

Recuperación y análisis de texto con R

Licenciatura en Ciencia Política (FCS-UdelaR)

Mag. Elina Gómez (UMAD)

elina.gomez@cienciassociales.edu.uy

www.elinagomez.com



Este trabajo se distribuye con una licencia Creative Commons Attribution-ShareAlike 4.0 International License

Aspectos generales

- Curso presencial (Sala C3)
- Miércoles 10 a 12hs.
- [Espacio virtual EVA](#)
- Requisito un 80% de asistencia.

Objetivos del curso

Bases teóricas:

- Contextualizar las **Ciencias sociales computacionales**
- Emergencia de nuevos recursos y técnicas para la investigación social en la era digital.

Objetivos del curso

Generalidades del lenguaje R:

- R como software libre y gratuito
- Comunidades y foros
- Tidyverse
- Manipulación básica de strings

Objetivos del curso

Exploración de fuentes de datos textuales:

- Exploración y obtención de datos de diversa índole, contemplando las diferentes fuentes posibles: OCR, web scraping, prensa digital, redes sociales, audio, Youtube, APIs.

Objetivos del curso

Análisis textual:

- Codificación manual de textos y creación de redes multinivel (categorías, códigos y citas) mediante la plataforma RQDA().
- Abordaje de los requerimientos previos (limpieza y homogeneización) para el análisis de textos.
- Trabajo con minería de textos, el cual se centrará en la noción de *corpus* y sus posibilidades analíticas, desde lo más descriptivo a la aplicación de técnicas más complejas.

Objetivos del curso

Análisis textual:

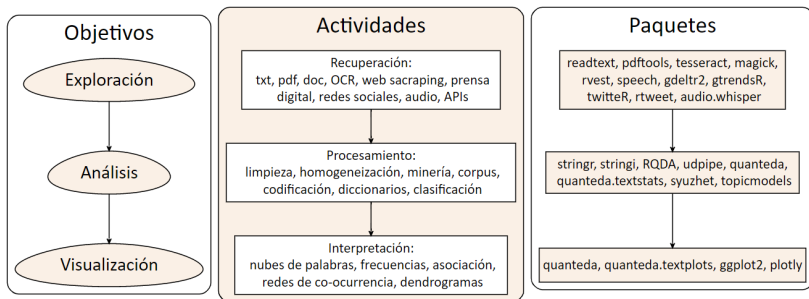
- Trabajo con diccionarios: Introducción al uso de diccionarios (manuales y automáticos), para la clasificación de documentos masivos según intereses particulares.
- Clasificación de textos: clasificación de textos según temas o emociones asociadas a partir de la aplicación de diferentes técnicas existentes.

Objetivos del curso

Visualización:

- Exploración de las diferentes posibilidades gráficas de visualización de los resultados del análisis textual (nubes de palabras, frecuencias, dendrogramas, etc.) y algunos ejemplos de visualización interactiva.

Esquema del curso



Metodología

- El enfoque del curso es práctico (hands-on)
- Trabajaremos con estrategia de live-coding y ejercicios prácticos para cada tema.
- Posibilidad de clonar repositorio GitHub y trabajar con proyecto y control de versiones.
- <https://github.com/elinagomez/analisistextoEPUdelar2023>

Tutorial R+ GitHub

Consejo

- Elegir un tema de interés
- Hacerse una pregunta inicial
- Identificar una fuente textual para responderla

Ideas / Ejemplos

tema	política forestal uruguay
preguntas	¿cómo se desarrolló el debate parlamentario acerca de las modificaciones a la ley forestal? ¿qué palabras fueron más frecuentes? ¿qué temas fueron los preponderantes? ¿qué emociones y sentimientos se desprenden de cada texto? ¿hay diferencias entre los diferentes partidos políticos?
fuentes	diarios de sesiones parlamento

Ideas / Ejemplos

tema	investigación ciencias sociales
preguntas	¿cuáles son los temas predominantes en la investigación en la FCS? ¿qué grado de pluralidad hay en las temáticas de investigación? ¿en qué nivel son citados los investigadores de la FCS?
fuentes	revistas de departamentos de FCS

Ideas / Ejemplos

tema	carnaval murgas
preguntas	¿qué temas preponderan en los textos de murga 2023? Son textos positivos, negativos o neutros? ¿hay diferencias según el puesto de concurso? ¿hay diferencias según participación de mujeres?
fuentes	textos de murgas

Ideas / Ejemplos

tema

conflictividad laboral

preguntas

¿qué características tuvo la
conflictividad laboral en
Uruguay en la segunda quincena
de agosto de 2021?

fuentes

la diaria

Ideas / Ejemplos

tema	estudios constitucionales
preguntas	¿qué similitudes y diferencias tienen las constituciones de Uruguay y Costa Rica en materia de libertad económica y civil?
fuentes	constituciones

Ideas / Ejemplos

tema	género y política
preguntas	¿cómo se componen los discursos parlamentarios alrededor de los productos de gestión menstrual y la menstruación en Uruguay y Colombia?
fuentes	diarios de sesiones, leyes

