Una introducción programática a R en RStudio

Diego Alburez Gutiérrez
alburezg@lse.ac.uk

London School of Economics

EncuestoEntusiastas - 2016

Objetivos

Al finalizar este taller, usted

1. entenderá la estructura básica de R;
2. podrá importar y manipular datos;
3. sabrá cómo crear escribir código reproducible en R
4. Conocerá procedimientos para describir la desigualdad de ingresos; y
5. tendrá una base sólida para seguir aprendiendo.

Contenido

Día 1

Qué es R[1]

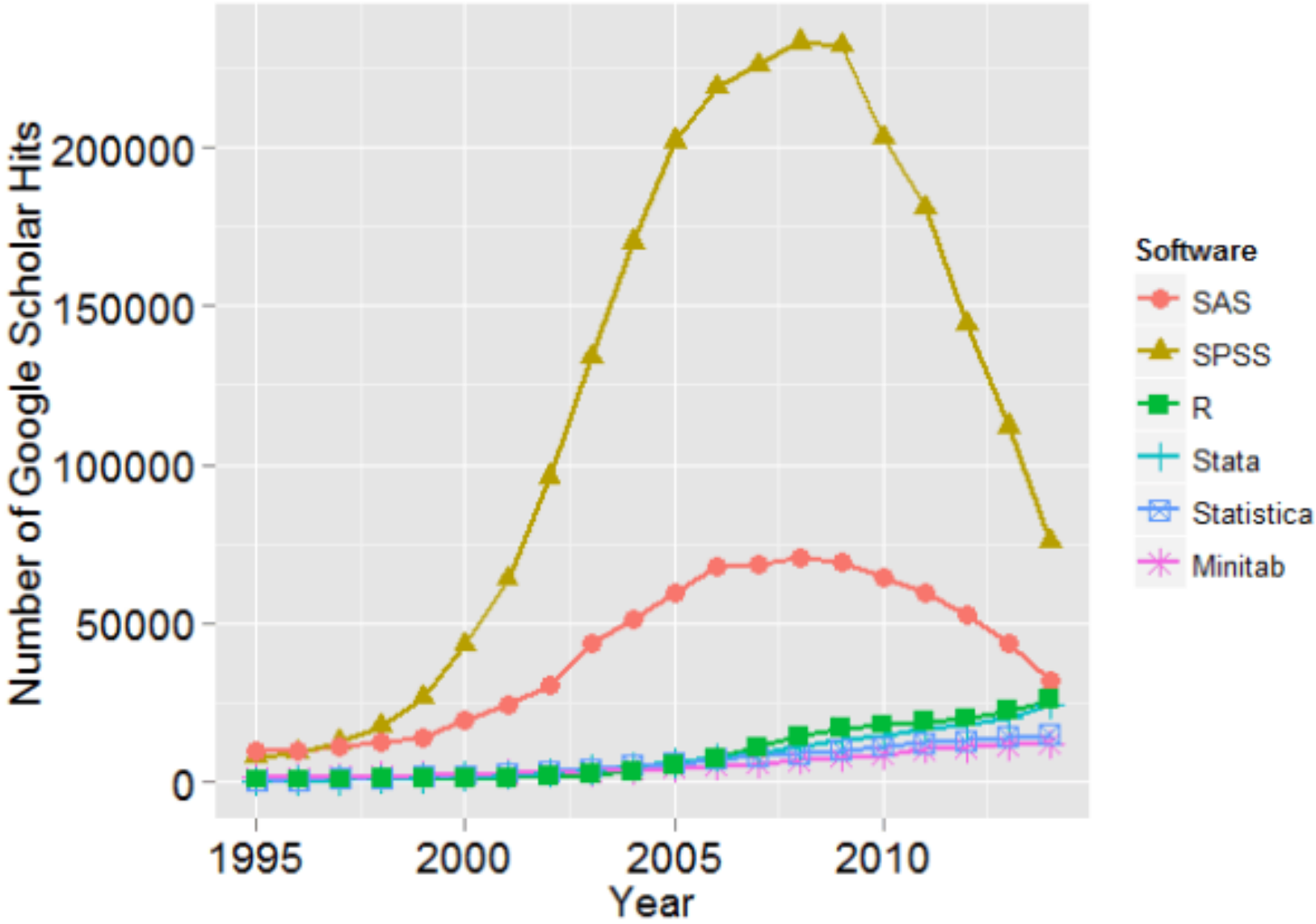
R is an integrated suite of software facilities for data manipulation, calculation and graphical display.

- an effective data handling and storage facility,
- a suite of operators for calculations on arrays, in particular matrices,
- a large, coherent, integrated collection of intermediate tools for data analysis,
- graphical facilities for data analysis and display either on-screen or on hardcopy, and
- a well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.

