

*powered by*

*R programming*

# Introducción general al uso de R - Clase 02 -

[minolicnp@gmail.com](mailto:minolicnp@gmail.com)

**Ignacio Minoli** *pwp*

Observatorio de Biodiversidad del Bosque Atlántico (OBBA)  
Instituto de Biología Subtropical (IBS) - CONICET - UNaM.



# Qué es un objeto en R?

- Un objeto es una estructura de datos que tiene algunos atributos y métodos que actúan sobre sus atributos. Ej. planos / casas / diferentes Clases.
- R es un lenguaje orientado a objetos porque todo el procesamiento gira en torno a los objetos y campos.
- Cada objeto tiene diferentes atributos y comportamiento. Clases.
- Básicamente hay 2 clases: S3 y S4.

```
S3 <- c("Listado donde se pasan los componentes junto con el nombre de clase",
       "adecuado seguido del resultado que se imprimirá", "Acceso y consulta co

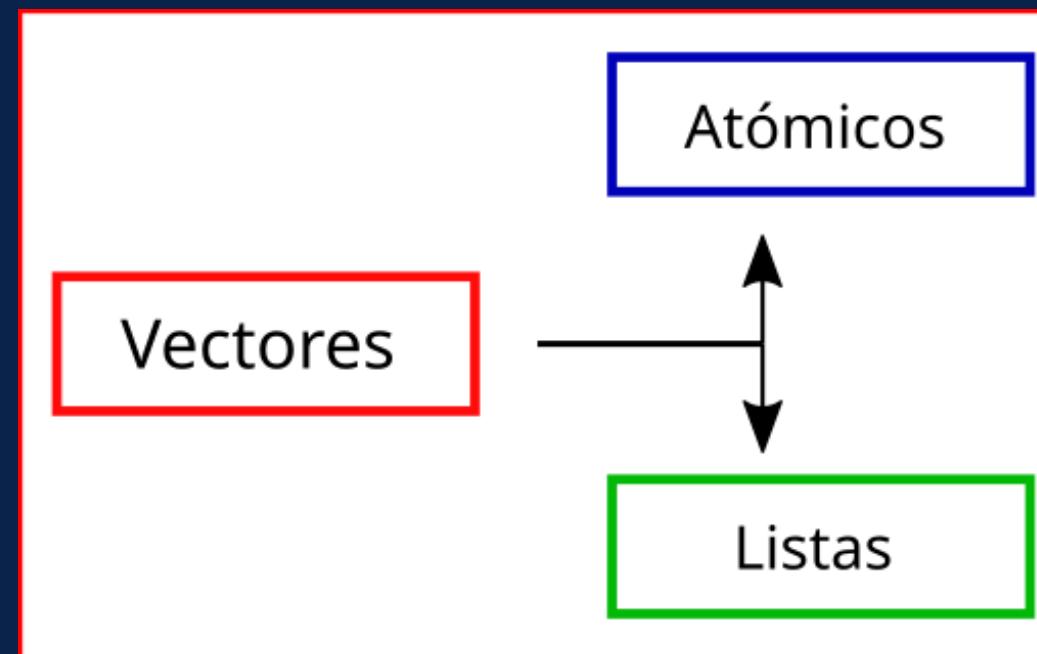
S3 <- c("Usa solo el primer argumento para enviar", "No requiere demasiado",
       "conocimiento como programador")

S4 <- c("Usado para multiples salidas", "Múltiples objetos")

S4 <- ("Utiliza los slots", "Acceso y consulta con @")
```

# Vectores

- Son considerados como los ladrillos estructurales que construyen la variedad de objetos que contienen datos.
- ¿Qué son los vectores? Es una secuencia de elementos que comparten el mismo tipo de datos.
- Soporta datos: logical, integer, double, character, complex, o raw data.



# Vectores

- Ejemplos

```
# Esto es un vector  
nums <- 4:-10  
nums
```

```
[1] 4 3 2 1 0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10
```

