



Tecnológico
de Monterrey

02. Repaso de VibeCoding

Módulo 5. Inteligencia Artificial

Jorge
Juvenal
Campos Ferreira

juvenal.campos@tec.mx

Programa de la clase

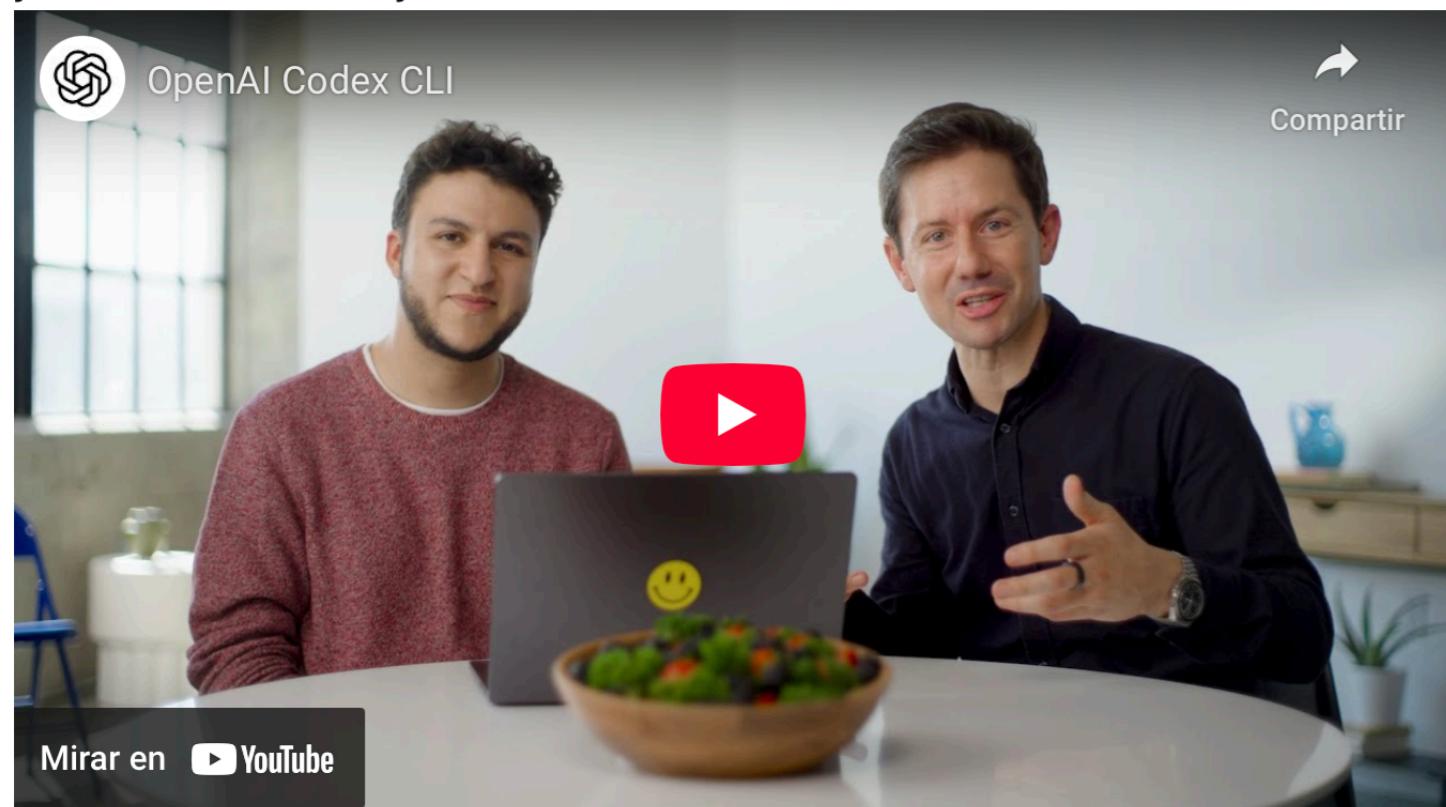
- Vibe Coding y sus peligros
- Configuración de Claude Code
- Glosario de IA
- Introducción a R y Rstudio
- Mapa de funcionalidades de Rstudio
- ¿Por qué R?
- Funciones más comunes
- Proyectos y Directorio de trabajo

OpenAI Codex CLI – Getting Started

Updated: hace 3 días

Overview

OpenAI Codex CLI is an open-source command-line tool that brings the power of our latest reasoning models directly to your terminal. It acts as a lightweight coding agent that can **read, modify, and run code on your local machine** to help you build features faster, squash bugs, and understand unfamiliar code. Because the CLI runs locally, your source code never leaves your environment unless you choose to share it.



For more details about Codex, please check out the Github repo directly -

<https://github.com/openai/codex>.

Les dije que seguramente
OpenAI iba a sacar una
versión propia y lo hizo unos
días después.

Verificación

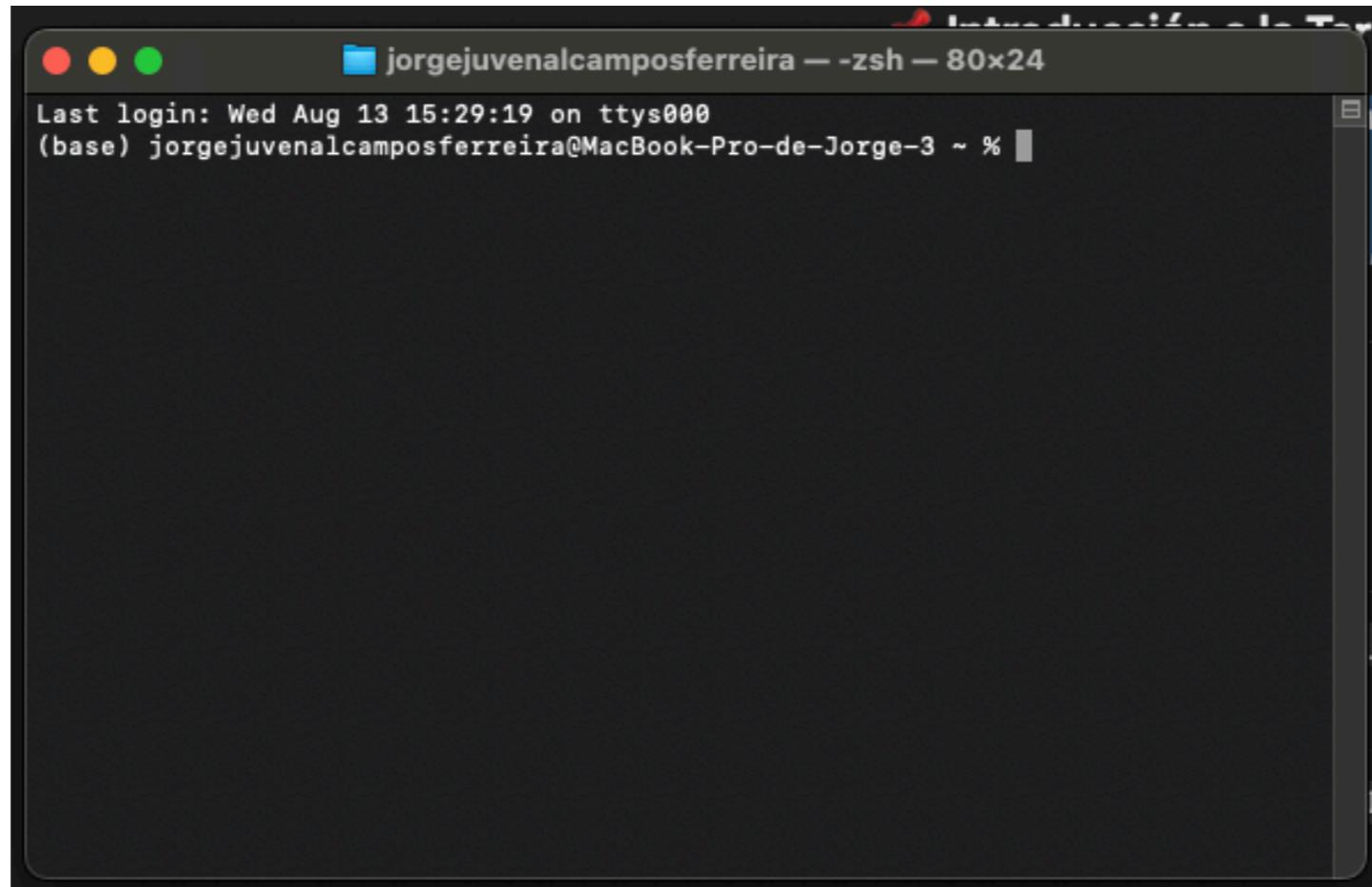
- ¿Quienes lograron instalar alguna herramienta para codificar con LLMs? ¿Cuál ha sido su experiencia?
- ¿Quienes no lo lograron? ¿Lo **intentaron**? ¿Qué **dificultades** se presentaron?



