Intro a R

Rodrigo Negrete Pérez

August 6, 2022

1. Material de los labs
2. Estructura de los labs
3. ¿Qué es R?
4. Interfaz
5. Creación y tipos de objetos
6. Vectores
7. Ejercicios Mauricio Romero

Inspiración de los labs.

A grandes razgos, baso estos labs en tres fuentes/profesores:

- Mauricio Romero (Profesor de Economía del ITAM)
  - https://mauricio-romero.com/teaching/microeconometria-aplicada-otono-2021/
- Nick Huntington (Profesor en quien se apoyó Mauricio)
  - https://nickchk.com/econometrics.html#learnR
- Curso de MPA de Adrián Lucardi-> Eventualmente lo llevarán.
- R for Data Science de Hadley Wickham y Garret Grolemund

Animo a checar estas fuentes ante cualquier inquietud adicional.

Section 1

Material de los labs

Github

Todo el material de apoyo de los labs estará en Github.

- rodrigonp/Curso_R

Section 2

Estructura de los labs

Introducción a R

El curso están pensados para principiantes en programación. Tres sesiones:

- Nociones básicas del lenguaje: data frames, funciones base
- Análisis de datos (Dplyrs)
- Ggplot

Cualquiera que quiera aprender R puede asistir.

MPA

- Mucho de lo que vamos a ver lo van a cubrir en MPA
- Si ya lo vieron, sirve para repasar, profundizar o resolver dudas
- Para los que son de MPA recomiendo ver la presentación Como_pros en el Github si es que ya saben R
- Les puedo asistir con dudas puntuales o con sus tareas.

Section 3

¿Qué es R?

Lenguaje orientado a objetos

A grandes razgos, con R podemos:

- Crear objetos
- Manipular objetos
- Ver objetos

¿Qué es un objeto? En nuestro caso, todo lo relacionado a datos: variables, observaciones, regresiones, tablas de regresión.

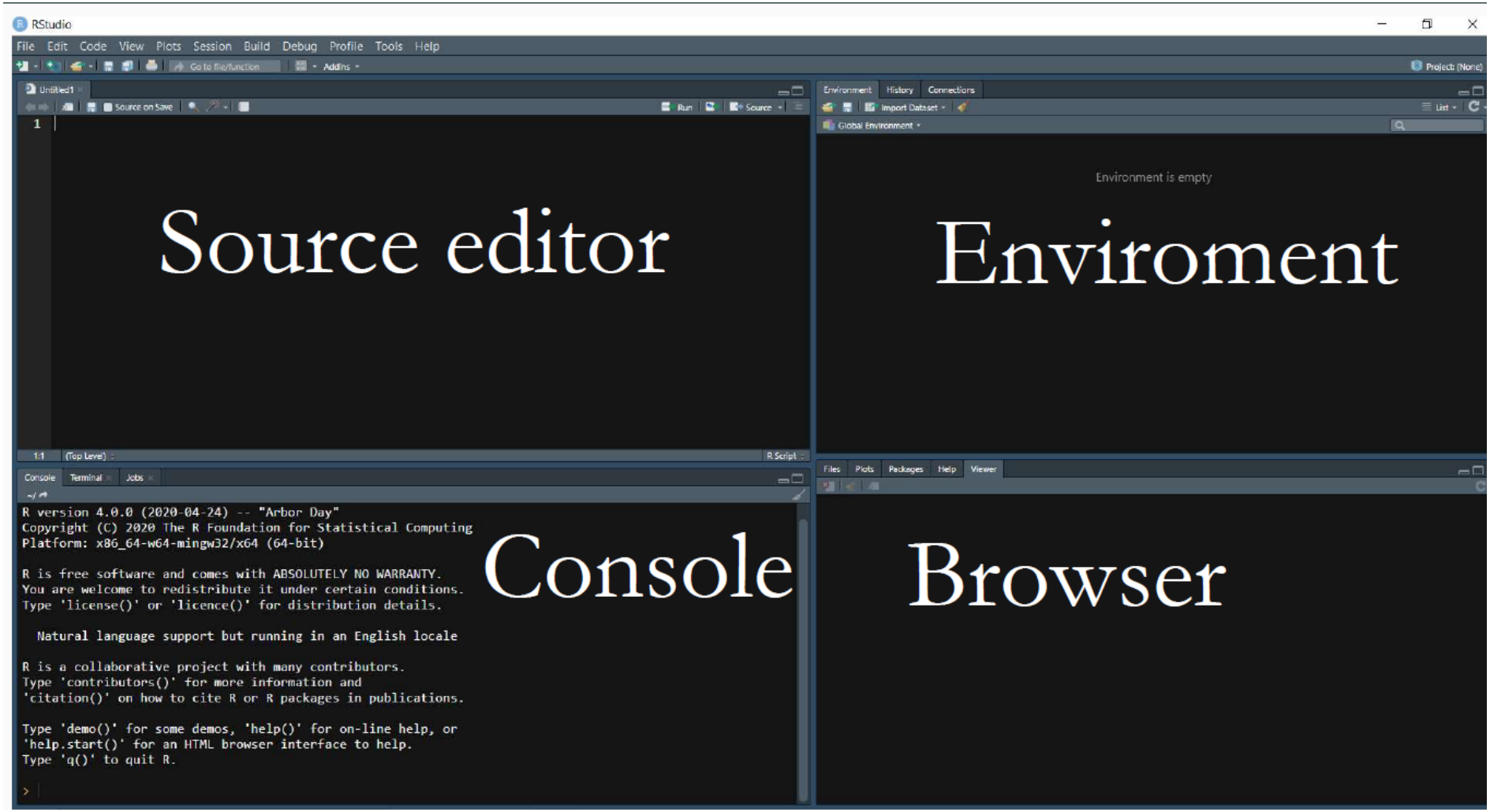
¿Por qué R?