[1] https://www.r-project.org/about.html

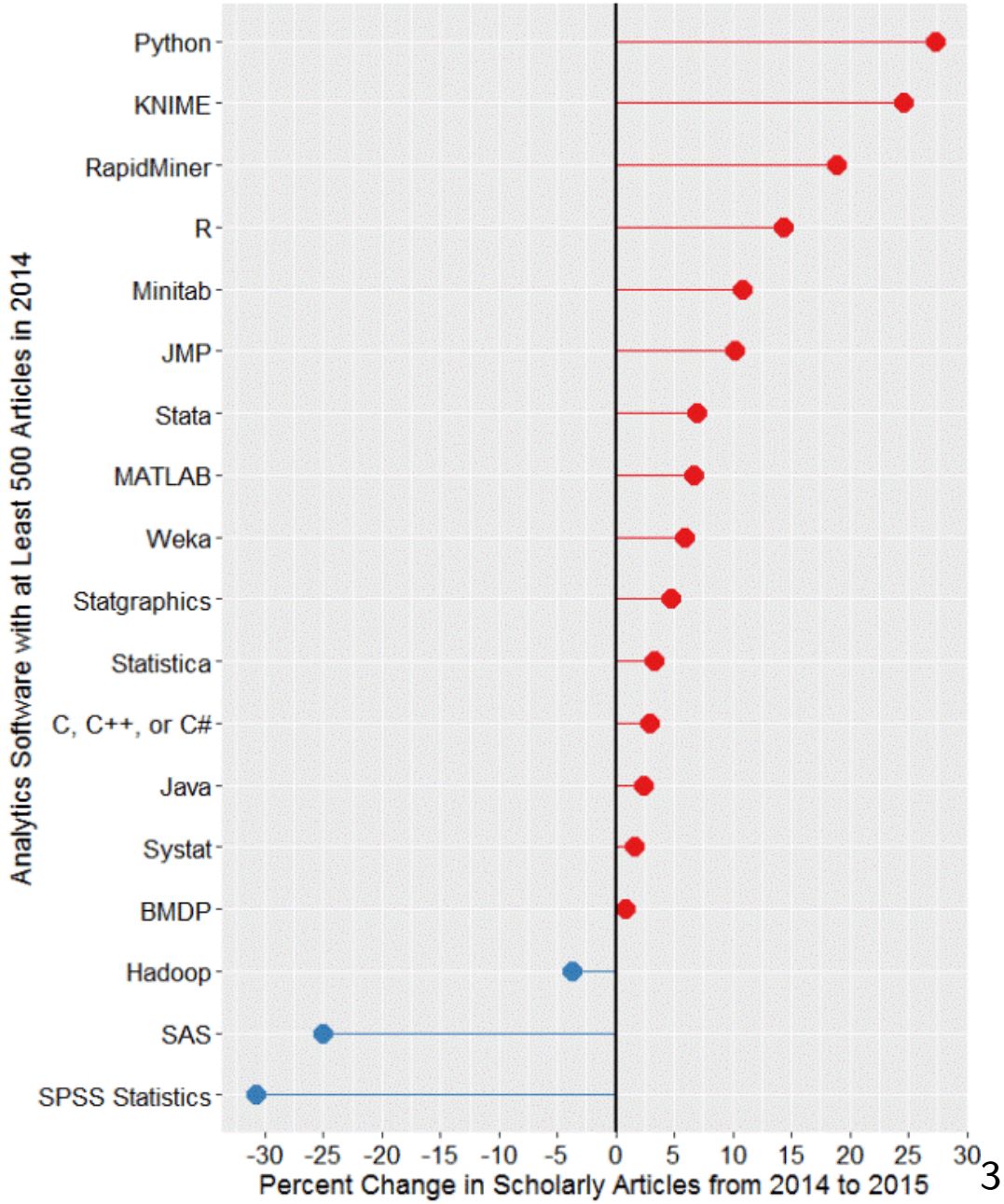
Por qué R

- Gratuito y colaborativo
  - Stackoverflow 139,342 temas
  - R-Bloggers
- Popularidad creciente
- Usado en varias disciplinas (SPSS = Statistical Package for the Social Sciences)
- En constante crecimiento
- Archivos portátiles (.csv y .txt)



2

[2] http://r4stats.com/articles/popularity/



[3]

[3] http://r4stats.com/articles/popularity/

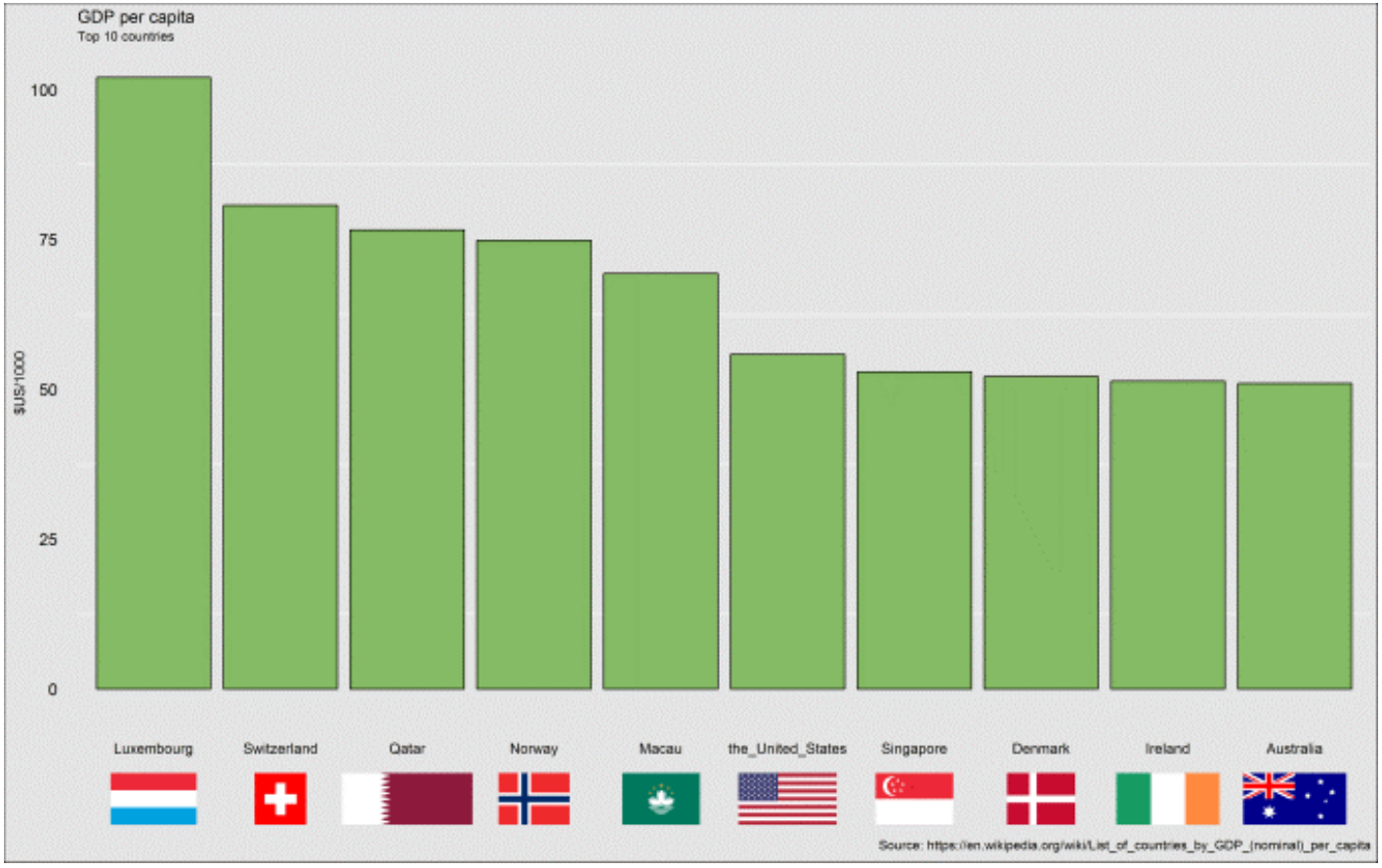
Qué puedo hacer con R

- Virtualmente cualquier análisis estadístico
- Análisis reproducible
- Gráficas
- Correr código python, SQL, Stata, etc.
- Integrar con LaTeX
- Generar reportes via markdown

Además..

