

Análisis de datos en fortificación de alimentos a gran escala con R

Tema I: Introducción a R y RStudio.

Dr. Maicel Monzón

Sumario

- Características generales del lenguaje R
- Creación, listado y remoción de objetos en memoria
- Objetos y tipos de datos
- Obtener ayuda

Porqué es importante esta conferencia ?

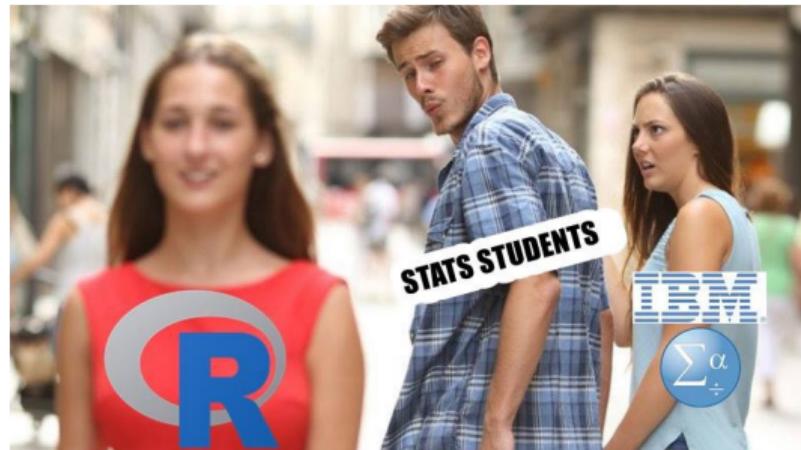
Sienta las **bases** para importar, ordenar y transformar datos, así como para crear **tablas** y **gráficos**

```
# Ejemplo: Crear un objeto y uso de vectores como argumentos
datos <- read_delim(file = "./static/csv/datos_fortificacion.csv")
# definir argumentos de una función a partir de vectores tipo cadena
tbl_summary(data=datos ,by = "area", include = c("hierro_mg","zinc_mg"))
```

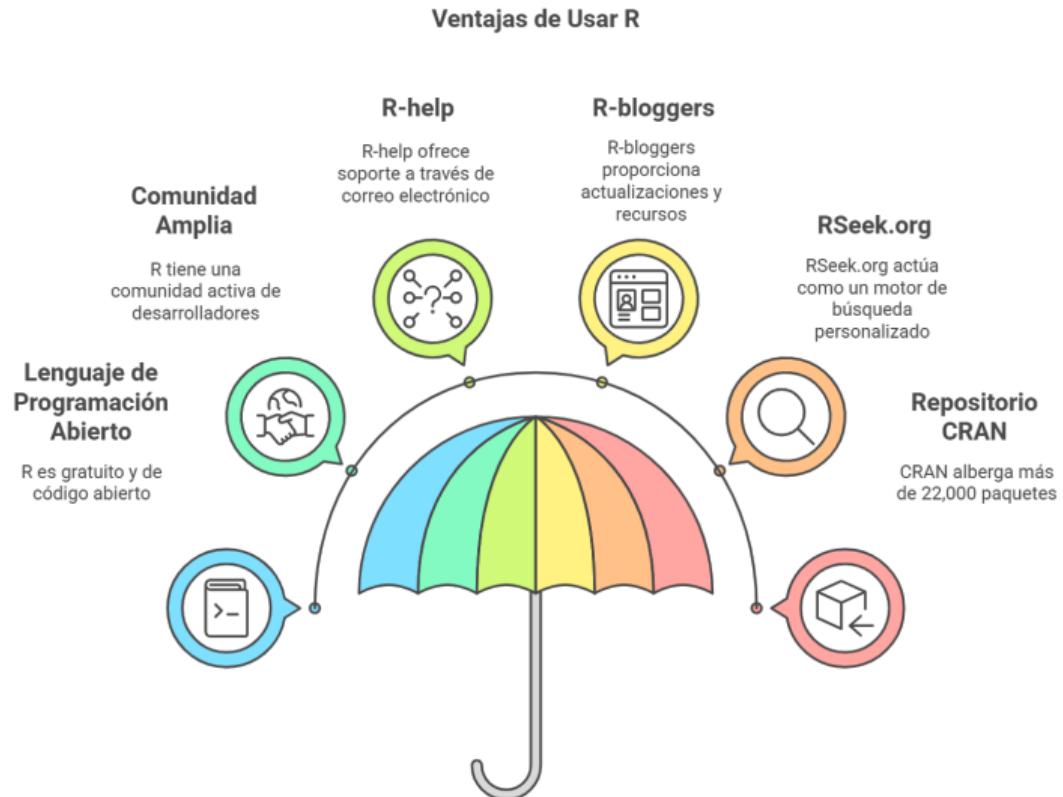
Characteristic	Rural N = 302 ¹	Urbano N = 198 ¹
hierro_mg	7.6 (4.2, 11.2)	7.2 (3.6, 10.9)
zinc_mg		
-	4 (1.3%)	0 (0%)
0.02	2 (0.7%)	0 (0%)
0.03	1 (0.3%)	0 (0%)
0.04	0 (0%)	1 (0.5%)

Qué es R?