Terminal

La **terminal** (o línea de comandos) es una herramienta para **interactuar directamente con el sistema operativo** mediante texto, sin usar ventanas o íconos.

Permite navegar por carpetas, ejecutar programas y administrar archivos con comandos.



Comandos básicos

Comando	Mac / Linux	Windows (PowerShell / CMD)	Descripción
cd	cd nombre_carpeta	cd nombre_carpeta	Cambia de directorio (moverse entre carpetas).
ls	ls o ls -l	dir	Lista los archivos y carpetas en el directorio actual.
pwd	pwd	cd (sin parámetros)	Muestra la ruta completa de la carpeta actual.
clear	clear	cls	Limpia la pantalla de la terminal.
mkdir	mkdir nombre_carpeta	mkdir nombre_carpeta	Crea una nueva carpeta.
rm	rm archivo.txt	del archivo.txt	Elimina un archivo.

Ejercicio práctico

Instale Claude Code o Gemini en su computadora
Ejecutelo y corra un prompt que genere un archivo para generar una
gráfica básica en ggplot.
Ejecute el código y verifique que funcione.

Vibe Coding



Es una **técnica de programación** dependiente de IA en la que una persona describe un problema en unas pocas oraciones como instrucciones para un LLM.

El LLM genera software, cambiando el rol del programador de la codificación manual a la guía, prueba y refinamiento del código fuente generado por IA

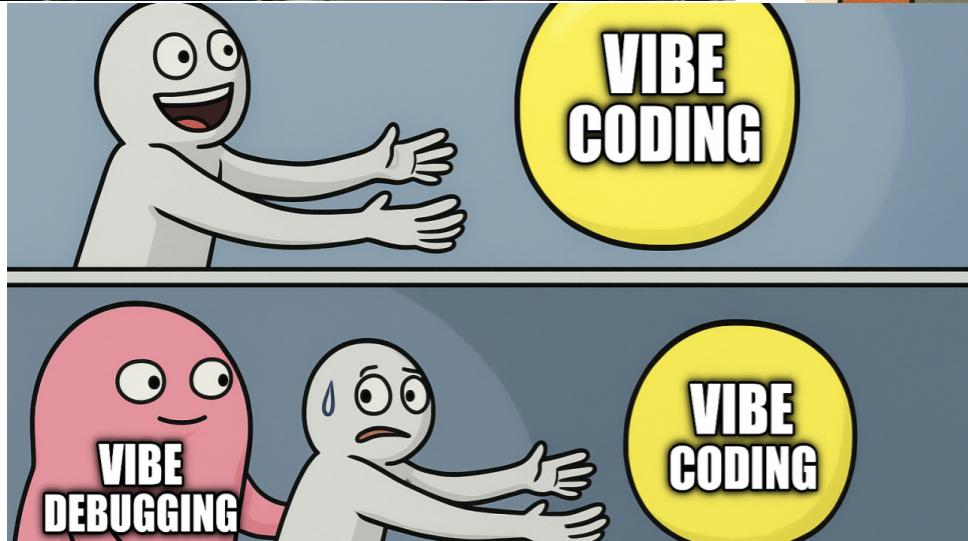
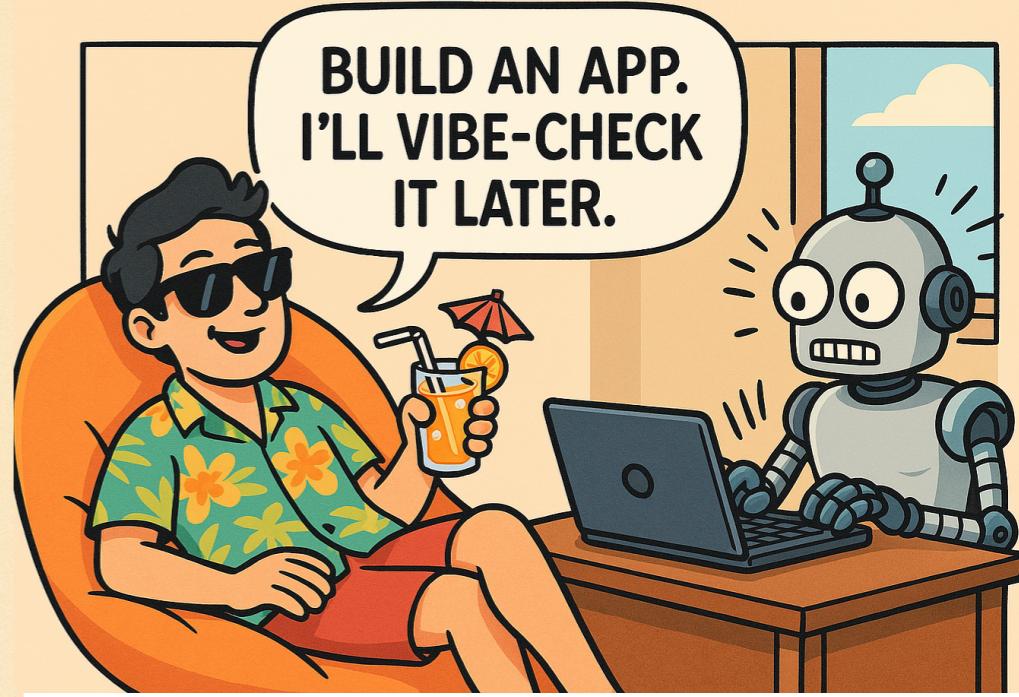
Vibe Coding



El **vibe coding** es un término acuñado por Andrej Karpathy (cofundador de OpenAI y ex-líder de IA en Tesla) en febrero de 2025. Como él mismo lo describió: "**There's a new kind of coding I call 'vibe coding', where you fully give in to the vibes, embrace exponentials, and forget that the code even exists**"