- Gratis: todo mundo lo usa, hasta autores.
- Como con la gramática: aprendes uno y entiendes el resto (parecido a Python)
- Replicabilidad

Section 4

Interfaz

Interfaz


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Source editor
Environment is empty
Enviroment
Console
R version 4.0.0 (2020-04-24) -- "Arbor Day"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (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.
Browser

Consola

- Podemos ejecutar código aquí, sin embargo, no se guarda
- Muestra los errores en otro color (QUE SIEMPRE HAY QUE LEER)
- Muestras las advertencias (warnings).

Errores vs Warnigns

- Los errores impiden que R corra el código
  - Dividimos sobre 0
  - Queremos sacar el promedio de una palabra
- Los Warnings son avisos: R pudo correr el código, pero hay algo sospechoso

Environment

Destacan el environment y y el historial

- El historial es un registro del código corrido
- El environment es un registro de los objetos que hemos creado
- Podemos borrarlo con rm(list=ls()) o con la escobita

Navegador

- Podemos ver archivos y paquetes instalados
- PANEL DE AYUDA
- Gráficas

Source

A pesar de que el código se puede ejecutar en la consola, aquí deberían trabajar el código. El código aquí se guarda: REPLICABILIDAD.

- Después de escribir, se puede ejecutar con ctrl+ enter, seleccionando previamente
  - o picando los botones arrriba en la pestaña del source, etc.
- R no ejecuta nada despues de #, así que se pueden hacer comentarios después de un #

Ayuda

- R puede autocompletar nombre de variables, funciones, etc.
- Podemos buscar una funcion en el navegador y saldrá la documentación. O también podemos ejecutar el commando ?función
- GOOGLEEN: R se aprende más en StackOverflow que en cursos.

Section 5

Creación y tipos de objetos

Creación de objetos

- Creamos objetos con <- , = o ->
- El nombre de un objeto solo puede incluir letras, números, guion bajo (_) y punto Hay muchos tipos de objetos, veamos los más básicos

```
numeric.var<- 1
character.var<-'Mexico'
factor.var<- factor(1, labels = 'one')
logic.var<- TRUE #Booleanos # TRUE= T, FALSE= F
```

- Los objetos creados se pueden ver en el environment

- R no siempre muestra el output.
- Podemos encerrar las tareas con () y R imprime el output