R es un lenguaje de programación y entorno de software libre para computación estadística y ciencia de datos con una gran capacidad para manipular, resumir, analizar y representar datos.

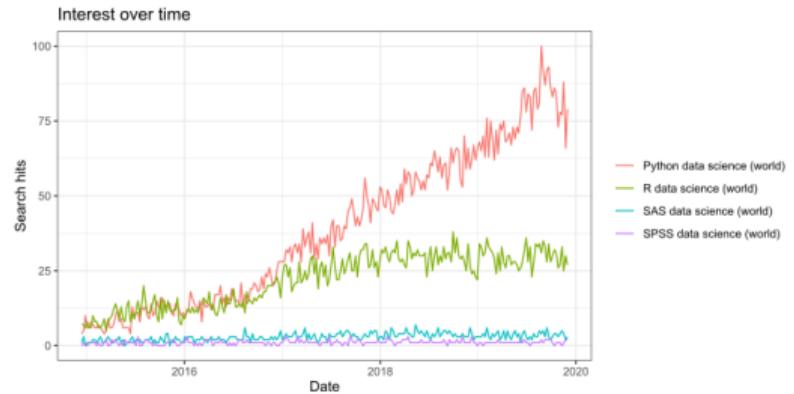


Porqué debo usar R?



Porqué debo usar R?

- Cuenta con la **mayor gama de métodos estadísticos** de alto rendimiento, robustos y validados por la comunidad científica.



Porqué debo usar R?

- R permite analizar **cualquier clase de datos** (datos rectangulares, texto, imagen, etc.) y **tamaños** (big data).