Vibe Coding





Tecnológico
de Monterrey

02. Introducción a RStudio

Módulo 5. Inteligencia Artificial

Jorge
Juvenal
Campos Ferreira

juvenal.campos@tec.mx

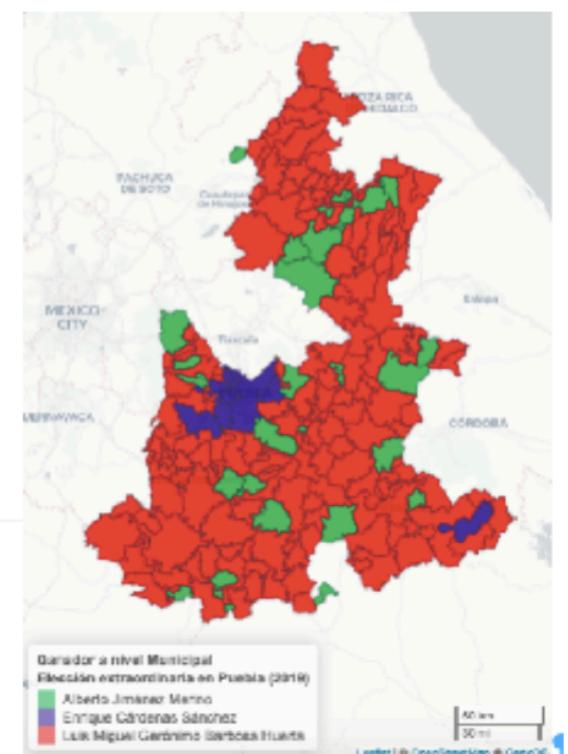
Tareas que se hacen con R

Tareas que se hacen con datos

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 )
```

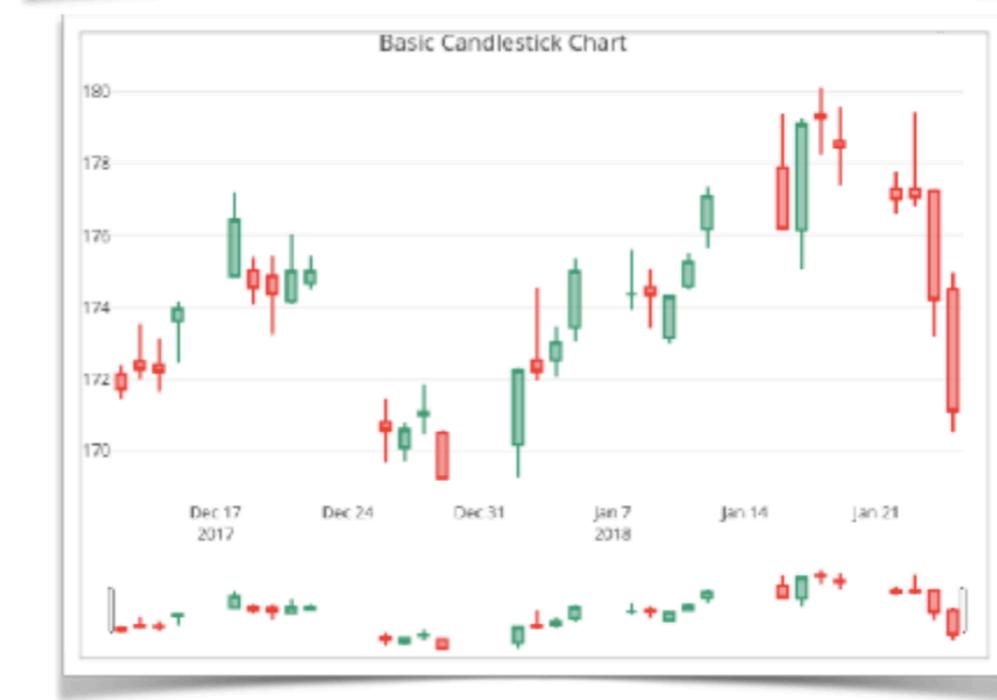
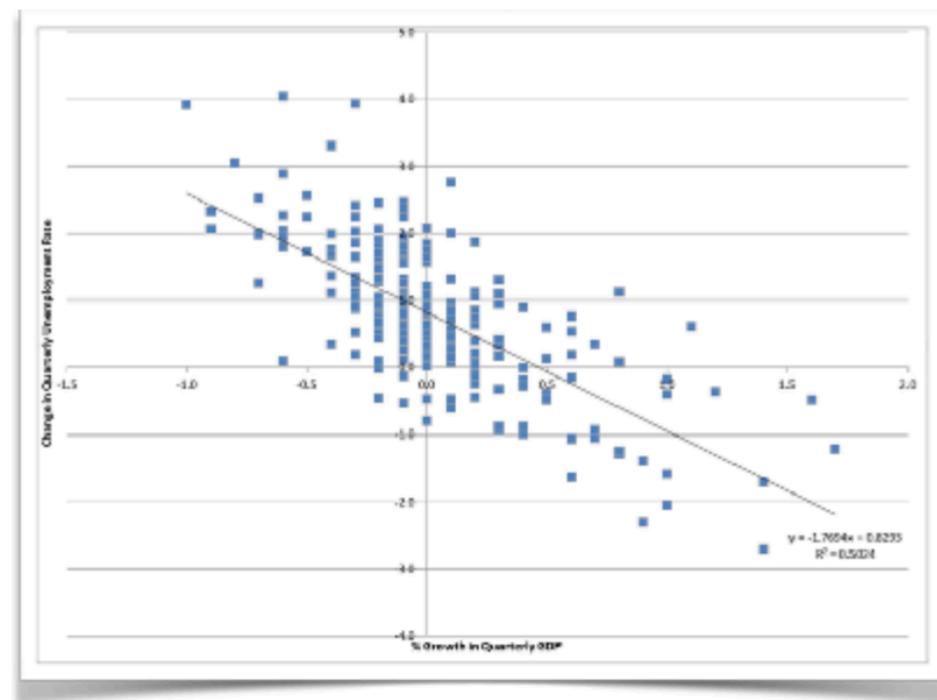
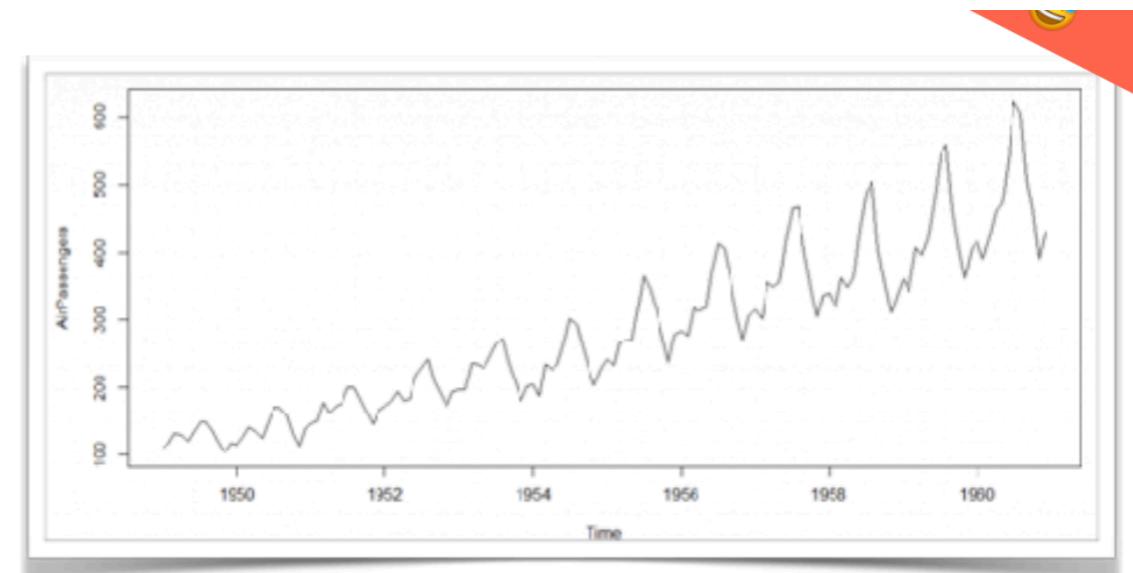


Tareas que se hacen con datos

2. Análisis estadístico y econometría.

library(base)

library(MASS)



Tareas que se hacen con datos

3. Machine Learning y Deep Learning.

`library(e1071)`

`library(tensorflow)`

`library(caret)`

`library(rpart)`

etc.

Classification



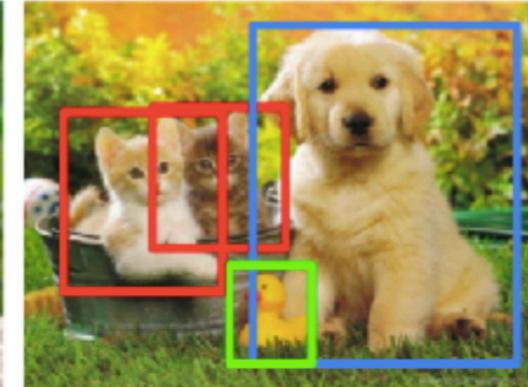
CAT

Classification + Localization



CAT

Object Detection



CAT, DOG, DUCK

Instance Segmentation



CAT, DOG, D

Single object

Multiple objects



Tareas que se hacen con datos

4. Análisis de texto.

library(tm)