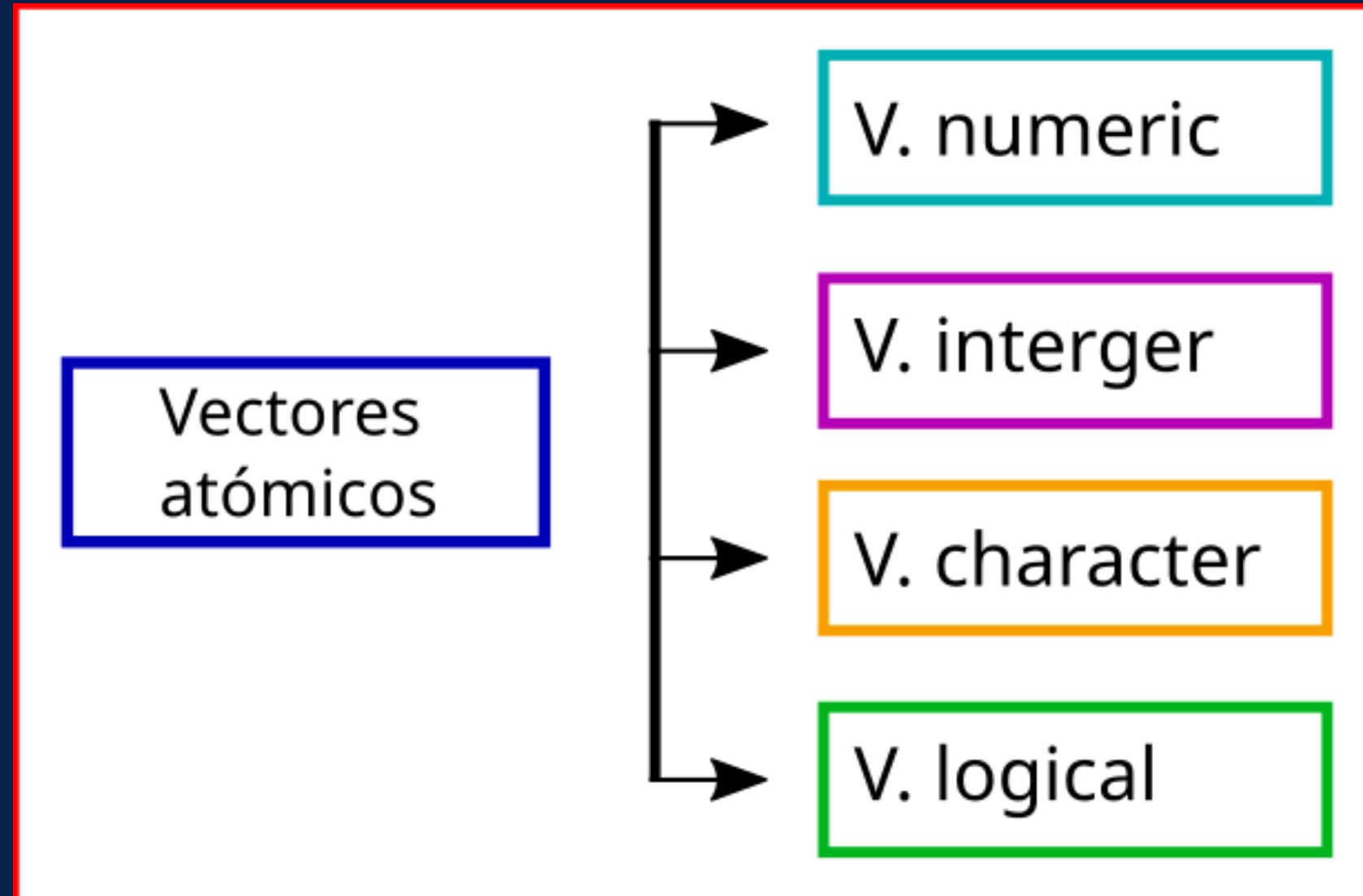
```
# Puedo crear una secuencia. Mínimo, máximo, e intervalo  
seq_vec <- seq(20, 25, by = 0.5)  
seq_vec
```

```
[1] 20.0 20.5 21.0 21.5 22.0 22.5 23.0 23.5 24.0 24.5 25.0
```

```
# Con esta función puedo ver que tipo de vector es  
class(seq_vec)
```

```
[1] "numeric"
```

# Tipos de vectores atómicos



# Vectores numeric

- Nos podemos ayudar con la función c()

```
# Vectores numeric. Números reales.  
num_vec <- c(2, 3, 4, 5)  
num_vec
```

```
[1] 2 3 4 5
```

```
# Evalúo la clase del objeto o vector  
class(num_vec)
```

```
[1] "numeric"
```

# Vectores interger

```
# Vectores interger. Números enteros.  
a <- as.integer(5)  
b <- 5L  
class(a) ; class(b)  
[1] "integer"  
[1] "integer"
```

```
int_vec <- c(1, 2, 3, 4, 5)  
class(int_vec)  
[1] "numeric"
```

```
int_vec <- as.integer(int_vec)  
class(int_vec)  
[1] "integer"
```

```
int_vec_1 <- c(1L, 2L, 3L, 4L, 5L)  
class(int_vec_1)  
[1] "integer"
```

# Vectores character

```
# Vectores character. Hay dos vías para crearlos. "" y as.character()
d <- 'Martin'
e <- "Josefina"
f <- 25
f <- as.character(f)
d
```

```
[1] "Martin"
```

```
    e
```

```
[1] "Josefina"
```

```
    f
```

```
[1] "25"
```

```
class(d) ; class(e) ; class(f)
```

```
[1] "character"
```

```
[1] "character"
```

```
[1] "character"
```

# Vectores character

```
# Vectores character.  
char_vec <- c(1,2,3,4,5)  
char_vec <- as.character(char_vec)  
char_vec  
[1] "1" "2" "3" "4" "5"
```

```
class(char_vec)  
[1] "character"
```

```
char_vec1 <- c("Martin", "Josefina", "Pérez", "Gómez")  
class(char_vec1)  
[1] "character"
```

# Vectores logical

- TRUE o FALSE.

```
# Vectores logical.  
A <- as.integer(5) ; A
```

```
[1] 5
```

```
B <- as.integer(6) ; B
```

```
[1] 6
```

```
C <- as.integer(7) ; C
```

```
[1] 7
```

```
X <- A > B  
Z <- B < C
```

```
X
```

```
[1] FALSE
```

```
Z
```

```
[1] TRUE
```

