



Tecnológico
de Monterrey

02. Ciencia de datos

Ciencia de datos para la toma de decisiones II

Jorge
Juvenal
Campos Ferreira

 juvenal.campos@tec.mx

Programa de la sesión de hoy

- ¿Qué es **Ciencia de datos**?
- **Tareas** que se hacen con datos
- **Estructura** de un código
- **Librerías** en R
- **Cargar archivos** de datos
- **Procesamiento** de datos con tidyverse

Datos

3



Fuente: World Press Photo (2020)

Datos

World Press Photo 2020



Fuente: World Press Photo (2020)

Metadatos

Título: *Straight Voice*
Autor: *Yasuyoshi Chiba*
Fecha: 19 de junio de 2019
Lugar: Khartoum, Sudan
Publicación: Agence France-Presse

Los metadatos son datos sobre los datos. Es decir, es información estructurada que describe, contextualiza o documenta otros datos, permitiendo entenderlos, localizarlos, usarlos y gestionarlos adecuadamente.

Datos

World Press Photo 2020



Datos Cuantitativos

¿Cuántas personas hay?

¿Cuántas están sonriendo?

¿Cuántos son hombres y cuántas son mujeres?

¿Cuántas cámaras hay?

Datos

World Press Photo 2020



Datos Cualitativos

¿Qué están diciendo?

¿Quién lo está diciendo?

¿Cómo lo está diciendo?

¿Por qué lo está diciendo?

¿Dónde lo está diciendo?

¿Qué es un dato?

Desde la ciencia de datos

Son: “*una colección de hechos, cifras, palabras, observaciones u otra información útil*” que deben ser procesados para transformarse en insights valiosos que mejoran la toma de decisiones.

—IBM

Son una “*representación simbólica (numérica, alfabética, algorítmica, espacial, etc.) de un atributo o variable cuantitativa o cualitativa*”.

—Wikipedia

Desde las ciencias sociales

Un dato es una **representación de la realidad social obtenida a través de un proceso de observación o medición**. No es solo una cifra aislada, sino un registro que adquiere sentido dentro de un marco conceptual

Una expresión de deseo por almacenar una vivencia

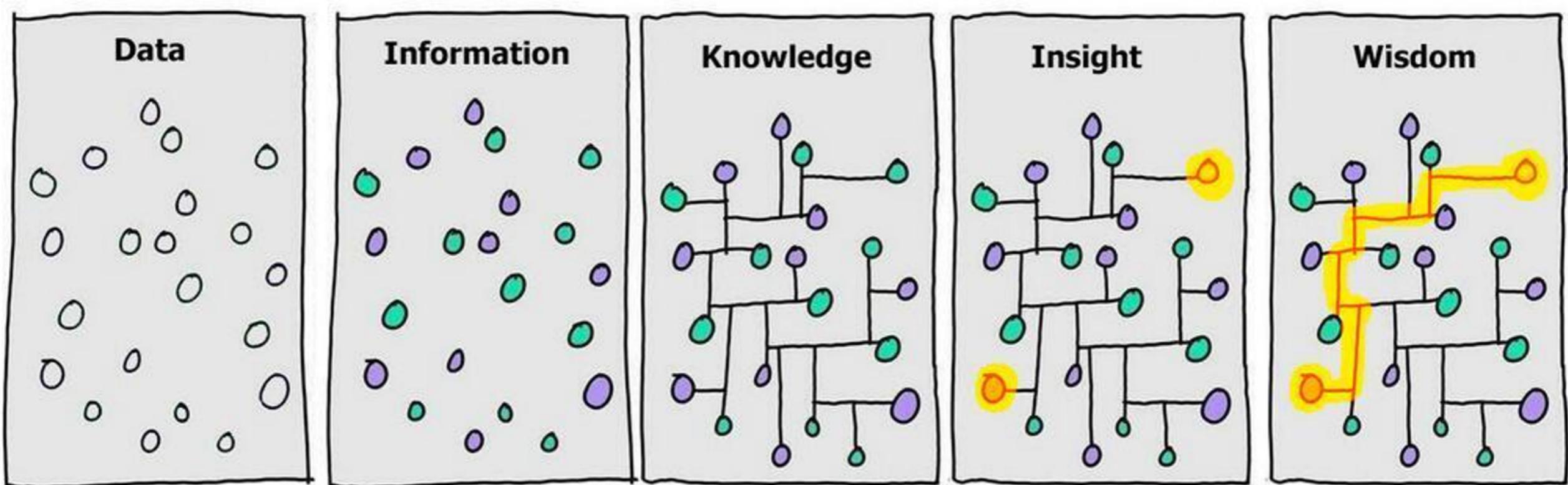
¿Qué es la ciencia de datos?

La **ciencia de datos** es un campo interdisciplinario que combina **estadística, matemáticas, programación, inteligencia artificial** y **conocimiento del dominio** para extraer información útil y generar conocimiento a partir de datos.

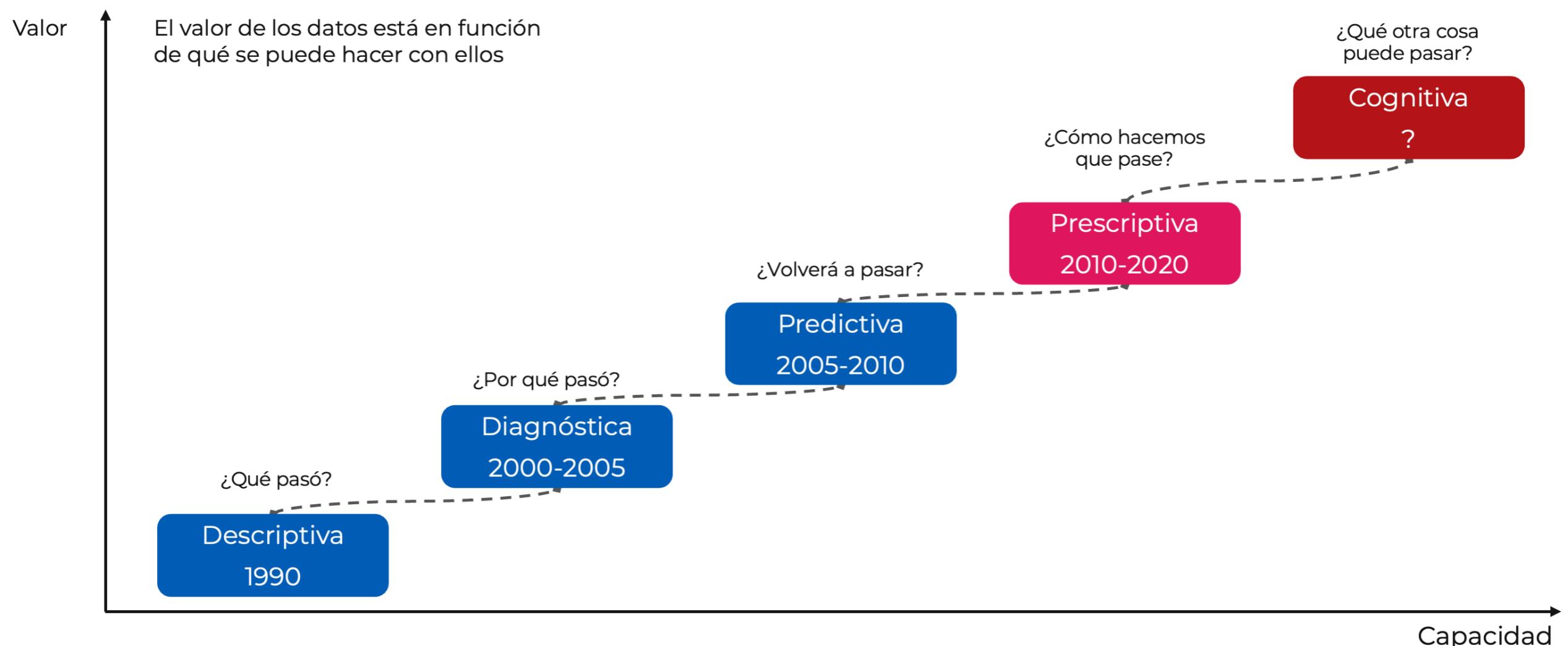


¿Cuál es el objetivo de la Ciencia de datos?

El objetivo principal de la **Ciencia de datos** es transformar grandes volúmenes de datos, muchas veces desordenados o no estructurados, en **hallazgos prácticos** que sirvan para la toma de decisiones.



El valor de los datos en la política pública



El valor de los datos en la política pública

TIPOS DE ANÁLISIS DE DATOS



Análisis descriptivo.

- ¿Qué es lo que sucede?
- Incorpora técnicas estadísticas para comprender el contexto actual de cierto problema público

Análisis de causalidad.

- ¿Por qué sucede?
- Utiliza técnicas estadísticas y econométricas para realizar inferencias causales y comprender cuáles son las causas que originan cierto valor público.



El valor de los datos en la política pública

TIPOS DE ANÁLISIS DE DATOS



Análisis predictivo

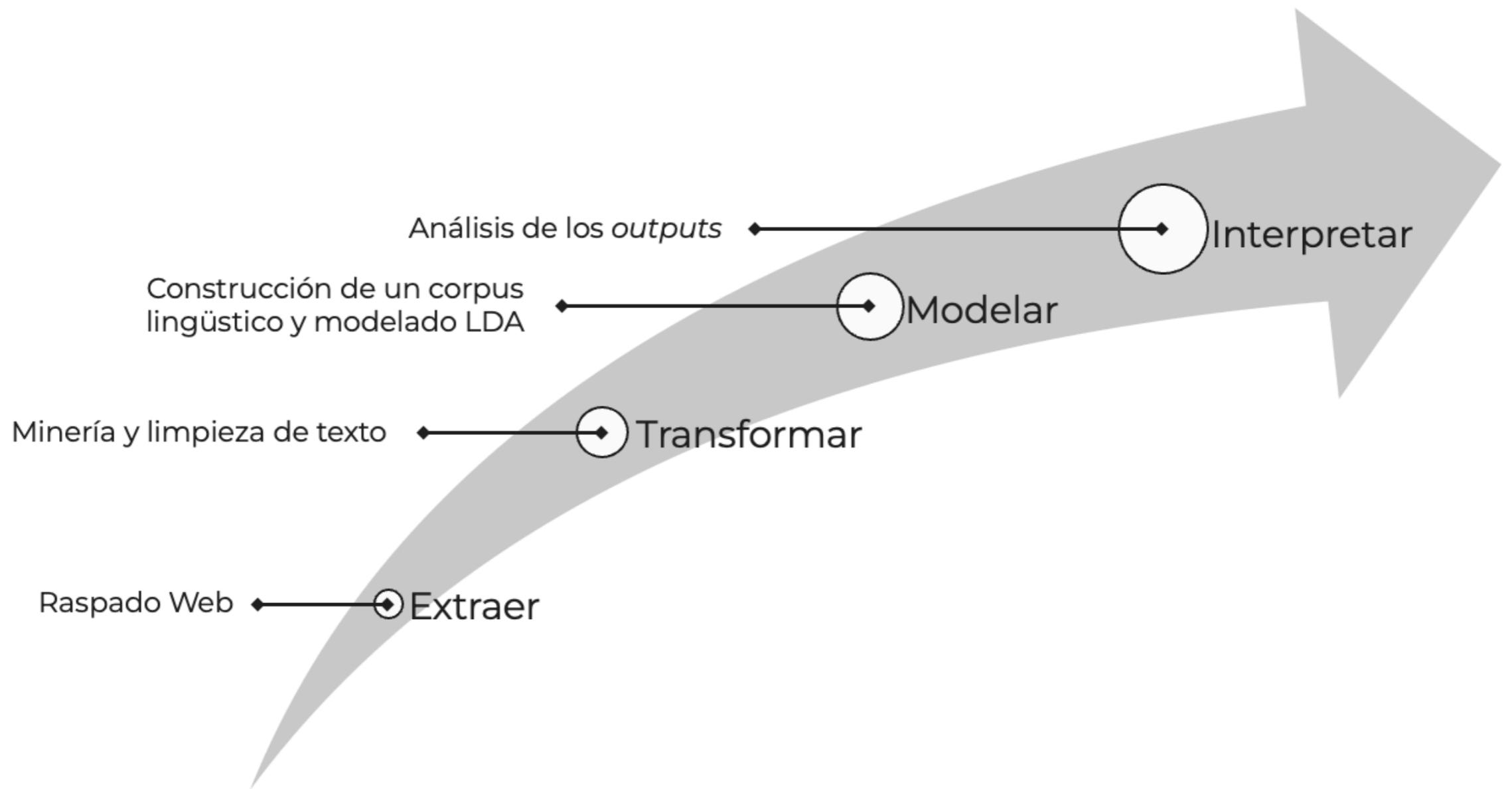
- ¿Qué pasará o qué podría suceder?
- Modelos estadísticos y predictivos para estimar las probabilidades de ocurrencia de un comportamiento o un fenómeno público

