Modelación Estadística

Actividada 1: Aprendiendo a usar R y R-Studio

Dra. Maite Mascaro y Dr. Edlin Guerra

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 del semestre para aprender a aplicar los conceptos más importantes de *Modelación estadística*, 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, como 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**.

Para instalarlo estos programas, descarguen los archivos instaladores desde:

- R
- RStudio

Lo que harán ahora es:

1. Descargar sus archivos de instalación y ejecutar los instaladores con las opciones que por omisión aparecen.

2. Luego, 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**.
3. 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. **Este archivo se debe entregar como comprobante de que hicieron la primera parte de la Actividad 1**.
4. 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 archivo de comandos; 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?

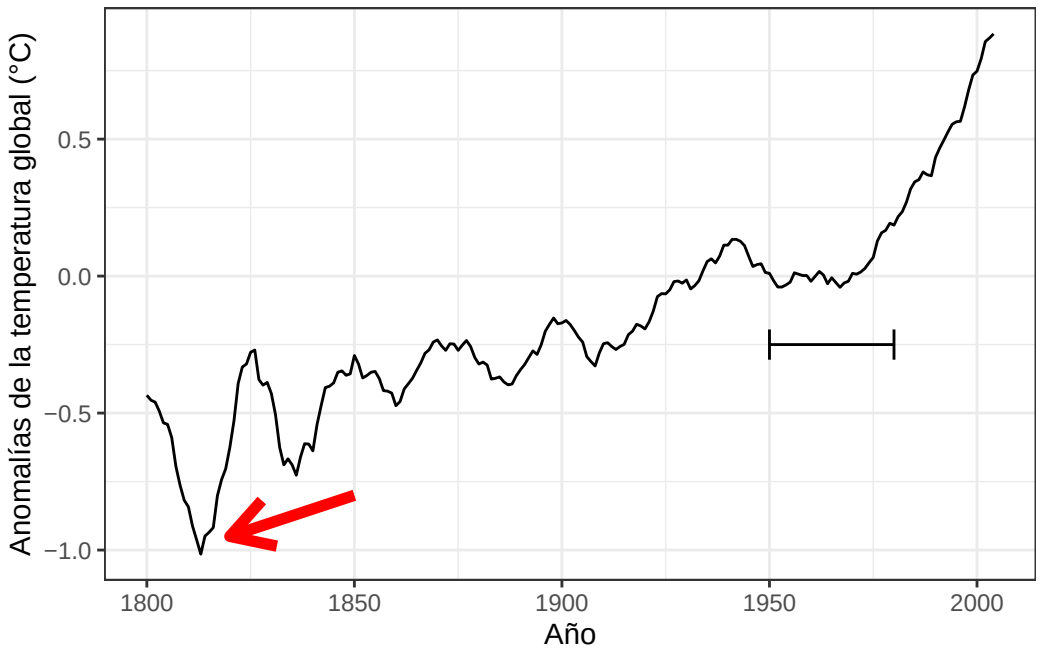
```
#Este código es para generar un gráfico sobre anomalías en la temperatura
#del planeta en las últimas décadas. Estas líneas de código van a instalar
#y carga tres paquetes, luego van a importar los datos y generar un gráfico
#com varios elementos gráficos añadidos.

# install.packages("gcookbook") #instalación del paquete tutorial para gráficos
# 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

#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 = "red",
    linewidth = 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 I: 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 `+` significa 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.

```
ls()
```

El nombre de un objeto se asigna con el operador '<-', y puede estar hecho de letras, números y puntuación. Usar el mismo nombre para dos objetos distintos implica perder la asignación del primer objeto. OTra cosa muy importante que sepan es que pueden crear anotaciones en su código siempre que se anteceda la anotación con el símbolo #, esto es muy importante pues nos permite comentar y describir todo lo que programemos. Ejecute cada una de las siguientes líneas una a una en la consola ¿qué obtuvo?:

```
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
Esto es un comentario
# Esto es un comentario
```

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)
```

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)
```

Objetos en R

En **R** se usan tres tipos de elementos: números (*numeric*), letras (*character*, siempre entre comillas), lógicos (*logical*). Estos elementos son usados para lamacenar datos en forma de 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 función `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)
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.

