IMM con R. Lab1: Aprendiendo a usar R con R-Studio

Dra. Maite Mascaro y Dr. Edlin Guerra Castro

11/01/2023

R, R-Studio y repositorios

El objetivo principal de este laboratorio es introducirlos a R y RStudio, las herramientas computacionales que utilizaremos a lo largo de este curso para aprender a aplicar los conceptos más importantes de *Análisis Multivariado con R*, pero en especial, para aprender a procesar y analizar datos reales.

El programa R, sus versiones actualizadas y todos los paquetes con funciones, asi como otra información relevante se encuentre en los repositorios de R conocidos como **CRAN** (Comprehensive R Archive Network). Los distintos servidores distribuidos en todo el mundo, conforman el CRAN y son conocidos como los **CRAN mirrors** (de espejo). Para descargar los paquetes requieres antes escoger un **CRAN mirror**, y la función que te permite escogerlo de una lista que aparece en la consola es `chooseCRANmirror`. Alternativamente, se pueden descargar paquetes desde otros repositorios, uno muy popular y que estaremos usando en esta asignatura es GitHub.

RStudio es un entorno de desarrollo integrado (IDE) para **R**. Incluye una consola, editor de comandos y líneas de programación que admite la ejecución directa de código, así como herramientas para graficar, documentar, registrar el historial de comandos ejecutados, acceder a archivos, y muchas cosas más desde la gestión de un espacio de trabajo. **RStudio** hace que el trabajar con **R** sea más poderoso, y a su vez simple. Para que tengan una idea, esta guía se escribió desde **RStudio**.

Una vez que se cargue la interface de *Rstudio*, generen un archivo de códigos vacío, siguiendo el símbolo de hoja en blanco y cruz verde justo debajo del menú **File**, es la primera opción. Un atajo para crearlo es con las teclas **Ctrl+Shift+N**. Este archivo vacío es un editor de texto inteligente, que reconoce ciertos caracteres y palabras para autocompletar los comandos. Ese espacio será su borrador de comandos y notas. En este archivo irá pegando y ejecutando todos los códigos que a continuación se explican. Acá un pequeño ejemplo, copien y peguen todo el código del recuadro de abajo en su editor de comandos. Luego, coloque el cursor al inicio de la primera línea de comando y presionen el botón **Run** en la esquina superior derecha de su achivo de códigos; alternativamente, presione **Ctrl+ENTER**. ¿Qué ocurrió? haga esto para las primeras seis líneas de comando. Cuando llegue a **p <- ggplot...** seleccione todas las líneas faltantes del código y presione Run o **Ctrl+ENTER**. ¿Qué ocurrió?

```
#Este código es para generar un gráfico sobre anomalías en la temperatura
#del planeta en las últimas décadas. Estas líneas de código van a instalar
#y carga tres paquetes, luego van a importar los datos y generar un gráfico
#com varios elementos gráficos añadidos.

install.packages("gcookbook") #instalación del paquete tutorial de gráficos gcookbook
install.packages("grid") #instalación de paquete gráfico grid
install.packages("ggplot2") #instalación de paquete gráfico ggplot2
library(gcookbook) #cargar paquete gcookbook en la sesión
library(grid) #cargar paquete grid en la sesión
library(ggplot2) #cargar paquete ggplot2 en la sesión

data("climate") #importar datos de clima global del paquete gcookbook
```

```r
#Generación del gráfico
p <- ggplot(subset(climate, Source == "Berkeley"), aes(x = Year, y = Anomaly10y)) +
  geom_line() +
  annotate(
    "segment",
    x = 1850,
    xend = 1820,
    y = -.8,
    yend = -.95,
    colour = "blue",
    size = 2,
    arrow = arrow()
  ) +
  annotate(
    "segment",
    x = 1950,
    xend = 1980,
    y = -.25,
    yend = -.25,
    arrow = arrow(
      ends = "both",
      angle = 90,
      length = unit(.2, "cm")
    )
  ) +
  theme_bw() +
  xlab("Año") +
  ylab("Anomalías de la temperatura global (°C)")
p

#Fin del código
```