# Vectores logical

```
# Vectores logical.  
A <- as.integer(5)  
B <- as.integer(6)  
C <- as.integer(7)  
log_vec <- c(A < B, A < C, B < A, B < C, C < A, C < B)  
  
# Output  
log_vec
```

```
[1] TRUE TRUE FALSE TRUE FALSE FALSE
```

```
# Check - tipos de objetos  
class(A) ; class(B) ; class(B) ; class(log_vec)
```

```
[1] "integer"  
[1] "integer"  
[1] "integer"  
[1] "logical"
```

# Posibles indexados en vectores



# 1 - Combinar vectores

- La función **c()** no se usa solo para crear un vector, se puede usar para combinarlos:

```
# Creo los vectores.
longitud <- c(15, 26, 44, 50, 12, 18)
especie <- c("Rhinella ornata", "Boana faber", "Leptodactylus fuscus",
           "Physalaemus cuvieri", "Scinax fuscovarius", "Odontophrynuas asper")
longitud_especie <- c(longitud, especie)
```

longitud

```
[1] 15 26 44 50 12 18
```

especie

```
[1] "Rhinella ornata"      "Boana faber"        "Leptodactylus fuscus"
[4] "Physalaemus cuvieri"  "Scinax fuscovarius"  "Odontophrynuas asper"
```

longitud\_especie

```
[1] "15"                "26"                "44"
[4] "50"                "12"                "18"
[7] "Rhinella ornata"   "Boana faber"       "Leptodactylus fuscus"
[10] "Physalaemus cuvieri" "Scinax fuscovarius" "Odontophrynuas asper"
```

## 2 - Operaciones aritméticas

- Se realizan miembro por miembro en vectores numéricos. Podemos sumar, restar, multiplicar o dividir dos vectores, etc.

```
alas <- c(15, 26, 44, 50, 12, 8)
patas <- c(7, 11, 3, 23, 5, 8)
```

```
alas ; patas
```

```
[1] 15 26 44 50 12 8
[1] 7 11 3 23 5 8
```

```
# Hago una suma de vectores
alas + patas
```

```
[1] 22 37 47 73 17 16
```

```
# Hago una resta de vectores
alas - patas
```

```
[1] 8 15 41 27 7 0
```

# 3 - Indexado lógico de vectores

- Usando un vector lógico puedo indexar un vector y crear así uno nuevo.

```
spp <- c("Cerdocyon thous", "Conepatus chinga", "Dasypus novemcinctus",
       "Herpailurus yagouaroundi", "Hydrochoerus hydrochaeris",
       "Lycalopex gymnocercus")
si_no <- c(TRUE, FALSE, TRUE, TRUE, FALSE, TRUE)
```

spp

```
[1] "Cerdocyon thous"          "Conepatus chinga"
[3] "Dasypus novemcinctus"    "Herpailurus yagouaroundi"
[5] "Hydrochoerus hydrochaeris" "Lycalopex gymnocercus"
```

si\_no

```
[1] TRUE FALSE  TRUE  TRUE FALSE  TRUE
```

spp[si\_no]

```
[1] "Cerdocyon thous"          "Dasypus novemcinctus"
[3] "Herpailurus yagouaroundi" "Lycalopex gymnocercus"
```

spp[!si\_no]

```
[1] "Conepatus chinga"        "Hydrochoerus hydrochaeris"
```

## 4 - Indexado numérico

- Es uno de los más usados. Se especifica entre corchetes [] el número de la posición que quiero indexar.
- Si uso valores negativos = todo el vector, menos esa posición.

```
taxa <- c("Cerdocyon thous", "Didelphis aurita", "Hydrodynastes gigas",
        "Myrmecophaga tridactyla", NA, "Procyon cancrivorus")
taxa
```

```
[1] "Cerdocyon thous"           "Didelphis aurita"
[3] "Hydrodynastes gigas"      "Myrmecophaga tridactyla"
[5] NA                         "Procyon cancrivorus"
```

```
taxa[2]
```

```
[1] "Didelphis aurita"
```

```
taxa[-1]
```

```
[1] "Didelphis aurita"          "Hydrodynastes gigas"
[3] "Myrmecophaga tridactyla"  NA
[5] "Procyon cancrivorus"
```

```
taxa[5]
```

```
[1] NA
```

```
taxa[919]
```

```
[1] NA
```

## 5 - Duplicar indexado

- Es la posibilidad de acceder más de una vez a un elemento de un vector una operación. Se utiliza **c()**

```
estadio <- c("Huevos", "Larvas", "Subadultos", "Adultos", "Vivos", "Muertos")
estadio
```

```
[1] "Huevos"      "Larvas"       "Subadultos"   "Adultos"      "Vivos"
[6] "Muertos"
```

```
# Duplico valores de interés
estadio[c(1, 5, 6, 5, 5)]
```

```
[1] "Huevos"    "Vivos"     "Muertos"    "Vivos"     "Vivos"
```