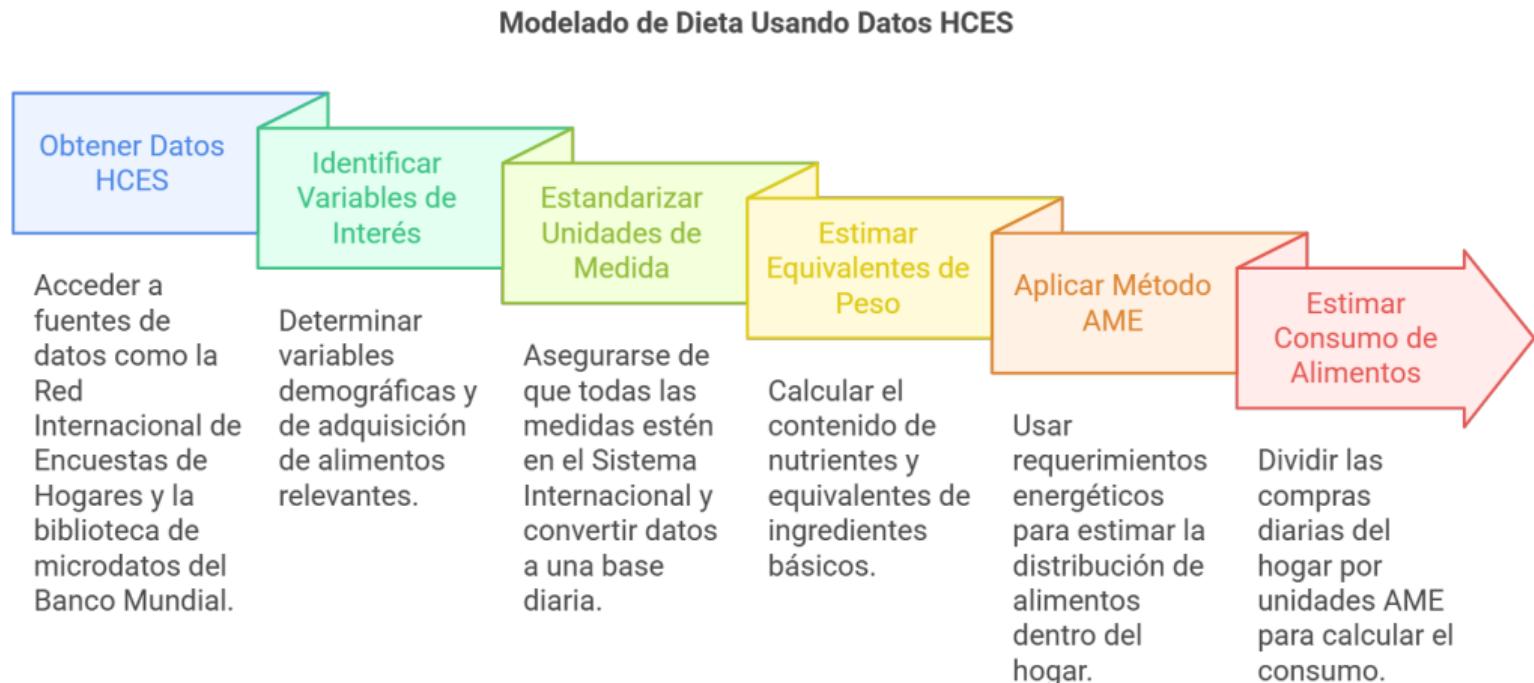


Figure 1: R, the language of data science

Aprovechando R para el Impacto del WFP



R para el modelado de dietas



Ejemplo con R base: cálculo de la ingesta familiar promedio de hierro

```
# calcular el consumo promedio de hierro en mg/día  
consumo_hierro <- c(15, 10, 20) # consumo diario en el hogar  
mean(consumo_hierro) # cálculo del consumo promedio
```

```
[1] 15
```

Resumiendo: clave de la presentación hasta el momento?

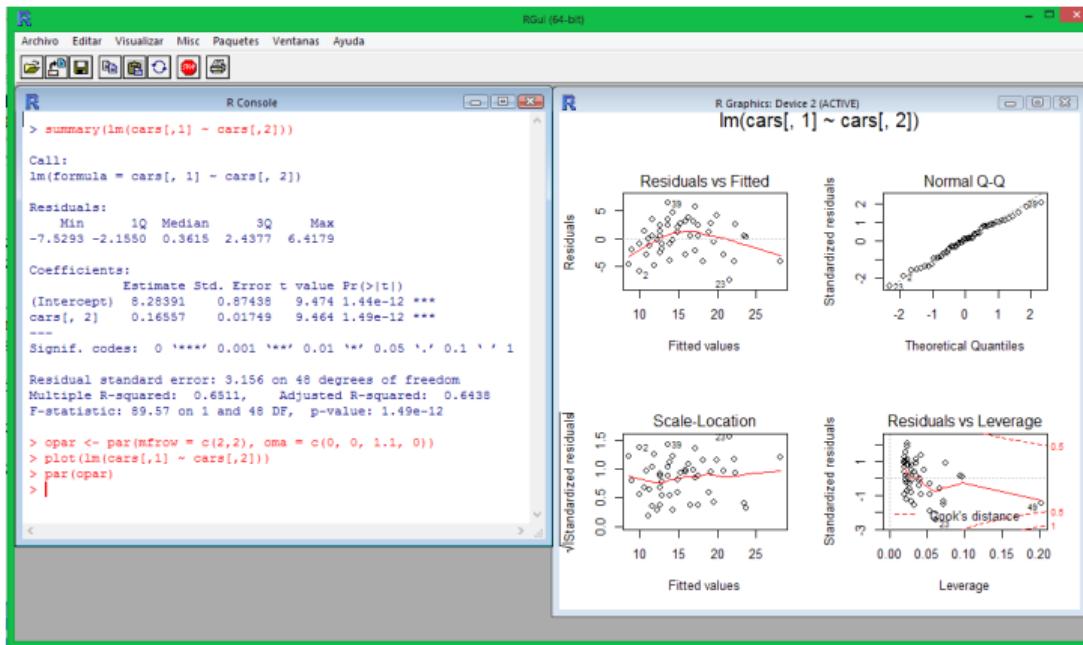
- R es un **entorno de desarrollo** de alta calidad
- R es un **lenguaje de programación** con **licencia GNU**
- Permite implementar procedimientos para **importar, ordenar, transformar, analizar, representar datos de muchos tipos**

Resumiendo: R está diseñado para usted!