```
#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 tres o más dimensiones ($k > 2$).

```
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). Cuando tenemos datos de un estudio que tienen distinta naturaleza, lo más recomendable es usar *dataframes*.

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. Una versión más moderna de las *dataframes* son las *tibble*, que en la práctica son iguales pero computacionalmente más eficientes.

```
# Obtenga los datos Iris en la consola
iris

#Extraiga los datos iris a su Environment
data(iris)

#haga click sobre <promise> de iris en su ambiente, luego explore visualmente.
# ¿Qué es iris? Busque en la ayuda el acrhivo de ayuda

class(iris) #Identifica el tipo de ibjeto que es irir
dim(iris)    #Pide las dimensiones de la tabla iris
names(iris)  #Pide los nombres de las columnas en iris
```

```r
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
```

Funciones

Cualquier operación en **R** que implique operación con datos, desde muy simples hasta muy complejos, se ejecuta con operaciones que están definidas en funciones identificadas con palabras. Por ejemplo `sum` sirve para sumar valores en un vector de datos. Las funciones pueden ser muy simples o muy complejas, algunas se ejecutan en fracciones de segundos, otras demoran horas incluso días. Básicamente, se requiere escribir el nombre de la función y, entre paréntesis, definir todos los argumentos o insumos que necesita la función para trabajar. Existen literalmente cientos de miles de funciones en **R**, y la cifra crece cada día más pues cualquiera puede crear su propia función. Acá un ejemplo de una función creada por nosotros mismos, que calcula el promedio de un vector de datos.

```r
# Generar 10 datos aleatorios
# con la función runif, con valor mínimo 5 y máximo 15
nn <- runif(n = 10, min = 5, max = 15)

# runif permite generar números aleatorios, noten los argumentos
# Pueden cambiar los valores en los argumentos para probar

mipromedio <- # nombre que asignamos a la función
  function(datos){ #función que crea funciones, y el argumento datos
    sumatoria = sum(datos) #Desarrollo de la función, primero suma todos los valores identifi
    N = length(datos) #luego identifica cuántos datos hay en datos
    prom = sumatoria / N #aplica la ecuación de promedio
```

```
    return(prom) #regresa el resultado guardado como prom
  }

mipromedio(datos = nn)
```

Paquetes

Debido a la complejidad y necesidad evidente de interacción de muchas funciones, estas suelen estar organizadas en paquetes o librerías. 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:

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

Alternativamente, puedes usar el acceso *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 estés 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`:

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” (*length* en inglés). Mida el grado de asociación ¿cómo lo haría?

```r
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:

```r
# Carguemos a ggplot2 en la sesión
library(ggplot2)

# construcción del gráfico base
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?
```

Parte 2: Exploración de datos con R

El objetivo de esta etapa de la actividad es que pongan en práctica los conceptos básicos aprendidos en el uso de R: manipular objetos, aplicar una función básica y generar un gráfico. Las **Instrucciones** son:

1. **Cargar los datos**: Utiliza el conjunto de datos 'iris.
2. **Manipular un objeto**: Confirma qye iris es un **data.frame**. Luego extrae las columnas `Sepal.Length` (largo del sépalo) y `Sepal.Width` (ancho del sépalo) de la especie *Iris virginica* en un nuevo dataframe. Asigna un nombre a este nuevo dataframe según lo que consideres apropiado. Repite la operación para la especie *Iris versicolor* e *Iris setosa*.

3. **Aplicar una función**: Calcula el promedio al largo del sépalo (`Sepal.Length`) y luego el promedio al ancho del sépalo (`Sepal.Width`) del dataframe filtrado y guárdalo en un objeto. **Elige un nombre para este objeto que sea representativo del resultado**. Repite la operación para las otras dos especies.
4. **Generar un gráfico**

    - Genera un gráfico de dispersión con los datos de *Iris virgínica*:
        - El eje X debe mostrar el largo del sépalo (`Sepal.Length`).
        - El eje Y debe mostrar el ancho del sépalo (`Sepal.Width`).
        - Agrega un título al gráfico que indique claramente qué se está graficando.
        - Repite la producción del gráfico para las otras dos especies.
        - Exportas las figuras como gráficos .JPG e insértalos en el Google Doc asignado en la actividad.

5. **Guardar resultados**: Todos los comandos usados deben ser reproducibles por el maestro, por ello, pedimos que suban el archivo .R donde ejecutó los comandos de esta parte de la actividad como entregable además del Google Doc.