- Si los ejecutamos, R los muestra. Si solo quieres ver el objeto, conviene hacerlo en la consola para no tener que borrarlo otra vez

```
numeric.var
```

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

- Podemos preguntarle a R si un objeto es de algun tipo con is.tipo

```
is.numeric(numeric.var)
```

```
## [1] TRUE
```

- y podemos usar as.tipo para cambiar entre tipos de variables

```
as.character(numeric.var)
```

```
## [1] "1"
```

```
as.numeric(numeric.var)
```

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

```
as.factor(character.var)
```

```
## [1] Mexico
## Levels: Mexico
```

Modificar un objeto lo sobreescribe: NO GUARDA DOS VERSIONES, ANULA LA ANTERIOR Y SE QUEDA CON LA NUEVA

```
numeric.var
```

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

```
numeric.var<-3+sqrt(9)+9^4-4+9/7
numeric.var
```

```
## [1] 6564.286
```

- nótese el uso de los operadores algebráicos comunes.
- recordemos que la consola no guarda, así que ahí podemos hacer cálculos rápidos.

Caracteres o strings

Son un trozo de texto encerrado entre comillas. Podemos usar dos comillas o el apóstrofe (que recomiendo porque es más rápido)

```
name<-'Rodrigo'
```

Lógicos o Booleanos

Podemos preguntarle algo a R y nos contesta con un TRUE o FALSE

```
a<-59

a>100
```

```
## [1] FALSE
```

- True y False se pueden escribir como su primera letra en mayúscula, T y F.

Como en casi todos los lenguajes, los operadores son:

- & para 'y', 'intersección de conjuntos'
- | para 'o inclusiva', 'unión de conjuntos'
- == 'igual a'
- >= 'mayor o igual a'
- ! 'no'
- != 'diferente'

Algo útil es que TRUE=1 y FALSE=0

```
T+3
```

```
## [1] 4
```

Práctica

```
is.logical(is.numeric(FALSE))
is.numeric(2)+is.character('hola')
T|F
T & F
```

```
## [1] TRUE
```

```
## [1] 2
```

```
## [1] TRUE
```

```
## [1] FALSE
```

Factores

Son variables categóricas mutuamente excluyentes. Se ven como caracteres, pero tienen niveles, que son el número de categorías.

```
consolas<-as.factor('xbox')
```

podemos añadir niveles de una manera mas sencilla

```
(levels(consolas)<-c('xbox','switch','ps5'))
```

```
## [1] "xbox"   "switch" "ps5"
```

Section 6

Vectores

Vectores o listas

Podemos concatenar objetos usando **c( )**. De preferencia, que sean del mismo tipo. Un vector es una lista de objetos, una colección de objetos. Podemos saber su longitud usando **length()**

```
vector<-c(1,4,5,6)
vector<-c(vector, 1,4,6,8,9)
length(vector)
```

```
## [1] 9
```

Slices (particiones)

Podemos llamar a partes específicas de los vectores utilizando paréntesis. Dentro de los paréntesis podemos especificar las posiciones deseadas o incluir un operador lógico. R nos va a devolver la parte del vector para cuya condición lógica es True

```
vector<-c(1,4,5,6)
vector[3]
```

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

```
vector[vector<5]
```

```
## [1] 1 4
```

:

Un operador habitual es ':' que se interpreta como de la posición x a la y.

```
vector<-c(1,4,5,6)
vector[1:3]
```

```
## [1] 1 4 5
```

También podemos hacer series

```
series<-1:10
series
```

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

Aunque lo mejor sería usar la función seq()

```
seq(10)
```

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

```
seq(2,10)
```

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

```
seq(2,10,2)
```

```
## [1]  2  4  6  8 10
```

!