Ideas / Ejemplos

tema	política usa
preguntas	¿qué sentimientos preponderan en los tweets de Biden y Trump?
fuentes	twitter

Recursos bibliográficos básicos

- Bit by bit (Matthew J. Salganik)
- Data Feminism o Feminismo de Datos (Catherine D'Ignazio y Lauren F. Klein)
- R para Ciencia de Datos (Hadley Wickham y Garrett Grolemund)
- Text Mining with R!(Julia Silge y David Robinson)
- Hojas de ruta en español
- Intro web scraping con R (Riva Quiroga)
- Tutoriales Ciencias Sociales Computacionales - SICSS

Repositorio con recursos varios

Objetivos de hoy

- Introducción conceptual y teórica
- Generalidades de la exploración y manipulación de datos con *tidyverse*.
- Manipulación de strings

Introducción conceptual y teórica

Contexto

- El abordaje metodológico y técnico que veremos en el curso se enmarca en las llamadas **ciencias sociales computacionales**

Introducción conceptual y teórica

"Una nueva disciplina como las Ciencias Sociales Computacionales (Cioffi-Revilla, 2017; Lazer et al., 2009; Mann, 2016) que aglutina la aplicación tanto de las ciencias de la complejidad como del análisis computacional en las Ciencias Sociales, aporta un enfoque innovador para la comprensión del comportamiento humano y social (. . .) Se promueven así no solo otro tipo de preguntas, sino también la cooperación y la colaboración entre disciplinas. Asimismo, se pueden explorar también nuevas hipótesis a partir de la disponibilidad de nuevos conjuntos de datos y nuevas capacidades de exploración de estos. El desarrollo de nuevos paradigmas y la colaboración entre diferentes campos de conocimiento tienen el potencial de promover nuevos escenarios de investigación, evitando la reiteración en construcciones interpretativas preexistentes (Cioffi-Revilla, 2017; Conte et al., 2012)." (Caro, Jorge et al., 2020)

Introducción conceptual y teórica

Características:

- Perspectiva transdisciplinar
- Dialogo con nuevos desarrollos vinculados a la ciencia de datos, el aprendizaje automático o estadístico, modelado, análisis espacial.
- Explotación del *big data* como fuente de datos y nuevos campos de estudio.

Sobre Big data

“Researchers are in the process of making a change akin to the transition from photography to cinematography.” (Salganik, 2018)

- Bit by Bit: Social Research in the Digital Age (Matthew J. Salganik, 2018). [Reseña en español](#)
- Big Data: datos en línea (búsquedas y rrss); registros administrativos.

Sobre Big data

- Diez características según Salganik (2018): *grande; siempre encendido; no reactivo; incompleto; inaccesible; no representativo; a la deriva; algorithmically confounded; sucio; y sensible.*
- Procesamiento: *“contar cosas; pronosticar cosas; y aproximar experimentos”.*
- Complementariedad con técnicas tradicionales (encuestas): preguntas enriquecidas y pregunta amplificada.
- Nuevas formas de colaboración (crowd-sourcing) y comunidades de desarrollo.
- Desafíos éticos, cajas negras

Panorama regional

- En América Latina existen diversos grupos académicos que trabajan esta línea ya sea desde las llamadas *humanidades digitales* o de las *ciencias sociales computacionales*

Panorama regional

Algunas iniciativas regionales interesantes:

- Programming Historian
- Laboratorio de Humanidades Digitales - UFBA
- Observatorio de conflictividad - UNMdP
- Nucleo de Innovación Social - Colegio de Sociólogos de la Provincia de Buenos Aires
- factor~data - Escuela IDAES (UNSAM)
- Observatorio de Redes
- Instituto Milenio Fundamento de los Datos
- Maestría en Humanidades Digitales - Uniandes

R

- ¿Qué es R?
- Consejos
- Generalidades de tidyverse
- Manipulación de strings

¿Qué es R?

- Un lenguaje de programación y un programa estadístico.
- Es **software libre**: se distribuye con licencia [GNU General Public License](#) que implica libertad de uso, modificación y distribución.
- Es **gratuito**, se descarga desde el [CRAN](#).
- Hay una **comunidad** mundial que usa R y lo mejora constantemente, hoy hay más de 10.000 [paquetes](#) disponibles para descargar
- Usuarios se ayudan entre sí: [stackoverflow](#), [talkstats](#), ([rusers](#)) y localmente [meetup R-Ladies Montevideo](#).

Consejos

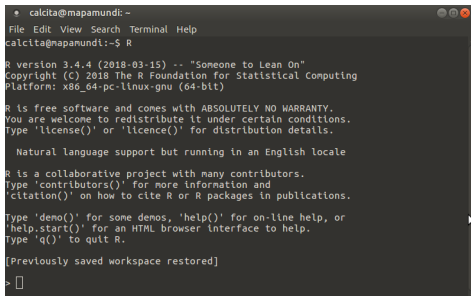
La curva de aprendizaje de R al comienzo suele resultar muy empinada. ¿Cómo podemos evitar o superar la frustración?

