



# CBIT200 - Introducción al análisis geoespacial

Horacio Samaniego

10/08, 2023

**Bienvenida**

# OBJETIVOS

**Cambio de nombre:** Introducción al análisis (y manipulación) de datos geoespaciales

1. Conocer y entender el concepto de *Investigación Reproducible* como una forma y filosofía de trabajo que permite que las investigaciones sean más ordenadas y replicables, desde la toma de datos hasta la escritura de resultados utilizando R.
2. Ser capaz de realizar análisis críticos de la naturaleza de los datos al realizar análisis exploratorios y reforzar conocimientos en estadística.
3. Realizar análisis de datos espaciales, poder visualizarlos y aplicar a preguntas de conservación y manejo de recursos naturales.
4. Aprender a utilizar, de forma proficiente, el lenguaje de programación R y la plataforma GitHub en un ambiente de trabajo colaborativo.

# Evaluaciones

*Evaluación*

*Ponderación*

Ejercicios & Tareas

$$\frac{1}{n} \sum_i^n nota\ tarea_i \qquad 50\%$$

Proyecto reporte

10%

Proyecto código

20%

Presentación

15%

Participación / Asistencia

5%

# Primeros pasos

- Evaluación sencilla (Informes reproducibles + presentación reproducible)
- Evaluación por pares
- Mucho trabajo personal guiado
- Pagina donde esta [todo el curso](#)
- Hacer:
  1. suscribirse a [servidor Discord](#)
  2. hacerse usuario de [GitHub](#)

# **Elementos básicos**

# Objetos

Todo en R es un objeto

- Identidad – *Nombre*
  - e.g. Teléfono, RORACO (Roble, Raulí, Coihue), BCS, 'uso', etc...
- Estado – *Característica*
  - e.g. verde, = '3', etc...
- Comportamiento – *Método*
  - e.g. riiing, uso



# Variables

Tipos de datos básicos en R:

Los tipos de datos básicos (o “primitivos”) más importantes son los tipos “numérico” (para números) y “carácter” (para texto). Otros tipos son el “entero”, que puede utilizarse para representar números enteros; el “lógico” para VERDADERO/FALSO, y el “factor” para variables categóricas.

# Variables

- *numeric* - (10.5, 55, 787)
- *integer* - (1L, 55L, 100L, letra "L" lo declara como un entero)
- *complex* - (9 + 3i, donde "i" es la parte imaginaria)
- *character* (string) - ("k", "R is exciting", "FALSE", "11.5")
- *logical* (boolean) - (TRUE or FALSE)
- *missing values* - NA (Not Available / No disponible)
- *factor* - Una categoría / nivel, ordenada, o no (i.e. )

Un factor es una variable nominal (categórica) con un conjunto de valores posibles conocidos denominados niveles. Pueden crearse utilizando la función `as.factor`. En R suele ser necesario convertir (cast) una variable de carácter en un factor para identificar grupos para su uso en pruebas y modelos estadísticos.

# Ejercicios

# Computo simple

1. Calcula la suma de 100.1, 234.9 and 12.01
2. La raíz cuadrada de 256
3. El logaritmo (base 10) de 100, y multiplique el resultado por el coseno de n. Ayuda: vea ?log and ?pi.
4. Suma acumulada de 2, 3, 4, 5, 6.
5. Suma acumulada anterior, pero en orden inverso.
6. Encuentre 10 números enteros aleatorios entre 0 y 100 Ayuda: ?sample, ?runif, o una combinacion de aquello

**Como se organizan los datos en R**

# Estructura de datos

- Vector: Un conjunto lineal de datos (secuencia génica, serie de tiempo)
- Matrix: Una tabla con solo números
- Data Frame: Una tabla donde cada columna tiene un tipo de datos (estándar dorado)
- List: Aquí podemos meter lo que queramos

The diagram illustrates the relationship between different R data structures. At the top, a bracket labeled "List" spans all four structures. Below it, another bracket labeled "Data frame" spans the "Vector" and "Matrix" structures. The "Vector" structure is shown as a vertical column of five elements: 2, 3, 4, 5, and 1. The "Data frame" structure is shown as a grid with three columns and five rows, containing elements 2, a, 0; 3, b, 3; 4, c, 7; 5, v, 3; and 1, f, 6. The "Matrix" structure is shown as a grid with three columns and six rows, containing elements 2, 8, 0; 3, 6, 3; 4, 5, 7; 5, 4, 3; and 1, 3, 6.