## 6 - Indexados por rangos

- Se “corta” el vector con un operador para sacar un rango o sección del mismo.
- Se utiliza :

```
clase <- c("Peces", "Anfibios", "Reptiles", "Aves", "Mamíferos")
clase
```

```
[1] "Peces"      "Anfibios"    "Reptiles"    "Aves"       "Mamíferos"
```

```
# Hago una selección por rango
clase[2:4]
```

```
[1] "Anfibios" "Reptiles" "Aves"
```

## 7 - Indexados por fuera del orden

- Se utiliza para poder reordenar los elementos del vector o hacer una selección más allá del orden original.

```
muestras <- c("testigo", "alcohol_100%", "alcohol_70%", "alcohol_40%", "alcoho  
muestras
```

```
[1] "testigo"      "alcohol_100%"  "alcohol_70%"  "alcohol_40%"  "alcohol_10%"
```

```
# Reordeno el vector  
muestras[c(5, 4, 3, 2, 1)]
```

```
[1] "alcohol_10%"  "alcohol_40%"  "alcohol_70%"  "alcohol_100%" "testigo"
```

# 8 - Miembros vectoriales con nombre

- Se utiliza normalmente con un set de datos o data frame.

```
almacenaje <- c("Semillas", "Frutos")
almacenaje
[1] "Semillas" "Frutos"
```

```
names(almacenaje) <- c("Primero", "Segundo")
# Asigno nombres a los miembros del vector
almacenaje
```

```
Primero      Segundo
"Semillas"   "Frutos"
```

```
# Indexación por nombre
almacenaje["Primero"]
```

```
Primero
"Semillas"
```

```
almacenaje[c("Primero", "Segundo")]
```

```
Primero      Segundo
"Semillas"   "Frutos"
```

# Ejemplos de usos de vectores

- Se utilizan vectores para **PCA** y en **machine learning**. Se extienden a valores propios y vectores propios, los que luego se utilizan para realizar la descomposición en espacios vectoriales.
- Los inputs para modelos de **deep learning** están en forma de vectores. Estos vectores consisten en datos estandarizados que se suministran a la capa de entrada de la **neuronal network**.
- Se utilizan vectores en el desarrollo de algoritmos de **support vector machine**.
- Las operaciones de vectores se utilizan para diversas operaciones en **neuronal network**, como el reconocimiento de imágenes y el procesamiento de texto.
- Los vectores también son la materia prima para crear **matrices**. Muchas funciones para análisis piden este tipo de input.
- Son esencialmente la base estructural del objeto - set de datos más usado en R: **data frame** o **tibble**.

# Data frame

- Es una estructura de datos de dos dimensiones (rectangulares) que pueden contener datos de diferentes tipos (heterogéneas).
- Esta estructura de datos es la más usada para realizar análisis de datos y es el input más común en la mayoría de los paquetes estadísticos.
- Es como una matriz, pero más flexible en sus características. Las **filas** de un data frame admiten datos de distintos tipos, pero sus **columnas** conservan la restricción de contener un único tipo de datos.
- En el contexto estadístico: las **filas** (rows) en un data frame son los, registros, observaciones, casos o individuos.
- En el contexto estadístico: las **columnas** (columns) representan atributos, rasgos o variables.
- Las **columnas** son esencialmente vectores y pueden ser manipuladas con todas las operaciones descriptas anteriormente.

# Data frame

- **Estructura Tabular:** los data frames organizan los datos en un formato tabular, semejante a una tabla o una hoja de cálculo, con filas y columnas.
- **Columnas de Tipos Variables:** cada columna en un data frame puede contener diferentes tipos de datos (numéricos, caracteres, factores, etc.). Sin embargo, todos los elementos dentro de una columna deben ser del mismo tipo de datos.
- **Vectores de Longitud Igual:** las columnas son esencialmente vectores, y todas las columnas dentro de un data frame deben tener la misma longitud. Esto asegura que cada fila corresponda a un conjunto completo de observaciones a través de todas las variables.
- **Nombres de Columnas:** los data frames tienen nombres de columnas que facilitan el acceso y la referencia a columnas específicas usando estos nombres. Los nombres de las columnas deben ser únicos dentro de un data frame.

# Data frame

- **Nombres o Índices de Filas:** similar a las columnas, los data frames tienen nombres o índices de filas, que ayudan a identificar y referenciar filas específicas. Por defecto, las filas están numeradas comenzando desde 1 a menos que se proporcionen explícitamente nombres de filas.
- **Manipulación de Datos:** los data frames ofrecen varias funciones y métodos para la manipulación de datos, incluyendo el subconjunto, filtrado, combinación, reestructuración y transformación de datos.

```
library(gt)
library(palmerpenguins)

penguins %>%
  head(3) %>%
  gt()
```

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
Adelie	Torgersen	39.1	18.7	181	3750	male	2007
Adelie	Torgersen	39.5	17.4	186	3800	female	2007
Adelie	Torgersen	40.3	18.0	195	3250	female	2007

# ¿Tibble o Data Frame?

- Los tibbles son una versión más sofisticada de data frames incluídos en la familia de paquetes tidyverse. <https://www.tidyverse.org/packages/>
- ¿“Ecosistema - Familia” tidyverse? Es un conjunto inter relacionado de paquetes con funciones orientadas a la ciencia de datos en R. Hay una gran cantidad de cosas que puede hacer con sus datos, como dividir en subconjuntos, transformar, visualizar, etc.