```
library(stringr)
```



Nube de palabras.

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



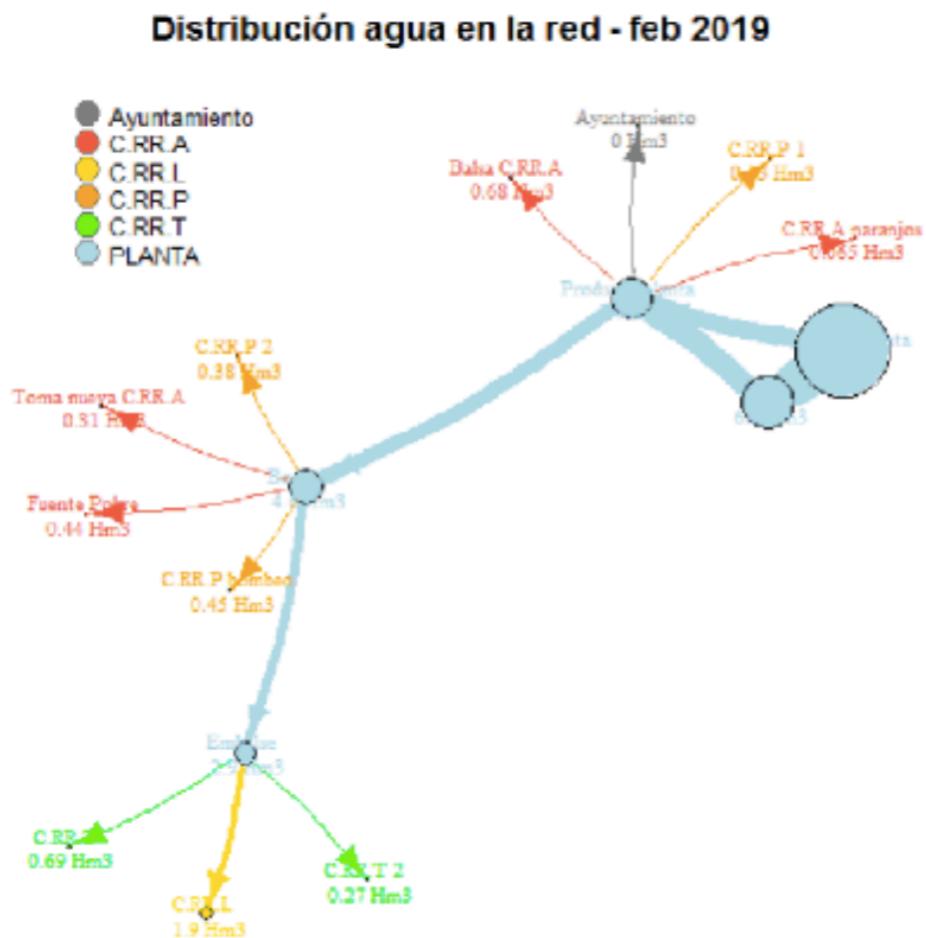
Nube de palabras.

Plan Nacional de Desarrollo. 2019.

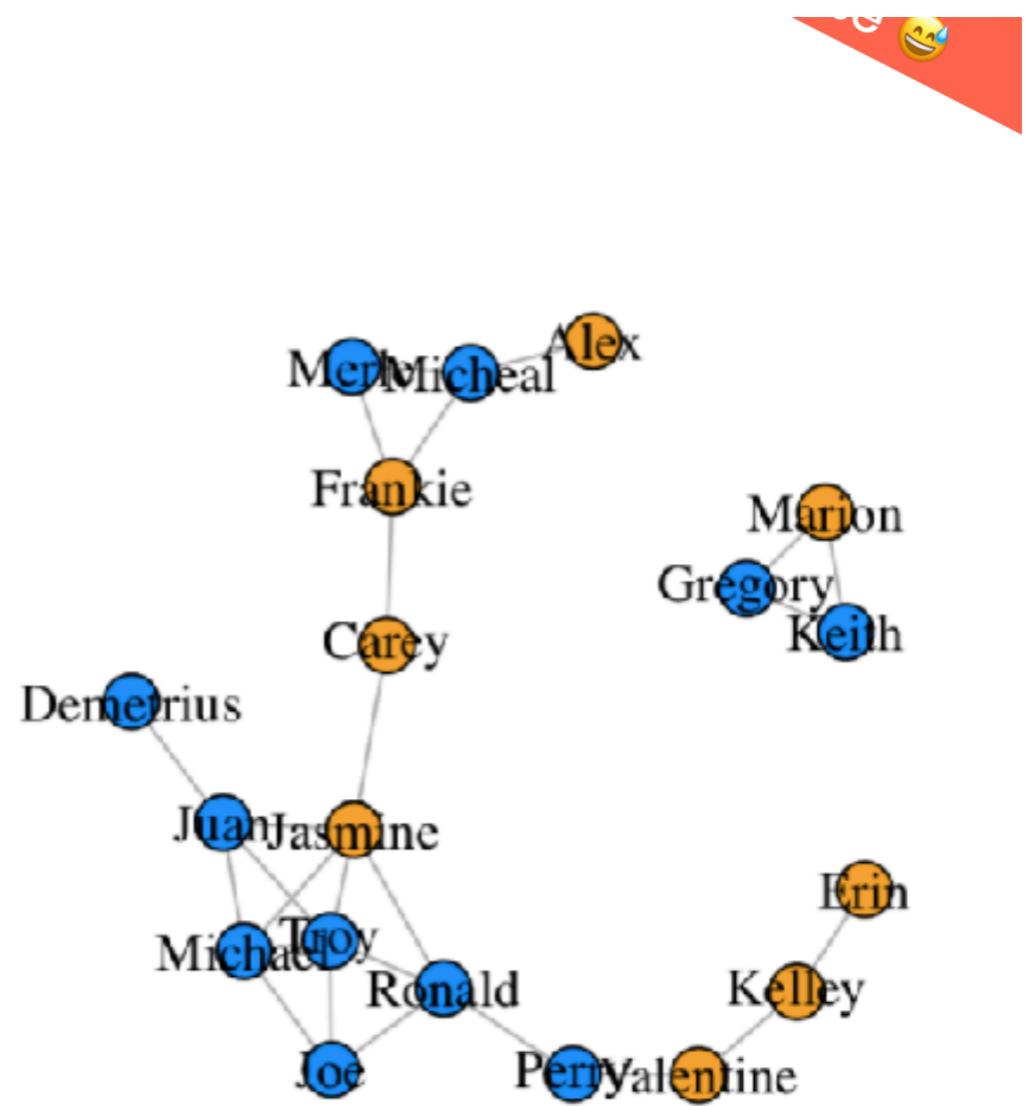
Tareas que se hacen con datos

5. Análisis de redes.

`library(igraph)`



Red de distribución de Agua



Red social de amigos en una prepa

Tareas que se hacen con datos



6. Visualización de datos.

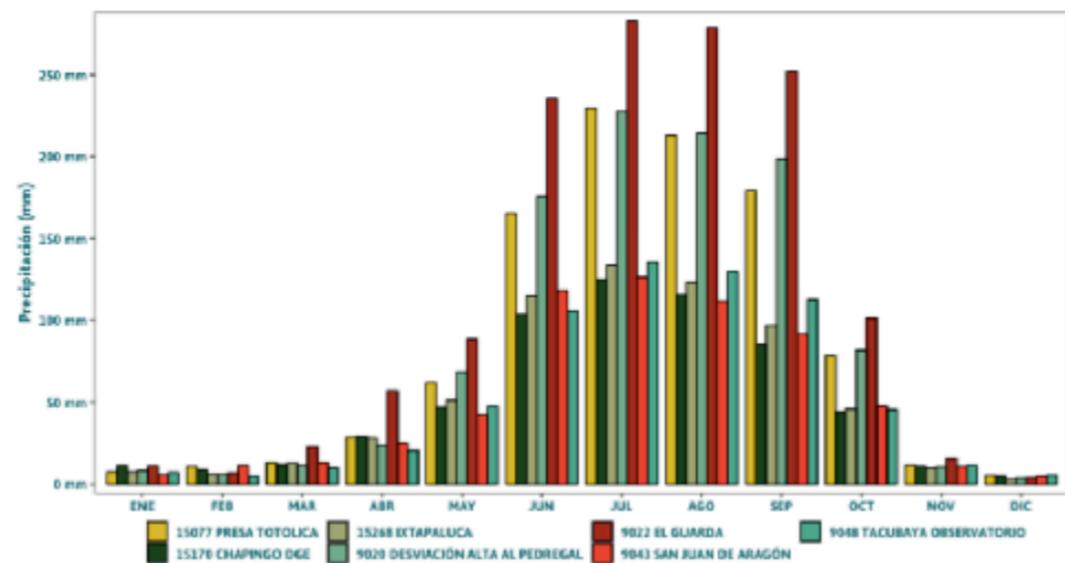
library(ggplot2)

library(plotly)

library(leaflet)

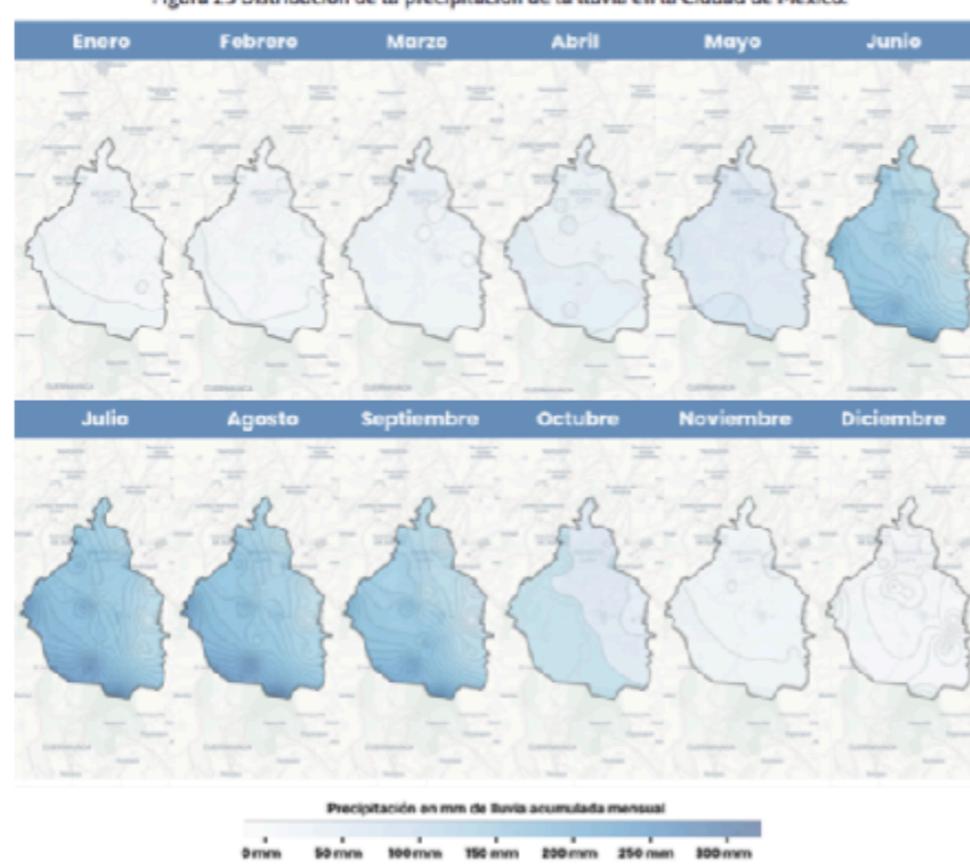
library(htmlwidgets)

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.

Figura 25 Distribución de la precipitación de la lluvia en la Ciudad de México.



Fuente: Elaboración Propia con datos del Atlas de Riesgo de la Ciudad de México, SGIRPC, 2019.

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

Tareas que se hacen con datos

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

library(rvest)

library(xml)