Análisis prescriptivo

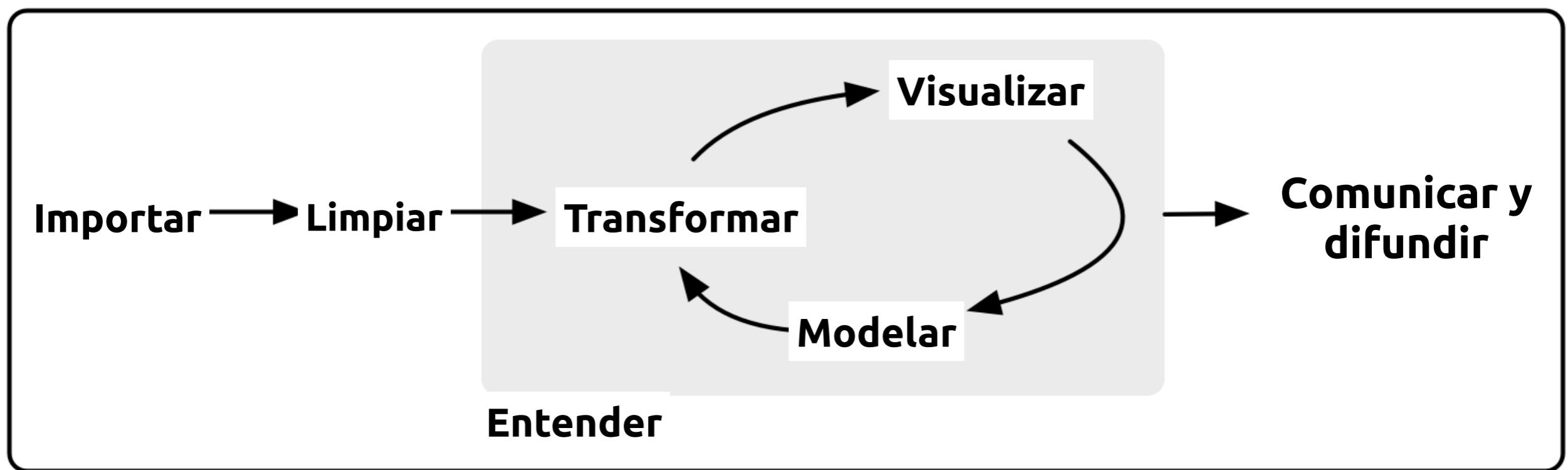
- ¿Qué se debería hacer?
- Permite la formulación de recomendaciones de política pública.



El proceso del trabajo con datos



El proceso del trabajo con datos



Programar

¿Por qué aprender a programar?

- 1. Actualmente, los datos son enormes**, por lo que se requieren herramientas especializadas para manejarlos
- 2. Automatización = más tiempo para pensar.** Programar te permite automatizar tareas para dedicarte a otras actividades
- 3. El mercado laboral demanda habilidades de programación.** R, Python y SQL entre lo más demandado
- 4. Permite hacer análisis, experimentar y simular escenarios**
- 5. Te da independencia y poder** para realizar tus propios cálculos

La explosión de los datos

Datos generados anualmente en el mundo



Tareas que se hacen con datos

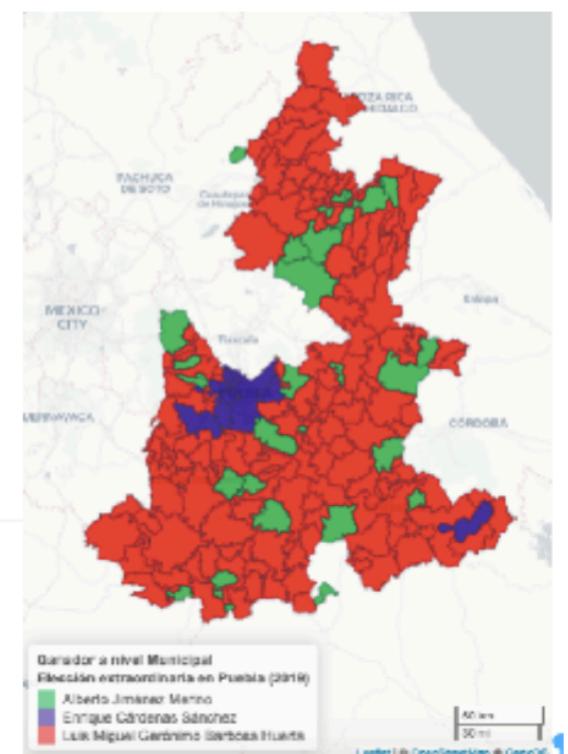


Tareas que se hacen con datos

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 )
```

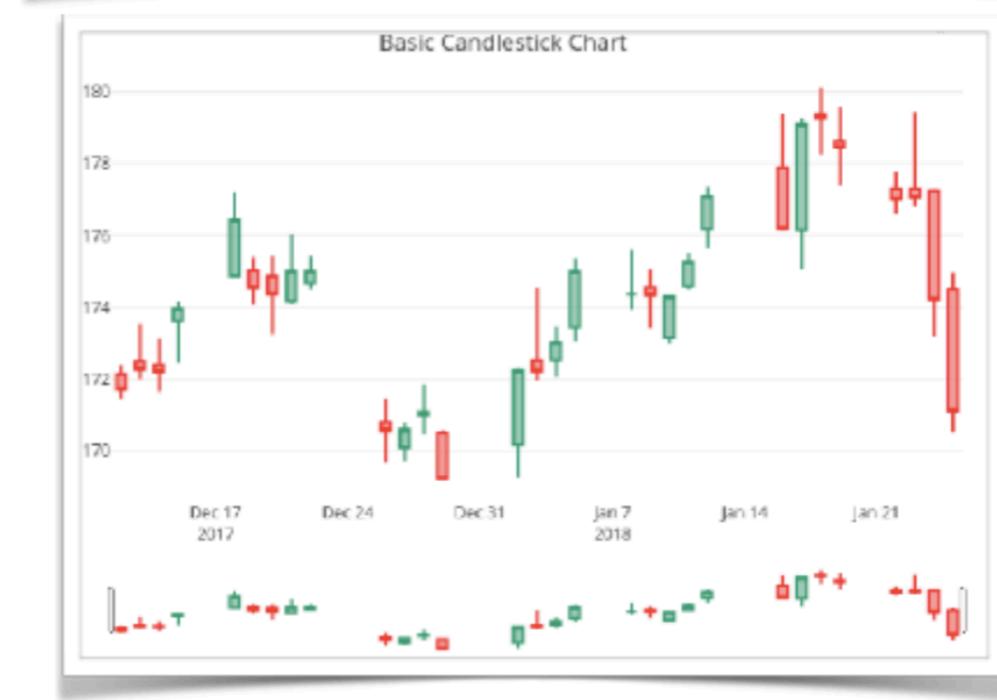
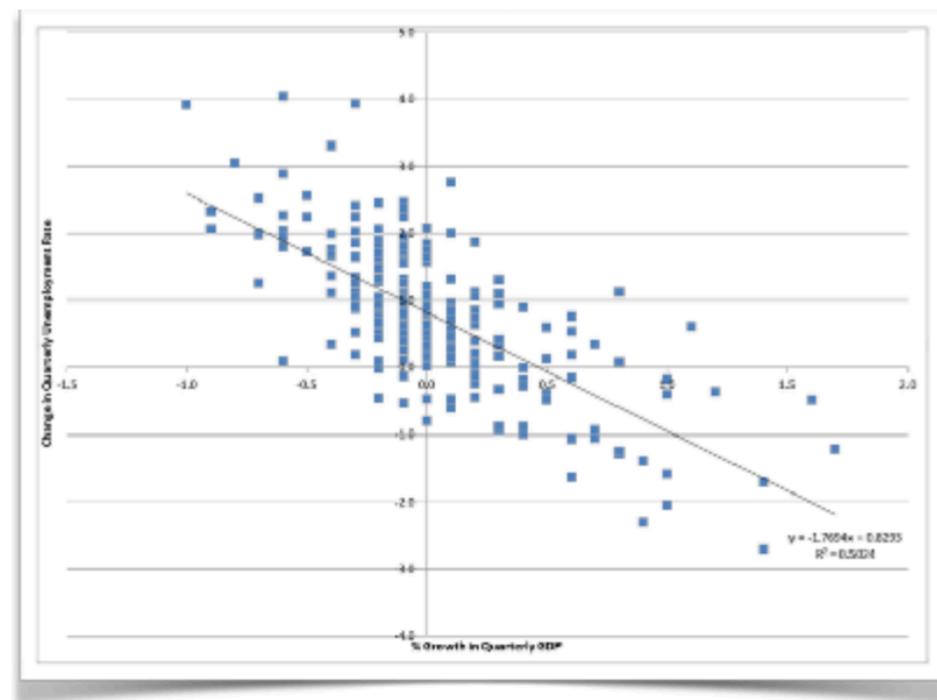
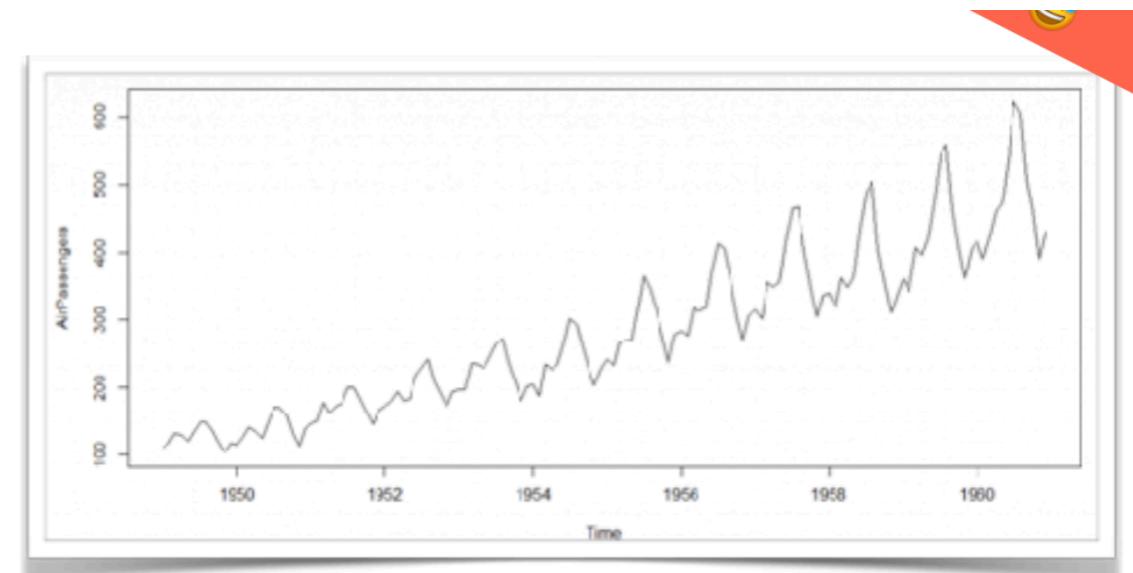


Tareas que se hacen con datos

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



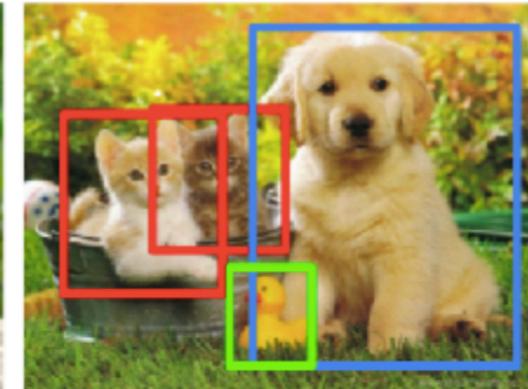
CAT

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.

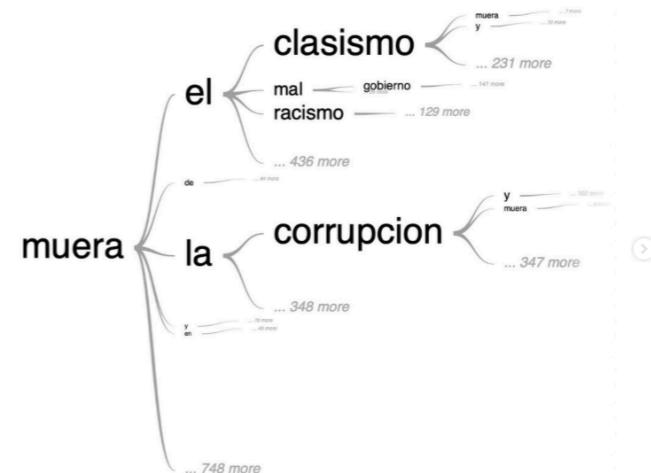
library(tm)