Parte 1: Objetos, funciones y paquetes

R es un lenguaje orientado a objetos, lo que significa que las variables, datos, funciones, resultados, etc., se guardan en la memoria activa del computador en forma de objetos con un nombre específico dado por el usuario en cada sesión. Los objetos se manipulan mediante funciones (que, a su vez, pueden ser tratados como objetos) y operadores. La ventana de la consola es donde se escriben los comandos, después de un indicador o prompt `>` que notifica cuando R está listo para recibir la siguiente instrucción. La tecla `esc` aborta la tentativa de esa línea de comando y da la señal para que aparezca un nuevo prompt. Dos prompts `> >` seguidos invalida esa línea de comando. Si aparece un signo de `+` es que la linea de comando está incompleta y requiere ser completada ante de devolver un resultado. Si aparece un mensaje de *Error* significa que él comando o instrucción no tuvo efecto. Si aparece un *Warning* significa que **R** efectuó la instrucción anterior, pero tuvo algún obstáculo mismo que es descrito inmediatamente. Con las flechas del arriba y abajo del teclado, aparece la linea de comando inmediata anterior y es una manera de no re-escribir dichas líneas cada vez. El signo de número `#` indica un comentario que no será tomado en cuenta hasta que aparezca un nuevo prompt.

Para poder ver los objetos que se encuentran en una sesión activa de **R**, se puede escribir la función de enlistar `ls`, o si estás en **R-Studio**, verifica directamente la pestaña **Environment** en el panel superior derecho.

```r
ls()
```

El nombre de un objeto se asigna con el operador '<-', '->' o '=', y puede estar hecho de letras, números y

puntuación. Nota: Usar el mismo nombre para dos objetos distintos implica perder la asignación del primer objeto

```
y.y <- 10 * 10
z.12 <- 81 / 9
unam <- "Universidad Nacional Autonoma de Mexico"

xx <- 4
xx
xx <- "xx ya no es el mismo"
xx
```

Para ver el tipo y longitud de un objeto se pueden usar las funciones `mode`, `class`, `length` o `str`. Úselas con los objetos creados:

```
#Solo con y.y, úsela con los otros dos objetos
mode(y.y)
class(y.y)
length(y.y)
str(y.y)
```

El ';' sirve para separar comandos en una misma linea, sin darle enter, y los textos siempre van entre comillas dobles:

```
A <- "mandarina"; compar <- TRUE; mode(A)
mode(compar)
```

Noten los números entre corchetes del lado izquierdo de la consola marcando el número de elementos que siguen en esa línea antes de llegar a una línea abajo. Luego de ejecutar las línea, ¿puede deducir qué hace `seq`?

```
muchos <- seq(0, 50)
muchos
```

R es sensible a mayúsculas, pero no a los espacios:

```
compar
Compar
sum    (3+2)
sum(3+2)
```

En **R** se usan tres tipos de lementos: números (*numeric*), letras (*character*, siempre entre comillas), lógicos (*logical*). Estos elementos son usados para generar objetos. Los objetos pueden clasificarse como:

A) *Vector*: una columna o una fila de elementos, que pueden ser numéricos, de caracteres de texto, de operadores lógicos, etc. Cuando se trata de una variable categórica, el vector puede ser tratado como un factor, y los niveles del factor corresponden a las categorías de dicha variable. Un vector se crea con la funcion `c`, deguido de paréntesis `()` que incluyen todos los elementos del vector separados por coma.

```
vect1 <- c(2,4,6,3,7,8,9,2)
vect1

vect2 <-c("esp", "ing", "por")
vect2

#Puedes preguntar si un vector tiene elementos de un tipo en particular:
is.numeric(vect1)
is.numeric(vect2)
is.character(vect1)
```

```r
is.character(vect2)

#Qué hace esto:
vect1[5]
vect2[2]
```

Una de las grandes fortalezas de **R** es que permite el acceso a los elementos de un objeto a través de una selección de subconjuntos de éstos. El *sub-setting* es una manera eficiente y flexible de acceder selectivamente a los elementos de un objeto, y se hace mediante el uso de corchetes `[]`.