- Otro operador es '!' que se interpreta como el complemento

```
vector<-c(1,4,5,6,15,3,20)
vector[vector<5]
```

```
## [1] 1 4 3
```

```
vector[! vector<5]
```

```
## [1]  5  6 15 20
```

Muchas funciones utilizan vectores

```
mean(vector); sd(vector); prod(vector)
```

```
## [1] 7.714286
```

```
## [1] 7.016986
```

```
## [1] 108000
```

o puedes operar con los vectores mismos, y funciona como un vector matemático

```
r<-c(1,4,6,4,2,5,9)
r*2
```

```
## [1]  2  8 12  8  4 10 18
```

```
r+r
```

```
## [1]  2  8 12  8  4 10 18
```

```
r>=4
```

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

Como vimos, podemos utilizar factores para ver cuantas observaciones pertenecen a una categoría

```
carreras<-as.factor(c('eco','eco','cpol','ri','ri'))
table(carreras)
```

```
## carreras
## cpol  eco   ri
##    1    2    2
```

%in%

Usamos el operador '%in%' para ver si un objeto pertenece a un vector. Denota si un objeto está dentro de un vector.

```
carreras<-c('eco','eco','cpol','ri','ri')
'mat' %in% carreras
```

```
## [1] FALSE
```

Otras funciones

rep() copia un vector y lo repite algunas veces.

- rep(*objeto a repetir*, *número de veces a repetir*)

```
rep(4,4)
```

```
## [1] 4 4 4 4
```

```
a<-c(1,7,9)
rep(a,3)
```

```
## [1] 1 7 9 1 7 9 1 7 9
```

Podemos hacer vectores de ceros o de texto vacío

```
numeric(5) #vector de 0 el numero de veces indicada
```

```
## [1] 0 0 0 0 0
```

```
character(6) # lo mismo, pero para characters
```

```
## [1] "" "" "" "" "" ""
```

sample() toma una muestra aleatoria. Se puede especificar el reemplazo

- sample(*objeto para tomar la muestra*, *tamaño de muestra*, replace= FALSE)

```
sample(1:10,3)
```

```
## [1] 4 6 7
```

```
a<-c(2,4,5,6,1,3,12,45,56)
sample(a,4)
```

```
## [1]  2  4 45  1
```

```
sample(a, 2, replace = F)
```

```
## [1] 56 45
```

Menciones honorables

- sort() acomoda las entradas del vector según se especifique
- unique() da las entradas únicas en un vector con posibles repeticiones
- max()
- min()
- length() da la longitud del vector

Particiones con vectores independientes

Podemos hacer particiones lógicas, incluso usando otro vector.

```
coin_toss<-sample(
  rep(c('aguila', 'sol')), 10 ,
  replace = T)

y<-1:10

y[coin_toss=='aguila']
```

```
## [1] 1 4 6 9
```

Section 7

Ejercicios Mauricio Romero

Ejericios tomados de Mauricio Romero

```
f<-c(2,7,5,1)
f^2
f + c(1,2,3,4)
c(f,6)
is.numeric(f)
mean(f >= 4)
f*c(1,2,3)
length(f)
length(rep(1:4,3))
f/2 == 2 | f < 3
as.character(f)
f[1]+f[4]
c(f,f,f,f)
f[f[1]]
f[c(1,3)]
f %in% (1:4*2)
```

```
## [1]  4 49 25  1
```

```
## [1] 3 9 8 5
```

```
## [1] 2 7 5 1 6
```

```
## [1] TRUE
```

```
## [1] 0.5
```

```
## Warning in f * c(1, 2, 3): longitud de objeto mayor no es m
## longitud de uno menor
```

```
## [1]  2 14 15  1
```

```
## [1] 4
```

```
## [1] 12
```

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

```
## [1] "2" "7" "5" "1"
```

```
## [1] 3
```

```
##  [1] 2 7 5 1 2 7 5 1 2 7 5 1 2 7 5 1
```

```
## [1] 7
```

```
## [1] 2 5
```

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

- Crea un factor que seleccione aleatoriamente entre seis personas que pueden ser hombre, mujer u otro género
- Averigua la suma acumulada entre 45 y 987 y súmale la media de todos estos números. (averigua la función para suma acumulada)
- Reacomoda h<-c(1,3,5,23,-4) de mayor a menor
- ¿Cuántos múltiplos de 4 hay entre 344 y 899? (averigua cómo sacar residuos no econométricos)

Section 8

Data Frames

Data Frames

En la sección anterior vimos vectores. El siguiente paso natural es analizar bases de datos

- No son otra cosa que un conjunto de vectores del mismo tamaño
- Los data frames son un tipo de objeto: algunas funciones requieren que las tablas sean Data frames.