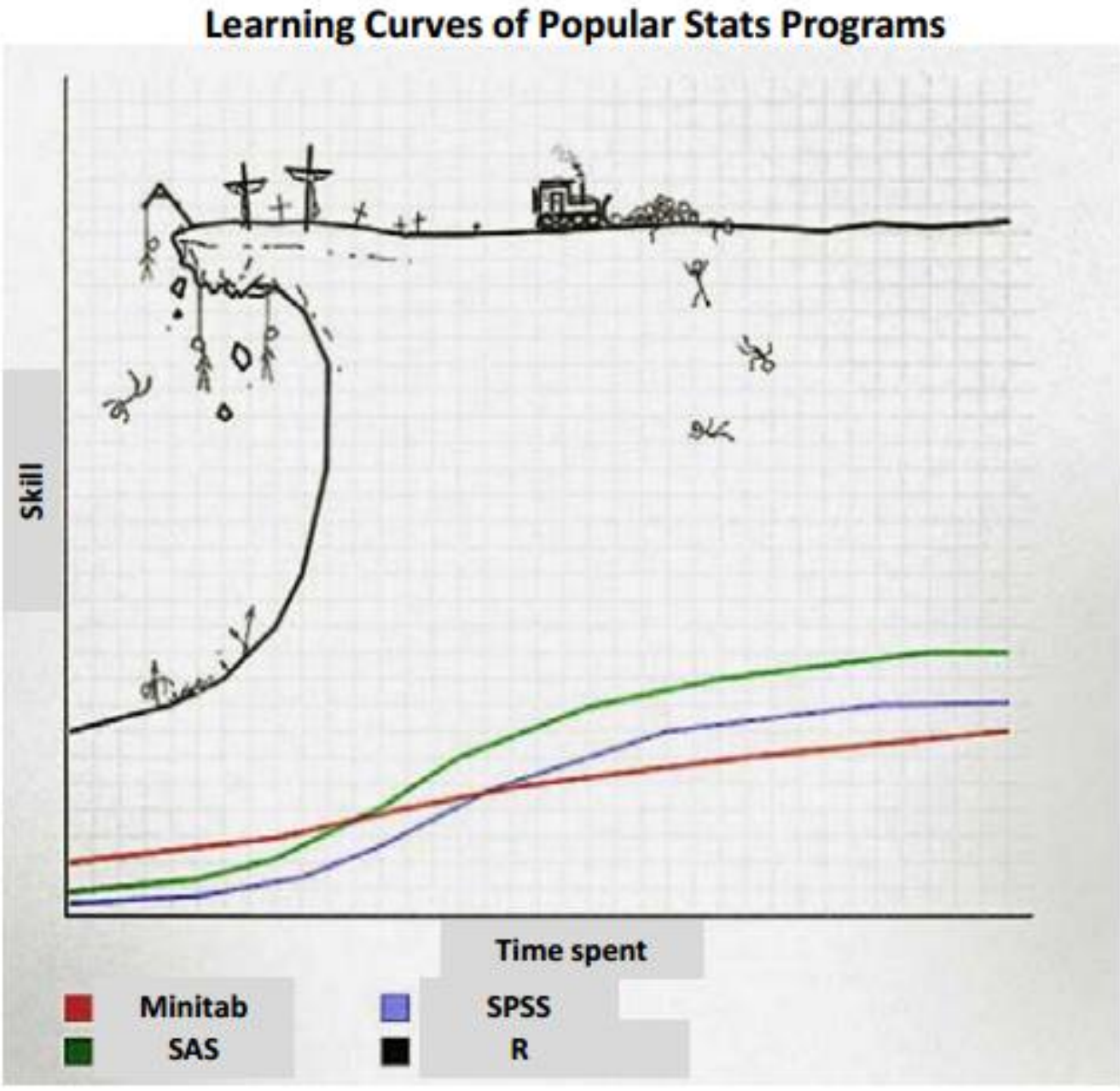
- Cualquier cosa: Lenguaje Turing completo
- Aplicaciones web: mortalidad, terror
- Análisis cualitativo, OCR, 'Cloud computing', etc.

Imágenes para identificar ejes?



Mapas, etc

Pero...


Learning Curves of Popular Stats Programs
Skill
Time spent
Minitab
SAS
SPSS
R

1. Día 1
   - Qué es R y por qué usarlo
   - RStudio: una interfaz de usuario para R
   - Estructuras de datos en R
   - Ejercicios para llevar
2. Día 2
   - Estructuras de control
   - Funciones
   - Importar y explorar datos
   - Ejercicios para llevar
3. Día 3
   - Operaciones vectorizadas
   - Aplicación: estimaciones de desigualdad en R
   - Estilo y Depuración
   - Recursos

RStudio

Principales elementos:

- Línea de comando
- Ventana de objetos/historial
- Visualizador
- Ventana para edición de código

Operadores de asignación

Dos operadores casi análogos:

- <- preferible por ahora
- = (para asignar argumentos en una función)

Ejemplo

```
 x <- 42
 x
[1] 42
 y <- x
 y
[1] 42
 x + y
[1] 84
```

Operadores aritméticos

- + Suma
- - Resta
- * Mult
- / Div
- ^ Expon

Ejemplo

```
 x <- 6^2
 x
[1] 36
 x*4
[1] 144
y <- (x*4+6)/3
y
[1] ?
```

Operadores lógicos

- < menor que; > mayor que
- <= menor igual que; >= mayor igual que
- == igual a
- != no es igual a
- | OR
- & AND

Ejemplo

```
 x <- 42
 x == 42
[1] TRUE
 x >= 6
[1] FALSE
 x = 6 # que hace esta linea?
```

Tipos de objetos

1. Vectores
2. Matrices
3. Data frames
4. Operadores de referencia (ojo, estos no son objetos!)
5. Listas
6. Funciones
7. Paquetes

Vectores

Contenedores de datos de menor nivel. Pueden ser:

1. Numéricos (numeric, integer); ej. 1 4 6
2. Caracteres (character); ej. "a" "b" "c"
3. Lógicos (logical); ej. TRUE FALSE
4. Fecha (date, POSIXt, ...); ej. "2009-09-15"
5. Categóricos (factor)
6. ...