List		
Vector	Data frame	Matrix
2	2 a 0	2 8 0
3	3 b 3	3 6 3
4	4 c 7	4 5 7
5	5 v 3	5 4 3
1	1 f 6	1 3 6

# Vector

- Secuencia lineal de datos
- Pueden ser de muchos tipos (numéricos, de caracteres, lógicos, etc.)
- Ejemplo `data(uspop)`
- para crear uno `c(1,4,6,7,8)`
- para subsetear un vector se pone el índice entre []
- `uspop[4]`, `uspop[2:10]`,  
`uspop[c(3,5,8)]`



# Errores + comunes

Las siguientes líneas de código contienen algunos errores comunes que impiden que se evalúen correctamente o dan lugar a mensajes de error. 1. Mire el código sin ejecutarlo y vea si puede identificar los errores y corregirlos todos. 2. Luego, ejecute también el código defectuoso copiando y pegando el texto en la consola (no escribiéndolo, R studio intentará evitar estos errores por defecto) (¡pero no todos producen errores!).

```
vector1 <- c('one', 'two', 'three', 'four, 'five', 'seven')
vec.var <- var(c(1, 3, 5, 3, 5, 1)
vec.mean <- mean(c(1, 3, 5, 3, 5, 1))
vec.Min <- Min(c(1, 3, 5, 3, 5, 1))
Vector2 <- c('a', 'b', 'f', 'g')
vector2
vector1 <- c('one', 'two', 'three
```

# Data Frame

- Una tabla, cada columna un tipo de datos (Numérico, lógico, etc)
- Cada columna un vector
- Ejemplo `data(iris)`
- Para subsetear  
`data.frame[filas,columnas]`
- Ejemplos `iris[,3]`,  
`iris["Petal.Length"]`,  
`iris[2:5,c(1,5)]`,  
`iris$Petal.Length`



# Ejercicio

Crea este data.frame en la variable z:

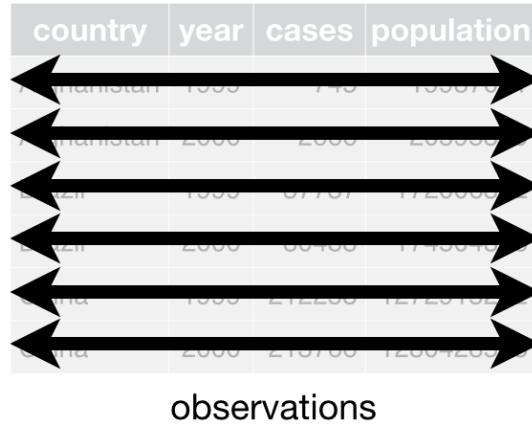
Numero	Letra	Medida
5	S	49.59
4	L	91.84
3	B	49.32
1	Q	72.25
2	A	65.51

# Principios de Tidydata

# Tidy Data

country	year	cases	population
Afghanistan	1990	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	128042583

variables



country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	128042583

values

- Cada columna una variable
- Cada fila una observación

Trabajemos tidy

# dplyr

- Paquete con pocas funciones muy poderosas para ordenar datos
- Parte del tidyverse
- *group\_by* (agrupa datos)
- *summarize* (resume datos agrupados)
- *filter* (Encuentra filas con ciertas condiciones)
- *select* junto a *starts\_with*, *ends\_with* o *contains*
- *mutate* (Genera variables nuevas)
- *pipe*: |>, o bien %>% (uso antiguo, requiere tener cargada la librería)
- *arrange* ordenar

# summarize y group\_by

- *group\_by* reune observaciones según una variable
- *summarize* resume una variable

```
library(tidyverse)
```

```
Summary.Petal <- summarize(iris, Mean.Petal.Length = mean(Petal.Length), SD.Petal.Length = sd(|
```

Mean.Petal.Length	SD.Petal.Length
3.758	1.765298

# summarize y group\_by (continuado)

```
Summary.Petal <- group_by(iris, Species)
```

```
Summary.Petal <- summarize(Summary.Petal, Mean.Petal.Length = mean(Petal.Length), SD.Petal.Length = sd(Petal.Length))
```

Species	Mean.Petal.Length	SD.Petal.Length
setosa	1.462	0.1736640
versicolor	4.260	0.4699110
virginica	5.552	0.5518947

# summarize y group\_by (continuado)

- Pueden agrupar por más de una variable a la vez

```
data("mtcars")
Mtcars2 <- group_by(mtcars, am, cyl)
Consumo <- summarize(Mtcars2, Consumo_promedio = mean(mpg), desv = sd(mpg))
```

am	cyl	Consumo_promedio	desv
0	4	22.90000	1.4525839
0	6	19.12500	1.6317169
0	8	15.05000	2.7743959
1	4	28.07500	4.4838599
1	6	20.56667	0.7505553
1	8	15.40000	0.5656854

**Dudas?**

# mutate

- Crea variables nuevas

```
DF <- mutate(iris, Petal.Sepal.Ratio = Petal.Length/Sepal.Length)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Petal.Sepal.Ratio
5.8	4.0	1.2	0.2	setosa	0.21
4.7	3.2	1.6	0.2	setosa	0.34
5.1	3.8	1.9	0.4	setosa	0.37

# Pipeline (

> )

- Para realizar varias operaciones de forma secuencial
- sin recurrir a parentesis anidados
- sobrescribir multiples bases de datos

```
x <- c(1,4,6,8)  
y <- round(mean(sqrt(log(x))),2)
```

- Que hice ahí?