```
library(stringr)
```



Nube de palabras.

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



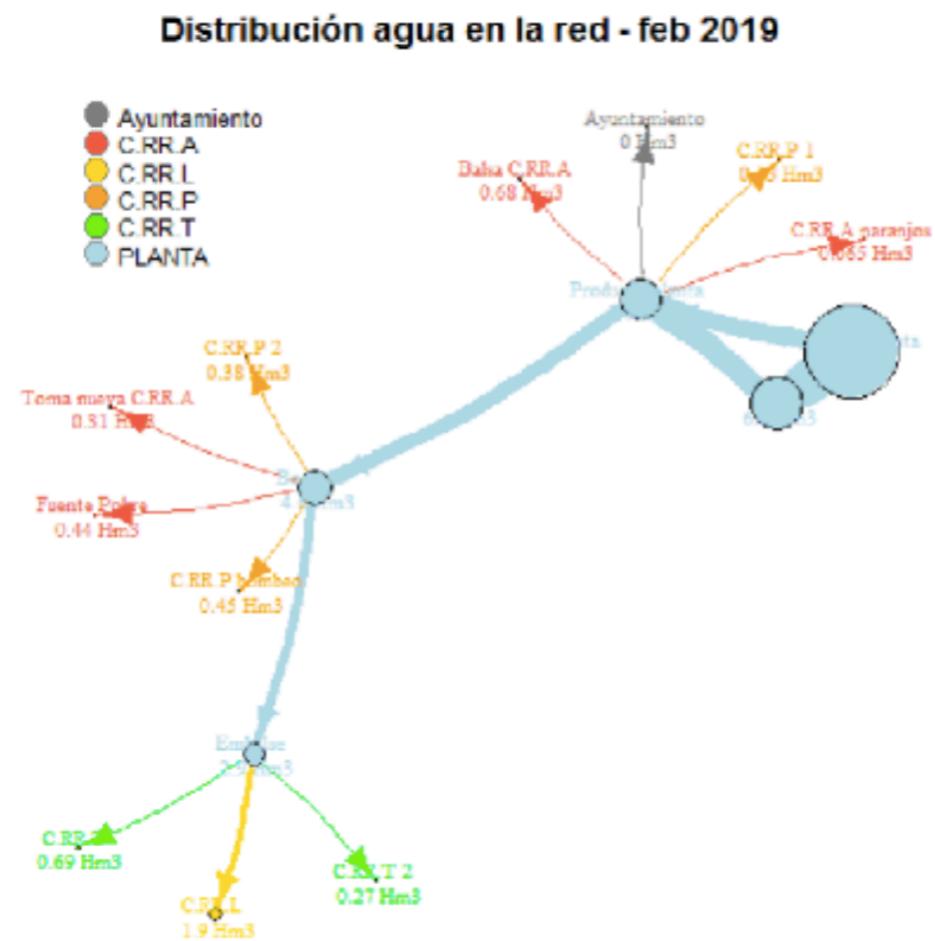
Nube de palabras.

Plan Nacional de Desarrollo. 2019.

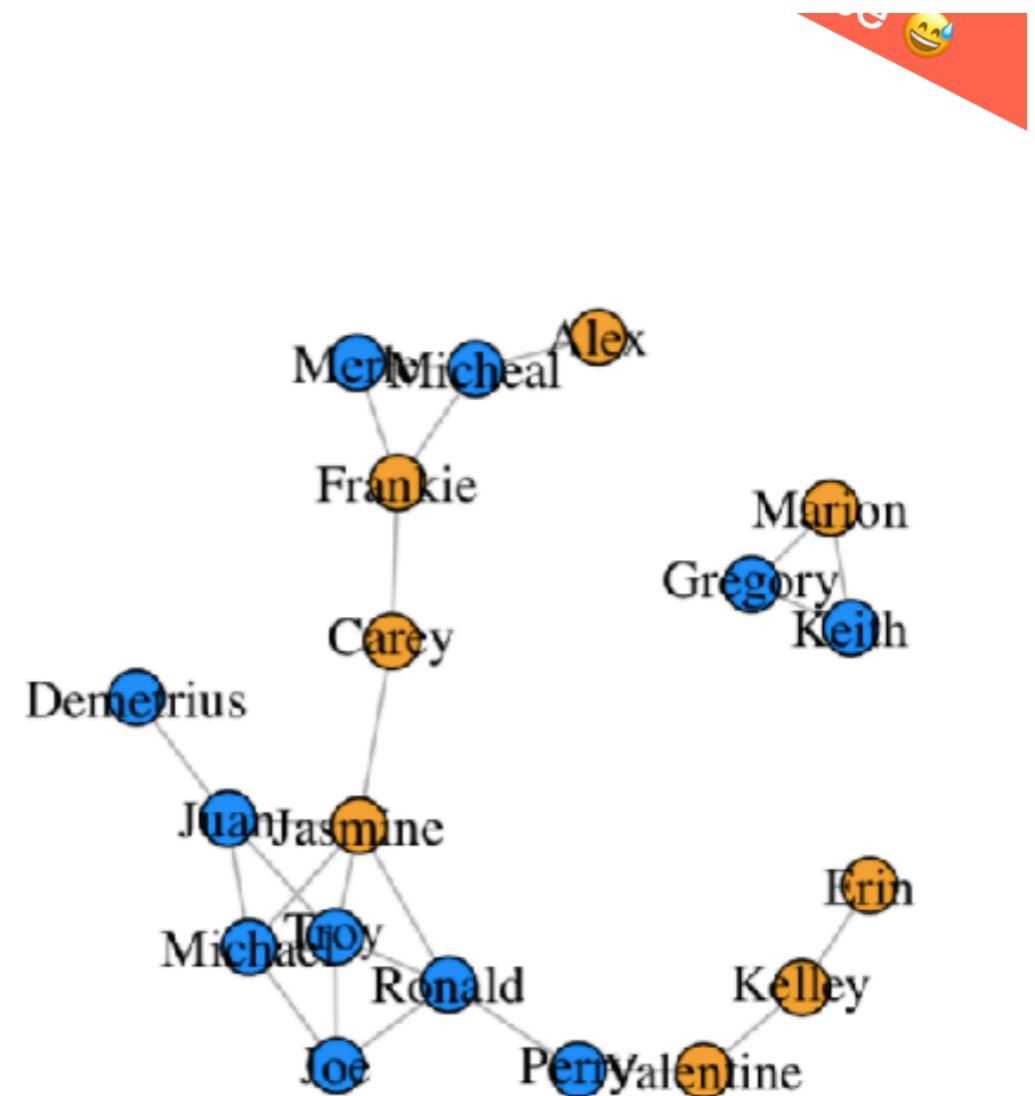
Tareas que se hacen con datos

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.

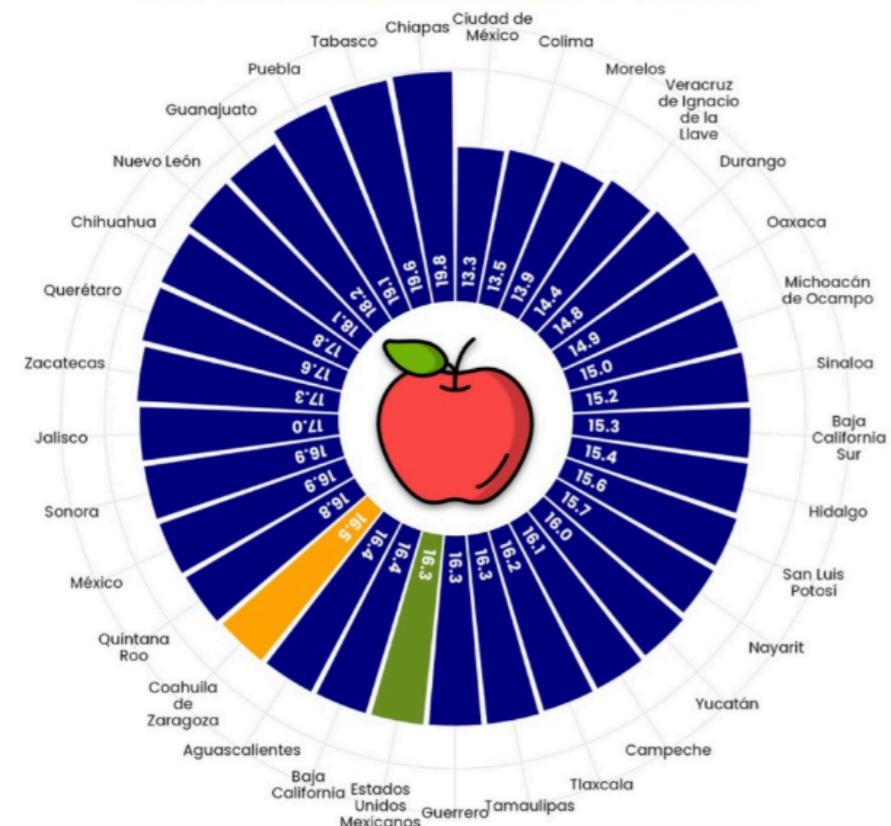
library(ggplot2)

library(plotly)

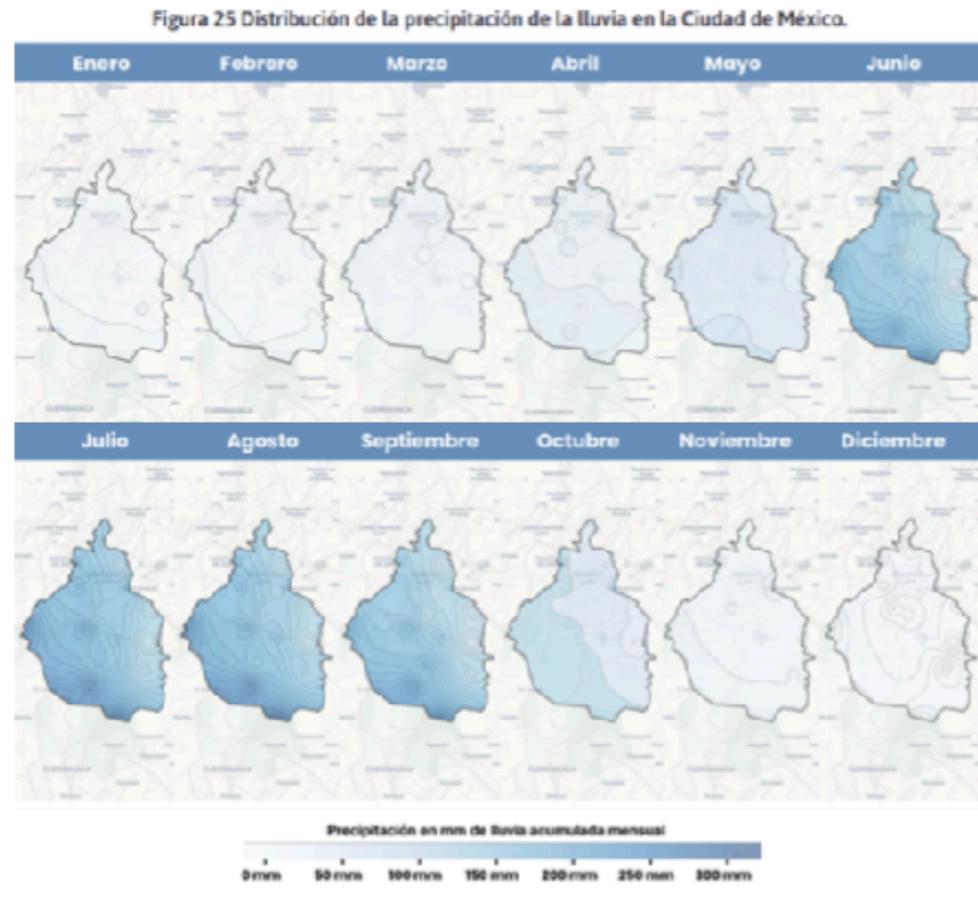
library(leaflet)

library(htmlwidgets)

Razón de **alumnos por profesor** en las entidades de México



Fuente: Elaboración propia con datos del INEGI. Ciclo 2021-2022.
@JuvenalCamposF - IG: juvenalcampos.dataviz



**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)

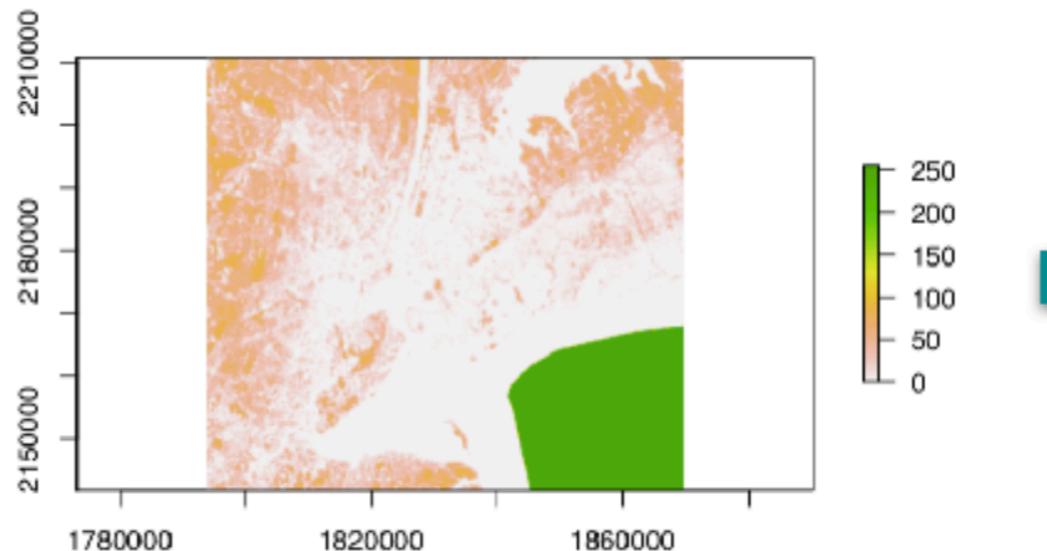
The screenshot shows the Mercado Libre website interface for used, semi-new, and new cars. At the top, there's a search bar and a banner for 'ENVÍO GRATIS EN TU PRIMERA COMPRA EXCLUSIVO EN APP'. The main search results for 'Autos usados, seminuevos y nuevos' show a featured listing for a 'SEMI NUEVOS FARRERA' Honda Civic 2022 at \$445,000. Below this, there are three more car listings: a silver Mercedes-Benz Clase C 3.5 400 Cgi At for \$499,850, a red Ford Mustang 5.0L V8 At for \$949,000, and a white Honda Cr-v 2.4 I-style Mt for \$266,800. The left sidebar includes filters for 'Vehículos verificados' and 'Publicados hoy', and sections for 'Tiendas oficiales' (13,094 results) and 'Ubicación' (listing various Mexican states). The bottom sidebar has sections for 'Marca' (Chevrolet, Volkswagen, Ford, Nissan, BMW, Mercedes-Benz, Honda, Toyota, Mazda) and 'Modelo'.

Tareas que se hacen con datos

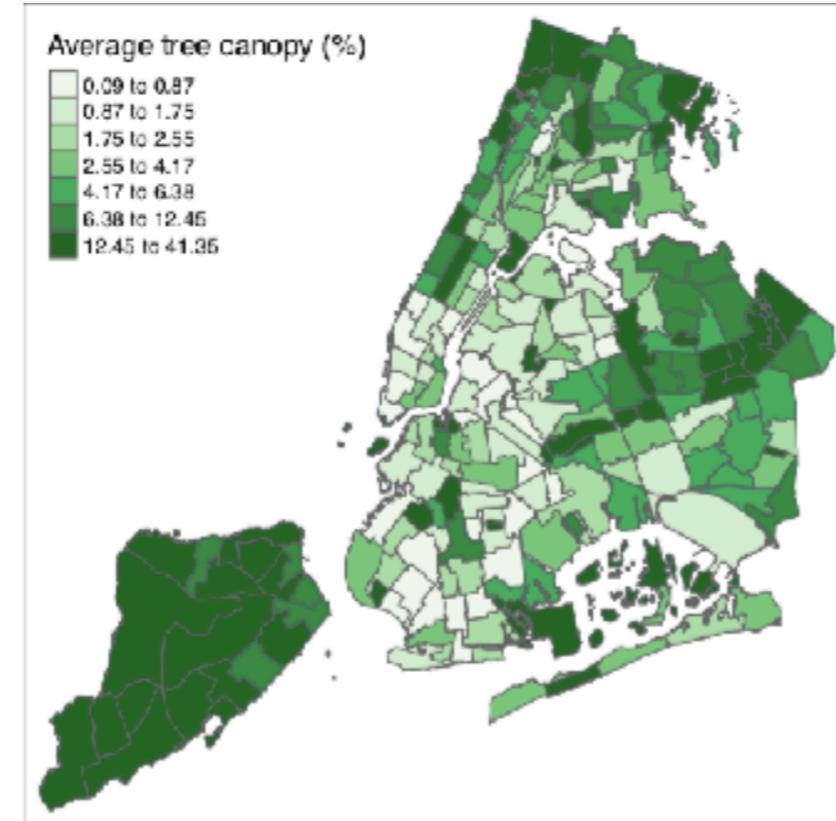
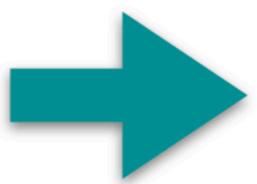
8. Análisis Geoespacial.

`library(sf)`

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



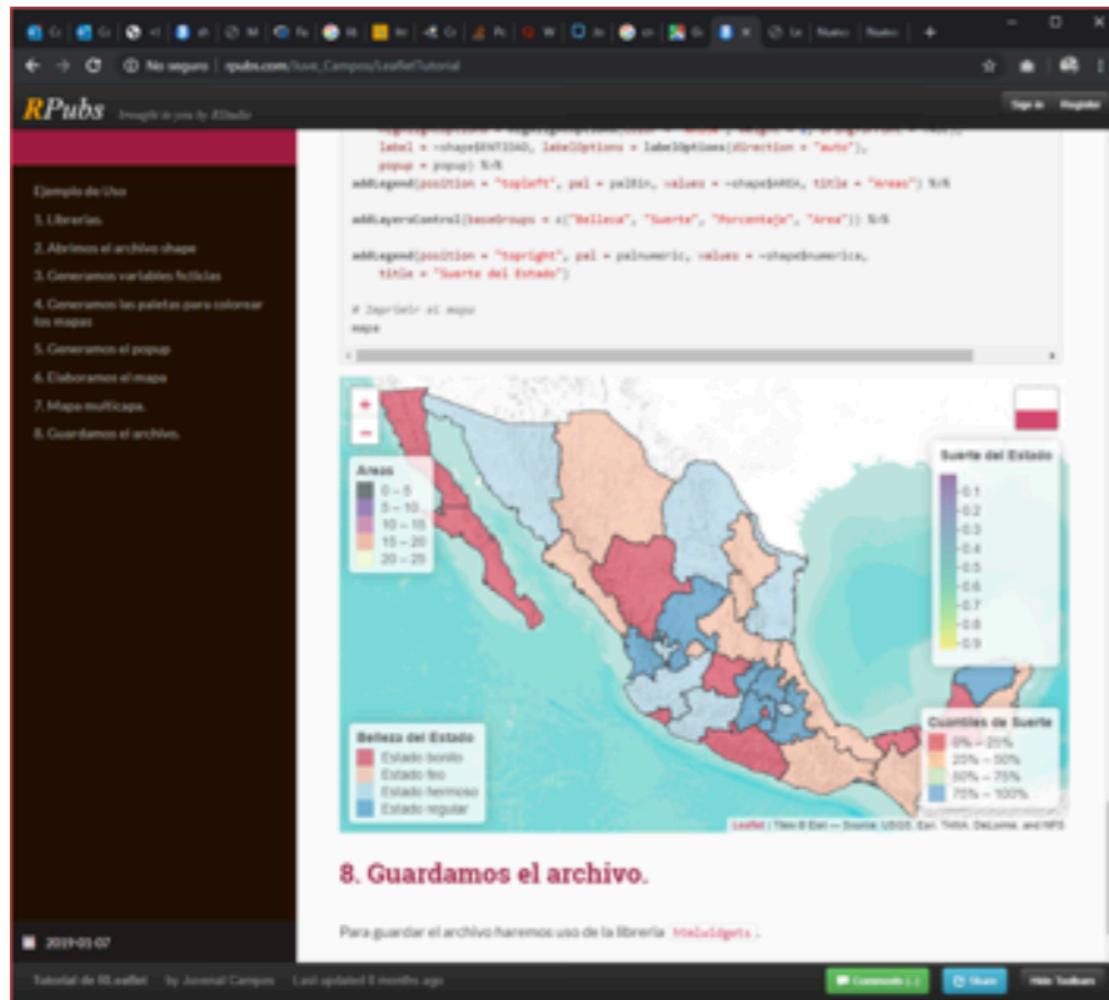
**Datos Crudos
(Raw Data)**



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 `mapview` function `crs()`.

When the geodata has been read in with `sf`, `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 `proj` string.

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

# Loading required package: sp
root <- "/Users/juvicam/Desktop/CienciaCorta/Geovisualization/lat_35_raster.tif"
raster.readr

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

# Reading layer 'trees' from data source '/Users/juvicam/Desktop/CienciaCorta/Geovisualization/lat_35_raster.tif'
# Simple feature collection with 65227 features and 7 fields
# geometry type: POLYGON
# dimension: XY
# bbox: -16.250000 40.40924 43.75000 60.91666
# 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/Geovisualization/lat_35_raster.tif'
# Simple feature collection with 128 features and 8 fields
# geometry type: POLYGON
# dimension: XY
# bbox: -16.250000 40.40912 43.75000 60.91666
# epsg (SRID): 4326
# projection: +proj=longlat +ellps=WGS84 +no_defs
# Read in the tree canopy raster
canopy <- raster(paste0(root, "/canopy.tif"))
```