Section 9

Repaso: Creación de DF

Aprovechemos lo aprendido para crear un DF. Creemos una base de datos de n=4000 alumnos del ITAM que contenga:

- un numero natural que l@ identifique
- sexo
- edad
- carrera: eco, cpol, ri, derecho, conta, mat
- promedio general

Para crear la base de datos necesitamos cada uno de los vectores

Una práctica común es poner el tamaño de la base de datos como una variable, para poder modificarla fácilmente posteriormente. Los primeros dos vectores sabemos cómo hacerlos

```
n<-4000
id<-1:n
sex<-sample(c('h','m','o'), n, replace = T)
```

Para crear edades aleatorias podemos usar las funciones de distribución incorporadas en R.

- R tiene incorporadas funciones para generar vectores que provengan de las distribuciones de probabilidad más comunes:
- runif() para la distribución uniform
- rnorm()
- rbinom()
- etc.

Solo se deben especificar los parámetros pertinentes y el tamaño del vector

- Adicionalmente, qnorm() se usaría para ver los cuantiles

En nuestro caso conviene usar la uniforme:

```
edad<-runif(n, min = 17, max = 27)
```

Que salgan decimales es extraño, apliquémosle la función piso.

```
edad<-floor(edad)
```

Creemos el resto de los vectores

```
carrera<-sample(c('eco','cpol','ri','derecho','conta','mat'),
prom<-runif(n, 6, 10)
```

Creación de Data Frames

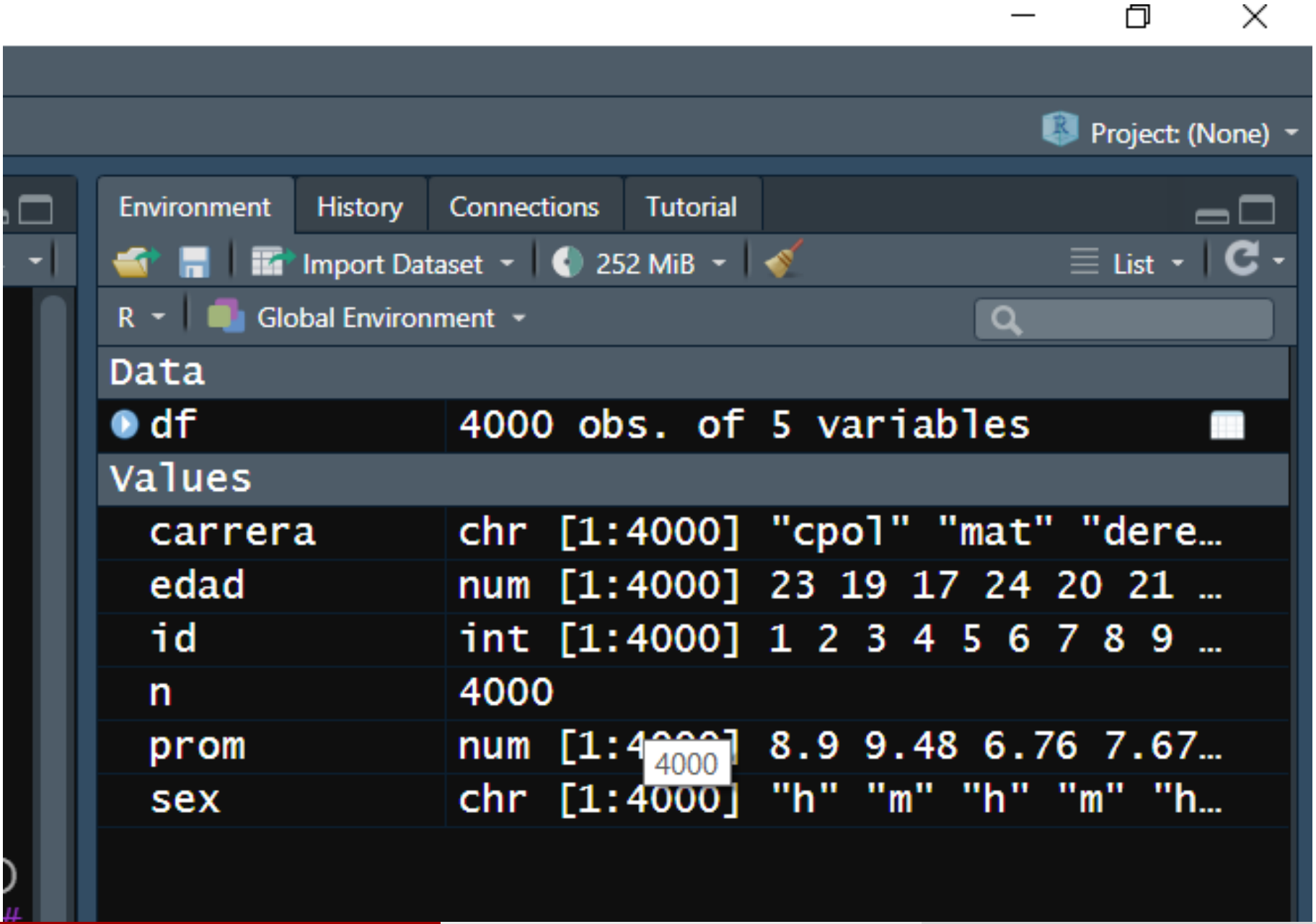
Crear data frames es muy sencillo, se hace con la funcion data.frame()

```
df<-data.frame(id, sex, edad, carrera, prom)
```