- **Usá** R a diario.
- **Traducí** a R una sintaxis sencilla de otro programa que conozcas.
- Recurrí a los **foros** y a la ayuda de R para encontrar las soluciones a los problemas que te surjan: stackoverflow
- Recurrí a otrxs **usuarixs** de R que conozcas.
- Prestá atención a los **mensajes** de error y advertencia.
- **Escribí** tus sintaxis en un script y **comentalas** detalladamente.
- **Reutilizá** sintaxis existentes.

[Hoja de ayuda de R](#)

Introducción a R: ¿cómo interactuamos con R?

- Abrimos R.
- Aparece un mensaje de apertura.
- Debajo el 'prompt' que es el símbolo `>` ('mayor') e indica que R está listo para recibir órdenes.



```
calcita@mapamundi: ~  
File Edit View Search Terminal Help  
calcita@mapamundi:~$ R  
  
R version 3.4.4 (2018-03-15) -- "Someone to Lean On"  
Copyright (C) 2018 The R Foundation for Statistical Computing  
Platform: x86_64-pc-linux-gnu (64-bit)  
  
R is free software and comes with ABSOLUTELY NO WARRANTY.  
You are welcome to redistribute it under certain conditions.  
Type 'license()' or 'licence()' for distribution details.  
  
Natural language support but running in an English locale  
  
R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to cite R or R packages in publications.  
  
Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.  
  
[Previously saved workspace restored]  
  
> 
```

Introducción a R: ¿cómo interactuamos con R?

- Escribiendo un código en la consola
- O escribiendo en el script y luego enviando el código a la consola para obtener un resultado: CTRL + ENTER
- El código utiliza una serie de operadores ('+', '*', '%in%', etc.), funciones, números, caracteres, etc.
- El script se guarda en un archivo .R

Introducción a R: ¿cómo interactuamos con R?

- Las órdenes elementales de R consisten en expresiones o asignaciones.
- Una **expresión**, se evalúa, se imprime el resultado y su valor se pierde.

```
2 + 3
```

```
## [1] 5
```

Introducción a R: ¿cómo interactuamos con R?

- Una **asignación**, crea un objeto y no se imprime el resultado.

```
x <- 2 + 3
```

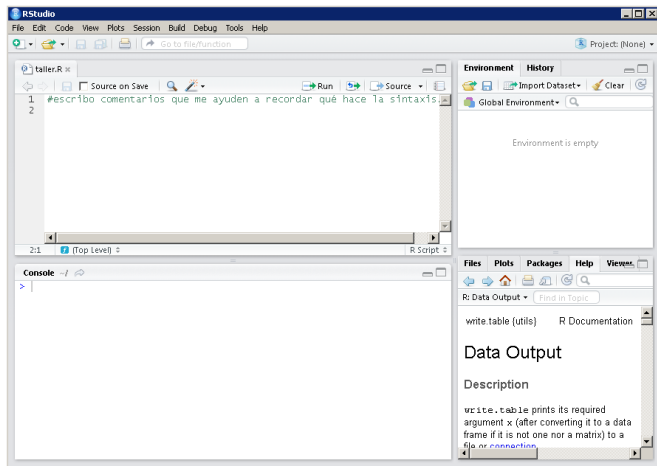
- Una asignación se hace utilizando el símbolo: <- o también con el símbolo =
- Mientras que otros programas estadísticos muestran directamente los resultados de un análisis, R guarda estos resultados como un 'objeto'.

Introducción a R

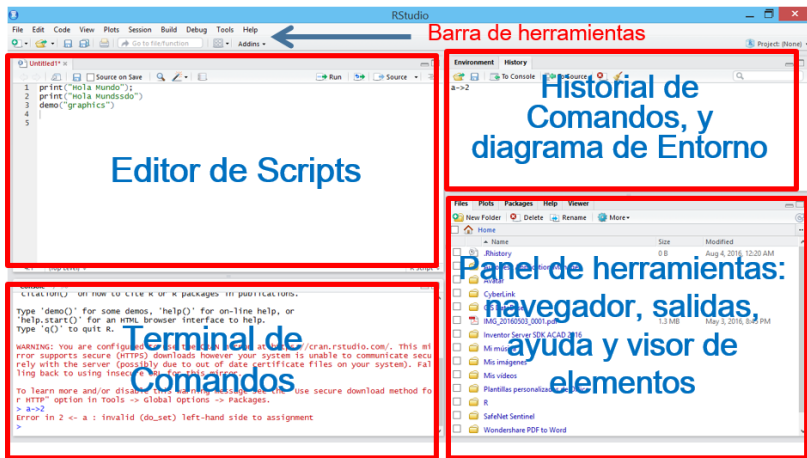
- R es un programa '**orientado a objetos**': variables, datos, funciones, resultados, etc., se guardan en la memoria RAM en forma de objetos con un nombre específico sin usar archivos temporales.
- Estos objetos se pueden modificar o manipular con **operadores** –lógicos, aritméticos, comparativos y especiales– y **funciones** –que a su vez son objetos–.
- Bajo este término se esconde la simplicidad y flexibilidad de R.

Introducción a RStudio

R es mucho más amigable utilizando un editor de código, elegimos **RStudio** porque da muchas facilidades.



