1. Instalación

Para los usuario de Linux recomiendo este link para instalar R compilándolo. Ésta es la mejor opción pues, de esta manera, se aprovecharán todas las características de su máquina. Pueden clonar el repositorio y en la terminal correr

```
./i_R.sh
```

Para descargar e instalar R en su versión precompilada, seguir las instrucciones de este link para el sistema operativo que estén utilizando.

2. Editores

Hay muchísimos, yo les recomiendo dos.

2.1. RStudio

Puedes descargar RStudio siguiendo las instrucciones para cada sistema operativo. RStudio es un IDE (integrated development environment) para R que incluye consola, editor de texto, memoria de gráficos, vista de objetos en el ambiente y otras herramientas útiles para desarrollar (RStudio Team 2016). En su versión más reciente, también autocompleta código y depura (*debugging*) "al vuelo", es decir, al mismo tiempo que se escribe, señala potenciales errores de código.

Hay que tener cuidad con el uso de la memoria RAM de este editor pues utiliza muchos recursos de la computadora y -cuando están usando una gran cantidad de datos o procesos muy pesados- RStudio suele detenerse fácilmente. Buenas prácticas en general: guardar seguido, seguir un flujo de trabajo (*workflow*) aunado a controlador de versiones (o algún tipo de respaldo) y, sobretodo, crear las funciones, lógica, algoritmos, con una muestra de los datos.

2.2. ESS

Emacs speaks statistics es el add-on favorito para los usuarios de `emacs` y `R` (Rossini y col. 2004). Soporta la edición de scripts para R, S-plus, SAS, Stata, OPenBUGS/JAGS. Para los que además ya están acostumbrados al enorme poder de Emacs, ésta será la mejor opción.

El editor interactivo es muy bueno y casi no tiene overhead de memoria.

3. El espacio de trabajo (Workspace)

El *espacio de trabajo* es el ambiente actual de trabajo en `R`. Incluye todos los objetos definidos por el usuario (vectores, matrices, funciones, dataframes, listas).

Una sesión de R inicia cuando abres la consola. Al terminar el trabajo se puede guardar la imagen del espacio de trabajo tal cual está, de manera que sea posible continuar *desde donde te quedaste* (Kabacoff 2015, p. 11).

3.1. Directorio de trabajo

El directorio de trabajo (*working directory*) es el directorio en tu computadora en el que estás trabajando en ese momento. Cuando se le pide a R que abra un archivo o guarde ciertos datos, R lo hará a partir del directorio de trabajo que le hayas fijado.

Para saber en qué directorio te encuentras, se usa el comando `getwd()`.

> Usa la mnemotécnica del inglés: *get working directory* ≡ *getwd*. Notarás como muchas funciones tienen un nombre que acorta lo que hacen.

```
getwd()
```

```
## [1] "/home/animalito/study/aprendeR/01_programacion_basica"
```

Para especificar el directorio de trabajo, se utiliza el comando `setwd()` (*set working directory*) en la consola. Y volvemos a

```
setwd("/home/animalito/study/")
getwd()
```

Ejercicio

1. Abre tu consola de `R` y escribe *setwd("/*.
2. Utiliza la tecla `tab` ⇥ para autocompletar las posibles rutas desde donde quiera que estes.
3. Escoge alguna (nuevamente usando la tecla tab para moverte entre las opciones). Si esto no funciona, teclea textualmente alguna de las rutas que ves.
4. Cierra la doble comilla y el paréntesis.
5. Teclea enter.
6. Debes encontrarte en la ruta elegida cuando tecleas `getwd()`.

Con lo que acabamos de hacer, R buscará archivos o guardará archivos en el folder `/home/animalito/study/`. En R también es posible navegar a partir de el directorio de trabajo. Como siempre,

- "../un_archivo.R" le indica a R que busque un folder arriba del actual directorio de trabajo por el archivo *un_archivo.R*.
- "datos/otro_archivo.R" hace que se busque en el directorio de trabajo, dentro del folder *datos* por el archivo *otro_archivo.R*

Rutas relativas vs. Rutas absolutas

El resultado que se muestra aquí al usar el comando `getwd()` depende de la computadora en la que se esta trabajando debido a que es una *ruta absoluta*. Nota como es diferente la ruta que obtienes al correr el comando en tu consola de `R`. Eso es porque se trata de una ruta absoluta, es decir, es tal que da la ruta (*path*) completo al directorio en cuestión. Puedes accesar todos los directorios o archivos usando su ruta absoluta.