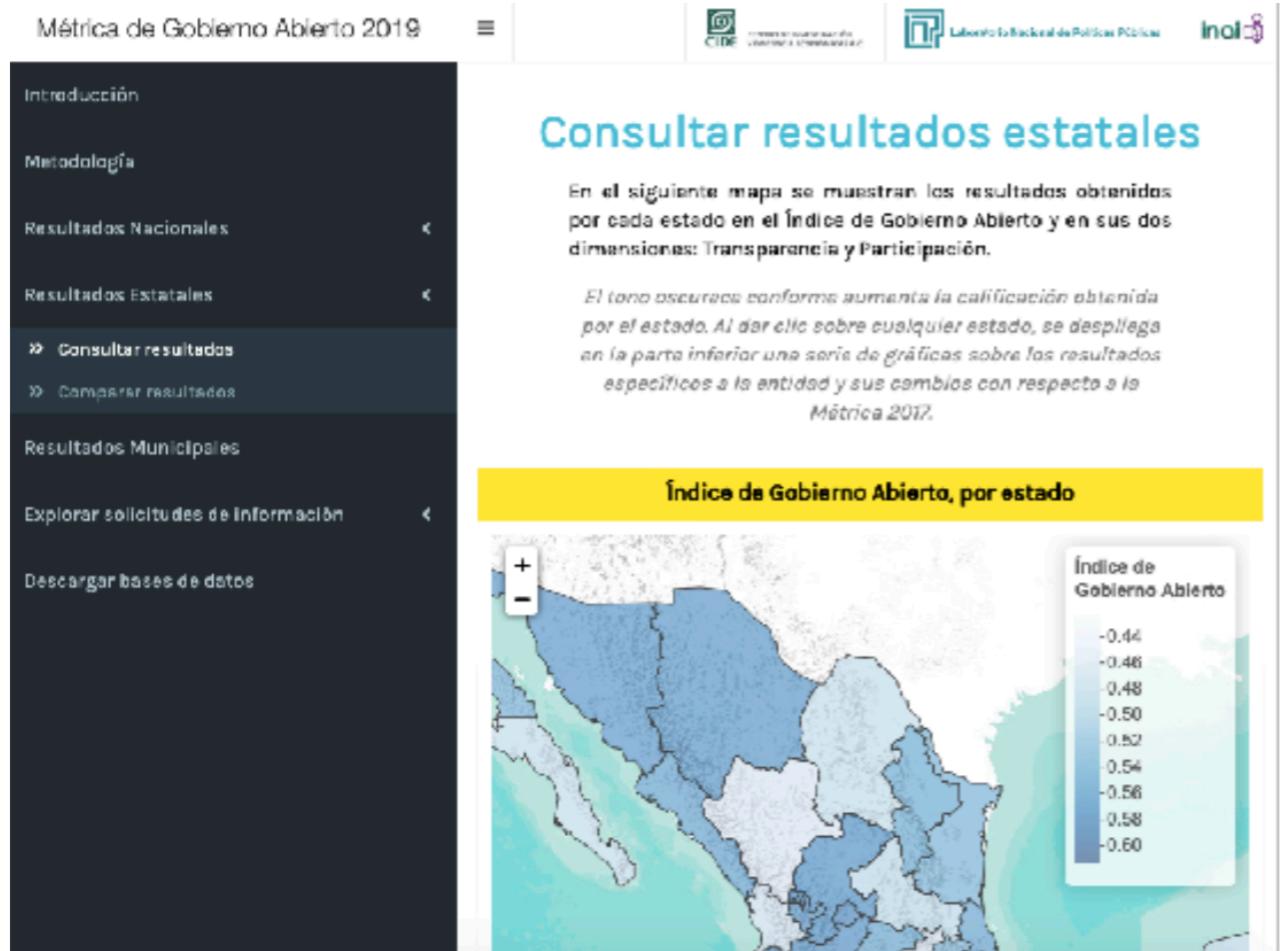
http://rpubs.com/Juve_Campos/LeafletTutorial