Recordar que:

- Cada vector puede albergar datos de un solo tipo
- Los vectores no pueden tener nombres numéricos ni incluir operadores (+,-,...)
- Los vectores no deben llamarse de la misma forma que funciones existentes.
- Usar el operador c() para crear vectores con más de un elemento.
- Usar comillas ("") para asignar valores de texto

Valores desconocidos (missing values)

- Representados como NA en R

```
localidad <- c("urbano", "rural", NA, NA,
               "rural")
localidad
[1] "urbano" "rural" NA NA "rural"
# Se evaluan mediante el comando 'is.na()'
is.na(localidad)
[1] FALSE FALSE  TRUE  TRUE FALSE
# Use '!'  para evaluar la funcion opuesta:
!is.na(localidad)
[1]  TRUE  TRUE FALSE FALSE  TRUE
```

Ejemplo

```
 x <- (1,2,3)
Error:  unexpected ',' in "x <- (1,"
x <- c(1,2,3)
x
[1] 1 2 3
mult <- 66
x * mult
[1] 66 132 198 # Por que?
x
[1] 1 2 3 # Por que?
resultado <- x * mult # Asignamos resultado
resultado
[1] 66 132 198
```

Vectores

Ejemplo

```
valores <- c("perro", 45, "estela") # Vector mixto
valores
[1] "perro"  "45"      "estela" # Que paso aqui?
dias1 <- c("lunes", "martes")
dias2 <- c("miercoles", "jueves")
dias1 + dias2
Error in dias1 + dias2 :  non-numeric argument to binary
operator
dias.trabajo <- c(dias1,dias2)
dias.trabajo
[1] "lunes" "martes" "miercoles" "jueves"
```

Matrices

Datos de una misma clase ordenados en filas y columnas.
Recordar que:

- Todas las columnas deben ser de una misma clase
- El comportamiento predeterminado es llenar la matriz por columnas

Sintaxis básica

```
matrix(vector, nrow = r, ncol = c,
 byrow = FALSE, dimnames=list(vector_rownames,
 vector_colnames))
```

Ejemplo

```
num <- c(1:4)
matriz1 <- matrix(num, nrow = 2, ncol = 2)
matriz1
     [,1] [,2]
[1,]    1    3
[2,]    2    4
matriz2 <- matrix(num,nrow=2,ncol=2,byrow=TRUE)
matriz2
     [,1] [,2]
[1,]    1    2
[2,]    3    4
matriz1^2
     [,1] [,2]
[1,]    1    4
[2,]    9   16
matriz1 + matriz2 # Que pasaria aqui?
```

Matrices

Ejercicio Crear una matriz con los siguientes nombres:

- Juan Osorio
- Clara Ixpata
- Pedrina Sic

Resultado esperado:

```
      [,1]       [,2]
[1,] "Juana"    "Osorio"
[2,] "Clara"    "Ixapata"
[3,] "Pedrina"  "Sic"
```

Solución

```
nombres <- c("Juana", "Osorio", "Clara",
 "Ixapata", "Pedrina", "Sic")
misnombres <- matrix(nombres, nrow=3, ncol=2,
 byrow=T)
misnombres
[1,] "Juana"    "Osorio"
[2,] "Clara"    "Ixapata"
[3,] "Pedrina" "Sic"
```

Hay una solución alternativa?

Data frames

- Más generales que matrices
- Pueden contener varios tipos de datos
- Similares a las bases de datos en SPSS y STATA
- Nota: todos los vectores que conforman una data frame deben tener la misma cantidad de elementos (ser del mismo largo)

Sintáxis básica

```
data.frame(..., stringsAsFactors = T)
```

Data frames

Ejemplo

```
 localidad <- c("urbano", "rural", NA, NA,
                "rural")
 television <- c("si","no","no",NA,"no")
 id <- 1:length(localidad)
 df <- data.frame(id = id, tel = television,
  loc = localidad)
 df
   id  tel     loc
1   1   si  urbano
2   2   no   rural
3   3   no    <NA>
4   4 <NA>    <NA>
5   5   no   rural
class(df)
[1] "data.frame"
```

Ejemplo

```
df$tel
[1] si    no    no    <NA> no
Levels: no si
class(df$tel)
[1] "factor"
df <- data.frame(id = id, tel = television,
  loc = localidad, stringsAsFactors = FALSE)
df$tel
[1] "si" "no" "no" NA   "no"
class(df$tel)
[1] "character"
```

Data frames

Factor o texto?

```
# Queremos agregar la columna 'comentarios' a nuestra df
# Que codigo usamos?
comentarios <- c("tiene dvd",NA,NA,NA,
  "esposo ausente")
df <- ?
```

Data frames

Precaución!

```
mala.df <- data.frame(id = 1:20,
   estado = c(0,1,0,0,1,0))
Error in data.frame(id, estado = ) :
arguments imply differing number of rows:  20, 6
# Qué pasó?
```

Data frames

Precaución!

```
# Qué pasó?
length(1:20)
[1] 20
length(c(0,1,0,0,1,0))
[1] 6
length(1:20) == length(c(0,1,0,0,1,0))
[1] FALSE
#
# Excepcion:  una columna contiene un valor único
data.frame(id = 1:2, estado = 1)
  id estado
1  1      1
2  2      1
3  3      1
```

Operadores de referencia

- usar $ para hacer referencia a una columna
- usar [row,col] para hacer referencia a una fila o columna

Tener en cuenta que

- En R, siempre se define la fila primero y la columna segundo
- datos[1,2] quiere decir 'fila = 1; col = 2' Es decir, el valor en la primera fila y segunda columna
- Los dos operadores de referencia pueden usarse en solitario o combinados
- Si uno de los argumentos de [row,col] falta, R lo interpretará en contexto

Ejemplo: Operadores de referencia

Ejemplo

```
vec <- c("piedra", "azucar", "molienda")
vec[1]
"piedra"
vec[1:3]
[1] "piedra"    "azucar"    "molienda"
vec[1,1] # que pasa aqui?
# para reemplazaar
vec[2] <- NA
vec
[1] "piedra"    NA    "molienda"
```

Ejemplo: Operadores de referencia

Hacer referencia a valores

```
nombres <- c("estela", "carlos", "pedro",
  "isabela", "juan", "herlinda", "mario",
  "clara")
sexo <- c("f","m","m","f","m","f","m","f")
df<-data.frame(nombres,sexo,stringsAsFactors=F)

df[1,] # que imprime esto?

df$nombres # que imprime esto?

df$sexo[3] # que imprime esto?
```

Ejemplo: Operadores de referencia

Substituir valores

```
df$nombres[2]
"carlos"
df$nombres[2] <- "julia" # que hace esto?
df$nombres[2]
"julia"
df$nombres[df$nombres == "julia"] <- "carlos"
df$nombres[2]
"carlos"
# para entender, veamoslo por partes
df$nombres == "carlos"
FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE
df$nombres[c(F,T,F,F,F,F,F,F)] <- "julia"
# es decir, sustituir valores que sean julia por carlos
```

Listas

Un vector compuesto de objetos. Pueden haber

- Listas de vectores
- Listas de data frames
- Listas de objetos mixtos

OJO! Usar operador '[[ ]]' para extraer elementos de una lista!