Ejemplo: Extraer precios e información de vuelos desde Google Flights

The screenshot shows the Google Flights interface. On the left, there's a sidebar with icons for Trips, Explore, and Hotels. The main area displays "Best departing flights" with four results:

Airline	Flight Details	Duration	Stops	Price
United - ANA	06:05 - 14:35*	18 h 30 m	1 stop 2 h 35 m SFO	MX\$20,089 round trip
United, ANA	07:30 - 15:20*	17 h 50 m	1 stop 1 h 35 m IAH	MX\$20,089 round trip
ANA	02:20 - 06:45*	14 h 25 m	Non-stop MEX-NRT	MX\$21,889 round trip
Aeromexico - UA	01:30 - 06:20*	14 h 50 m	Non-stop MEX-NRT	MX\$31,471 round trip

Below this, a note says "Prices are currently typical for your trip." and there's a "Details" button. Further down, under "Other departing flights", there are two additional entries:

Airline	Flight Details	Duration	Stops	Price
United, ANA	17:30 - 14:20 ?	30 h 50 m	2 stops ▲ IAU, ORB	MX\$20,089 round trip
United, ANA	17:30 - 14:20 ?	30 h 50 m	2 stops ▲ IAU, ORB	MX\$20,089 round trip

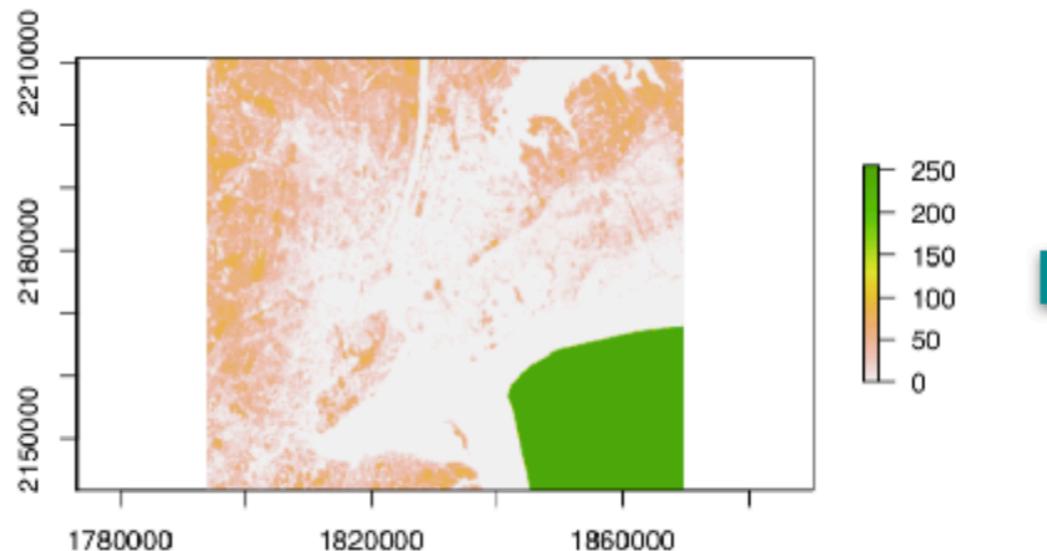