LaTeX pdf

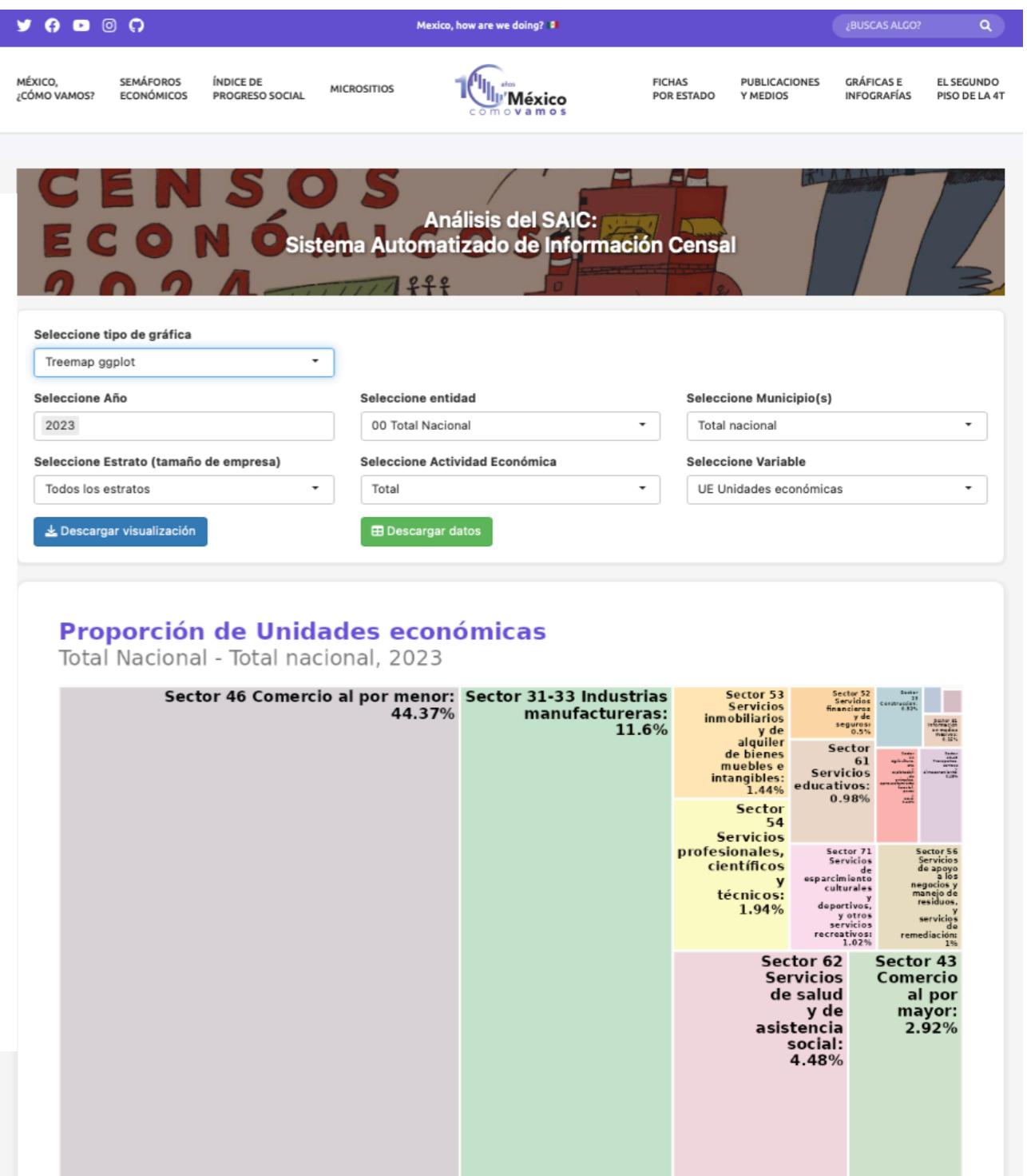
Tareas que se hacen con datos

12. Web Apps.

library(shiny)



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





Tecnológico
de Monterrey

Introducción al entorno de R y Rstudio

Ciencia de datos para la toma de decisiones II

Jorge
Juvenal
Campos Ferreira

 juvenal.campos@tec.mx

¿Qué es R?



Lenguaje de programación *creado en 1993 por Ross Ihaka y Robert Gentleman en la Universidad de Auckland, Nueva Zelanda*

20,000+

Paquetes

2.3M+

Usuarios

#1

Lenguaje estadística

Usado por: Banco Mundial • Google • Facebook • Twitter • Microsoft • Uber

¿Por qué R?



1 Diseñado para estadística

La herramienta perfecta para análisis estadístico avanzado



2 Ecosistema para Ciencias Sociales

Comunidad activa con paquetes especializados (econometría, encuestas, APIs)

3 100% Gratuito y Open Source

Acceso total al código y a la mayoría de sus funciones, sin licencias costosas 💰

4 Reproducible y profesional

Genera reportes verificables y profesionales



R y Python



R y Python se parecen más de lo que suena, e incluso pueden utilizarse juntos.
Ambos son lenguajes interpretados y multiplataforma (corren en Mac, Windows y Ubuntu).