Ejemplo

```
x <- c("a","b")
y <- 1:100
lista <- list(x,y)
lista[[1]] # notar operador '[[]]'
[1] "a" "b"
```

Para qué queremos listas?

- Permitir a una función incluir varios objetos en la salida
- Agrupar/ordenar objetos
- Aplicar funciones apply a varios objetos
- ...

Paquetes

Colección integrada y coherente de funciones para algún tema concreto. Básicamente en R existen:

- Paquete 'básico'
- Paquetes creados por usuarios en CRAN
- Paquetes alternativos o Beta (Github, etc.)

Para utilizar paquetes

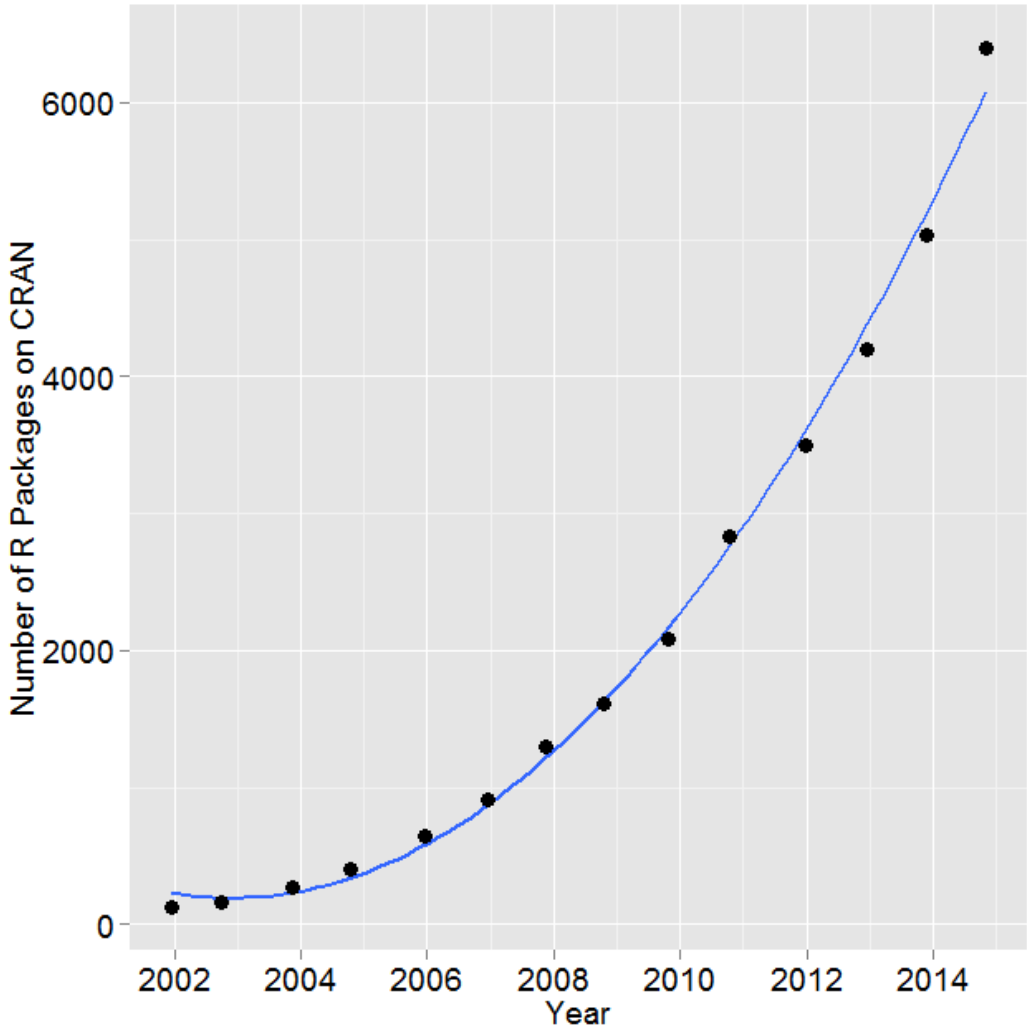
```
install.packages("dplyr")  # instalar paquete
library(dplyr) # cargar el paquete para sesion actual
?dplyr # archivos de ayuda sobre el paquete
?select # ayuda de funcion especifica del paquete
```

Paquetes

Algunos paquetes útiles

```
 # para manipular datos
install.packages("dplyr")
  # para trabajar con fechas
install.packages("lubridate")
  # para importar de y exportar a otros formatos
install.packages("foreign")
  # para trabajar con datos de encuestas
install.packages("survey")
  # para graficar
install.packages("ggplot2")
      # para crear aplicaciones web
install.packages("shiny")
```

Paquetes en CRAN



Ejercicios para llevar

Ejercicio 1: matrices

Cree esta matriz usando una única línea de código

```
      [,1] [,2] [,3] [,4] [,5]
 [1,]    1   21   41   11   31
 [2,]    3   23   43   13   33
 [3,]    5   25   45   15   35
 [4,]    7   27   47   17   37
 [5,]    9   29   49   19   39
 [6,]   11   31    1   21   41
 [7,]   13   33    3   23   43
 [8,]   15   35    5   25   45
 [9,]   17   37    7   27   47
[10,]   19   39    9   29   49
```

Solución 1: matrices

```
# Una solucion:
# crear vector
num <- seq(from=1,to=50,by=2)
# luego, crear la matriz
matrix(num,nrow=10,ncol=5,byrow=F)

# O, todo en una linea:
matrix(seq(1,50,2),nrow=10,ncol=5,byrow=FALSE)

# que hace el argumento 'byrow=FALSE'?
```

Ejercicio 2: data frame

#1. Cree esta data frame:

cuest	num	nacimiento	altura	peso	sexo
13	1	13/09/1989	174	66.1	"f"
13	4	05/08/1988	154	60.5	"f"
12	5	06/12/2000	130	110.3	"f"
11	1	08/08/1988	160	156.1	"m"
14	9	04/09/1999	136.6	134.7	"m"
14	1	01/06/2016	NA	NA	"m"

#2. Cree esta otra data frame:

id	estado_civil
14_9	1
14_1	NA
12_5	NA
13_1	1
11_1	0
13_4	1

Ejercicio 3: data frame

1. Substituya todas los valores desconocidos por ” .”
2. Concatene las columnas 'cuest' y 'num' para crear una nueva columna con un identificador único
3. Agregue la columna 'estado civil' de la segunda data frame a la primera
   - Cuántas columnas tiene la nueva base?
   - Qué clase tiene cada columna? Es la clase adecuada para el tipo de datos que contiene?
4. El peso de los cuestionarios 11,12 y 14 está en libras; transfórmelo a kg
5. Cree una nueva columna para guardar la razón de la altura al peso ($altura \div peso$) para cada fila
6. Ordene la base de datos con base en esta nueva columna
7. Guarde la columna 'nacimiento' como clase 'Date'

Ejercicio 4: listas[4]

Si:

```
w <- c(2, 7, 8)
v <- c("A", "B", "C")
x <- list(w, v),
```

Qué comando usamos para reemplazar "A" en x por "K"?