https://www.google.com/flights?lite=0#flt=MEX.NRT.2019-10-21*NRT.MEX.2019-11-05;c:MXN;e:1;sd:1;t:f

Tareas que se hacen con datos

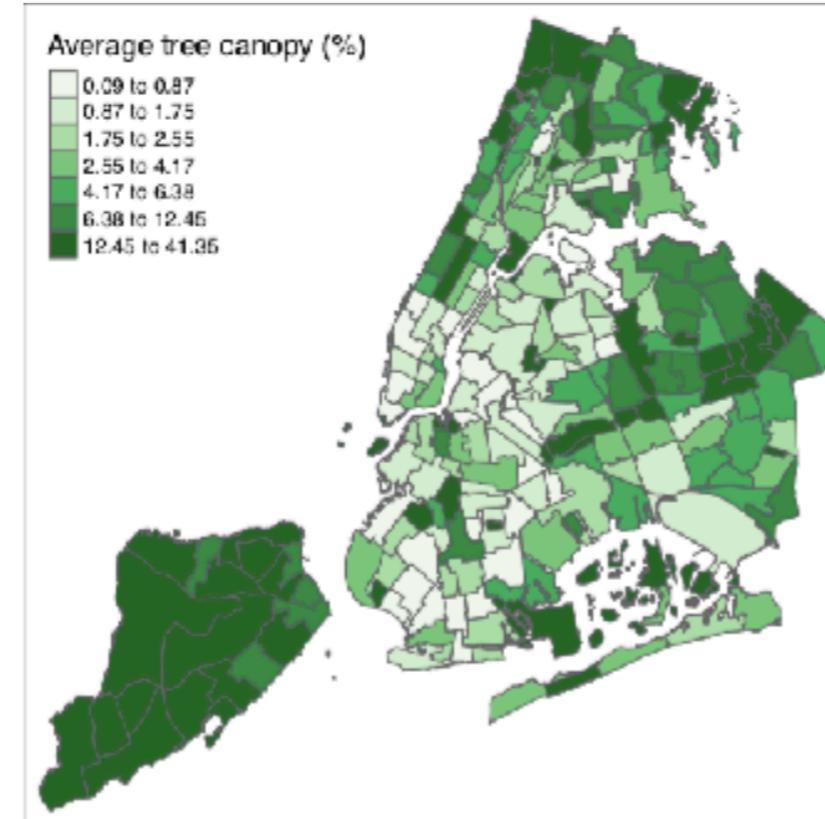
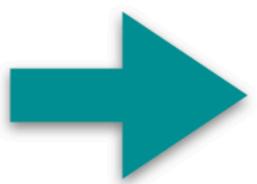
8. Análisis Geoespacial.

`library(sf)`

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



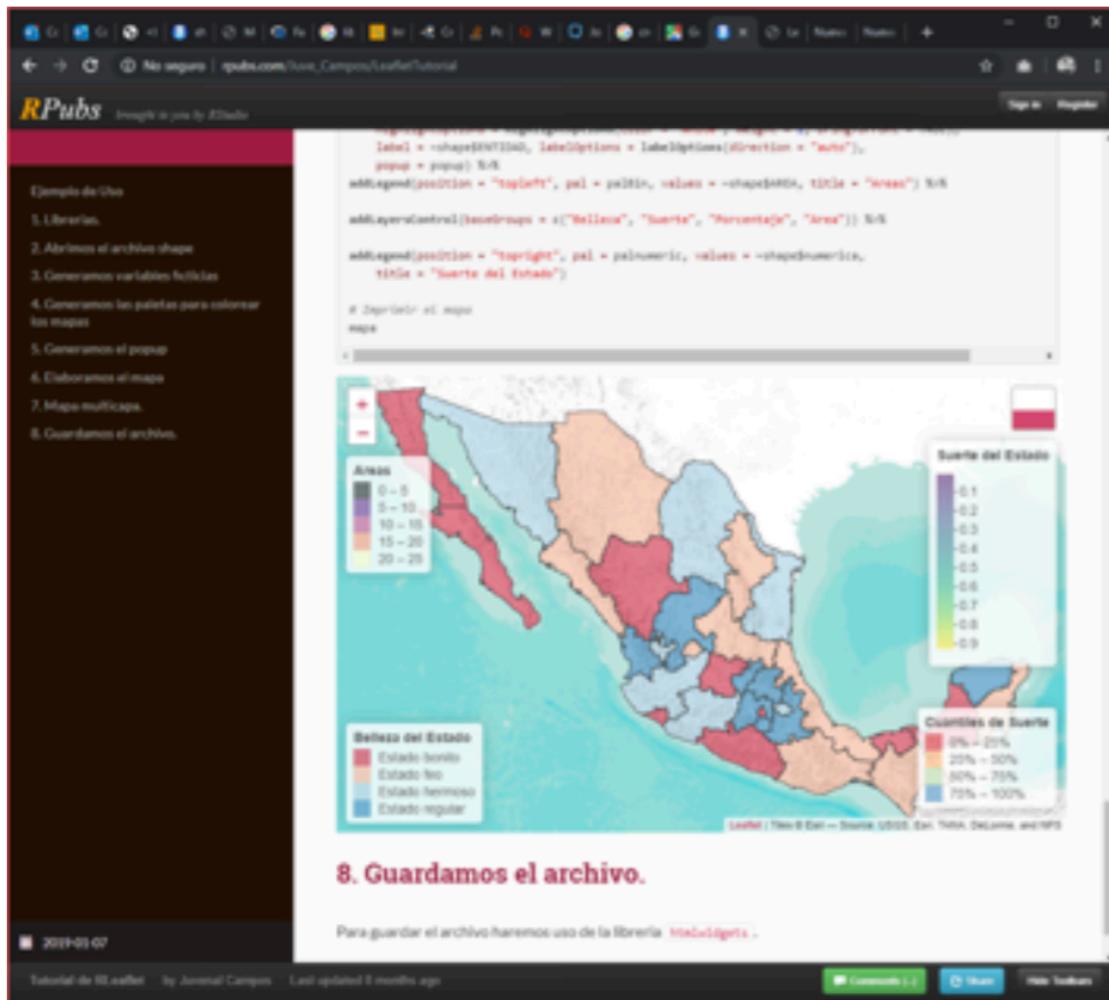
**Datos Crudos
(Raw Data)**



Datos Procesados que permiten llegar a conclusiones

Tareas que se hacen con datos

11. Páginas web y reportes (*.pdf, *.doc, diapositivas, tableros estáticos). library(markdown)



8. Guardamos el archivo.

Para guardar el archivo haremos uso de la librería [readrigner](#).

Unit 2 of the course
Juventud Campos - Based on DevelopML Classification Course
17/1/2018

Vector and raster coordinate systems

In order to perform any kind of analysis with more than one layer, you have to establish the same coordinate 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 geodata has been read in with `sf` or `raster` 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 projection.