Ambos sirven para importar datos, limpiar/transformar, visualizar, modelar y automatizar

Ambos conectan con lo mismo: bases de datos (PostgreSQL, MySQL, BigQuery), archivos (Parquet), y herramientas de “data engineering” (Spark, etc.)



Aspecto

R



Python

Aspecto	R	Python
Enfoque original	Estadística y análisis de datos	Lenguaje generalista (software en general)
Mejor cuando...	Análisis exploratorio, estadística, visualización, reportes	Automatización, ingeniería, IA moderna, integración con sistemas
Curva de aprendizaje (data)	Muy amigable si vienes de estadística/economía	Muy amigable si vienes de programación general
Manipulación de datos	tidyverse (dplyr, tidyr) muy expresivo	pandas (y polars) muy potente, a veces más verboso
Visualización	ggplot2 (estándar de facto en "grammar of graphics")	matplotlib / seaborn / plotly (varía según librería)
Estadística clásica	Muy fuerte (ecosistema amplio y maduro)	Fuerte, pero muchas cosas vienen "adaptadas" del mundo estadístico
Machine Learning "tradicional"	caret, tidymodels, mlr3	scikit-learn (muy dominante)
Deep Learning / IA moderna	Menos dominante (interfaces existen, pero menos central)	Dominante (PyTorch, TensorFlow, etc.)
NLP	Buenas opciones (quanteda, tidytext)	Muy fuerte (transformers, spaCy, etc.)
Reporting / documentos reproducibles	RMarkdown / Quarto (muy integrado)	Jupyter / Quarto (muy usado)
Dashboards / apps	Shiny (muy potente y rápido para prototipos)	Streamlit / Dash / FastAPI + frontend
Producción (APIs/servicios)	Possible (plumber), menos común en industria	Muy común (FastAPI, Flask, etc.)
Rendimiento	Puede ser más lento en loops; se acelera con vectorización/C++	Similar; se acelera con numpy/Cython/numba/C++
Manejo de paquetes/entornos	renv, pak; CRAN/Bioconductor	venv/poetry; pip/conda; PyPI
Comunidad típica	Estadística, academia, analistas, data viz	Ingeniería, data science, IA, software
Interoperabilidad	Puede llamar Python (reticulate)	Puede llamar R (rpy2)
"Sabor" del código	Muy orientado a datos (pipes, data frames)	Más generalista (estructuras, clases, scripts)

R como calculadora

Operaciones básicas

```
2 + 3  
5 * 4  
10 / 2  
2 ^ 8
```

Funciones matemáticas

```
sqrt(16)  
abs(-5)  
round(3.7)
```

Salida en consola:

```
> 2 + 3  
[1] 5  
> 5 * 4  
[1] 20  
> sqrt(16)  
[1] 4
```

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**

Ambiente/entorno global

- El ambiente global en RStudio es el espacio de trabajo donde R guarda todos los objetos que se crean en la sesión
- Aquí viven los objetos que creamos con la `<-`
- En la sesión, vamos a poder acceder a los objetos creados en este ambiente.

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.

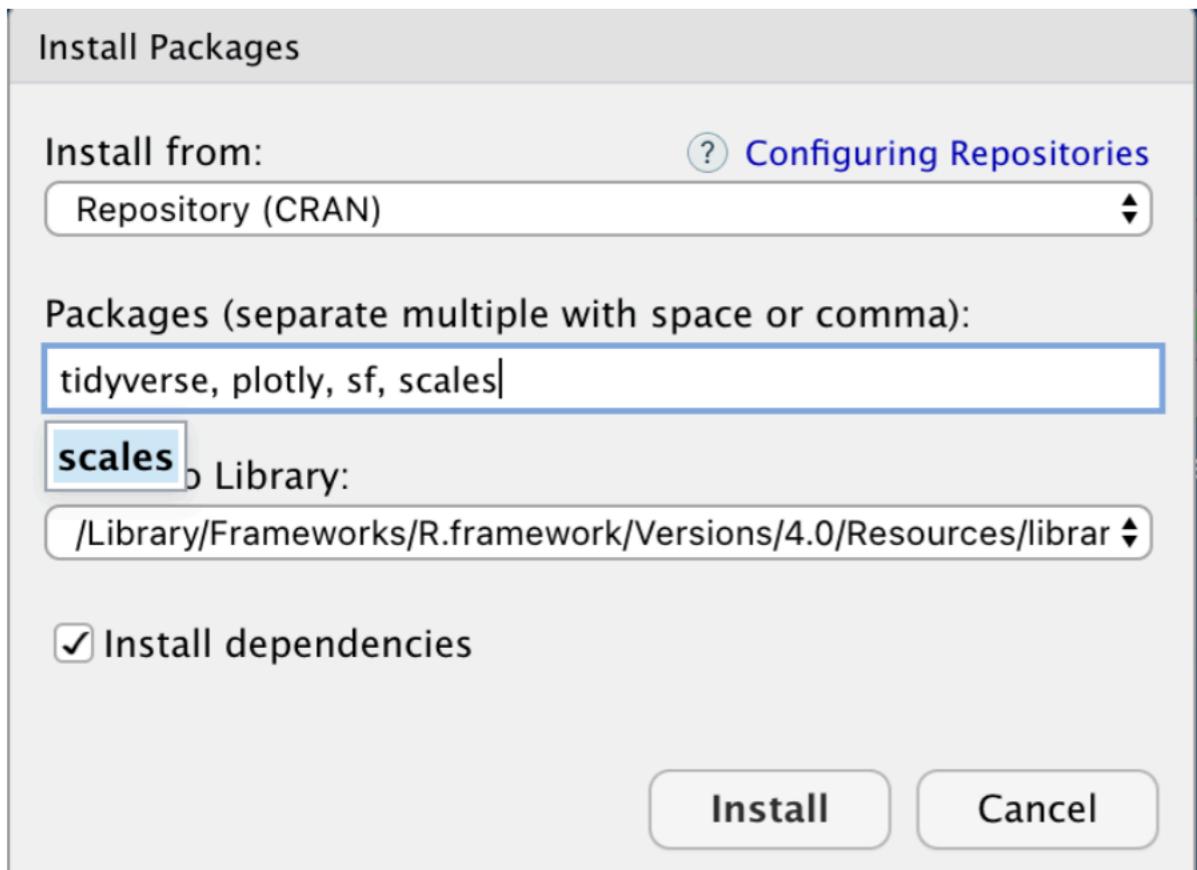
Instalación

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



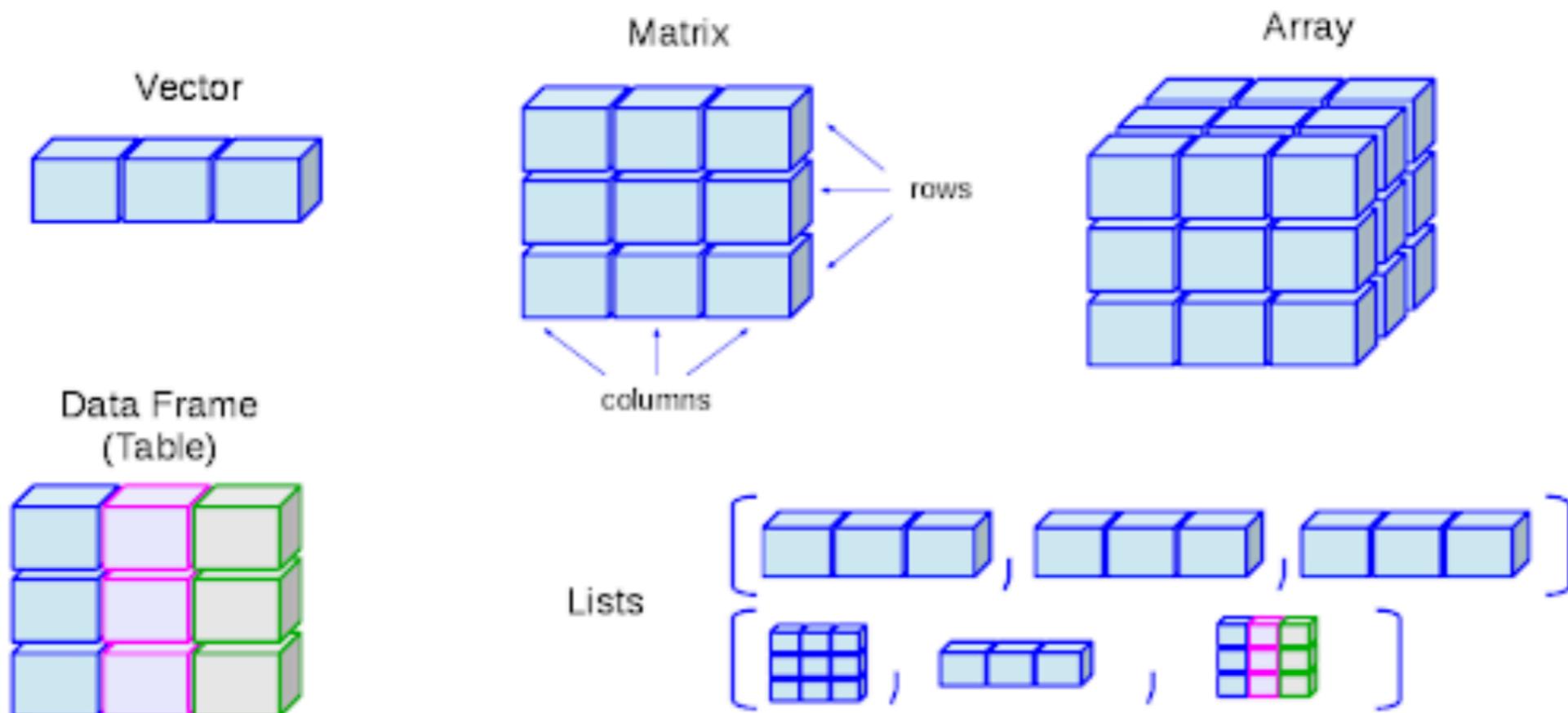
★ Instalar librerías



Llamar librerías



Estructura de Datos

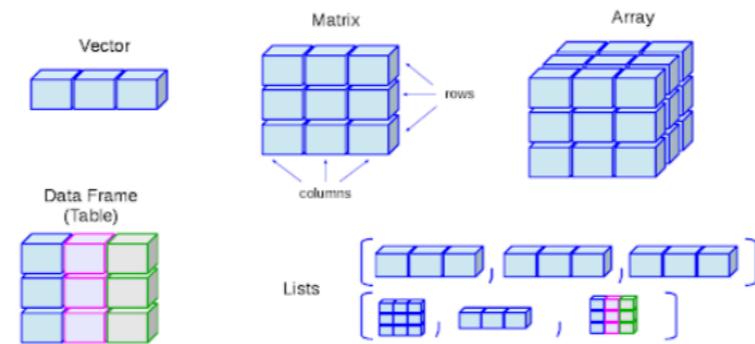


Estructura de Datos



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:

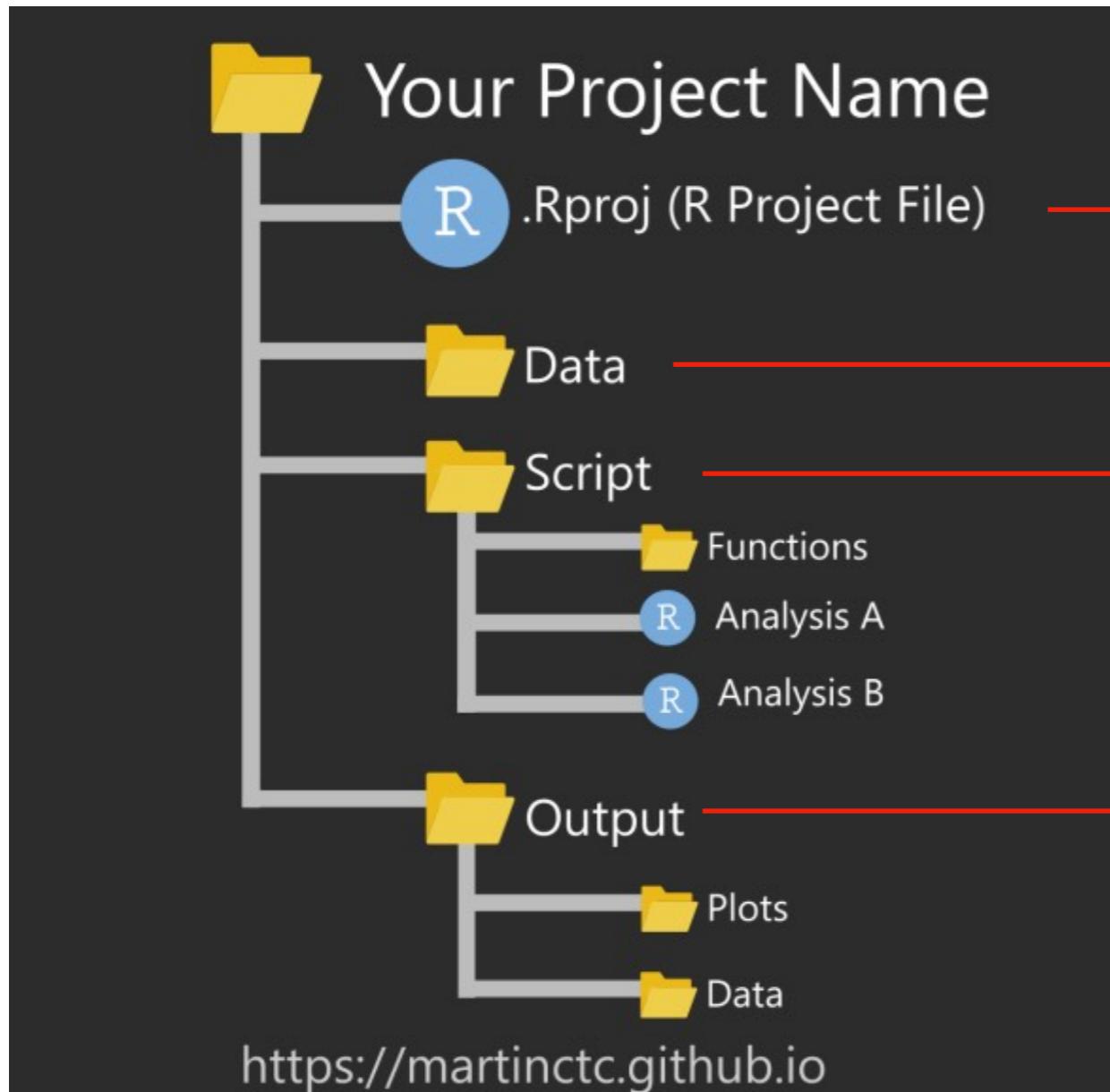
```
# Tres tipos de datos combinados  
c("Hola", TRUE, 1)
```

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.

Configuración de un proyecto básico de R



Archivo de proyecto (.Rproj)

Donde se guardan los archivos de datos

Donde se guardan los scripts

Donde se guardan los resultados del
proceso de trabajo (datos,
visualizaciones, reportes, modelos).

Estructura (sugerida) de un código o script

- 1. Configuración**
- 2. Carga de librerías**
- 3. Carga de parámetros y funciones propias**
- 4. Carga de datos**
- 5. Limpieza de datos**
- 6. Análisis exploratorio de datos**
- 7. Generación de funciones**
- 8. Generación y pruebas de modelos**
- 9. Generación de gráficas**
- 10. Guardado de archivos**

```
# -----
# Autor: Juvenal Campos
# Fecha: 2025-08-18
# Proyecto: Ejemplo para el Tec de Monterrey
# Descripción: Script dummy que ilustra la estructura básica de un flujo en R.
# -----  
  
# 1) Configuración -----
options(digits = 3)
set.seed(42)  
  
# 2) Carga de librerías -----
library(dplyr)
library(readr)
library(ggplot2)  
  
# 3) Parámetros y funciones propias -----
params <- list(  
  input = "data/input.csv",      # <- reemplaza si tienes archivo  
  output_dir = "out",  
  target = "mpg"  
)  
paleta_colores <- c("#6950d8", "#3CEAFA", "#00b783",  
  "#ff6260", "#ffaf84", "#ffbd41")  
if (!dir.exists(params$output_dir)) dir.create(params$output_dir, recursive = TRUE)  
  
clean_names <- function(x){  
  names(x) <- tolower(gsub("[^a-z0-9_]+", "_", names(x)))  
  x  
}  
rmse <- function(y, yhat) sqrt(mean((y - yhat)^2))  
  
# 4) Carga de datos -----
# df <- read_csv(params$input)          # <- real
# Dummy para ilustrar:
df <- clean_names(mtcars)
```

Estructura (sugerida) de un código o script

1. Configuración
2. Carga de librerías
3. Carga de parámetros y funciones propias
4. Carga de datos
- 5. Limpieza de datos**
- 6. Análisis exploratorio de datos**
- 7. Generación de funciones**
- 8. Generación y pruebas de modelos**
- 9. Generación de gráficas**
- 10. Guardado de archivos**

```
# 5) Limpieza de datos -----
df <- df %>%
  mutate(across(everything(), ~(.v) v)) %>%
  filter(!is.na(.data[[params$target]]))

# 6) Análisis exploratorio (EDA) -----
print(summary(df))
print(df %>% summarise(n = n(), p = ncol(.)))

# 7) Generación de funciones -----
# (ya definimos `rmse()` arriba como ejemplo)

# 8) Modelado (dummy) -----
mod <- lm(mpg ~ wt + hp, data = df)
pred <- predict(mod, df)
cat("RMSE:", rmse(df$mpg, pred), "\n")

# 9) Gráficas -----
p <- ggplot(df, aes(wt, mpg)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)

ggsave(file.path(params$output_dir, "scatter.png"), p,
       width = 5, height = 4, dpi = 150)

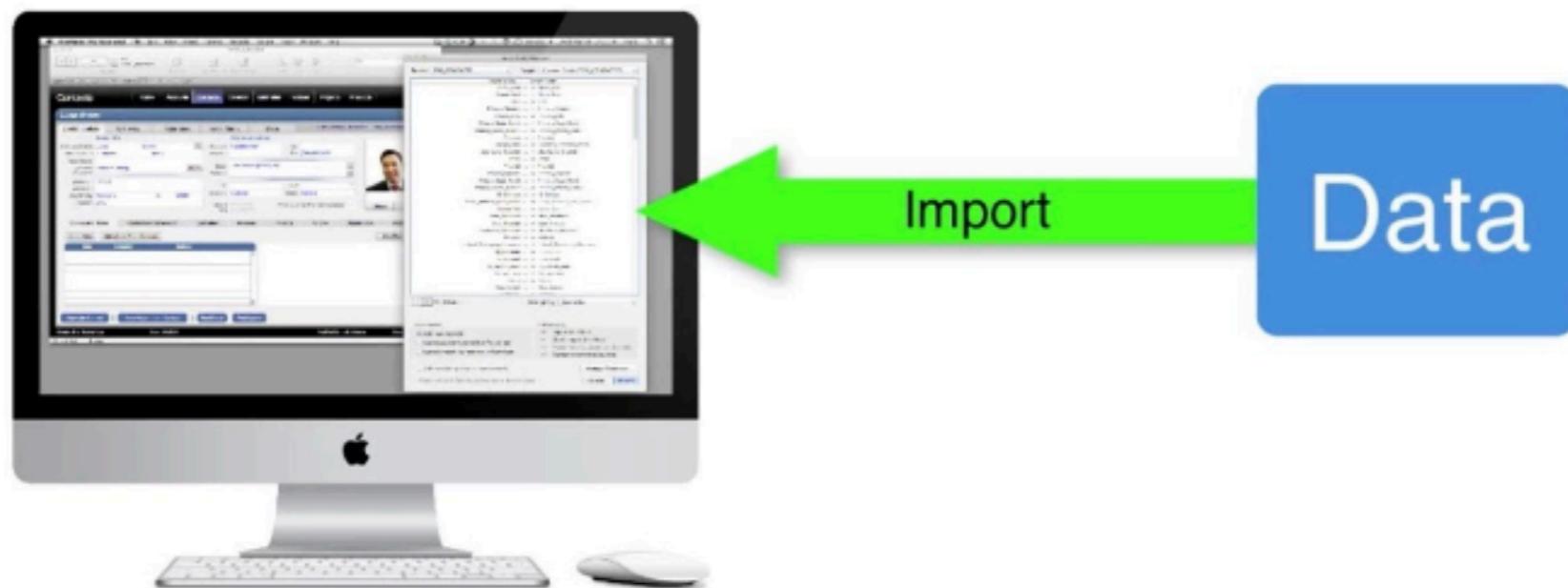
# 10) Guardado de archivos -----
write_csv(df, file.path(params$output_dir, "data_clean.csv"))
saveRDS(mod, file.path(params$output_dir, "model.rds"))
cat("Archivos guardados en:", normalizePath(params$output_dir), "\n")
```

Importar de datos

Importar datos

Cargar datos de un archivo externo a nuestra sesión de R.

IMPORTING DATA



Directorio de trabajo

El directorio de trabajo es la carpeta de la computadora en la cual vamos a estar trabajando.

En esta carpeta deben ir (idealmente) todos los archivos que contienen los datos que vamos a utilizar.

En esta carpeta también se van a guardar los archivos que vamos a generar derivados de nuestro análisis (visualizaciones, otras bases de datos, etc.).

Directorio de trabajo

The screenshot shows the RStudio interface with the following components:

- Top Bar:** Shows the title "Presentación - main - RStudio" and a tab "Presentación — 4. Importar bases de datos".
- Environment Tab:** Displays the message "Environment is empty".
- File Browser:** Shows the directory structure:
 - Home > Documents > GitHub > cursoRBasico > 4. Importar bases de datos > Presentación
 - Files: .Rhistory (20 B, Jul 7, 2021, 12:51 PM), clase4_files (15.7 KB, Mar 2, 2021, 11:34 PM), clase4.html (9.2 KB, Mar 2, 2021, 11:34 PM), clase4.Rmd (205 B, Jul 7, 2021, 1:04 PM), Datos, Keynote, Presentación.Rproj, Untitled.
- Code Editor:** An R Markdown file named "clase4.Rmd" is open, showing code blocks 9 and 10. A large yellow arrow points from the text "La dirección del Directorio de trabajo" to this code.

```
9 css: ["shinobi"]  
10  
11 nature:  
12
```
- Console:** Shows the command "elp." followed by "Type 'q()' to quit" and a prompt "> |".

Ubicaciones de un archivo

Para acceder al contenido de un archivo, el programa debe saber donde se ubica.