1. x[[2]] <- "K"
2. x[[2]][1] <- "K"
3. x[[1]][2] <- "K"

[4] tomado de http://www.r-bloggers.com/list-exercises/

Día 2

Argumentos condicionales

Permiten definir acciones basadas en la evaluación de condiciones lógicas.

Hay dos princiales:

- if
- else

Importante! A diferencia de Stata:

- El condicional if solo evalúa un argumento a la vez
- Si la condición contiene más de un argumento, solo se evalúa el primero y se imprime una advertencia ('warning')
- Para evaluar múltiples argumentos lógicos, use ifelse

Sintaxis básica

```
if (condicion) {accion1}
else {accion2}
# O bien:
if (condicion1) {accion1}
else if (condicion2) {accion2}
```

Ejemplo

```
# Condicional simple:
 x <- "presente en el hogar"
 if (x == "presente en el hogar") # notar '=='
   print("presente")
 if (x == "ausente del hogar")
   print("ausente")
 if (x != "presente en el hogar" &
     x !="ausente del hogar")
   print("desconocido")

# Ahora, que pasa si probamos:
num <- 1:10
if (num < 5) print("Numero chico")
```

ifelse

- ifelse es la forma vectorizada de if
- Uso: aplicar argumentos condicionales a múltiples elementos de un vector
- Similar a condicionales if else en Stata
- Pueden haber varios condicionales subsumidos
- Estructura: si condición se cumple, ejecutar acción1, de lo contrario, acción2
- NOTA: Recordar siempre asignar la función a un objeto para conservar el output

Sintaxis básica

```
ifelse(condicion, si, no)
```

Ejemplo ifelse

```
# datos espurios:
deptos <- c("JUT","HUE","SM","QTZ","CH","ZAC")
set.seed(42) # que es esto?
reg <- sample(deptos,10000,replace=T)
sexo <- sample(c("m","f"),10000,replace=T)
hacinamiento <- sample(0:1,10000,replace=T)
# recodificar por region:  "oriente" o "occidente"
reg<-ifelse(
  reg=="JUT" | reg=="CH" | reg=="ZAC", # cond1
  "oriente",  # accion1
  "occidente" # accion2
  )

table(reg)
```

Ejercicio ifelse

Utilice ifelse y (table o sum) para responder la pregunta:
Hay mayor prevalencia de hacinamiento entre mujeres en oriente que en occidente?

Solución ejercicio ifelse

```
# oriente:
# crear vector logico para identificar mujeres:
fem_or<-ifelse(sexo=="f" & reg=="oriente",T,F)
# conservar solo mujeres:
hac_or <- hacinamiento[fem_or]
sum_or <- sum(hac_or) # casos de hacinamiento
oriente <- sum_or/length(hac_or) # prevalencia
# occidente:
fem_oc<-ifelse(sexo=="f" & reg=="occidente",T,F)
hac_oc <- hacinamiento[fem_oc]
sum_oc <- sum(hac_oc)
occidente <- sum_oc/length(hac_oc)

oriente > occidente # evaluar pregunta
```

Bucles 'while'

Ejecuta una operacion mientras una condición sea satisfecha.

- El bucle realizará la operación en tanto el resultado de evaluar la condición sea TRUE
- Bucles infinitos con while(T) operacion

Ejemplo

```
# Sintaxis:
while(condicion logica) {operacion}
# Ejemplo:
anio_actual <- 2016 # Fecha actual
respuesta <- ""  # Por que hay que crear esta var?
while(respuesta != anio_actual) {
print("Te␣pregunto:")
respuesta <- readline("En␣que␣anio␣estamos?␣>␣")
}
```

Para interrumpir bucles

- next
- break

Ejemplo Un bucle que cuente en forma descendente desde 100 y se detenga al llegar a 50.

```
 # Como hacerlo utilizando next y break?
foo <- 100
while(foo >50) { # definir condicion
foo <- foo - 1
print(foo)
}
# Como hacerlo utilizando next y break?
```

Ejemplo: next, break

```
# Primera alternativa:
foo <- 100
while(T) {
foo <- foo - 1
print(foo)
if (foo <= 50) break # condicion
}
# Segunda alternativa:
foo <- 100
while(T) {
foo <- foo - 1
print(foo)
if (foo > 50) next # notar direccion de '>'
print("Fin del bucle") # se imprime una sola vez
break
}
```

Bucles 'for'

Estructura de control que permite realizar una operación iterativamente bajo ciertas condiciones.

- Pueden considerarse casos especiales de 'while'
- usa operadores '[ ]' para pasar valor actual del vector de cuenta dentro del bucle

Sintaxis básica

```
for (v en secuencia) {operacion}
```

Ejemplo: for

Instrucciones: imprima las siguientes líneas en la consola de R.

```
# Resultado esperado:
[1] "Hola␣estela"
[1] "Hola␣carla"
[1] "Hola␣pedrina"
[1] "Hola␣isabela"
[1] "Hola␣juana"
[1] "Hola␣herlinda"
[1] "Hola␣maria"
[1] "Hola␣clara"
```

Solución ejemplo: for

```
# Una solución:
# Crear un vector con los nombres para iterar:
nombres <- c("estela","carla","pedrina",
  "isabela","juana","herlinda","maria","clara")
# definir el bucle:
for (n in 1:length(nombres)) {
saludo <- paste("Hola", nombres[n]) # Notar [n]
print(saludo)
}
# O, mas compacto:
for (n in 1:length(nombres))
  print(paste("Hola", nombres[n]))
```

Funciones

Conjunto de operaciones, condicionales, etc. Se caracterizan por:

- Recibir una entrada (input) - OPCIONAL
- Realizar una operación - NECESARIO
- Devolver una salida (output) - OPCIONAL

Sintaxis básica

```
function(entrada) {
operacion
return(salida)
}
```

Ejemplo

```
# primero, definir la funcion
resta_argumentos <- function(x,y) {
z <- x - y
return(z)
}
resta_argumentos(1,99)  # ejecutar funcion
[1] -98
 # podemos definir condicionales en la funcion
resta_positivos <- function(x,y) {
 z <- x - y
 if (z < 0) print("Error: Salida negativa")
 else return(z)
}
resta_positivos(1,99)
[1] "Error: Salida negativa"
```

Funciones subsumidas (embedded)

- Es posible definir funciones que 'llamen' a otras funciones,
- siempre que no exista recursividad infinita.