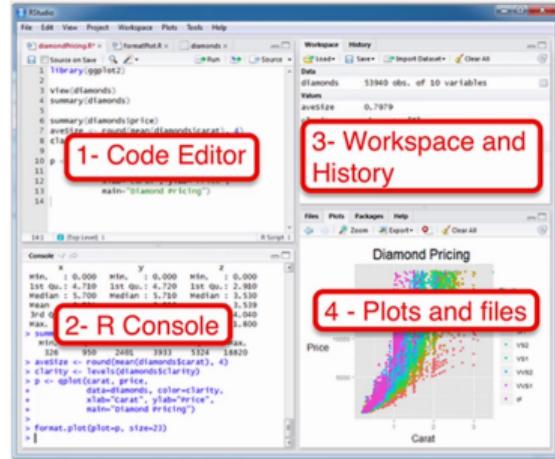


Base R

- conjunto integrado de herramientas diseñado para crear, analizar y visualizar datos estadísticos de manera eficiente



Rstudio es el IDE (Entorno de desarrollo integrado) más popular



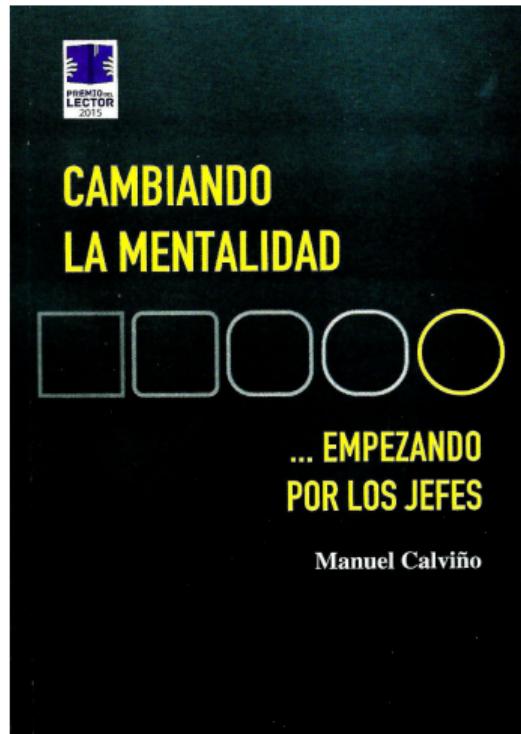
<https://posit.co/download/rstudio-desktop/>

R es un lenguaje de programación

- Lenguaje o idioma artificial (formal)
- Diseñado para expresar **computaciones**
- Llevadas a cabo por **computadoras**



R es un lenguaje de programación (formal)



R es un lenguaje de programación orientado a objeto (POO)



- Objeto: en programación es como una **pieza de juego de Lego**
- Objeto: tiene atributos (como color o tamaño) y hace **acciones propias** (como unirse a otras piezas)
- Objeto: Interactúa con otros objetos (*modularidad*) para funcionar como un sistemas más complejo.

La modularidad permite la reutilización del código



Funciones

Conjuntos de instrucciones reutilizables en R.



Script

Archivos de código que ejecutan tareas específicas.

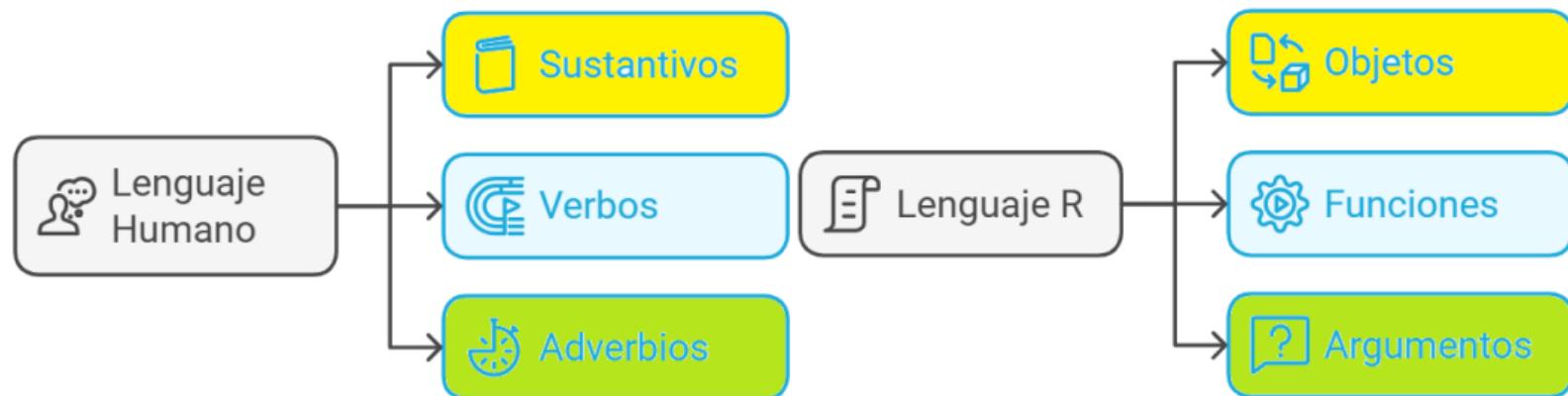


Bibliotecas

Colecciones de funciones y scripts predefinidos.