Introducción a RStudio



Introducción RStudio

RStudio se organiza en 4 ventanas:

- **Ventana superior izquierda:** se escribe el documento, sintaxis, etc.
- **Ventana inferior izquierda:** se ubica la consola donde se imprimen resultados
- **Ventana superior derecha:** se muestran los objetos creados y el historial de comandos ejecutados. También se mostrarán los archivos modificados de un proyecto de control de versiones
- **Ventana inferior derecha:** se muestran los archivos, gráficos, paquetes y ayuda

Introducción RStudio

- Para ejecutar una línea de código, colocar el cursor sobre esa línea y ejecutar **Ctrl + ENTER**.
- Para ejecutar varias líneas de código, debemos seleccionarlas todas y ejecutar **Ctrl + ENTER**.
- El símbolo **>** en la consola, indica que R está listo para recibir un comando, mientras que si aparece el símbolo **+**, hay una sentencia no finalizada.
- Mensajes de error (**Error**): errores en la sintaxis (no ejecuta la/s líneas erróneas, pero sí el resto)
- Mensajes de advertencia (**Warnings**): no necesariamente hay un error (ejecuta los comandos y solo te advierte de posibles inconvenientes).
- Importa saber qué tipo de objetos son los que estamos trabajando: no toda función se puede aplicar a cualquier tipo de objeto.

Creación de objetos

- Al nombrar archivos u objetos, evitar usar tildes y ñ. No dejar espacio entre caracteres.
- Nombre de un objeto debe comenzar con una letra (A-Z, a-z)
- Puede incluir letras, dígitos (0-9), puntos (.) y guión bajo (_).
- R discrimina entre letras mayúsculas y minúsculas: x y X son objetos diferentes.
- Son nombres válidos *raiz2* o *raiz_2*, no así *2raiz* ni *raiz-2*

Objetos: tipos de elementos y clases de objetos

Vector (contiene **elementos** de un **mismo tipo**) []

```
a=c(1,0,1,1,1,0) # vector numérico
```

```
b=c("1","0","1","1","1","0") # vector caracter
```

```
d=c(T,F,T,T,T,F) # vector lógico
```

```
f=factor(b,levels=c(1,0),labels=c('Sí','No')) # factor
```

Matriz (contiene **vectores** de una **misma clase**) [,]

```
e=matrix(c(1,0,1,1,1,0),nrow=3) # matriz numérica
```

Data Frame (contiene **vectores** de **cualquier clase**) \$

```
g=data.frame(e) # marco de datos, base de datos
```

Lista (contiene **objetos** de **cualquier clase**) [[]]

```
e=list(g,f) # lista
```

Objetos: tipos de elementos y clases de objetos

Todo objeto pertenece a un tipo (`typeof()`), pero a su vez, tiene instancias de una o varias clases (`class()`), con determinados atributos (`attributes()`), y tiene una estructura específica (`str()`).

Objeto	clase	tipo
<code>x <- 1</code>	numeric	double
<code>x <- 'hola'</code>	character	character
<code>x <- TRUE</code>	logical	logical
<code>x <- iris</code>	data.frame	list

Objetos: tipos de elementos y clases de objetos

Tipo	Verificación	Cambio
vector	<code>is.vector</code>	<code>as.vector</code>
list	<code>is.list</code>	<code>as.list</code>
data.frame	<code>is.data.frame</code>	<code>as.data.frame</code>
matrix	<code>is.matrix</code>	<code>as.matrix</code>
logical	<code>is.logical</code>	<code>as.logical</code>
factor	<code>is.factor</code>	<code>as.factor</code>
character	<code>is.character</code>	<code>as.character</code>
numeric	<code>is.numeric</code>	<code>as.numeric</code>
double	<code>is.double</code>	<code>as.double</code>

Funciones

¿Qué es una función?

- Una función es un conjunto de instrucciones que operan sobre unos argumentos y producen un resultado.
- Una función esconde líneas de código que permite reutilizarlo una y otra vez de manera sencilla.
- Las funciones tienen nombres descriptivos -en inglés- y se acompañan de paréntesis curvos.
- Dentro de los paréntesis se definen los valores de sus argumentos.
- La mayoría tiene al menos un argumento obligatorio y el resto con valores por defecto.

Funciones

- La función `setwd()` tiene como único argumento *'dir'* y es obligatorio asignarle un valor que debe ser una ruta a una carpeta específica en la computadora:

```
# define el directorio de trabajo  
#setwd(dir = 'home/username/Desktop/')
```

Algunas funciones no necesitan que se defina ningún argumento: `getwd()`, `dir()`, `ls()`

Carga de datos R

- `load(file = "ruta/archivo.Rdata")` *Carga archivo en formato .RData*
- `save(obj1, obj2, obj3, file = "ruta/archivo.Rdata")` *Respalda algunos objetos en formato .RData*
- `save.image("ruta/respaldo.RData")` *Respalda todo el entorno de trabajo*

A diferencia de otras funciones para cargar archivos, la función `load()` es la única en que no se debe asignar a un objeto, ya que para que un archivo sea guardado como Rdata debió haber sido un objeto de R y por ende, al cargar el archivo se carga en la memoria el objeto asociado.