- Hay 4 diferencias clave entre tibbles y data frames: **printing, subsets, clases de columnas e impresión en consola.**

# Beneficios: Tibbles

- **Impresión en Consola Mejorada:** los tibbles ofrecen mejores capacidades de impresión, mostrando un resumen conciso de los datos, lo que facilita la visualización y comprensión de conjuntos de datos grandes.
- **Consistencia:** los tibbles tienen un comportamiento más consistente en diferentes operaciones, reduciendo comportamientos inesperados en comparación con los data frames.
- **Manipulación y Manejo Moderno de Datos:** diseñados para abordar algunas de las limitaciones y peculiaridades de los data frames, los tibbles proporcionan un enfoque más moderno para trabajar con datos tabulares en R.

# Matrices

- Es la extensión de un vector a dos dimensiones. Diferentes características en una matriz: **data** el vector, **nrow** número de filas, **ncol** es el número de columnas, **dim** número de dimensiones, **dimnames** nombres asignado a filas y columnas.

```
matriz <- matrix(1:8, nrow = 4)
matriz
```

	[,1]	[,2]
[1,]	1	5
[2,]	2	6
[3,]	3	7
[4,]	4	8

```
# Un vector con dimensión
m <- 1:12
```

```
# Lo transformo en matriz
dim(m) <- c(4, 3)
m
```

	[,1]	[,2]	[,3]
[1,]	1	5	9
[2,]	2	6	10
[3,]	3	7	11
[4,]	4	8	12

# Arrays

- Es cuando se extiende un vector a más de dos dimensiones. Permiten representar datos multidimensionales de un único tipo. Por ej. si creamos un array de dimensiones (2, 3, 2) se crearán 2 matrices rectangulares cada una con 2 filas y 3 columnas.
- Para crearlos se utiliza la función `array()`. Se pueden asignar nombres a las dimensiones.

```
array_3d <- array(1:12, dim = c(2, 3, 2), dimnames = list(
  c("sp_uno", "sp_dos"),
  c("trat_testigo", "trat_fuego", "trat_agua"),
  c("matriz_1999", "matriz_2024")))

array_3d
, , matriz_1999

      trat_testigo trat_fuego trat_agua
sp_uno          1            3            5
sp_dos          2            4            6

, , matriz_2024

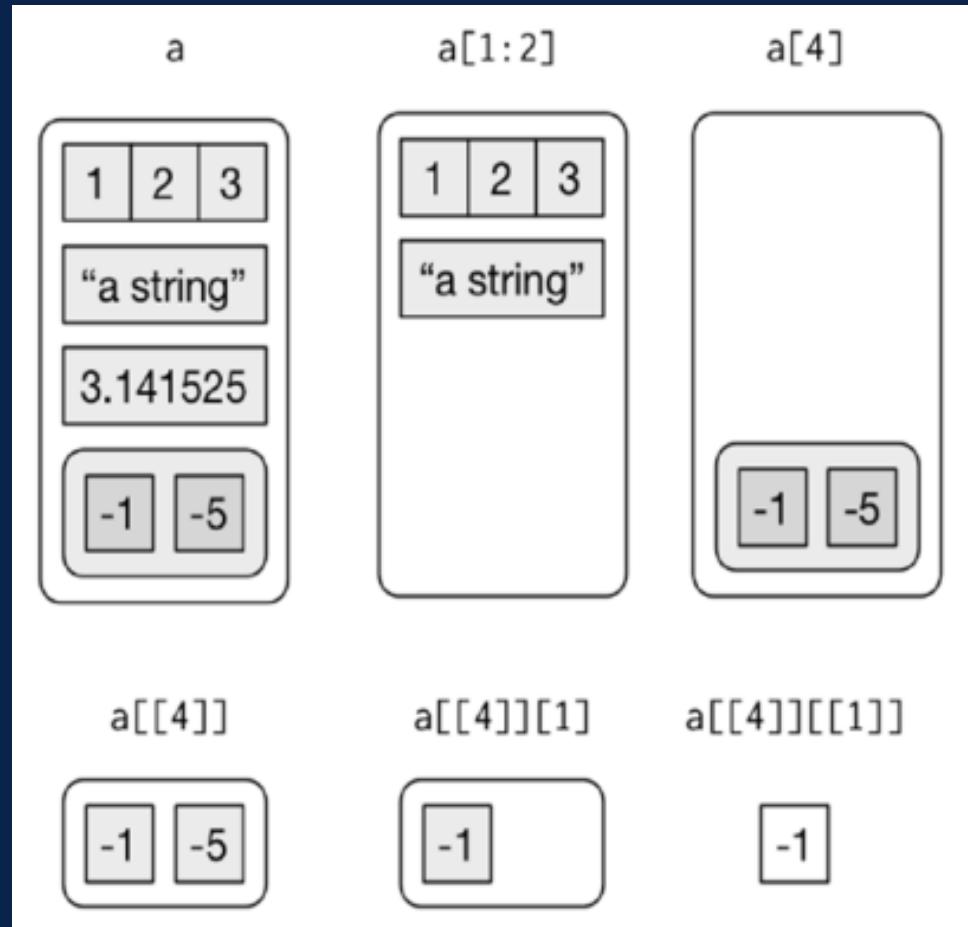
      trat_testigo trat_fuego trat_agua
sp_uno          7            9           11
sp_dos          8           10           12
```

