RData files and saving workspaces. The History section includes options for saving history and removing duplicates. The Other section contains checkboxes for wrap-around navigation, update notifications, and crash reporting. At the bottom are OK, Cancel, and Apply buttons.



# Introducción al código de R

# Objetos y funciones



- En R, todo lo que existe es un objeto.
- En R, todo lo que ocurre es una función.



# Objetos y funciones

Los **objetos** son el lugar de la memoria en el cual vamos a guardar información.

Podemos crear nuestros **objetos** o podemos tomar **objetos** hechos por alguna librería.

Los **objetos** son sujetos de ser afectados por las **funciones**.

Las **funciones** son las acciones que le vamos a aplicar a un objeto para obtener un resultado, el cual va a ser un objeto.

Los **objetos** (y las **funciones**) se guardan en el ambiente. El ambiente es el lugar donde se almacenan los **objetos** de la sesión.



# Objetos y funciones

Los **objetos** son el lugar de la memoria en el cual vamos a guardar información.

Podemos crear nuestros **objetos** o podemos tomar **objetos** hechos por alguna librería.

Los **objetos** son sujetos de ser afectados por las **funciones**.

Las **funciones** son las acciones que le vamos a aplicar a un objeto para obtener un resultado, el cual va a ser un objeto.

Los **objetos** (y las **funciones**) se guardan en el ambiente. El ambiente es el lugar donde se almacenan los **objetos** de la sesión.



# Objetos

Para guardar un **objeto**, utilizamos el operador flechita (`<-`) o el operador igual (`=`).

Si no utilizamos estos operadores, no estamos guardando nada en memoria y, por lo tanto, no lo podremos usar más adelante en nuestro proceso de trabajo.

```
nombres <- c("Joaquín", "María")
sabe.r <- c(TRUE, FALSE)
edad <- c(29, 30)
numero_al_azar <- runif(n = 2, min = 0, max = 10)
datos <- tibble(nombres,
                 sabe.r,
                 edad,
                 numero_al_azar)
```



# Objetos

La sintaxis para guardar un objeto es:

*nombre\_objeto <- contenido\_del\_objeto*

El nombre del objeto puede ser cualquiera, tratando de respetar ciertas reglas, como no usar palabras reservadas (como **TRUE** o **FALSE**), no empezar con símbolos (\_ / !, etc.), no empezar con números (1-9) y, de preferencia, no usar símbolos especiales (ñ, 漢字, 'संस्कृतम्', etc.).

```
nombres <- c("Joaquín", "María")
sabe.r <- c(TRUE, FALSE)
edad <- c(29, 30)
numero_al_azar <- runif(n = 2, min = 0, max = 10)
datos <- tibble(nombres,
                 sabe.r,
                 edad,
                 numero_al_azar)
```



# Objetos

Como vemos en este ejemplo, también se puede guardar como objetos el resultado de funciones; en este caso, estamos guardando el resultado de la función `c()`, de la función `runif()` y de la función `tibble()`.

```
nombres <- c("Joaquín", "María")
sabe.r <- c(TRUE, FALSE)
edad <- c(29, 30)
numero_al_azar <- runif(n = 2, min = 0, max = 10)
datos <- tibble(nombres,
                 sabe.r,
                 edad,
                 numero_al_azar)
```

# Funciones



Las **funciones** son las **acciones** que vamos a realizar sobre los **objetos**. Estas pueden ya estar precargadas de las **librerías base** o pueden provenir de **librerías** descargadas de manera externa.

Las funciones se llaman con la siguiente sintaxis:

```
nombre_funcion(argumento_1 = "argumento_1",
               argumento_2 = "argumento_2",
               ... ,
               argumento_n = "argumento_n")
```

Los **argumentos** son como palanquitas a las que hay que moverle para que las funciones funcionen de manera adecuada.

# Funciones



Las **funciones** tienen nombre y apellido. Muchas veces las vamos a encontrar en la literatura de la manera que sigue:

```
dplyr::filter()  
sf::read_sf()  
base::sum()  
leaflet::leaflet()
```

En este caso, lo que va a la izquierda de los ***dos-dos puntos*** es la **librería o paquetería** (apellido) de la cual provienen, mientras que lo que va al lado derecho es el nombre de la función.