Carga de datos csv o txt

Las rutas deben escribirse con la barra / y no la contrabarra \. Si el archivo está en el directorio de trabajo, alcanza solo con llamar al archivo sin especificar la ruta.

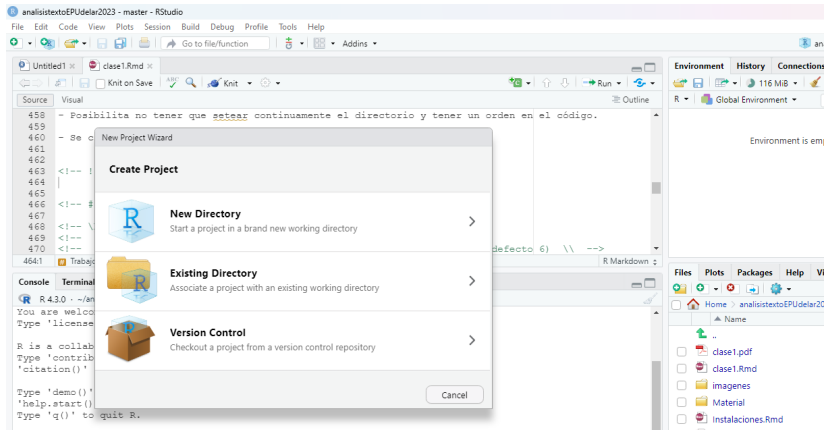
Existen paquetes que facilitan la carga de datos, un ejemplo es [rio](#)

```
datos <- read.table(file="ruta/archivo.txt", header=FALSE,
  sep=" ")# otros posibles valores de sep= son ",", ";" o "/t"
datos <- read.csv(file="ruta/archivo.csv", header=TRUE,
  sep=",")
datos <- read.delim(file="ruta/archivo.txt", header = FALSE,
  sep = "/" )
```

Trabajo con proyectos

- Se recomienda el trabajo por **Proyectos** ya sea a nivel local o conectando con repositorio Git.
- Posibilita no tener que setear continuamente el directorio y tener un orden en el código.
- Se crea desde la barra de herramientas: File > New Project
- Se aloja en un archivo *.Rproj*
- [Tutorial R + GitHub](#)

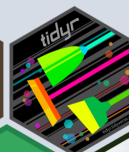
Trabajo con proyectos



Ejercicio 1: Trabajo con proyectos

- 1 Crear un proyecto que se llame *CursoRAT2024* (.Rproj)
- 2 Crear una carpeta que se llame *Material* y otra *Datos*
- 3 Cargar en la carpeta Datos, la base world en RData que está en EVA.

Generalidades de tidyverse



Generalidades de tidyverse

- En R, paquetes como **ggplot2**, **dplyr** y **stringr** han adquirido gran popularidad, y ahora forman parte de **tidyverse**
- Estos paquete se basan en los principios de 'tidy data' para mayor consistencia y velocidad de procesamiento:
- Cada Variable forma una columna.
- Cada observación forma una fila.
- Cada tipo de unidad observacional forma una tabla.

Generalidades de tidyverse

Recomendamos la consulta:

- [Hoja de ruta de dplyr en español](#)
- [Hoja de ruta de ggplot2 en español](#)
- [Hoja de ruta stringr en español](#)

Selección por atributos

- *Base R* incluye las funciones de selección `[`, `subset()` y `$`. y **dplyr** aporta `select()`, `slice()`, `filter()`, and `pull()`.

Selección por atributos

- `select()` selecciona columnas por nombre o por posición.
- `pull()` extrae una columna como un vector
- `slice()` es el equivalente de `select()` pero para las filas.
- `filter()` es el equivalente en **dplyr** al `subset()` del *base* de R.
- Otras funciones: `contains()`, `starts_with()` y `num_range()`

Conectando funciones

- Un beneficio de **dplyr** es su compatibilidad con el operador `%>%`.
- Este 'pipe' de R, toma su nombre del pipe de Unix `|`, y es parte del paquete **magrittr**
- Su función es “conectar” la salida de un comando anterior al primer argumento de la siguiente función.
- Esto permite encadenar comandos de análisis de datos, pasando el marco de datos de una función a la siguiente.
- Una ventajas adicional es que fomenta la adición de comentarios a funciones autónomas y permiten líneas simples comentadas sin romper el código.