Analogía entre R y el lenguaje humano

Los **objetos** son como **sustantivos** y las **funciones** son como **verbos** que expresan acciones.



Analogía: sustantivos y objetos

humano: Los sustantivos representan cosas o conceptos

R: Los objetos contienen datos.

```
# Ejemplo: Se asignan concentraciones un kg de Harina a un objeto  
hierro_por_kg <- 30      # mg de hierro  
zinc_por_kg <- 15        # mg zinc  
yodo_por_kg <- 50        # µg yodo
```

Analogía: verbos y funciones

humano: Los verbos expresan **acciones** en las lenguas humanas.

R: las **funciones** son los “**verbos**” que realizan **acciones** sobre los objetos.



Ejemplo en R: Funciones realizando acciones (asignar valores y crear objeto)

```
# El operador <- asigna un valor a un objeto
# creo un objeto (hierro en el hogar) con c()
hierro <- c(15, 10, 20)
# calculo la suma de consumo de Fe por Kg de Harina en el hogar
consumo_total <- sum(hierro) # lo asigno a objeto otro objeto
print(consumo_total)
# 45
```

nota: (Alt y -) imprime el operador de asignación <-

Analogía: Adverbios y argumentos

humano: Los adverbios que modifican los verbos en lenguas humanas

R: Los **Adverbios** son los **argumentos** de las funciones

```
# Se usan objetos para crear un vector con la función combinar "c()"  
# Varios tipos de datos se usan como argumentos para crear vectores  
vitaminas <- c("A", "D", "C")      # vector tipo cadena (micronutrientes)  
consumo_hierro <- c(150, 200, 180) # vector tipo numérico (Fe mg )  
fortificado <- c(TRUE, FALSE, TRUE) # vector tipo lógico (fortificación)
```

Resumiendo: analogías del lenguaje R con lenguaje humano

- humano: **sustantivos** y **verbos** se combinan para **expresar pensamientos**
- lenguaje R: **objetos** y **funciones** se combinan para **realizar operaciones**

Ejemplo: Combinación de funciones y objetos para realizar operaciones

humano: calcula la media del consumo de hierro en un hogar

R: Como sigue:

```
# R Calcula la media con la función mean pasando el objeto hierro como argumento
consumo_hierro <- c(150, 200, 180) # crea Objeto consumo de hierro
mean(x = consumo_hierro) # calcula la media aritmética
```

```
[1] 176.6667
```

R como Lenguaje

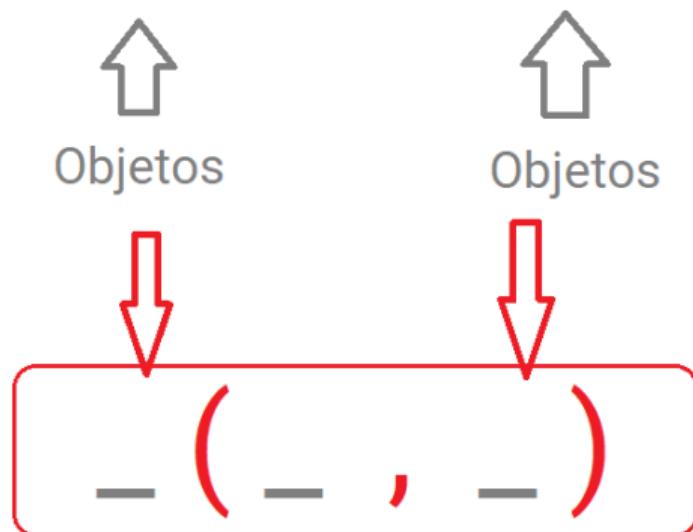
Estructura y Sintaxis: Al igual que el español combina sustantivos y verbos según reglas gramaticales, R combina objetos y funciones para realizar cálculos.

Características del Lenguaje: R tiene su propio vocabulario y sintaxis, permitiendo dar comando y estructurar soluciones a problemas

Y si ahora se va la corriente, y solo puedo seleccionar la diapositiva más Importante!

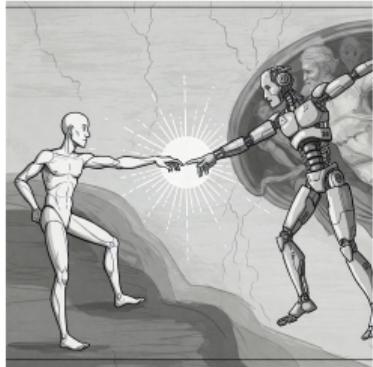
En R, "todo lo que sucede es una llamada a función"

FUNCTION(argumento 1, argumento 2,...)