Section 10

Unidad de observación y variables

El recién creado Data Frame aparece en el environment.



Unidad de observación

- Definición: es la unidad mínima en la que puede cambiar el valor de una variable
- Es cada una de las filas de la base de datos. En nuestro caso, los alumnos.
- Si una categoría aparece varias veces, por sí sola no puede ser la unidad de observación.
- Ejemplos:
  - Diputados
  - País-año
  - alumnos-semestre

Tipos de datos

4 tipos de datos:

- Cross-sectional/ de corte transversal
- Series de tiempo
- pooled cross sections
- panel/ "time series cross-sectional"

Cross_sectional/ de corte transversal

- Cada unidad aparece una sola vez. Es una fotografía de la unidad tomada en un punto particular del tiempo
- NO HAY UNIDADES DE TIEMPO
- Ejemplos:
  - Encuestas

Series de tiempo

- Observaciones de UNA entidad a lo largo del tiempo.
- El orden importa
- Ejemplos:
  - PIB
  - Tasas de interés

Pooled cross section

- Combinaciones de al menos dos *cross-sections*
- mismas variables son analizadas para al menos dos periodos de tiempo, pero sin seguir a las mismas unidades.
- Seguir individuos es costoso: mejor tomas muestras representativas en distintos periodos de tiempo.
- Ejemplos:
  - LAPOP
  - ENIGH

Panel / Time series cross-sectional

- Una serie de tiempo para cada miembro cross-sectional
- un conjunto de entidades es observado varias veces en el tiempo
- Ejemplos:
  - Líderes mundiales
  - Datos OCDE

Observaciones y variables

- Las observaciones son las filas: las entidades mínimas que estamos observando
- Las **variables** son aquello que estamos observando de la unidad de observación: las columnas

Tipos de datos: resumen

(a) *cross-sectional*

id	time	x_1	x_2	x_3	...	x_n
A	1				...	
B	1				...	
C	1				...	
D	1				...	
E	1				...	
F	1				...	
G	1				...	
H	1				...	
...	...	...	...	...	...	...
N	1				...	

(b) *time series*

id	time	x_1	x_2	x_3	...	x_n
A	1				...	
A	2				...	
A	3				...	
A	4				...	
A	5				...	
A	6				...	
A	7				...	
A	8				...	
...	...	...	...	...	...	...
A	*T*				...	

(c) *pooled cross-sectional*

id	time	x_1	x_2	x_3	...	x_n
A	1				...	
B	1				...	
C	1				...	
D	2				...	
E	2				...	
F	2				...	
...	...	...	...	...	...	...
X	*T*				...	
Y	*T*	...	...	...	...	...
N	*T*				...	