B) *Matrix*: es un arreglo bidimensional de columnas y renglones, sobre el cual se pueden aplicar operaciones algebraicas. Cada elemento de una matriz puede ser accedido con `[,]`, delante de la coma iría el número de la o las filas, luego de la coma, el número de la o las columnas. La combinación específica de una fila y columna lleva al valor de la celda.

```r
#creando una matriz combinando tres vectores
matr1 <- rbind(c(1,2,3),c(4,5,6),c(7,8,9))
matr1
is.factor(matr1)
is.numeric(matr1)

#creando una matriz con una función

matr2 <- matrix(data = seq(1:9), nrow = 3, ncol = 3, byrow = TRUE)

#Note la diferencia entre matr2 y matr3 si cambiamos el argumento byrow a FALSE
matr3 <- matrix(data = seq(1:9), nrow = 3, ncol = 3, byrow = FALSE)

#Selección de segunda y tercera columna de matr2
matr2[,c(2,3)]

#Selección de primera fila de matr2
matr2[1,]

#Selección el valor de la primera fila y segunda columna de matr2
matr2[1,2]
```

C) *Array*: es un arreglo de dimensiones k>2.

```r
n <- 3
k <- 2
j <- 4
samp <- array(dim = c(n,k,j))
samp

is.factor(samp)
is.numeric(samp)
```

Los vectores, matrices y arreglos solo pueden tener elementos del mismo tipo (e.g. numéricos, lógicos, letras)

D) *Dataframe*: es una tabla compuesta de uno o más vectores de la misma longitud, pero con elementos que pueden ser de diferentes tipos. Es el formato ideal para bases de datos, ya que las variables suelen ser de diferente naturaleza (i.e. continuas, nominales, etc.). Se puede acceder a ellas usando la sintaxis de matrices, pero también son el signo `$` para identificar a la columna por su nombre.

```r
iris
data(iris)
```

```r
#haga click sobre <promise> de iris en su ambiente, luego explore visualmente.
# ¿Qué es iris?

dim(iris)    #Pide las dimensiones de la tabla iris
names(iris)  #Pide los nombres de las columnas en iris
iris[,"Species"] # Selecciona la columna por su nombre
iris[,5] # Selecciona la columna por su número de columna
iris$Species     # Selecciona la columna por su asignación en la tabla 'iris'
```

E) *List*: Este objeto puede ser visto como un estante, ya que agrupa ordenadamente objetos de diferente tipo (e.g. vectores, arreglos, tablas, otras listas, etc). Se usa mucho para devolver los resultados de una función que se encuentran en la forma de una colección de objetos:

```r
mi_lista <- vector(mode = "list")

mi_lista[[1]] <- iris
mi_lista[[2]] <- y.y
mi_lista[[3]] <- unam

mi_lista
```

Parte 2: Estadística descriptiva

Las funciones están organizadas en paquetes. El paquete denominado `base` constituye el núcleo de **R** y contiene las funciones básicas del lenguaje. Otro paquete muy importante es `stats` e incluye las funciones estadísticas más importantes y básicas de **R**. Ambos ya vienen preinstalados en **R**. Existen muchos paquetes, a medida que se requiera el uso de alguno específico se irá indicando para que lo descarguen e instalen. Por ahora les adelanto el uso de un set de paquetes agrupados en una familia de paquetes muy usados para ordenar, limpiar, modelar, reproducir, comunicar y graficar datos; este grupo de paquetes se les denomina tidyverse. Para instalarlos pueden escribir en la consola:

```r
#Para instalar ggplot2 (realizar gráficos de alta calidad)
install.packages("ggplot2")

#Para depurar y reordenar bases de datos, instala: tidyr
install.packages("tidyr")

#Para administrar bases de datos: usa dplyr
install.packages("dplyr")

#Para importar datos desde Excel: readxl
install.packages("readxl")

#Para análisis en ecología de comunidades usa vegan
install.packages("vegan")

#para análissi de diversidad usa iNEXT
install.packages("iNEXT")

#Para instalar todos los paquetes del Tidyverse, incluyendo ggplot2, tidyr, dplyr, etc:
install.packages("tidyverse")
```