Ejemplo

```
menor <- function(num) # func.  auxiliar 1
  print("Menor de 50")
mayor <- function(num) # func.  auxiliar 2
  print("Mayor de 50")
decidir <- function() { # funcion principal
 num <- readline("Escriba valor de 1 a 100 > ")
 num <- as.numeric(num)
 if (num <= 50) menor(num)
 else if (num > 50) mayor(num)
}
decidir() # que pasa al ejecutar funcion?
```

Scoping rules: Lexical scoping

Dadas las normas de ambientación de R

- Toda variable definida dentro de una función existe únicamente dentro de esa función (y las funciones que dependan de ella)
- la única excepción es el objeto indicado en la funcion 'return()'

Ejemplo

```
 # Primero, definir una funcion sin valor de salida
asignar.valor <- function(valor) {
 x <- valor
}
# Ahora el ejemplo...
asignar.valor(20)
print(x)
Error:  object 'x' not found
```

Importar datos

- Mejor formato: .csv
- Para importar de Stata, SPSS hay varios paquetes (foreign, RStata, etc.)
- Recordar asignar datos a un objeto en R para conservar
- Al escribir ruta de archivo (en 'file'), usar / en lugar de \
- stringsAsFactors es importante

Sintaxis básica

```
read.csv(file, header = TRUE, sep = ",",
         stringsAsFactors = TRUE)
read.spss(file, use.value.labels = TRUE,
          to.data.frame = FALSE, ...)\\
read.dta(file,convert.factors=TRUE,...)
```

Explorar datos

Datos estan en: 'Dropbox/curso R/data/ensmi_h.csv'

```
# Localizar archivo:
install.packages("utils")
library(utils)
path <- choose.files()
# Abrir archivo:
hogares <- read.spss(path,
                     to.data.frame = T)
# para explorar nuestros datos
str(hogares) # resumen de variables
head(hogares[,1:5]) # Muestra primeras 10 lineas
tail(hogares[,1:5]) # ultimas 10 lineas
nrow(hogares) # numero de filas
ncol(hogares) # numero de columnas
```

Ejemplo: recodificar variable

```
# MHALTITUD registra altitud de encuesta
hist(hogares$mhaltitud) # histograma
max(hogares$MHALTITUD) # valor maximo
min(hogares$MHALTITUD) # valor minimo
# Nos interesas altitudes menores a 3000m
hogares <- hogares[hogares$MHALTITUD < 3000,]
mean(hogares$MHALTITUD) # media de columna
median(hogares$MHALTITUD) # mediana de columna
View(hogares) # abrir en pestania
edit(hogares) # abrir para edicion
```

Tabulaciones y descripción

```
nombres<-c("estela","carlos","pedro","isabela",
           "juan","herlinda","mario","clara")
sexo <- c("f","m","m","f","m","f","m","f")
df <- data.frame(nombres, sexo)
df$id <- 1:length(sexo)
df$riqueza <- sample(1:100, length(sexo))
table(df$sexo) # tabular factores
# mean,median,25th and 75th quartiles,min,max
summary(df)
# con el paquete Hmisc:
install.packages("Hmisc")
library(Hmisc)
describe(df)
```

Ejercicios para llevar

Ejercicio 1: for

Escriba un bucle que

- iniciando en 1,
- en cada iteración genere un número entero aleatorio entre 0 y 100;
- en cada iteración imprima la suma de todos los números generados hasta ese momento; y
- se interrumpa cuando la suma de todos los números aleatorios hasta ese momento exceda 10,000
- (puntos extra: que al finalizar el bucle, imprima el número de iteraciones realizadas)

```
# Use 'sample' para numeros enteros aleatorios
sample(0:1, 10, replace = T)
[1] 1 1 0 1 1 1 1 0 1 1
```

Solución 1: for

```
# Una solucion:

set.seed(42) # que es esto?
suma <- 0
while(suma <= 10000 ) {
  aleatorio <- sample(1:100,1)
  suma <- suma + aleatorio
  print(suma)
}
```

Solución 1 (puntos extra): for

```
set.seed(42)
suma <- 0
iter <- 0
while(T) {
  iter <- iter + 1
  aleatorio <- sample(1:100,1)
  suma <- suma + aleatorio
  print(suma)
  if (suma >= 10000) {
    print(paste("iteraciones:",iter))
    break
  }
}
```

Ejercicio 2: for

1. Utilice un bucle 'for' y argumentos condicionales para imprimir el siguiente texto en la consola:

```
# Dados dos vectores:
nombres<-c("estela","carlos","pedro","isabela",
 "juan","herlinda","mario","clara")
sexo <- c("f","m","m","f","m","f","m","f")
# Resultado esperado:
[1] "estela - femenino"
[1] "carlos - masculino"
[1] "pedro - masculino"
[1] "isabela - femenino"
[1] "juan - masculino"
[1] "herlinda - femenino"
[1] "mario - masculino"
[1] "clara - femenino"
```

Solución 2: : for

```
# Una solucion:

for (n in 1:length(nombres)) {
  if (sexo[n] == "m") etiqueta<-"masculino"
  else if (sexo[n] == "f") etiqueta<-"femenino"
  print(paste(nombres[n], "-",etiqueta))
}
# Esto se puede vectorizar para optimizar
# Y este es el proximo ejercicio...
```

Ejercicio 3: ifelse

1. Repita el ejercicio 1 sin usar bucles:
2. en cambio, use ifelse o algún operador de asignación
3. Imprima el siguiente texto en la consola (no se preocupe si todo se imprime en una única línea):

```
# Resultado esperado:
[1] "estela - femenino"   "carlos - masculino"
"pedro - masculino"   "isabela - femenino"
[5] "juan - masculino"    "herlinda - femenino"
"mario - masculino"   "clara - femenino"
```

Solución 3: ifelse

```
## Una solución

# Crear etiquetas basado en sexo:
etiqueta <-
    ifelse(sexo == "f", "femenino", "masculino")
# Crear vector para imprimir
res <- paste(nombres, "-", etiqueta)
print(res) # Imprimir vector
```

Ejercicio 4: funciones

Instrucciones

- Funcion que imprima los primeros 10 valores de la secuencia Fibonacci iniciando con 1:

$$F_n = F_{n-1} + F_{n-2}$$

- Una función que, dado un vector, imprima la media, desviación estándar, y (puntos extra) la mediana

$$\sigma = \sqrt{1/N \sum_{i=1}^{N} (x_i - \mu)^2}$$

- Una función que, dada una base de datos y un número o letra, regrese como valor de salida la base de datos con ese número o letra substituido por NA.