En R, "todo lo que existe en un objeto"

Y ahora que podemos hablar en R, que faremos en lo adelante?



- Crear objetos
- Definir vectores
- Conocer los diferentes tipos de vectores(atómicos , factores, marcos de datos, listas)
- Realizar subconjunto

Los objetos en R se crean:

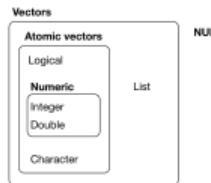
- Asignándoles un valor
- Como resultado de un cálculo
- Leyendo datos de un archivo (Ej. importar)
- etc.

Los vectores son la estructura de datos fundamental de R base para crear otras estructuras

- vectores atómicos: elementos del mismo tipo (homogéneos)
- vectores recursivos: pueden tener elementos de varios tipos (heterogéneos)
- NULL: vector genérico de longitud cero

Tipos de vectores atómicos

1. Lógicos: - (*TRUE o FALSE*), o (*T o F*)
2. Dobles: - *forma decimal* (*0.1234*), *científica* (*1.23e4*)
3. Enteros: - *seguidas de L* (*1234L*, *1e4L*)
4. Cadenas: - *encerrados por comillas* (“*vitamina*”)



Objeto vector atómico

la función "c()" abreviatura de **combinar** hace vectores más largos

```
anemia <- c(TRUE, FALSE)
cucharadas <- c(1L, 6L, 10L)
ingesta_hierro <- c(1, 2.5, 4.5)
micronutrientes <- c("hierro", "Ácido Ascórbico", "Zinc")
```

Objeto vector atómico

Cuando intentas **combinar** elementos de diferentes tipos para hacer vectores más largos ocurre la **Coerción**

```
# ingesta diaria de hierro en mg
ingesta_hierro <- c(8, 7, "11 mg")
# se convierte en vector de cadena
ingesta_hierro <- c("8", "7", "11 mg")
# las cadenas van entre comillas
```

Factores

- Es un vector que solo puede contener valores predefinidos
- Se utiliza para almacenar datos cualitativos o categóricos

Factores

- Los factores se construyen a partir de vectores de enteros (atómicos) y añaden un atributo para definir los niveles (etiquetas categóricas)
- Evitan la falta de consistencia Ejemplo sexo: “Fem”, “Femenino”, “F”

Factores

```
sex_char <- c(1, 2, 1)
sex_factor <- factor(x = sex_char,
                      levels = c(1, 2),
                      labels = c("masculino","femenino"))
```

```
table(sex_factor)
```

```
sex_factor
```

```
masculino  femenino
```

```
2
```

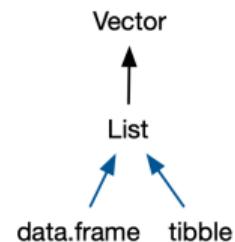
```
1
```

Listas

```
# cada elemento puede ser de cualquier tipo (datos heterogenea)
# construir una lista con datos de fortificación
datos_malawi <- list(
  c("Aceite", "Azúcar", "Harina de trigo"),
  list(
    vitamina_a = c(1000, 700, 80), # µg por 100g
    hierro = c(NA, NA, 3.0)       # mg por 100g
  ),
  c("Norte", "Centro", "Sur"),
  c(76.1, 55.6, 44.2) # de hogares que consumen
)
str(datos_malawi)
```

Marcos de datos y tibbles

- lista con nombre de vectores (columna)
- la longitud de cada uno de sus vectores debe ser la misma a diferencia de las listas



Marcos de datos (dataframe) creacion

Puede crear un marco de datos al proporcionar pares nombre-vector

```
library(tidyverse)
# Crear dataframe de fortificación
df_fortificacion <- tibble(
  vehiculo = c("Aceite vegetal", "Azúcar", "Harina de trigo"),
  micronutriente = c("Vitamina A", "Vitamina A", "Hierro"),
  cantidad_por_100g = c(1000, 700, 3.0), # µg para vitamina A
  cobertura = c(76.1, 55.6, 44.2), # % de hogares consumidores
  region_principal = c("Norte", "Centro", "Sur") # región
)
str(df_fortificacion)
```

tibbles (Subconjunto)

- Selección por nombre, por posición o por valor
- Se usan operadores \$ y [[

Selecting Vector Elements	
By Position	
<code>x[4]</code>	The fourth element.
<code>x[-4]</code>	All but the fourth.
<code>x[2:4]</code>	Elements two to four.
<code>x[-(2:4)]</code>	All elements except two to four.
<code>x[c(1, 5)]</code>	Elements one and five.
By Value	
<code>x[x == 10]</code>	Elements which are equal to 10.
<code>x[x < 0]</code>	All elements less than zero.
<code>x[x %in% c(1, 2, 5)]</code>	Elements in the set 1, 2, 5.
Named Vectors	
<code>x['apple']</code>	Element with name 'apple'.