```
x <- c(1,4,6,8)  
y <- x |> log() |> sqrt() |> mean() |> round(2)
```

```
## [1] 0.99
```

# Pipeline (

> )

- Muchos objetos intermedios

```
DF <- mutate(iris, Petal.Sepal.Ratio = Petal.Length/Sepal.Length)
BySpecies <- group_by(DF, Species)
Summary.Byspecies <- summarize(BySpecies, MEAN = mean(Petal.Sepal.Ratio), SD = sd(Petal.Sepal.
```

Species	MEAN	SD
setosa	0.2927557	0.0347958
versicolor	0.7177285	0.0536255
virginica	0.8437495	0.0438064

# Pipeline (

> )

- Con pipe

```
Summary.Byspecies <- summarize(group_by(mutate(iris, Petal.Sepal.Ratio = Petal.Length/Sepal.Length), Species), MEAN = mean(Petal.Length), SD = sd(Petal.Length))
```

Species	MEAN	SD
setosa	0.2927557	0.0347958
versicolor	0.7177285	0.0536255
virginica	0.8437495	0.0438064

# Pipeline (

>) otro ejemplo

```
library(tidyverse)
MEAN <- iris |> group_by(Species) |> summarize_all(.funs = list(Mean = mean, SD =sd))
```

Species	Sepal.Length_Mean	Sepal.Width_Mean	Petal.Length_Mean	Petal.Width_Mean	Sepal.Length_SD	Sepal.W
setosa	5.006	3.428	1.462	0.246	0.3524897	0.379
versicolor	5.936	2.770	4.260	1.326	0.5161711	0.313
virginica	6.588	2.974	5.552	2.026	0.6358796	0.321

**Mas dudas?**

# Filter

- Selecciona según una o más variables

simbolo	significado	simbolo_cont	significado_cont
>	Mayor que	!=	distinto a
<	Menor que	%in%	dentro del grupo
==	Igual a	is.na	es NA
>=	mayor o igual a	!is.na	no es NA
<=	menor o igual a		

# Ejemplos de filter agregando a lo anterior

```
data("iris")
DF <- iris |> filter(Species != "versicolor") |> group_by(Species) |> summarise_all(mean)
```

Species	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
setosa	5.006	3.428	1.462	0.246
virginica	6.588	2.974	5.552	2.026

# Ejemplos de filter

```
DF <- iris |> filter(Petal.Length >= 4 & Sepal.Length >= 5) |> group_by(Species) |> summarise(
```

Species	N
versicolor	39
virginica	49

# Más de una función

```
data("iris")
DF <- iris |> filter(Species != "versicolor") |> group_by(Species) |> summarise_all(.funs= lis
```

Species	Sepal.Length_Mean	Sepal.Width_Mean	Petal.Length_Mean	Petal.Width_Mean	Sepal.Length_SD	Sepal.Width_SD
setosa	5.006	3.428	1.462	0.246	0.3524897	0.379023
virginica	6.588	2.974	5.552	2.026	0.6358796	0.322446

# Select

- Selecciona columnas dentro de un data.frame, se pueden restar

```
iris |> group_by(Species) |> select(Petal.Length, Petal.Width) |> summarize_all(mean)
```

```
iris |> group_by(Species) |> select(-Sepal.Length, -Sepal.Width) |> summarize_all(mean)
```

```
iris |> group_by(Species) |> select(contains("Petal")) |> summarize_all(mean)
```

```
iris |> group_by(Species) |> select(-contains("Sepal")) |> summarize_all(mean)
```

# Resultado

Species	Petal.Length	Petal.Width
setosa	1.462	0.246
versicolor	4.260	1.326
virginica	5.552	2.026



# Ejercicios

```
Casos_Activos <- read_csv("https://raw.githubusercontent.com/MinCiencia/Datos-COVID19/master/o...
```

Usando la base de datos del repositorio del ministerio de ciencias, genera un dataframe que responda lo siguiente:

- ¿Que proporción de las comunas ha tenido en algun momento mas de 50 casos por cada 100.000 habitantes?
- Genera un dataframe, donde aparezca para cada comuna que haya tenido sobre 50 casos por cada 100.000 habitantes, cuantos días ha tenido sobre ese valor.
- Genera una tabla de cuales comunas han tenido sobre 50 casos por cada 100.000 habitantes y de esas comunas crea una variable que sea la prevalencia máxima de dicha comuna.

# Bonus (Esto requiere investigar no basta con lo que aprendimos)

- Ve cuales son las 10 comunas que han tenido la mayor mediana de prevalencia, para cada una de estas 10 comunas, genera una tabla con la mediana, prevalencia máxima y fecha en que se alcanzó la prevalencia máxima
- Nos vemos a las 12:45

# Para la otra clase, es necesario:

- Crearse cuenta de [github](#)
- Instalar los paquetes knitr, rmarkdown y kableExtra