```
#Read data
# Load the packages
library(sf)
# Linking to GDAL 3.6.1, Gdal 2.1.3, proj.4 4.9.3
library(raster)

# Loading required package: sp
root <- "/Users/juvicam/Desktop/CienciaCorta/Geospatial/Analisis_sf/R/shapefiles"
trees_sf

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

# Reading layer 'trees' from data source '/Users/juvicam/Desktop/CienciaCorta/Geospatial/Analisis_sf/R'
# with 6527 features and 7 fields
# geometry type: POINT
# dimension: XY
# bbox: -16.2500 40.4093 45.7500 46.0100
# epsg (SRID): 4326
# projection: +proj=longlat +ellps=WGS84 +no_defs
# Read in the neighborhood shapefile
neighborhoods <- st_read(paste0(root, "/neighborhoods/neighborhoods.shp"))

# Reading layer 'neighborhoods' from data source '/Users/juvicam/Desktop/CienciaCorta/Geospatial/Analisis_sf/R'
# with 128 features and 8 fields
# geometry type: POLYGON
# dimension: XY
# bbox: -16.2500 40.4093 45.7500 46.0100
# epsg (SRID): 4326
# projection: +proj=longlat +ellps=WGS84 +no_defs
# Read in the tree canopy raster
canopy_sf <- raster(paste0(root, "/canopy.tif"))
```

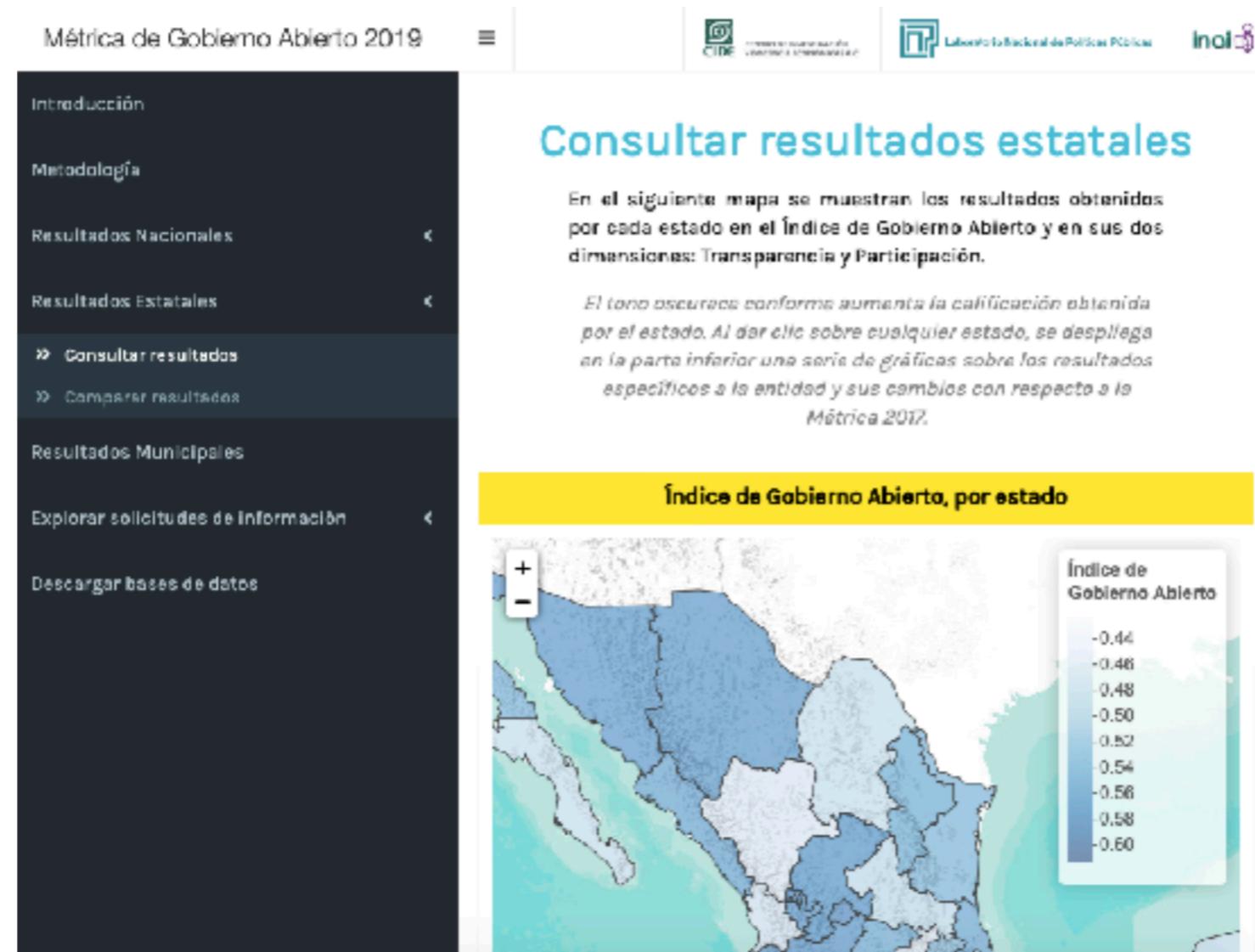
http://rpubs.com/Juve_Campos/LeafletTutorial

IATEX pdf

Tareas que se hacen con datos

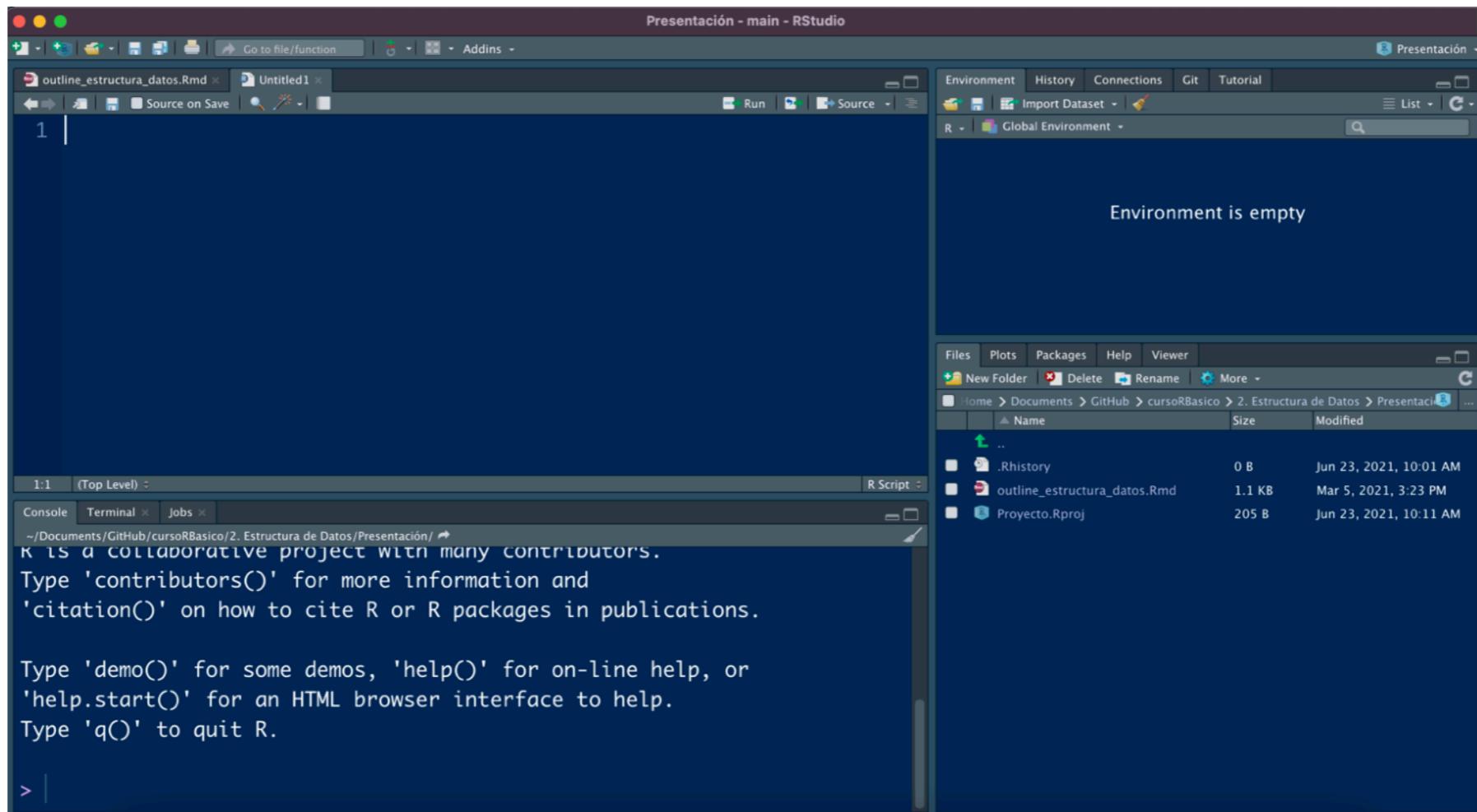
12. Web Apps.

library(shiny)



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

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: 'años' (num [1:2] 29...), 'pers...' (chr [1:2] "J..."), 'sabe...' (logi [1:2] T...), and 'Escuela' (chr [1:2] "C...").
- 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:

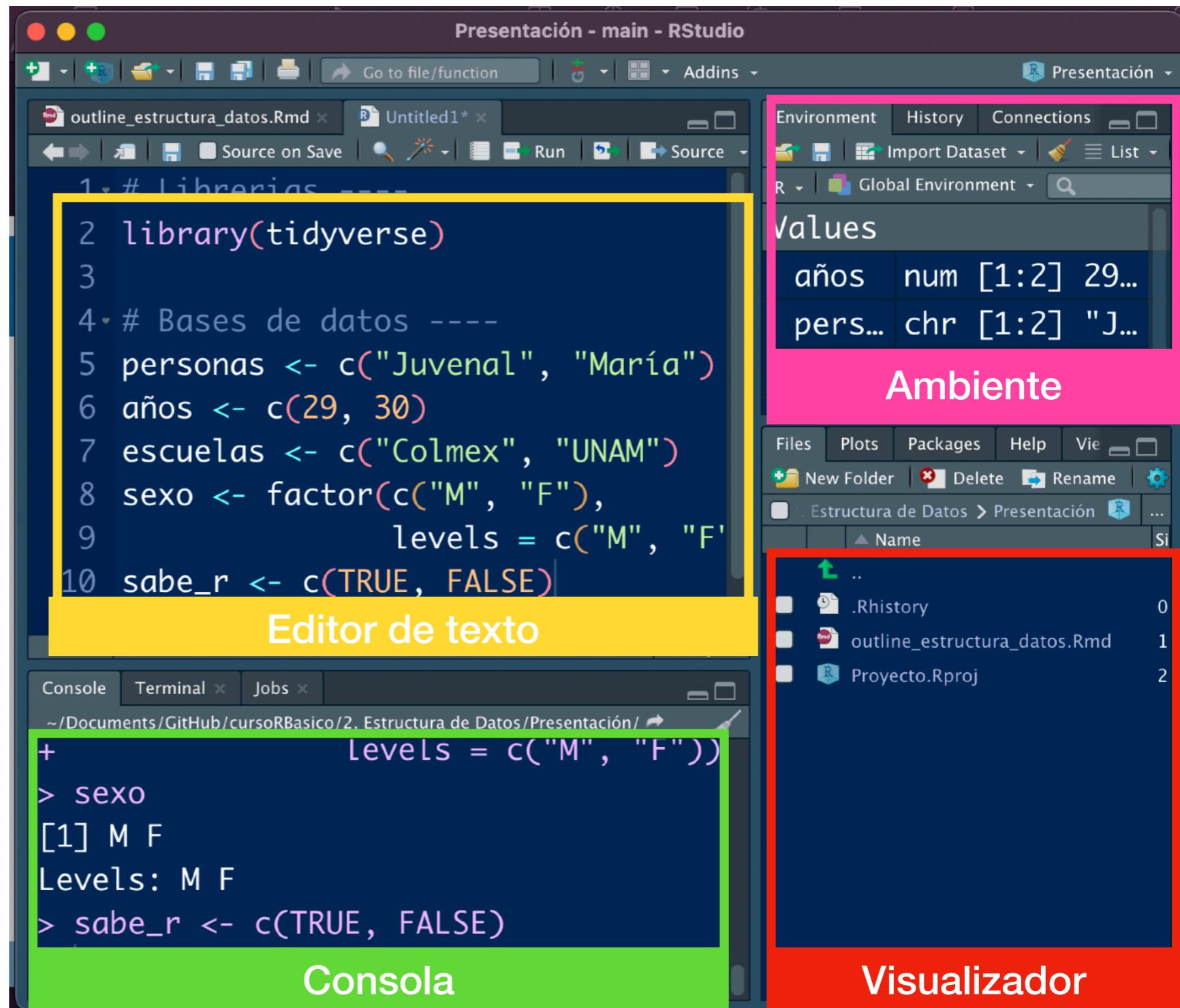
- Editor de texto**: A yellow box highlights the code editor window containing R code for creating variables (personas, años, escuelas, sexo, sabe_r) and setting levels for the sexo factor.
- Consola**: A green box highlights the console window showing the execution of the R code and its output.
- Environment**: A pink box highlights the environment pane showing the global variables: años, pers..., sabe... (with their respective types and values).
- History**: A pink box highlights the history pane showing the commandLevels = c("M", "F") and its output [1] M F.
- 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

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 area 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.



Archivos nativos de R

Al trabajar con R y RStudio, generamos cuatro tipos de archivos nativos de R, estos son los siguientes:

- *.Rproj,
- *.RData,
- *.R y
- *.rds.

Los scripts son aquellos archivos que terminan en *.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

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.).

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()`.

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.

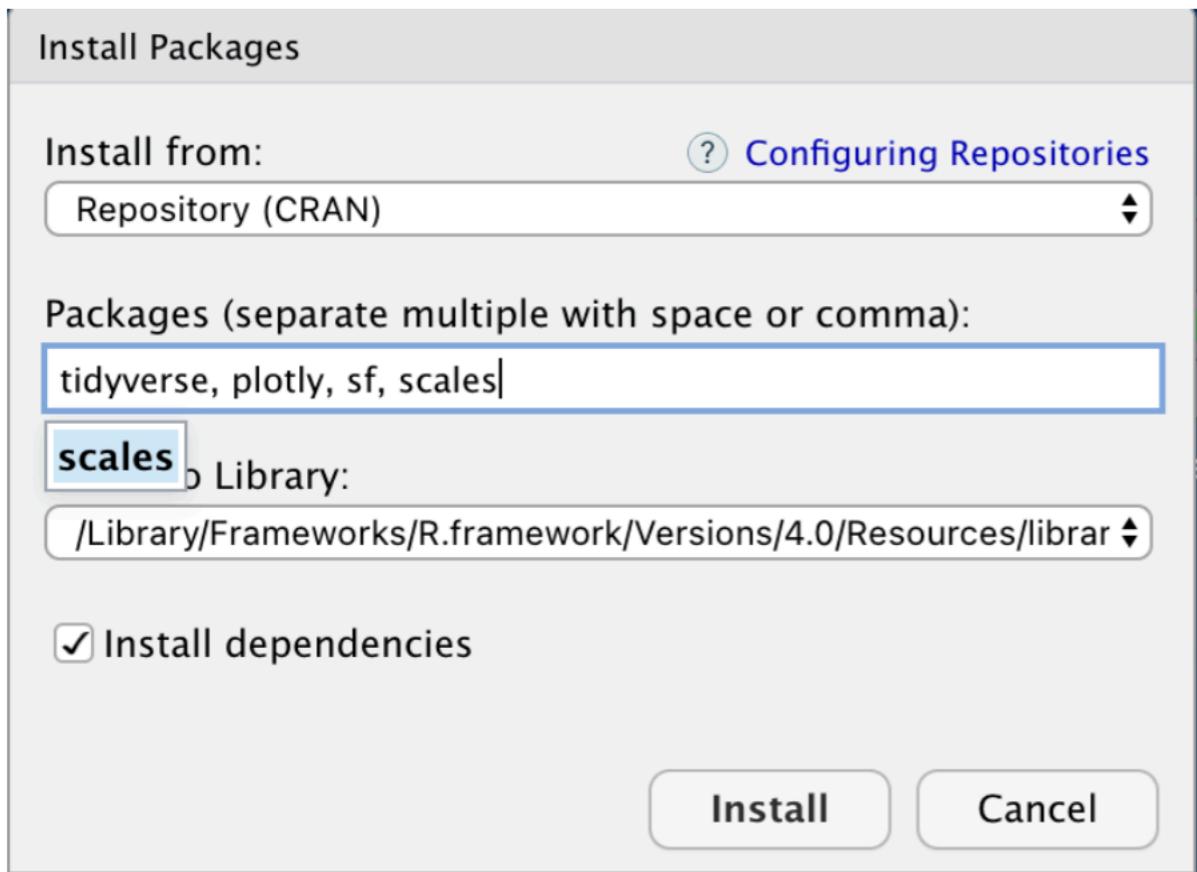
Instalación

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



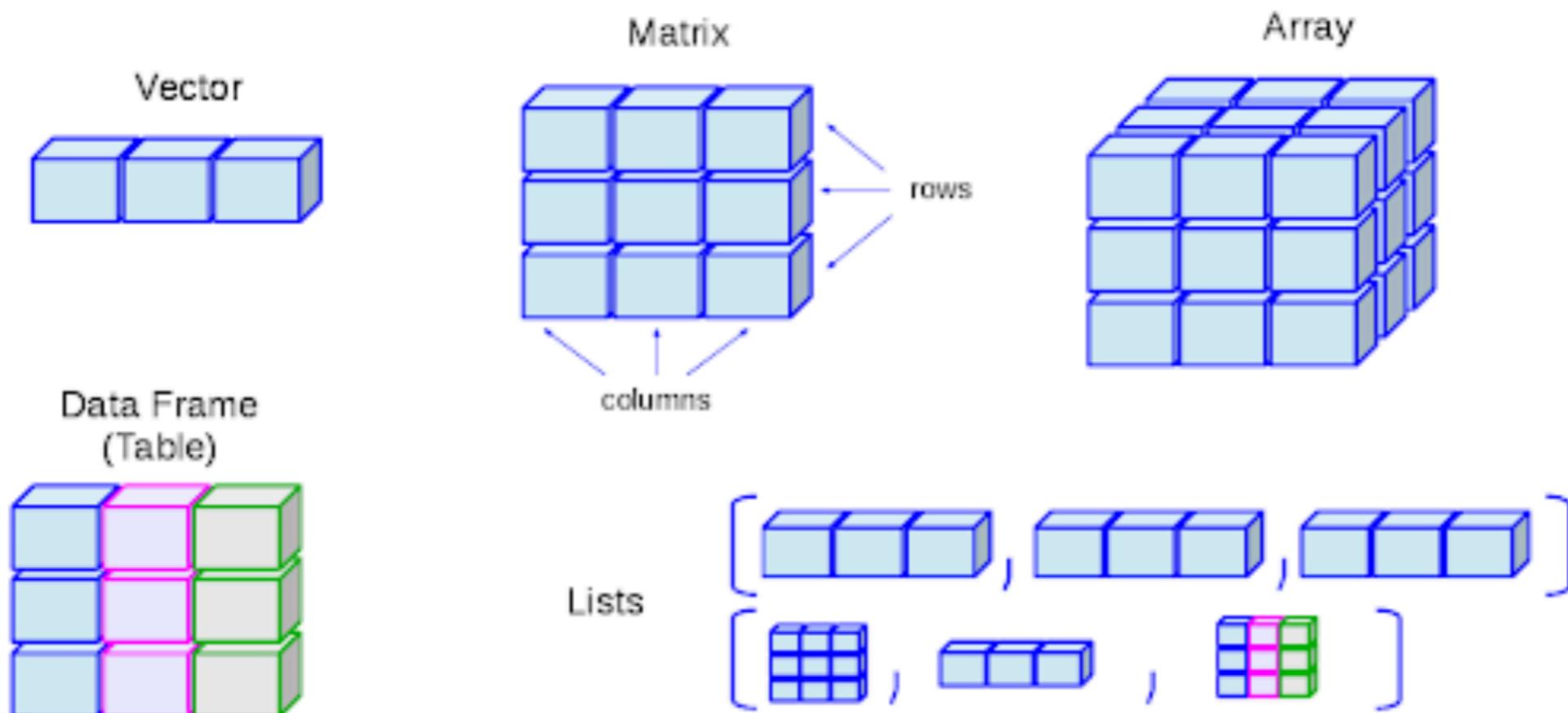
★ Instalar librerías



Llamar librerías



Estructura de Datos

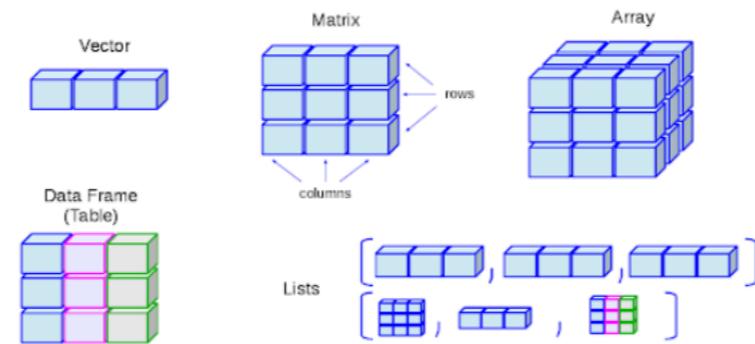


Estructura de Datos



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