# Listas

- Es excelente para ser usado como “contenedor” de cualquier clase de objeto: números, vectores, matrices, funciones, data.frames, incluso otras listas.
- Se crean con la función **list()** y el indexado es diferente a los vectores y data frames.
- Su estructura es más compleja y se requiere entenderla para poder indexar o filtrar los datos a ser consultados.
- Muchos paquetes con funciones complejas dan como salidas objetos - listas.
- El entendimiento de su estructura, manejo, subsets, y pasado a tablas de los resultados obtenidos en este tipo de objeto, siempre representa un desafío y es necesario un buen manejo de R.
- Cuando la función **str()** no es suficiente para entender la estructura de la lista, siempre se puede recurrir a la documentación del paquete utilizado.

# Listas - Indexado

- La distinción entre [] y [[]] es importante en las listas. El operador [[]] navega jerárquicamente por la lista, y [] devuelve una sub lista.



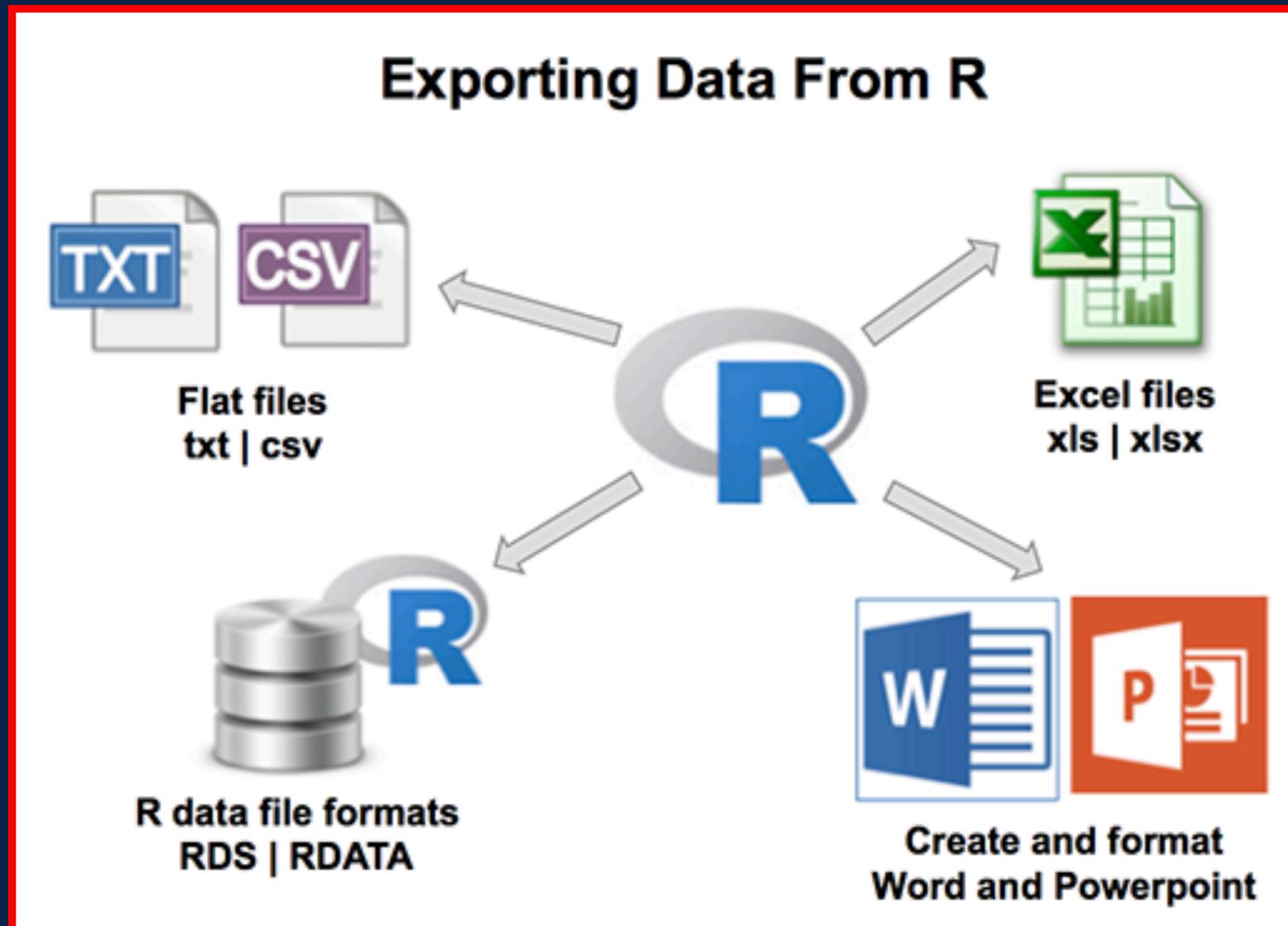
# Importación - Checklist previo

- Si trabajan con hojas de cálculo, lo habitual es que la primera fila sea considerada el header y la primera columna sea considerada como el identificador de observaciones, registros, casos o unidad de muestreo.