Si llamamos la **librería** de origen, meter el apellido ya no va a ser necesario. :3. Si no se le pone apellido, es que proviene de **base**

# Librerías

## ★ Definición

Las *librerías* son un conjunto de objetos y funciones programados por terceros, que podemos instalar en nuestra sesión de R para potenciar las funciones que podemos realizar.

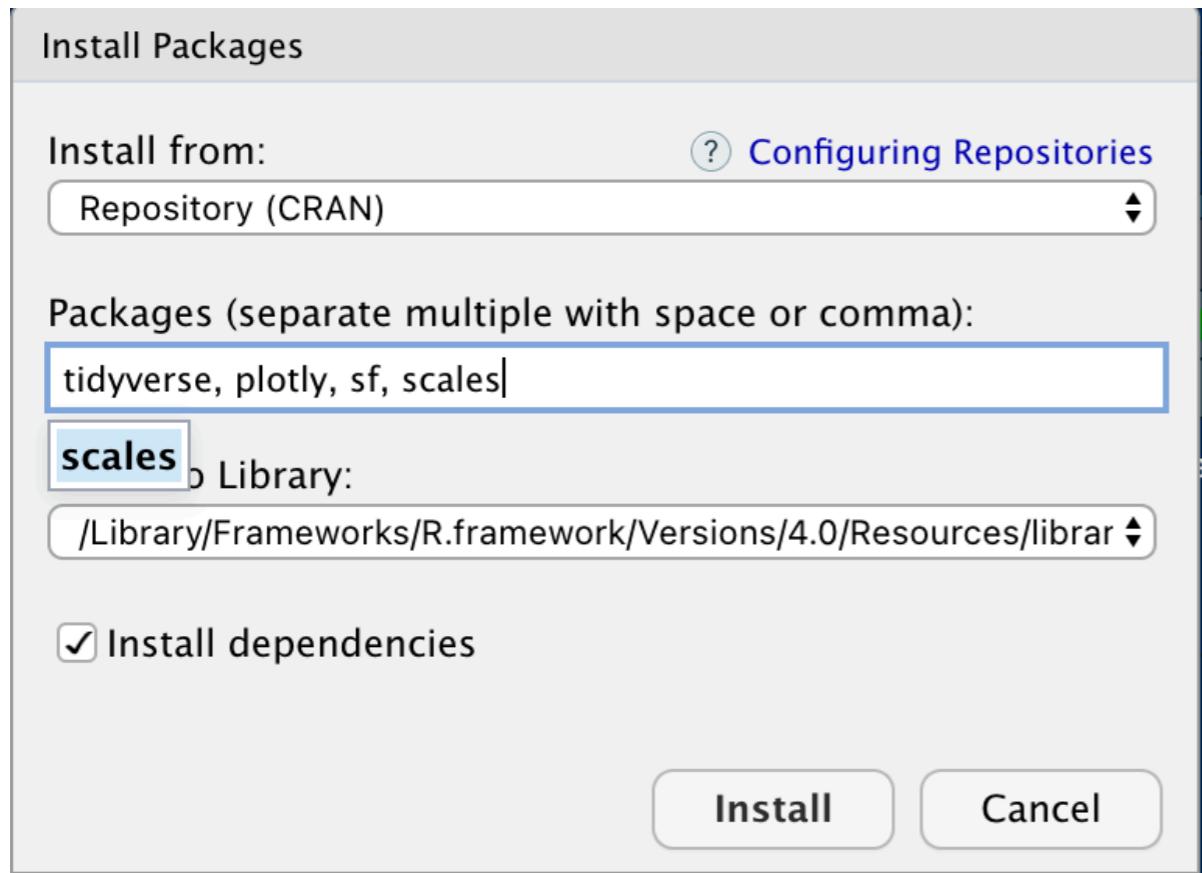
## Instalación

Las instalamos en la sección del **Visualizador**, en la sección de “*Packages*” > *Install* y escribimos la librería que queremos instalar. Otra opción es con la función *install.packages(“librería\_a\_instalar”)*