Agregación de atributos: ejemplo

- Las operaciones de agregación resumen los conjuntos de datos por una 'variable de agrupación' (típicamente una columna de atributo) o un objeto espacial.
- Para calcular el número de personas por continente en base al objeto `world`
- usaremos la columna `pop` que contiene la población por país y la variable de agrupación `continent`.
- En la base R esto se hace con `aggregate()` y `sum()` de la siguiente manera:

Agregación de atributos

```
aggregate(pop ~ continent,  
           FUN = sum,  
           data = world,  
           na.rm = TRUE)
```

Agregación de atributos

`summarize()` es el equivalente en **dplyr** de `aggregate()`, y usa la función `group_by()` para agrupar la variable. Y se implementaría así:

```
group_by(world, continent) %>%  
  summarize(pop = sum(pop, na.rm = TRUE))
```

Agregación de atributos

- Este enfoque es flexible, lo que permite nombrar las columnas resultantes.
- El omitir la variable de agrupación pone todo en un grupo.
- Esto significa que `summarize()` se puede usar para calcular la población total de la Tierra (~ 7 mil millones) y el número de países.
- Utilicemos `sum()` and `n()` para generar las columnas `pop` y `n_countries`

Agregación de atributos

```
world %>%  
  summarize(pop = sum(pop, na.rm = TRUE),  
            n_countries = n())
```

Agregación de atributos

Combinemos todo lo anterior para identificar los 3 continentes más poblados (usando `dplyr::n()`) y el número de países que contienen:

```
world %>%  
  dplyr::select(pop, continente = continent) %>%  
  group_by(continente) %>%  
  summarize(población = sum(pop, na.rm = TRUE),  
            n_paises = n()) %>%  
  slice_max(n = 3, order_by = población)
```


Agregación de atributos

Table 10: Los 3 continentes más poblados, y su número de países.

continente	población	n_paises
Africa	1147005839	51
Asia	4306025131	47
Europe	739178065	39

Ejercicio 2: Tidyverse

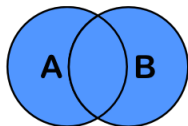
Tidyverse

- 1 Cargar la base world en formato .RData
- 2 Imprimir una tabla con los tres continentes con mayor territorio

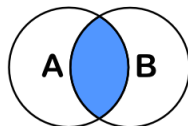
Combinando objetos

- Join combina tablas basadas en una variable compartida ("key")
- **dplyr** presenta varias funciones para ello:
 - `left_join()` - Une las filas coincidentes de b en a
 - `right_join()` - Une las filas coincidentes de a en b
 - `inner_join()` - Une reteniendo solo las filas de ambos conjuntos,
 - `full_join` - Une los datos conservando todos los valores, todas las filas
 - `semi_join()` - Todas las filas en a que tienen una coincidencia en b
 - `anti_join()` - Todas las filas en a que no tienen una coincidencia en b
- Estos nombres de funciones siguen las convenciones utilizadas en el lenguaje de bases de datos [SQL](#).

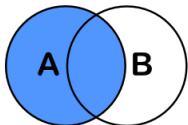
Combinando objetos



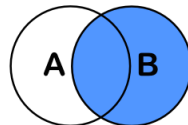
full_join(A,B)



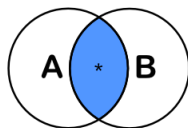
inner_join(A,B)



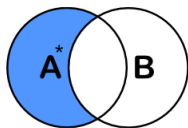
left_join(A,B)



right_join(A,B)



semi_join(A,B)



anti_join(A,B)

Combinando objetos

- Las funciones “..._join” de **dplyr** trabaja tanto con `data.frames`
- El orden de los factores altera el resultado... la clase del primer objeto es la que conserva el resultado.
- Nos centraremos en los `left` e `inner` “..._join” que son los más utilizados, que utilizan la misma sintaxis que los otros tipos de unión.

Combinando objetos

dplyr	base
<code>left_join(x, y)</code>	<code>merge(x, y, all.x=TRUE)</code>
<code>right_join(x, y)</code>	<code>merge(x, y, all.y=TRUE)</code>

Manipulación de *strings*

Antes de realizar un análisis o de construir un modelo de aprendizaje, la discusión de datos es un paso crítico para preparar los datos de texto sin procesar en un formato apropiado.

El texto puede ser considerado como una colección de documentos y un documento puede ser analizado en cadenas.

En la limpieza de texto, los patrones de búsqueda se definen en expresiones regulares (abreviadas como `regex()` o `regexp()`) para “encontrar y eliminar” o “buscar y reemplazar” cadenas.

- No es simple ni intuitivo trabajar con expresiones regulares, [acá](#) pueden ver un taller realizado por Stephanie Orellana y Riva Qiroga

Stringr

- Para manipulación de strings hay varias funciones básicas de *R base*: `nchar()`, `paste()`, `grep()`, `grepl()`, `sub()`, `gsub()`, `substr()`, `substring()`, `regexec()`, entre otras
- Sin embargo, en el curso veremos principalmente las funciones de manipulación de strings que se incluyen en la librería [stringr](#) de Tidyverse.
- Una de las ventajas de stringr es que todas la funciones comienzan con `str_` a diferencia del base que hay que memorizar distintos nombres



Stringr

- [Hoja de ruta stringr](#)
- En esta [Introducción a la ciencia de datos](#) encuentran una buena introducción al procesamiento de cadenas



Stringr

Funciones para separar un texto:

str_split

```
x <- "Este es un curso de Recuperacion y analisis de texto con R"

# divido por espacio en blanco
str_split(x, " ") # string, pattern

#> [[1]]
[1] "Este" "es" "un" "curso" "de" "Recuperacion" "y" "analisis" "de"
[10] "texto" "con" "R"

# la cadena puede ir en el argumento string sin crear un objeto previo
str_split("Este es un curso de Recuperacion y analisis de texto con R", " ")

# variante con regex
str_split(x, "\\s") # string, pattern
# otra variante divido por palabra
str_split(x, boundary("word")) #string, boundary()
```