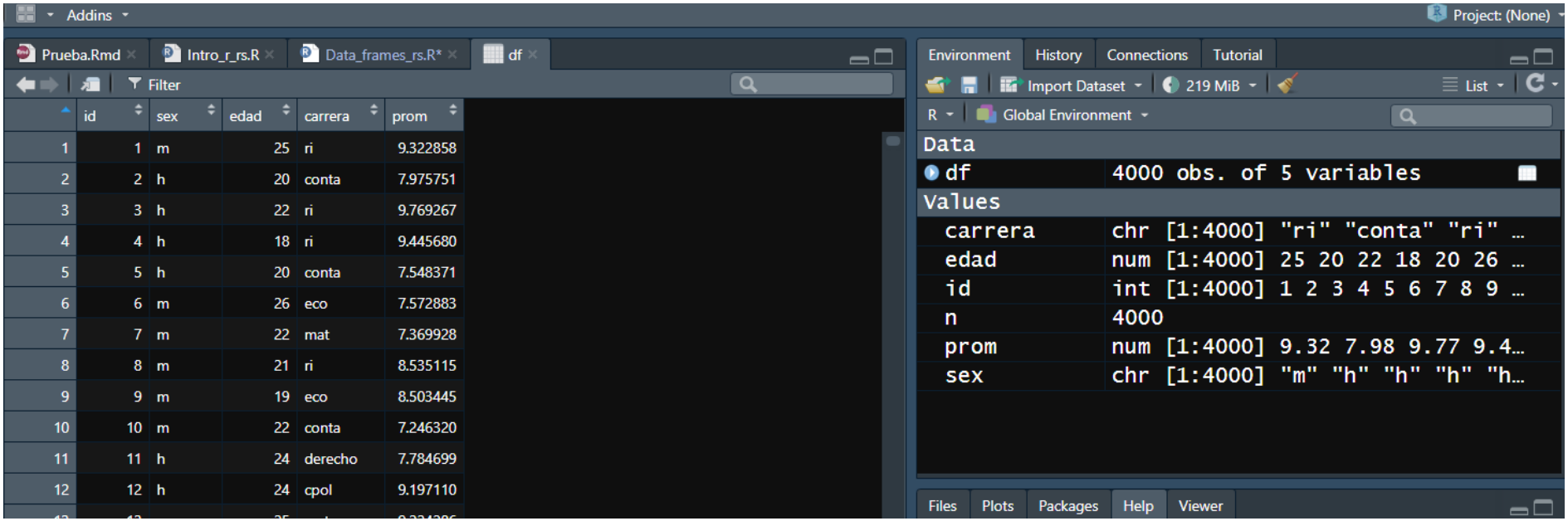
(d) panel

id	time	x_1	x_2	x_3	...	x_n
A	1				...	
A	...				...	
A	*T*				...	
B	1				...	
B	...				...	
B	*T*				...	
...	...	...	...	...	...	...
N	1				...	
N	...	...	...	...	...	...
N	*T*				...	

Section 11

Semillas

Podemos hacer click sobre el df en el environment para que nos lo muestre R



	id	sex	edad	carrera	prom
1	1	m	25	ri	9.322858
2	2	h	20	conta	7.975751
3	3	h	22	ri	9.769267
4	4	h	18	ri	9.445680
5	5	h	20	conta	7.548371
6	6	m	26	eco	7.572883
7	7	m	22	mat	7.369928
8	8	m	21	ri	8.535115
9	9	m	19	eco	8.503445
10	10	m	22	conta	7.246320
11	11	h	24	derecho	7.784699
12	12	h	24	cpol	9.197110

- ¡Pero cada quien obtendría un df distinto!
- Las variables de sexo, edad, carrera y promedio las obtuvimos usando cierta aleatoriedad

Semillas

- En realidad, R no utiliza valores aleatorios, sino pseudo-aleatorios a través de algoritmos.
- Gracias a dichos algoritmos podemos replicar los valores pseudo-aleatorios usando la función set.seed()

```
set.seed(2022)
```

si corremos la función set.seed() y el mismo número, y luego corremos la misma base de datos, deberíamos obtener el mismo df.

```
df<-data.frame(
  id=1:n,
  sex=sample(c('h','m','o'), n, replace = T),
  edad=floor(runif(n, min=17, max=27)),
  carrera=sample(c('eco','cpol','ri','derecho','conta','mat'),
                 n, replace = T),
  prom=runif(n, 6, 10)
  )
```

- Nota que también podemos crear el df directamente, sin generar las variables.