# Instalar librerías



## Opción 1



## Opción 2

```
> install.packages(c("tidyverse",
  "plotly", "sf", "scales"))
```

# Instalar librerías



★ Instalar librerías



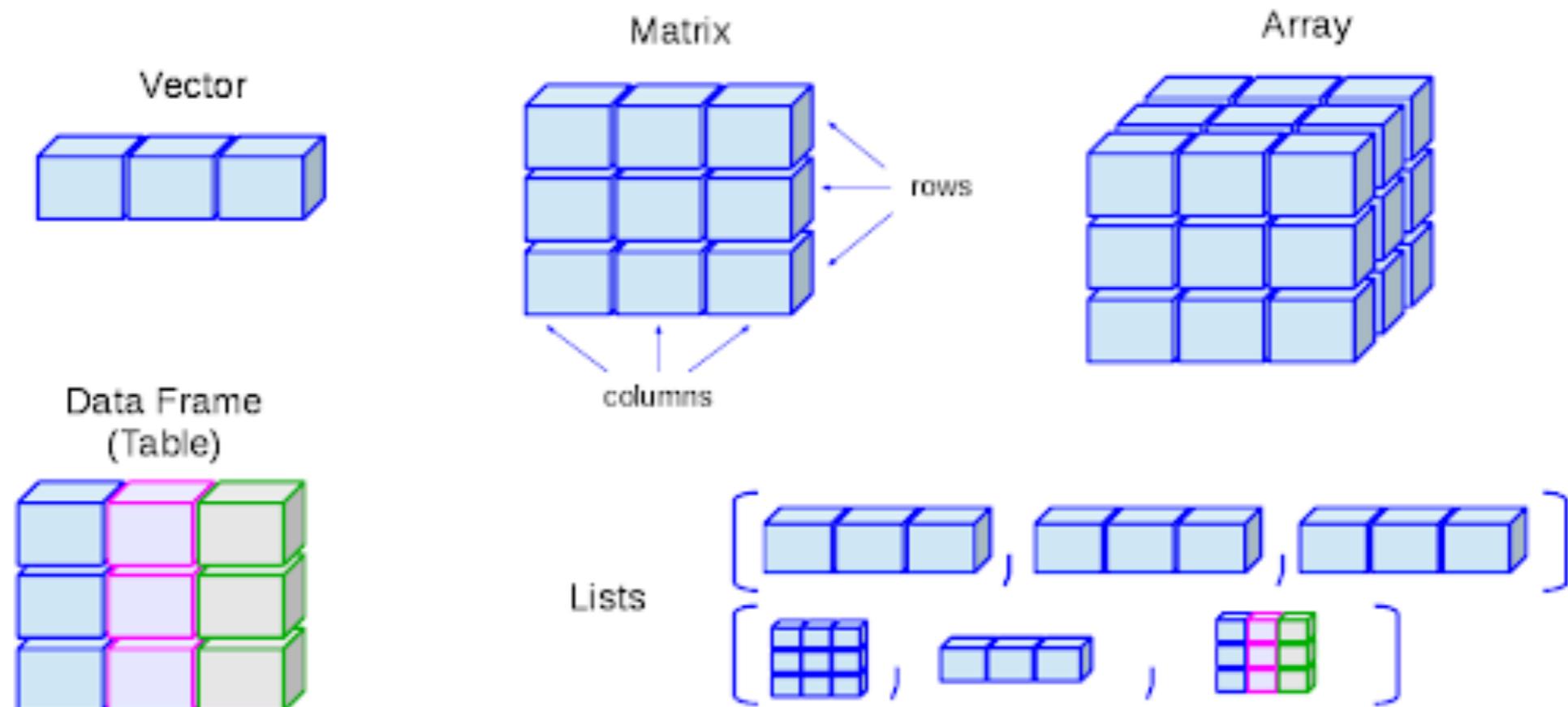
Llamar librerías





# Estructuras de datos

# Estructura de Datos

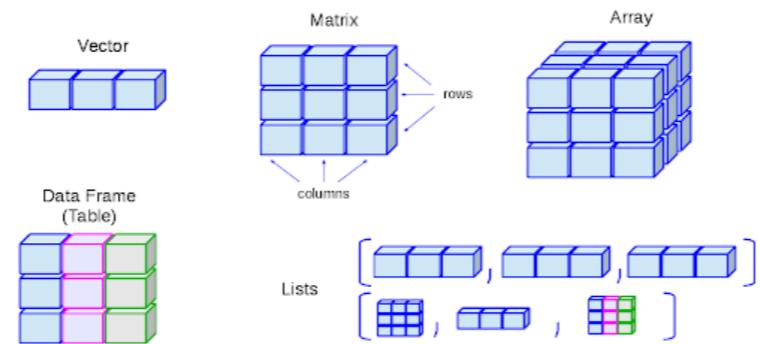


# Estructura de Datos



Los datos suelen agruparse en estructuras básicas de datos:

1. **Escalares**, vectores de 1x1
2. **Vectores**, arreglos de nx1 (n renglones y 1 columna)
3. **Matrices**, arreglos cuadrados de n x n (n renglones y n columnas). Un único tipo de dato.
4. **Arrays**, matrices de matrices multidimensionales n x n x ... x n.
5. **DataFrames**, arreglos cuadrados de n x n (n renglones y n columnas). Múltiples tipos de dato.
6. **Listas**. Contenedores de cualquier cosa.



# Estructura de Datos



Se pueden generar o extraer de muchas formas. Las mas sencillas son las siguientes:

**Vectores:** Usamos la función combine, **c()**.

```
vector_1 <- c(1,2,3,4,5)
```

**Matrizes:** Usamos la función **matrix()** y le metemos como ingredientes para esa matriz vectores y argumentos como *ncol* o *nrow* para definir las dimensiones.

```
vector_1 <- c(1,2,3,4,5)
vector_2 <- 10:14
mtx <- matrix(c(vector_1, vector_2), ncol = 2)
```

# Estructura de Datos



Se pueden generar o extraer de muchas formas. Las mas sencillas son las siguientes:

**Dataframes:** Podemos construir dataframes con la función **data.frame()** o con la función **tibble::tibble()**, y le pasamos como ingredientes vectores del mismo tamaño para que sirvan de columnas.

```
df <- data.frame(vector_1,  
                  vector_2,  
                  letras = c("a", "b", "c", "d", "e"))
```

*Este es el tipo de estructura más usado; cuando leemos datos de excel, por ejemplo, nos va a construir automáticamente un DF.*

# Estructura de Datos



Se pueden generar o extraer de muchas formas. Las mas sencillas son las siguientes:

**Listas:** Podemos construir listas con la función **list()**. Como acá le podemos meter dentro la estructura de datos que sea, le metemos como ingredientes lo que sea.

```
lista <- list(df, # Le metemos un df  
             mtx, # le metemos una matriz  
             vector_2, # Le metemos un vector  
             vector_1, # Le metemos otro vector  
             mtcars, # Le metemos otro df pre-construido  
             list(df, vector_2), # Le podemos meter listas  
             sum() # Le podemos meter funciones  
           )
```



# **Tipos de datos (vectores atómicos)**

# Vectores atómicos



Los tipos de datos que podemos almacenar dentro de nuestras estructuras de datos son los siguientes:

## Básicos:

- **Character**, para almacenar texto.
- **Numeric**, para almacenar números.
- **Logical**, para almacenar valores lógicos (TRUE/FALSE o T/F).

## Compuestos:

- **Factor**, para almacenar datos categóricos.
- **Date**, para almacenar fechas.

# Vectores atómicos



- **Character**, para almacenar texto.

```
character <- c("Hola",
             "Adios",
             "Buenas tardes",
             "Buenas Noches")
```

```
character_2 <- c("1", "2",
                  "tres", "cuatro", "5")
```

```
macondo <- c("Muchos años después, frente al pelotón de
fusilamiento, el coronel Aureliano Buendía había de recordar
aquella tarde remota en que su padre lo llevó a conocer el hielo.
Macondo era entonces una aldea de veinte casas de barro y
cañabrava construidas a la orilla de un río de aguas diáfanas
que se precipitaban por un lecho de piedras pulidas, blancas y
enormes como huevos prehistóricos.")
```



# Vectores atómicos

- **Numeric**, para almacenar números.

```
# Numéricos
enteros <- c(1L, 2L, 3L, 4L) # Enteros
reales <- c(1,2,3,4) # reales
reales_2 <- c(1e6, 3.1416, 2) # reales con notación científica
```

- # Operaciones ----

```
# Suma
c(1,3,4,5) + 5 # [1] 6 8 9 10
# Resta
c(1,3,4,5) - 5 # [1] -4 -2 -1 0
# Multiplicación:
c(1,3,4,5) * 5 # [1] 5 15 20 25
# División:
c(1,3,4,5) / 2 # [1] 0.5 1.5 2.0 2.5
# Exponenciación:
c(1,3,4,5) ^ 2 # [1] 1 9 16 25
```

```
# Secuencias ----
1:100 # Numeros del uno al 100, de uno en uno
seq(from = 0, to = 100, by = 2)
# Numeros del cero al 100, de dos en dos
```