Día 3

Bucles y operaciones vectorizadas

Las operaciones vectorizadas aplican una función a cada elemento de un vector 'simultáneamente' sin iterar en cada elemento.

- En R, todos los objetos son vectores
- Cada vector es de una clase específica

Considere la operación:

```
 t <- c(5,2,8)
 s <- c(9,5,1)
 # Es cada elemento en t mayor que su contraparte en n?
 # Solucion1:  usar bucle
 for (n in 1:length(t))
   print(t[n] > s[n])
# Solucion2:  vectorizar operacion
 t > s
```

Asignación de memoria[5]

Ejemplo 1

1. Buscar al vector en la memoria
2. Crear vector para acomodar mś datos (reasignar memoria)
3. Copiar los datos viejos e insertar nuevos
4. Borrar el vector viejo

Ejemplo 2 (más veloz)

1. R no debe reasignar en la memoria a cada iteración

```
 # Ejemplo1
j <- 1
for (i in 1:10) {
    j[i] = 10
}
 # Ejemplo2
 j <- rep(NA, 10)
for (i in 1:10) {
    j[i] = 10
}
```

Bucles y operaciones vectorizadas[6]

Por qué vectorizar?

- Optimización
- Evitar 'efectos secundarios' de bucles
  - for(n in 1:10) print(n)
  - Este loop asigna una variable temporal 'n' al environment actual
  - Esto puede causar problemas más adelante si se reusa 'n'
- Mantener balance optimización-esfuerzo

[6] http://www.noamross.net/blog/2014/4/16/vectorization-in-r--why.html

Cómo vectorizar?

- Pensar en términos de vectores
- Evitar usar bucles for donde sea posible
- Utilizar ifelse para aplicar operaciones condicionales a vectores
- Para vectorizar la aplicacioón de una función a lo largo de un vector, una data frame o una lista, utilizar funciones de la familia apply

Muchas operaciones se pueden vectorizar!

```
nombres <- c("Juana", "Pedrina", "Clara")
apellido1 <- c("Ixapata", "Osorio", "Sic")
n_completo <- paste(nombres, apellido1)
n_completo <- paste(n_completo, "Chen")
```

Ejemplo: Recodificar variables

datos > hogares.sav

```
setwd("C:/Users/USUARIO/Documents")
hogares2 <- read.csv("hogares.csv")
# mhp15 Fuente de agua para beber.
class(hogares2$MHP15)
levels(hogares2$MHP15)
# crear variable binaria:  agua pozo
pozo<-ifelse(
      hogares2$MHP15=="Pozo mec nico/manual",
      1, 0)
# crear variable binaria:  rio o lluvia
prec<-ifelse(hogares2$MHP15=="Agua de lluvia"|
      hogares2$MHP15==" R  o/acequia/manantial",
      1, 0)
hogares2$prec <- prec # opcional
```

Funciones apply

Familia de funciones utilizada para aplicar una función de forma iterativa a una serie de objetos.

- apply: aplica una función a filas o columnas de matriz (retorna lista, matriz o vector)
- lapply: aplica una función a un vector (retorna lista)
- **sapply** aplica una función a un vector (retorna vector o matriz)
- vapply: similar a sapply, con ciertos parámetros predefinidos

```
sapply(X, FUN, ..., simplify = TRUE,
  USE.NAMES = TRUE)
```

Ejemplo: calcular media de varias columnas

```
 id <- 1:10000
 # crear valores aleatorios para las columnas
 riqueza <- runif(10000,min=0.2, max=1)
 nutricion <- runif(10000,min=0.3, max=1)
 distancia <- runif(10000,min=0, max=0.7)
 prevalencia <- runif(10000,min=0, max=1)
 df <- data.frame(id,riqueza,nutricion,
  distancia,prevalencia)
mean(df$riqueza) # media de una columna
# Como calcular la media de cada columna?
```

sapply

sapply aplica una función a cada columna de una matriz base de datos

Retorna: vector o matriz

```
sapply(df, mean) # Que nos da esto?
# Otro ejemplo:
# Calcular valor minimo de cada columna mas uno
min_mas_1 <- function(col) {
 val <- min(col) + 1
 val
}
sapply(df, min_mas_1)
```

Estimaciones de desigualdad

Plan para cálculo de desigualdad

- Abrir 'curso R/ejercicios/desigualdad.R'
- Datos: Enei 2010 en 'curso R/data/enei.sav

Objetivos:

1. Importar datos a R
2. Descripción básica de datos
3. Análisis basado en cuantiles
4. Análisis usando indices de desigualdad

Estilo

- Tabulación
- Cambiar estilo de visualización
- Formato automático de texto
- Ocultar bucles
- Comentar código!

Depuración y manejo de errores

Permite detectar errores en funciones

- Es recomendable ubicar errores mediante 'print()'
- Si esto no funciona, existe la funcion 'debug()'

Ejemplo

```
fun <- function(x) x
debug(fun)
```

Depure: encuentre el error en la funcion

```
capitalBelice <- function(){
  while(T) {
    n <- readline("capital␣de␣belice?␣>␣")
    if (n != "belmopan") print("No␣lo␣creo")
    else if (n == "belmopan") {
      m <- readline("Esta␣segura/o?␣(s/n)>␣")
      if (m != "s" & m != "n") {
        print("Escriba␣un␣valor␣valido")
      } else if (m == s) {
        print("Muy␣bien!")
      } else if (m == "n") {
        print("Intente␣de␣nuevo.")
      }
    }
  }
}
```

Otros recursos

- Documentación en R (usando '?' en R o en CRAN)
- Comunidades de usuarios (ETHZ, stackoverflow)
- Ejercicios sobre varios temas
- Buscador rseek o rsitesearch
- Lista de distribución de R-bloggers
- Coursera: R programming
- Para manipulación de datos: Tutorial de dplyr (video)
- Una introducción a la poneración de datos de encuestas usando 'survey'
- Libros de texto: R in Action, The Art of R Programming, Cookbook for R

Tiempo de respuesta en Stackoverflow