- Evitar nombres, valores o campos con espacios en blanco. Cada palabra será tomada como una variable separada y da errores en el número de elementos por linea en el data frame.
- Para unir o concatenar palabras dentro del set de datos unirlas con puntos o guión bajo.
- Nombres cortos y abreviados son más adecuados que largos.
- Evitar el uso de nombres con estos símbolos: ?, \$, %, ^, &, \*, (, ), -, #, ?, <, >, /, |, [ , ]. {
- Si usan excel, borrar todos los comentarios para evitar columnas NAs. Preferentemente aquellos valores faltantes completar como NA.
- Preferentemente, evitar el uso de comas al utilizar texto almacenado. Cuidar de que no queden saltos de linea o tabulaciones.

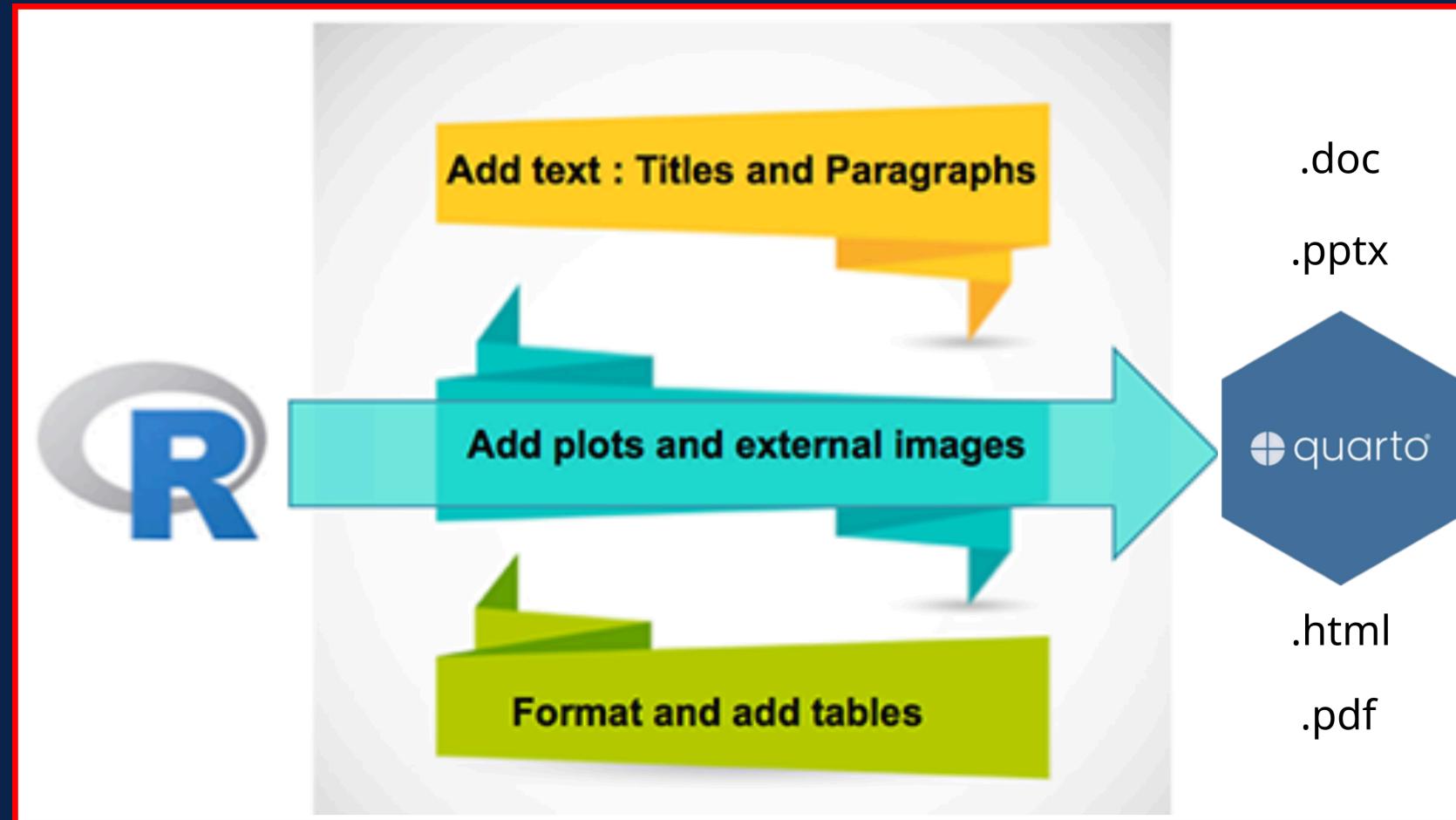
# Importación

- Importaciones más comunes en R:
  - Archivos tablas texto plano `read.table()`
  - Archivos Comma Separated Value `read.csv()`
  - Archivos tablas texto plano `read.delim()`
  - Archivos WinOffice `read_excel()`
  - Archivos de texto `fromJSON()`
  - Tablas HTML `readHTMLTable()`
- Archivos de otros software estadísticos:
  - Archivos SPSS                  Archivos Stata
  - Archivos Systat                Archivos SAS
  - Archivos Minitab               Archivos RDA o RData
- Importar desde bases de datos:
  - Data utilizando SQL.
- Data a través de Webscraping:
  - Datos no estructurados `rvest`, `read_html()`, `html_nodes()`