Alternativamente, puedes usar la ventala *Tools/install packages...*, se desplegará una ventana para que escribas el nombre del paquete a instalar. Los paquetes se instalan una sola vez, siempre que estes en el

mismo computador. Para usarlos debes incluirlos en tu sesión de trabajo cada vez que se inicia la sesión. Esto se logra con la función `library`:

```
library("tidyverse")
```

Antes de usar paquetes, hagamos un análisis exploratorio a los datos *iris*. Antes de copiar el código, busque en la pestaña *Help* qué es *iris*. Efectuamos un gráfico de dispersión entre las variables "largo". Mida el grado de asociación ¿cómo lo haría?

```
plot(iris$Sepal.Length, iris$Petal.Length)

#¿qué hace cor()?
cor(iris$Sepal.Length, iris$Petal.Length)
```

Usemos el paquete `ggplot2` para mejorar el gráfico:

```
pp <- ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length, colour = Species))+
      geom_point()

pp

#Mejoremos con capas, cada capa es agregada con el símbolo +
pp +  theme_bw()+ #fondo blanco
      xlab("Largo del sépalo (cm)")+ #edición del título del eje x
      ylab("largo del pétalo (cm)")+ #edición del título del eje y
      scale_y_continuous(breaks = seq(1,7,1))+ #especificación de unidades en y
      scale_x_continuous(breaks = seq(4,8,1))+ #especificación de unidades en x
      #agregar el valor de correlación al gráfico
      annotate(geom = "text", x= 4.3, y = 6.5, parse=TRUE, label= "italic(r) == 0.87")


#¿cuál gráfico le gustó más?
```

Llegados a este punto, vamos hacer algo de estadística descriptiva. Calculen promedio, varianza y desviación estandar a la variable largo del sepalo (**Sepal.Length**). Usen para ello los códigos suminitrados y responda las preguntas en el cuadro.

```
## PREGUNTAS

# 1. Copia el comando que sigue. ¿Qué se calculó?
xx <- iris$Sepal.Length
sum(xx) / length(xx)

# 2. Busca y aplica una función que ejecute la linea de comando anterior.
# (PISTA: escribe '?mean' en la cónsola y enter para ampliar tu búsqueda)

# 3. Calcula la mediana de usando la función correspondiente.

# 4. Calcula la varianza y desviación estándar de usando el comando 'var'

# 5. ¿Cuál es la diferencia entre estas dos fórmulas? ¿Representan lo mismo?

sum((xx - mean(xx)) ^ 2) / length(xx)
sum((xx - mean(xx)) ^ 2) / (length(xx) - 1)

# 6. Con base en el valor de la varianza y usando operadores aritméticos,
# calcula la desviación estándar a esta variable.
```

```r
# Confirma tu resultado usando la función en R correspondiente.


# 7. Todas estas estimaciones ignoran las posibles diferencias en el largo del sépalo
# entre las especies. ¿Qué le dice este gráfico?
boxplot(Sepal.Length~Species, data = iris)


# 8. Calcule el promedio para cada especie usando la función aggregate.
# (PISTA: escribe '?aggregate' en la cónsola y enter para ver cómo se usa la función;
# de las cuatro formas de uso, recomiendo el modo S3 para fórmulas).
# Nombre al resultado como prom.sep


# 9. Calcule la desviación estándar para cada especie usando la función aggregate.
# Nombre al resultado con el nombre sd.sep

# 10. Combine ambos resultados en una data frame usando el siguiente código e interprete
# la tabla con base en el gráfico previamente generado.

resultado <- data.frame("Especies" = prom.sep$Species,
                        "Promedio" = prom.sep$Sepal.Length,
                        "Desv. Estandar" = sd.sep$Sepal.Length)
```

Parte 3: Algebra de matrices

1. Crea una matriz con 4 filas ('nr') y 3 columnas ('nc'), con los números del 1 al 12 usando la función 'matrix' y llámala mat1.

```r
rm(list = ls())
mat1 <- matrix(1:12, nr = 4, nc = 3)
mat1
```

a) ¿Qué sucede si sólo defines el número de columnas?
b) ¿Qué sucede si defines el número de columnas pero adicionas el argumento 'byrow=T' en la función?