Stringr

Funciones para combinar un texto:

`str_c`

```
x <- "Este es un curso de Recuperacion y analisis de texto con R"
y <- "Es un curso de educación permanente."

# combino los vectores con un punto seguido de un espacio
str_c(x, y, sep = ". ") # string1, string2, separador

#> [1] "Este es un curso de Recuperacion y analisis de texto con R. Es un curso de educación permanente."
```

Stringr

Funciones para reemplazar un texto:

str_replace

```
x <- "Este es un curso de Recuperacion y analisis de texto con R"

# reemplazo primera ocurrencia de "de" por un .
str_replace(x, "de", ".") # string, pattern, replacement

#> [1] "Este es un curso . Recuperacion y analisis de texto con R"

# reemplazo todas las ocurrencias de "de" por un .
str_replace_all(x, "de", ".") # string, pattern, replacement

#> [1] "Este es un curso . Recuperacion y analisis . texto con R"
```

Stringr

Funciones para pasar mayúscula/minúscula:

`str_to_upper` / `str_to_lower`

```
x <- "Este es un curso de Recuperacion y analisis de texto con R"

str_to_upper(x) # string
#> [1] "ESTE ES UN CURSO DE RECUPERACION Y ANALISIS DE TEXTO CON R"

str_to_lower(x) # string
#> [1] "este es un curso de recuperacion y analisis de texto con r"
```

Stringr

Funciones para eliminar espacios en blanco:

`str_trim`

```
x <- " Este es un curso de Recuperacion y analisis de texto con R "
```

elimino espacios al inicio de la cadena

```
str_trim(x, side = "left") # string, side = c("both", "left", "right")
```

```
#> [1] "Este es un curso de Recuperacion y analisis de texto con R "
```

elimino espacios al inicio y final de la cadena

```
str_trim(x) # por defecto side = both
```

```
#> [1] "Este es un curso de Recuperacion y analisis de texto con R"
```

Stringr

Funciones para eliminar espacios en blanco:

`str_squish`

```
# elimino espacios al inicio y final de la cadena y reemplaza a un espacio el resto  
# str_squish(string)  
str_squish(" Este es un  curso de Recuperacion  y analisis de texto con R\t")  
  
#> [1] "Este es un curso de Recuperacion y analisis de texto con R"
```

Stringr

Ejemplo extrayendo texto de internet:

```
library(readtext)
url_texto <- "https://www.ingenieria.unam.mx/dcsyhfi/material_didactico/Literatura_Hispanoamericana_Conter
# Extraemos el texto
mario <- readtext(url_texto)

library(stringr)

# divido el texto en oraciones

# usando %>%
mario_sentencias <- str_split(mario$text, boundary("sentence"))%>% # divido el texto en oraciones
  unlist()%>% # convierto el texto en un vector
  str_trim("both") # elimino espacios

# sin %>%
mario_sentencias2 <- str_trim(unlist(str_split(mario$text, boundary("sentence"))), "both")

# compruebo que son iguales
identical(mario_sentencias, mario_sentencias2)
#> [1] TRUE
```

stringr, R-base y stringi

- `stringr` tiene muchas más funciones, algunas sin equivalente directo en R-base [Comparativo stringr - base](#)
- `str_view()`, `str_detect()`, `str_extract()`, `str_sub()`, `str_count()` son algunas de las funciones más útiles para el procesamiento
- Además existe el paquete [stringi](#) que tiene aún más funciones que `stringr`

Ejercicio 3: stringr

Manipulación de strings

- 1 Cargue, extraiga de internet o cree una cadena de texto
- 2 Aplique, de forma separada, al menos tres funciones de stringr
- 3 Simplifique el ejercicio anterior utilizando el `%>%`

Caracteres especiales

Para construir consultas que incluyan metacaracteres, i.e.

`\\$ * \\+ \\. \\? \\[\\\\ \\. \\^ \\. \\{ \\. \\| \\. \\(\\. \\)`

Se debe agregar una retrobarra `\\`

Metacaracteres especiales

`\\t` : Tabulador

`\\n` : Nueva línea

`\\v` : Tabulación vertical

`\\f` : Salto de formulario

`\\r` : Salto de línea

Cuantificadores

Los Cuantificadores especifican cuantas veces el patrón consultado pueda ocurrir.

$*$: coincide al menos 0 veces.

$+$: coincide al menos 1 vez.

$?$: coincide a lo sumo 1 vez.

$\{n\}$: coincide exactamente n veces.

$\{n, \}$: coincide al menos n veces.

$\{, m\}$: coincide a lo sumo m vez.

$\{n, m\}$: coincides entre n y m veces.