# Exportación



# Exportación



# Operadores

- Aritméticos

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
/	División
^ o **	Exponencial

```
# Ejemplos de operadores
x <- 5
y <- 16
x + y
```

```
[1] 21
```

```
x * y
```

```
[1] 80
```

# Operadores

- Relacionales

Operador	Descripción
<	Menor
>	Mayor
<=	Menor o igual que
>=	Mayor o igual que
==	Igual a
!=	No igual a

```
x <- 5 ; y <- 16  
  
x <= 5 ; x != 5 ; y == 16
```

```
[1] TRUE  
[1] FALSE  
[1] TRUE
```

# Operadores

- Lógicos

OPERADOR	DESCRIPCIÓN
!	NOT Lógico
&	AND Lógico por elementos
&&	AND Lógico
	OR Lógico por elementos
	OR Lógico

- Asignación y Otros

Operador	Descripción
<-, <<-, =	Asignación hacia la izquierda
->, ->>	Asignación hacia la derecha
;	ejecuta 2 funciones en una misma linea
:	rango

# Operadores. Asignación y Otros

```
x <- 5  
y <- 16  
x ; y
```

```
[1] 5  
[1] 16
```

```
x = 5  
x
```

```
[1] 5
```

```
x <- 1:15  
x
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

# R básico vs tidyverse

- La historia y evolución de R fue marcada por la “bifurcación” creada por **tidyverse** de **Hadley Wickham**. Cambios en el tiempo.
- El data wrangling o limpieza manipulación de datos, se afirma que ocupa aproximadamente el 80% del tiempo de un análisis de datos.
- Estos paquetes trabajan compartiendo ciertos principios en su funcionamiento, estructura de datos y desde la optimización del código con un potente operador: el **Pipe (%>%)**.
- El **Pipe (%>%)** permite encadenar funciones para realizar de forma acciones más complejas en los datos. Este operador pasa el elemento que está a su izquierda o anterior como un argumento de la función que tiene a la derecha o posterior.
- El operador pipe podemos leerlo como “entonces” y permite encadenar sucesivas llamadas a funciones.

# R básico vs tidyverse

```
# R Básico
head(iris, n = 7)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa

```
# Con tidyverse
iris %>% head(n = 7)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa

# R básico vs tidyverse

```
# R Básico
df1 <- subset(iris, Sepal.Length > 4)
df2 <- aggregate(df1$Petal.Width, list(df1$Species), FUN=mean)
df2
```

```
Group.1      x
1   setosa 0.246
2 versicolor 1.326
3 virginica 2.026
```

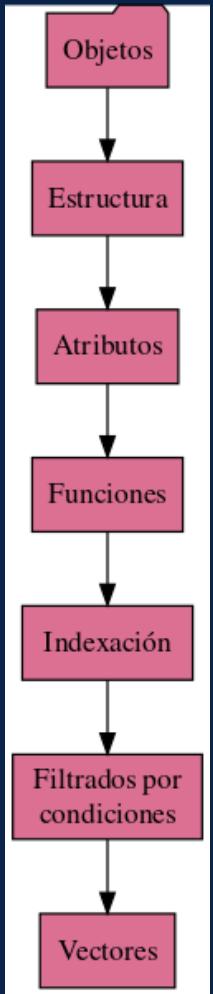
```
# Con tidyverse
library(tidyverse)

iris %>%
  filter(Sepal.Length > 4) %>%
  group_by(Species) %>%
  summarise(media = mean(Petal.Width))
```

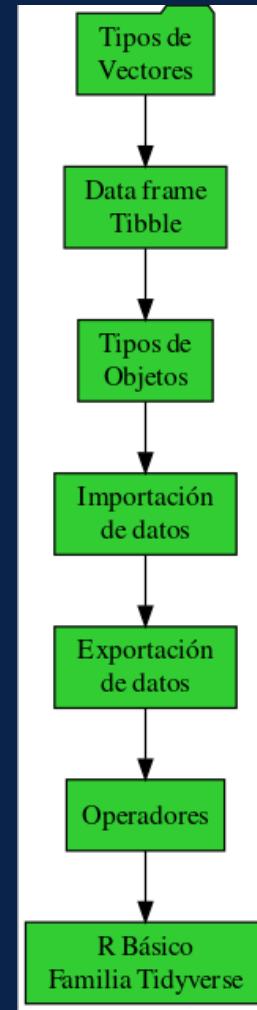
```
# A tibble: 3 × 2
  Species     media
  <fct>     <dbl>
1 setosa     0.246
2 versicolor 1.33 
3 virginica  2.03
```

# Resumen

## Objetos, Indexación y Funciones



## Data Frame, Tibble, Operadores y Tidyverse



# QR - presentación

**PDF Presentación: Clase 02 - Teoría**



# Fin. Clase 02 - Teoría

iii Muchas gracias !!!

¿ Preguntas ? ... ¿ Consultas ?