# Vectores atómicos

- **Logical**, para almacenar valores lógicos (TRUE/FALSE o T/F).

```
# VALORES LÓGICOS
logical <- c(TRUE, TRUE, TRUE, TRUE, FALSE)
logical_2 <- c(T, T, T, T, F)
# Son lo mismo uno y otro
```

```
# Como resultado de comparaciones:
c(1,2,3,4,5) < 3 # [1] TRUE TRUE FALSE FALSE FALSE
c("a", "b", "c") == "c" # [1] FALSE FALSE TRUE
c("Morelos", "Zacatecas", "Aguascalientes", "CDMX") %in%
  c("Morelos", "Zacatecas") # [1] TRUE TRUE FALSE FALSE
```



# Vectores atómicos

- **Factor**, para almacenar datos categóricos.

```
# PASO 1: GENERO UN VECTOR NORMAL
partidos_ganadores <- c("PAN",
                         "PRI",
                         "MORENA",
                         "MORENA",
                         "PRI",
                         "MORENA",
                         "MORENA",
                         "MORENA")

# PASO 2: LO TRANSFORMO A CATEGÓRICO.
partidos_con_categorias <- factor(partidos_ganadores,
                                      levels = c("PAN", "PRI", "MORENA"),
                                      ordered = F)

> partidos_con_categorias
[1] PAN      PRI      MORENA MORENA PRI      MORENA MORENA MORENA
Levels: PAN PRI MORENA
```



# Vectores atómicos

- **Factor**, para almacenar datos categóricos.

```
# Generamos el vector de texto
tamanios_playera <-
  c("mediano", "grande", "chico", "chico",
    "mediano", "grande", "grande", "mediano",
    "grande", "chico", "chico", "mediano")

# Sobreescribimos
tamanios_playera <- factor(tamanios_playera,
  levels = c("grande", "mediano", "chico"),
  ordered = T)

tamanios_playera
```

```
[1] mediano grande chico chico mediano grande grande mediano
[9] grande chico chico mediano
Levels: grande < mediano < chico
```



# Vectores atómicos

- **Date**, para almacenar fechas.

```
fecha <- Sys.Date()
fecha

fechas_2 <- as.Date(c("2020-02-02",
                      "2019-03-04",
                      "1991-07-08"),
                      format = "%Y-%m-%d")
fechas_2
```

```
> fecha <- Sys.Date()
> fecha
[1] "2021-06-23"
> fechas_2 <- as.Date(c("2020-02-02",
+                         "2019-03-04",
+                         "1991-07-08"),
+                     format = "%Y-%m-%d")
> fechas_2
[1] "2020-02-02" "2019-03-04" "1991-07-08"
```



# Función class(x)

La función **class(x)** es la función con la cual podemos saber qué tipo de dato tiene el objeto “x”.

```
# Función class(x)
class(fechas_2) # [1] "Date"
class(partidos_con_categorias) # [1] "factor"
class(tamanios_playera) # [1] "ordered" "factor"
class(logical_2) # [1] "logical"
class(vector_1) # [1] "numeric"
class(character_2) # [1] "character"
```



# Función length(x)

La función **length(x)** nos permite saber el tamaño (# de elementos) que contiene un vector.

```
# Función length(x)
length(fechas_2) # [1] 3
length(partidos_con_categorias) # 8
length(tamanios_playera) # 12
length(logical_2) # [1] 5
length(vector_1) # 5
length(character_2) # 5
```



# Coerción.

Supongamos que mezclamos dos o mas tipos de datos:

```
# Tres tipos de datos combinados  
c("Hola", TRUE, 1)
```

Como R no entiende que tipo de dato es el vector, y como necesita que sea de un solo tipo, lo va a transformar al tipo más simple: en este caso, todo lo va a hacer (coercionar) a TEXTO:

```
> c("Hola", TRUE, 1)  
[1] "Hola" "TRUE" "1"
```

Tenemos que tener un solo tipo de datos; si no se cumple esto, R va a transformarlos por nosotros.

# Gracias :3