Cuantificadores

El cuantificador refiere al caracter inmediatamente anterior

■ Ejemplos

```
str_detect(string, pattern)
```

```
vec <- c("AB", "A1B", "A11B", "A111B", "A1111B", "A2B", "A1q2")
```

```
str_detect(vec, "A1*B") # `*` : coincide al menos 0 veces.  
#> [1] TRUE TRUE TRUE TRUE TRUE FALSE FALSE
```

#detecta coincidencia siempre que a una A le siga una B
#y en los casos en que una A y una B estén separadas por 1 o muchos 1.
#Si hay algún otro caracter en el medio no detecta coincidencia

```
str_detect(vec, "A1+B") # `+` : coincide al menos 1 vez.  
#> [1] FALSE TRUE TRUE TRUE TRUE FALSE FALSE  
#detecta coincidencia siempre que una A y B estén separadas por 1 o muchos 1.  
#Si hay ninguno o algún otro caracter en el medio no detecta coincidencia
```

```
str_detect(vec, "A1?B")  
#> [1] TRUE TRUE FALSE FALSE FALSE FALSE FALSE  
#detecta coincidencia siempre que una A y B estén juntas o separadas por un 1.  
#Si algún otro caracter o más de un 1 en el medio no detecta coincidencia
```

Posición

`^` : Al inicio de la cadena.

`$` : Al final de la cadena.

`\b` : En los extremos de la palabra.

`\B` : No en los extremos de la palabra.

`\<` : Al inicio de la palabra.

`\>` : Al final de la palabra.

Posición

■ Ejemplos

```
str_detect(string, pattern)

vec <- c("abxxxx", "xxxxabxxxx", "xxxxxab")

str_detect(vec, "^ab")
#> [1] TRUE FALSE FALSE
#detecta coincidencia sólo cuando "ab" está al inicio de la cadena

str_detect(vec, "ab$")
#> [1] FALSE FALSE TRUE
#> #detecta coincidencia sólo cuando "ab" está al final de la cadena
```

Caracteres por clases

`[[:digit:]]` or `\\d` or `[0-9]` : dígitos 0 1 2 3 4 5 6 7 8 9

`\\D` or `[^0-9]` : no-dígitos

`[[:lower:]]` or `[a-z]` : letras minúsculas

`[[:upper:]]` or `[A-Z]` : letras mayúsculas

`[[:alpha:]]` or `[[:lower:]][[:upper:]]` or `[A-z]` :
caracteres alfabéticos

Caracteres por clases

`[[:alnum:]]` or `[[:alpha:]][[:digit:]]` or `[A-z0-9]`:
caracteres alfanuméricos

`\\w` or `[[:alnum:]]_` or `[A-z0-9_]` : caracteres de palabra

`\\W` or `[^A-z0-9_]` : No caracteres de palabra

`[[:xdigit:]]` or `[0-9A-Fa-f]` : dígitos hexadecimales (base 16) 0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f

Caracteres por clases

■ Ejemplos

```
str_detect(string, pattern)

vec <- c("12345", "hola", "HOLA", "Hola", "Hola12345", "$#&/(#)")

str_detect(vec, "[[:alnum:]]")
#> [1] TRUE TRUE TRUE TRUE TRUE FALSE
#detecta coincidencia si hay caracteres alfanuméricos

str_detect(vec, "[[:digit:]]")
str_detect(vec, "\\d")
str_detect(vec, "[0-9]")
#> [1] TRUE FALSE FALSE FALSE TRUE FALSE
#detecta coincidencia si hay números

str_detect(vec, "[[:alpha:]]")
str_detect(vec, "[A-z]")
#> [1] FALSE TRUE TRUE TRUE TRUE FALSE
#detecta coincidencia si hay letras

str_detect(vec, "[A-Z]")
#> [1] FALSE FALSE TRUE TRUE TRUE FALSE
#detecta coincidencia si hay letras mayúsculas

str_detect(vec, "[a-z]")
#> [1] FALSE TRUE FALSE TRUE TRUE FALSE
#detecta coincidencia si hay letras minúsculas
```

Caracteres y espacios

`[[:blank:]]` : espacios y tabulaciones

`[[:space:]]` or `\s` : todo tipo de caracteres de espaciado:
tabulación, nueva línea, tabulación vertical, salto de formulario,
Salto de linea, espacios

`\\S` : Caracteres que no sean espacios

`[[:punct:]]` : caracteres de puntuación ! " # \$ % & () * +
, - . / : ; < = > ? @ [] ^ _ { | } ~

Caracteres y espacios

`[[:graph:]]` or `[[:alnum:]][[:punct:]]` : caracteres gráficos (legibles)

`[[:print:]]` or `[[:alnum:]][[:punct:]]\\s` : caracteres sin espacios

`[[:cntrl:]]` or `\\c` : caracteres de control, como `\\n` or `\\r`, etc.

Expresiones regulares

Las expresiones regulares mas complejas combinan los ejemplos anteriores (y otros)

■ Ejemplo:

Necesito una expresión regular que detecte solo el quinto elemento de este vector

```
vec <- c("12312342312345", "hola", "HOLA", "Hola", "Hola12345", "$#&/(#")  
  
str_detect(vec, "^[A-Z].*\\d$") # inicio mayuscula, fin número, pueden existir caracteres en medio  
#> [1] FALSE FALSE FALSE FALSE TRUE FALSE
```