Podemos acceder a archivos que se encuentren en:

- A) Tu computadora**
- B) En internet**

Archivos de internet

Si el archivo se encuentra en internet, una de dos:

a) Lo leemos directamente desde internet.

Hay archivos que se pueden leer desde internet sin necesidad de descargarlos a la compu. Esto depende de la función que usemos, del tipo de archivo y de los permisos de acceso (generalmente solo se puede con archivos de texto plano o archivos únicos).

b) Lo descargamos a nuestra computadora.

Lo descargas y ya. Lo lees desde tu compu. Esto pasa generalmente con exedes, archivos shapefile o archivos *.dta.

Archivos locales

Son los archivos que viven en nuestra computadora.

Para acceder a ellos, necesitamos:

- 1) **La función de R correcta (y la librería que la contiene).**
- 2) **La ubicación global o la ubicación local de nuestro archivo.**

Ruta Global y Ruta Relativa

La ruta global y la ruta local son las ubicaciones de un archivo dentro de nuestra computadora.

La ruta global es la ruta absoluta y única de un archivo en la computadora, y es una forma de identificar el espacio en la memoria que ocupa dicho archivo.

La ruta local es la ruta relativa a otra ubicación en nuestra computadora.

Ruta Global y Ruta Relativa



Ruta absoluta



Ruta relativa

Ruta Global y Ruta Relativa

Las ventajas de usar la ruta absoluta es que siempre va a funcionar en tu computadora. Y ya.

La ventaja de la ruta relativa es que la puedes generar de forma en que funcione en tu computadora y en las computadoras de otras personas, lo cual mejora la replicabilidad de tu análisis y permite compartirlo con otras personas.

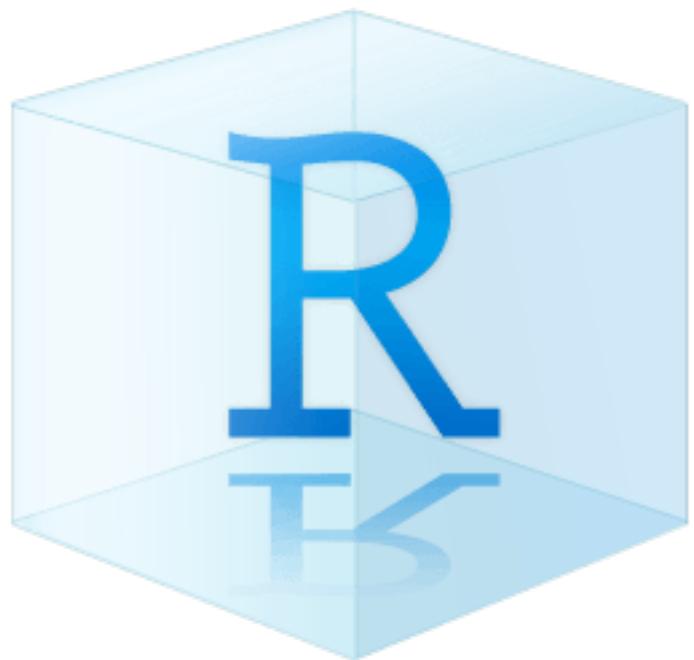
Creando rutas relativas

Hay 3 formas de crear rutas relativas en R:

- 1) Con la función setwd()**
- 2) Con la función here::here()**
- 3) Utilizando archivos de proyecto de RStudio.**

La que más recomendamos es la opción 3.

Archivo de proyecto (.Rproj)



=



Archivos más utilizados

Los archivos más utilizados para guardar datos son los siguientes:



Información tabular

Objetos de R



Información geográfica

Otros formatos

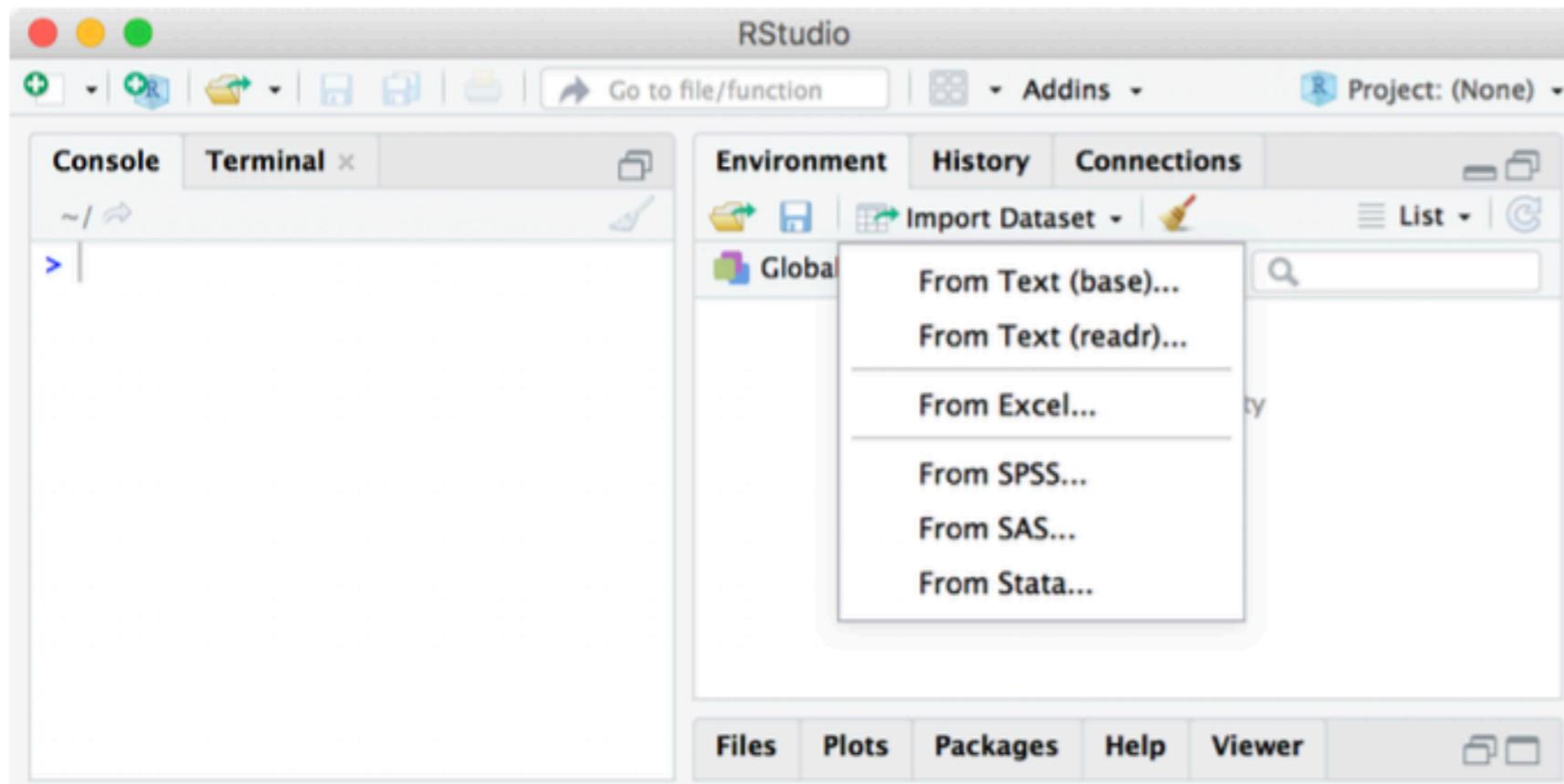
Librerías para importar datos

- `library(readr)`: Archivos texto plano.
- `library(readxl)`: Para excel
- `library(haven)`: Para archivos de Stata, SAS o SPSS
- `library(foreign)`: Para archivos de Stata, SAS o SPSS
- `library(sf)`: Leer archivos geográficos.

Formas de importar datos

Podemos importar archivos de dos maneras:

- 1) Con el **asistente**.*
- 2) Con **código puro**.

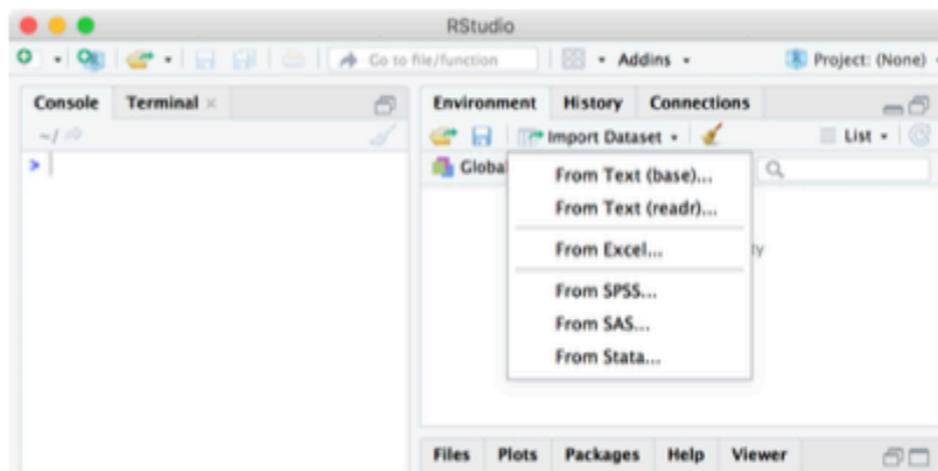


*Sólo para archivos de texto plano, excel, SPSS, Stata y SAS

Formas de importar datos

Método 1: ASISTENTE DE RSTUDIO.

- También puede hacerse el mismo procedimiento a través del Menú:
 - File > Import Dataset > From __ > Browse > Archivo > Import
- **Ventajas:**
 - **Acceso directo al archivo** y R genera la ruta y función correctas
 - Permite **agregar parámetros simples** para la importación (v.g. sheets, skip)
 - **Visualización** de cómo será importado el conjunto de datos
- **Desventajas:**
 - Supone **pasos extra** (copiar y pegar el código) para replicabilidad
 - No permite agregar todos los parámetros (v.g. stringsAsFactors, encoding)



Formas de importar datos

Método 2: SÓLO CON CÓDIGO

- Generando el código directamente:
 - Requiere cargar las librerías específicas y establecer la ruta del archivo
- Ventajas:
 - Puedes importar múltiples archivos a la vez y de todos los formatos posibles
 - Agregar argumentos específicos para una importación más sofisticada
 - Incluir importación de archivos dentro de loops o funciones
 - Si te sabes la función, es más rápido de escribir.
- Desventajas:
 - Requiere saber la función exacta
 - Hay que conocer como es el archivo por dentro
 - Pueden ocurrir errores de los que nos hubiéramos percatado con el otro método.

Problemas de importación

1. Encodings raros.

El **encoding** es el conjunto de símbolos utilizados por un programa para desplegar texto.

Tu sesión de RStudio trabaja por default con un encoding que **podría ser distinto** al que utilizan en otras oficinas/empresas, al que utilizan otros programas o al que usan en otros países.

Por esto mismo, **puede ser necesario el modificar el encoding** utilizado al abrir un archivo o base de datos.

Problemas de importación

2. Errores humanos.

Los más comunes son:

- 1) **Escribiste mal la ruta** del archivo y R no lo encuentra.
- 2) **No conoces bien el archivo** y no te saltaste las líneas necesarias para que se abriera bien.
- 3) **No especificaste la hoja correcta de excel** y te abrió la primera
- 4) No especificaste el **encoding correcto**.

Fin de la clase