2. Usando subsetting (Parte 1) selecciona la primera columna y la primera fila de mat1 de manera separada.
3. Ahora crea mat2 conteniendo los números del 13 al 24 y con las mismas dimensiones de mat1,
    a) ¿Como harías para sumar las dos matrices?
    b) Adiciona el número 3 a cada elemento de la matriz mat1.
    c) Resta el número 1 a mat1.
    d) ¿Como imaginas que se haría la sustracción (resta) entre mat1 y mat2?
    e) Dado el resultado que obtuviste, explica cuál es la relación matemática entre la mat 1 y la mat2.
4. La matriz traspuesta se calcula mediante la función `t()`. Obtén la matriz transpuesta de mat2, y explica en qué consiste trasponer una matriz.
5. El operador '*' es usado para obtener el producto escalar de una matriz.
    a) Usando este operador estima el doble de mat1. ¿El resultado es una matriz o es un escalar?
    b) ¿Qué sucede si multiplicas mat1 por mat2 uando el operador '*'?
    c) ¿Qué sucede si multiplicas mat1 por mat2 uando el operador '%*%'?
    d) ¿Qué sucede si multiplicas mat1 por la matriz traspuesta de mat2 usando el operador '%*%'?
    e) ¿Por qué tuviste que trasponer la mat2?

f) Calcula a mano el valor que corresponde a la primera fila y primera columna del resultado de multiplicar mat1 por la traspuesta de mat2.

6. Para obtener la división de los elementos de dos matrices (elemento a elemento) se puede multiplicar la primera matriz por una que contenga los inversos de los elementos de la segunda. Escribe el código para "dividir" la mat3 entre 2. Verifica el resultado.

7. Aplica la función 'diag' a mat1 y explica que hace esta función.

8. La matriz inversa se calcula utilizando la función 'solve'. Aplica 'solve' a mat2.

   a) Genera una matriz mat4 de 2 filas y 2 columnas con los elementos del 1 al 4 ordenados por columnas.
   b) Obtén la matriz inversa de mat4. ¿Cuál es el problema de mat2?
   c) Intenta obtener la inversa de mat3. ¿Cuál es el problema de mat3?

9. El determinante de una matriz se calcula con la función 'det'. ¿Porque no podemos calcular el determinante de mat1?

10. Las siguientes instrucciones son para visualizar matrices.

    a) Aplica la función 'plot' a mat1 y explica que hace esta función aplicada a una matriz.
    b) Aplica la función 'image' a mat1 e intenta interpretar el resultado gráfico.
    c) El siguiente código elabora la gráfica utilizando el ggplot2. Primero transforma la matriz en un dataframe, para después apilar todos los elementos en una única columna usando la función 'stack'. Después adiciona una columna al dataframe, indicando el número de celdas que se desea tener en el eje de las x (en este caso usa el número de renglones de mat1).
    d) Finalmente llama a la libreria ggplot2 y hace el gráfico con la función 'qplot'.

```
# Transforma la matriz en un dataframe
data.frame(mat1)

# Apila todos los elementos en una única columna usando la función 'stack'
stack(data.frame(mat1))

# Adiciona una columna al dataframe, indicando el número de celdas en el eje de las x
# (en este caso usa el número de renglones de mat1)

mat.s<-data.frame(stack(data.frame(mat1)),celdas=1:dim(mat1)[1])
mat.s

# Llama al paquete ggplot2 y grafica con la función 'qplot'
library(ggplot2)
qplot(
  data = mat.s,
  x = celdas,
  y = ind,
  fill = values,
  geom = c("raster", "text"),
  label = values,
  xlab = "Rows",
  ylab = "Cols"
)

#El argumento 'fill' indica la intensidad del color de cada celda de la gráfica, que
#correponde a los valores de los elementos de la matriz (la columna 'values'). Copia el
#código e identifica cada comando con una accion descrita para verificar lo que hace.
#Los argumentos 'image', 'raster' y 'label' son para indicar el tipo de gráfico, y los
```

```
#valores de las leyendas en cada celda.
```

Al terminar, recuerde guardar su archivo de comandos, no requiere salvar el espacio de trabajo. Durante la clase les mostraré algunas formas más prolijas para salvar información y análisis.