Subconjunto por nombre

```
# Extract by name  
df_fortificacion$vehiculo  
  
[1] "Aceite vegetal"   "Azúcar"           "Harina de trigo"  
  
df_fortificacion[["vehiculo"]]  
  
[1] "Aceite vegetal"   "Azúcar"           "Harina de trigo"
```

Subconjunto por posición

```
# Extract by position  
df_fortificacion[[2]]
```

```
[1] "Vitamina A" "Vitamina A" "Hierro"
```

Subconjunto usando tubería

```
df_fortificacion %>% .$vehiculo
```

```
[1] "Aceite vegetal"   "Azúcar"           "Harina de trigo"
```

```
#> [1] 0.434 0.395 0.548 0.762 0.254
```

```
df_fortificacion %>% .[["vehiculo"]]
```

```
[1] "Aceite vegetal"   "Azúcar"           "Harina de trigo"
```

```
#> [1] 0.434 0.395 0.548 0.762 0.254
```

tubería

En R, el operador `%>%` (tubería) permite encadenar operaciones de manera fluida, similar a cómo leerías una oración.

humano: en los datos de fortificación selecciona la región y luego calcula su frecuencia absoluta

R: como sigue

```
df_fortificacion %>%
  select(region_principal) %>%
  count()
```

```
# A tibble: 1 x 1
```

```
  n
  <int>
1     3
```

En R hay muchos enfoques para hacer una misma acción!

Dollar sign syntax

```
goal(data$x, data$y)
```

SUMMARY STATISTICS:

one continuous variable:
`mean(mtcars$mpg)`

one categorical variable:
`table(mtcars$cyl)`

two categorical variables:
`table(mtcars$cyl, mtcars$am)`

one continuous, one categorical:
`mean(mtcars$mpg[mtcars$cyl==4])`
`mean(mtcars$mpg[mtcars$cyl==6])`
`mean(mtcars$mpg[mtcars$cyl==8])`

Formula syntax

```
goal(y~x|z, data=data, group=w)
```

SUMMARY STATISTICS:

one continuous variable:
`mosaic::mean(~mpg, data=mtcars)`

one categorical variable:
`mosaic::tally(~cyl, data=mtcars)`

two categorical variables:
`mosaic::tally(cyl~am, data=mtcars)`

one continuous, one categorical:
`mosaic::mean(mpg~cyl, data=mtcars)`

tilde

Tidyverse syntax

```
data %>% goal(x)
```

SUMMARY STATISTICS:

one continuous variable:
`mtcars %>% dplyr::summarize(mean(mpg))`

one categorical variable:
`mtcars %>% dplyr::group_by(cyl) %>%
dplyr::summarize(n())`

two categorical variables:
`mtcars %>% dplyr::group_by(cyl, am) %>%
dplyr::summarize(n())`

one continuous, one categorical:
`mtcars %>% dplyr::group_by(cyl) %>%
dplyr::summarize(mean(mpg))`

the pipe

En R cada módulo da oportunidades diferentes!

Using Packages

install.packages('dplyr')

Download and install a package from CRAN.

library(dplyr)

Load the package into the session, making all its functions available to use.

dplyr::select

Use a particular function from a package.

data(iris)

Load a built-in dataset into the environment.

Otras cosas te sera facil recordar!

- Crear objetos con el operador de asignación “<-”
- Que los datos de cadenas van entre “vitamina”
- Combinar datos para formar un vector con “c()”
- La formula para pedir ayuda ?(“nombre_función”)

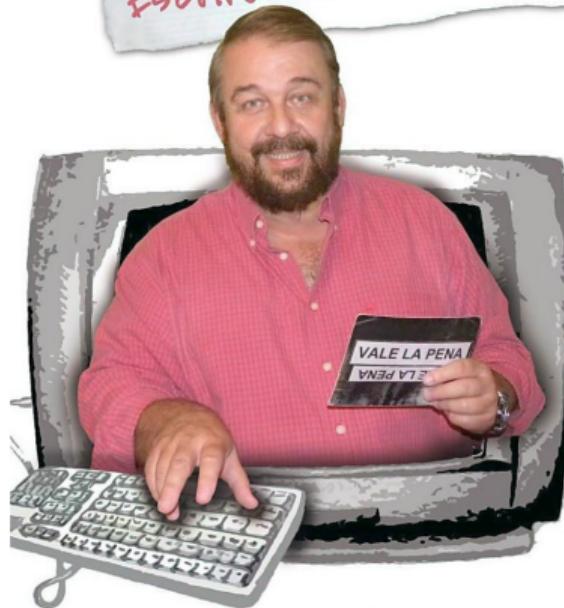
Siempre quedan cosas importates que aprederas con el tiempo!