Hacia una mayor replicabilidad

Siempre que trabajemos con datos aleatorios en un proyecto hay que incluir la semilla para que nuestros resultados sean replicables.

Section 12

Visualizaciones de datos

- Como vimos, podemos hacer click en el df en el environment para ver el df.
- Podemos nombrarlo en la consola
- Podemos usar funciones preestablecidas para darnos una mejor idea de los datos que estamos viendo.

Funciones para visualizar

- summary() Muestra un pequeño resumen para cada variable: media, max, min.
- head() Muestra las primeras observaciones
- str() muestra algunas variables y qué tipo de objeto son.

Particiones de Df

- Podemos extraer los vectores de las variables con '$'

```
df$prom
```

```
##    [1] 7.611360 9.223240 9.265613 6.508555 7.800716 9.60806
##    [9] 6.686259 8.734800 6.865451 9.125295 8.237313 6.52348
##   [17] 7.396828 7.278871 8.220931 8.038379 7.268943 9.37439
##   [25] 9.198753 7.675358 7.794355 7.731782 6.147772 8.15219
##   [33] 6.270802 8.241269 9.191177 6.746612 7.135915 7.32094
##   [41] 8.786593 8.647539 6.136653 9.604074 6.514588 9.58391
##   [49] 8.265105 9.244561 6.781065 6.260725 9.797496 7.41810
##   [57] 8.134904 6.193265 6.689217 8.572138 7.187714 7.92694
##   [65] 8.538962 8.234297 7.795317 8.680072 6.702359 9.48452
##   [73] 7.325837 6.348668 8.719591 7.889563 9.063951 8.76501
##   [81] 9.919212 9.312787 6.699627 7.898962 6.300744 7.00294
##   [89] 8.087187 7.703957 9.945827 8.954098 9.245917 7.84609
##   [97] 6.531157 8.601563 6.921033 9.756296 8.267582 6.05319
```

Podemos operar con este vector como antes

```
df$prom[5]
```

```
## [1] 7.800716
```

```
mean(df$prom)
```

```
## [1] 7.990082
```

Slices de DF

Podemos especificar partes del df

```
df[filas, columnas]
```

- Por ejemplo, si queremos las primeras dos filas y las columnas 2-3 y 5

```
df[1:2, c(2:3,5)]
```

```
##   sex edad    prom
## 1   o   24 7.61136
## 2   m   17 9.22324
```

Section 13

Algunas funciones importantes para DF

with()

A menudo, conviene usar la función with() para operar con los vectores en lugar de llamarlos con $

- Por ejemplo, calculemos el promedio de los promedios.

```
with(df, mean(prom))
```

```
## [1] 7.990082
```

Section 14

Ejercicios

Con el df creado

- Calcula el promedio de los hombres que estudian economía.
- Calcula la proporción de mujeres que estudian matemáticas. Para esto, recuerda que el promedio de una dummy es la proporción de observaciones que cumplen la característica.
- ¿Cuál es el id del hombre con promedio más alto en la carrera de Ciencia Política? Para esto, recuerda la función max() o sort()
- Crea una variable que identifique con un número la carrera: 1= eco, 2=ri, 3=e.o.c. Para esto, concatena ifelse()
- Obtén el DF de las mujeres que estudian RRII, pero con slices.

Creación de DF tipo panel

Creemos una base de datos de vacunación. Hay tres tipos de individuos: niños, adultos y audltos mayores. Hay tres periodos de tiempo: 1,2,3. Los adultos mayores se vacunan en el primer periodo; los adultos en el segundo; niños en el tercero. Crea una base de datos tipo panel con un identificador, una variable de edad (con distribucion uniforme y redondeada al entero menor), una variable de tiempo y una dummy que valga uno si al individio-tiempo le corresponde una vacuna.