Conditions	<code>a == b</code>	Are equal	<code>a > b</code>	Greater than	<code>a >= b</code>	Greater than or equal to	<code>is.na(a)</code>	Is missing
	<code>a != b</code>	Not equal	<code>a < b</code>	Less than	<code>a <= b</code>	Less than or equal to	<code>is.null(a)</code>	Is null

Pero solo te hace falta intentarlo!

VALE LA PENA

Escritos con Psicología



Manuel Calviño

Hoja de trucos

Base R Cheat Sheet

Getting Help

Accessing the help files

`?mean`
Get help of a particular function.

`help.search('weighted mean')`
Search the help files for a word or phrase.

`help(package = 'dplyr')`
Find help for a package.

More about an object

`str(iris)`
Get a summary of an object's structure.

`class(iris)`
Find the class an object belongs to.

Using Packages

`install.packages('dplyr')`
Download and install a package from CRAN.

`library(dplyr)`
Load the package into the session, making all its functions available to use.

`dplyr::select`
Use a particular function from a package.

`data(iris)`
Load a built-in dataset into the environment.

Vectors	
Creating Vectors	
<code>c(2, 4, 6)</code>	2 4 6 Join elements into a vector
<code>2:6</code>	2 3 4 5 6 An integer sequence
<code>seq(2, 3, by=0.5)</code>	2.0 2.5 3.0 A complex sequence
<code>rep(1:2, times=3)</code>	1 2 1 2 1 2 Repeat a vector
<code>rep(1:2, each=3)</code>	1 1 1 2 2 2 Repeat elements of a vector

Programming	
For Loop	
<code>for (variable in sequence){ Do something }</code>	Example
<code>for (i in 1:4){ j <- i + 10 print(j) i <- i + 1 }</code>	Example

Vector Functions	
If Statements	
<code>sort(x)</code> Return x sorted.	<code>rev(x)</code> Return x reversed.
<code>table(x)</code> See counts of values.	<code>unique(x)</code> See unique values.

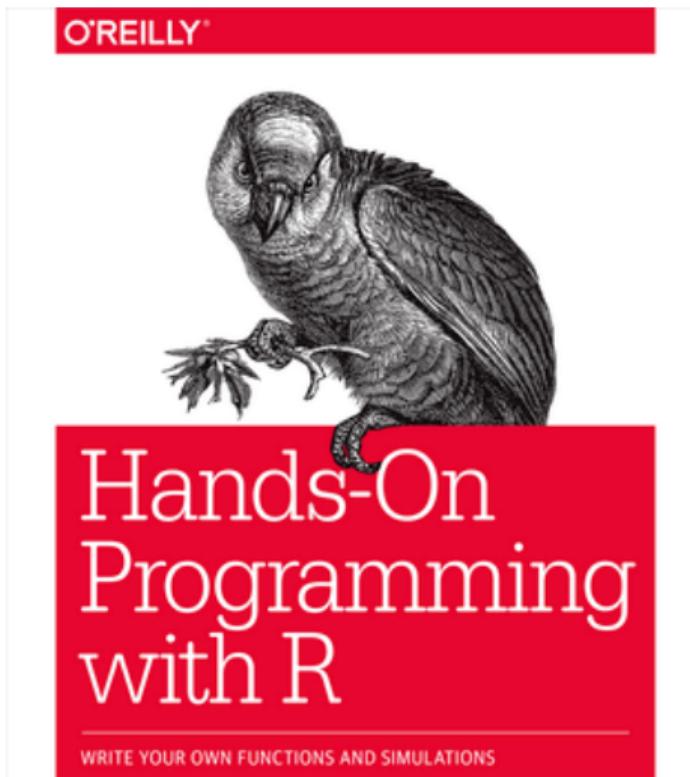
Selecting Vector Elements	
Functions	
<code>x[4]</code> The fourth element.	<code>if (condition){ Do something } else { Do something different }</code>
<code>x[-4]</code> All but the fourth.	<code>Example</code>

Reading and Writing Data	
<code>Input</code>	<code>Output</code>

`df <- read.table('file.txt')` `write.table(df, 'file.txt')` Read and write a delimited text file.

© Ben Shneiderman

Bibliografía



Garrett Grolemund
Foreword by Hadley Wickham

disponible

Próxima actividad: Desafío 1. Hierro “Cimientos de Código”

Desafío: "Construyendo Nutrilandia"