En investigación reproducible (*reproducible research*), en investigación colaborativa o incluso cuando trabajas en varias computadoras es una buena idea usar rutas relativas en lugar de absolutas. Esto hace que el código sea menos dependiente de una estructura de archivos o computadora en particular (Gandrud 2013, p. 67).

En general, es *buena práctica* configurar el código de un proyecto con rutas relativas. En `R` en particular, cuando guardas un `Rmarkdown` y lo corres desde la línea de comandos (o lo *tejes* desde `RStudio`), la ruta que está fija -como si hubieras usado el comando `setwd()` es en donde *vive* ese archivo, es decir, el directorio en donde está guardado el mismo.

Desde cualquier `script` puedes llamar a otros usando este tipo de ruta como en el ejemplo anterior.

3.2. Ejemplos básicos

La consola permite hacer operaciones sobre números o caracteres (cuando tiene sentido).

```
# Potencias, sumas, multiplicaciones
2^3 + 67 * 4 - (45 + 5)
```

```
## [1] 226
```

```
# Comparaciones
56 > 78
```

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

```
34 <= 34
```

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

```
234 < 345
```

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

```
"hola" == "hola"
```

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

```
"buu" != "yay"
```

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

```
# módulo
10 %% 4
```

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

Estas operaciones también pueden ser realizadas entre vectores[1].

```
# Creamos un vector con entradas del -1 al 12 y lo asignamos a la variable x
x <- -1:12
# Lo vemos
x
```

```
##  [1] -1  0  1  2  3  4  5  6  7  8  9 10 11 12
```

```
# Le sumamos 1 a todas las entradas
x + 1
```

```
##  [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13
```

```
# Multiplicamos por 2 cada entrada y le sumamos 3
2 * x + 3
```

```
##  [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27
```

```
# Sacamos el módulo de cada entrada
x %% 5
```

```
##  [1] 4 0 1 2 3 4 0 1 2 3 4 0 1 2
```

3.3. Comandos útiles

Para enlistar los objetos que están en el espacio de trabajo

[1] Revisaremos más adelante con detalle la definición de vectores en la sección **??**.

```r
ls()
```

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

Para eliminar todos los objetos en un workspace

```r
rm(list = ls()) # se puede borrar solo uno, por ejemplo, nombrándolo
ls()
```

```
## character(0)
```

También se puede utilizar/guardar la historia de comandos utilizados

```r
history()
history(max.show = 5)
history(max.show = Inf) # Muestra toda la historia

# Se puede salvar la historia de comandos a un archivo
savehistory(file = "mihistoria") # Por default, R ya hace esto
# en un archivo ".Rhistory"

# Cargar al espacio de trabajo actual (current workspace) una
# historia de comandos en particular
loadhistory(file = "mihistoria")
```

Es posible también guardar el workspace -en forma completa- en un archivo con el comando `save.image()` a un archivo con extensión *.RData*. Puedes guardar una lista de objetos específica a un archivo *.RData*. Por ejemplo:

```r
x <- 1:12
y <- 3:45
save(x, y, file = "ejemplo.RData") #la extensión puede ser arbitraria.
```

Después puedo cargar ese archivo. Prueba hacer:

```r
rm(list = ls()) # limpiamos workspace
load(file = "ejemplo.RData") #la extensión puede ser arbitraria.
ls()
```

Nota como los objetos preservan el nombre con el que fueron guardados.

4. Paquetes (*libraries*)

R puede hacer muchos análisis estadísticos y de datos. Las diferentes capacidades están organizadas en paquetes o librerías. Con la instalación estándar resumida en la sección 1, se instalan también los paquetes más comunes (también llamado el *base* o R-básico). Para obtener una lista de todos los paquetes instalados se puede utilizar el comando `library()` en la consola o en un script.

Existen una gran cantidad de paquetes disponibles además de los incluidos por omisión (*default*).

4.1. CRAN

Comprehensive R Archive Network (CRAN 2016) es una colección de sitios que contienen exactamente el mismo material, es decir, son espejos (*mirrors*) de las distribuciones de R, las extensiones, la documentación y los binarios. El master de CRAN está en Wirtschaftsuniversität Wien en Austria. Éste se "espeja" (*mirrors*) en forma diaria a muchos sitios alrededor del mundo. En la lista de espejos se puede ver que para México

están disponibles el espejo del ITAM, del Colegio de Postgraduados (Texcoco) y Jellyfish Foundation (CRAN 2016).

Los espejos son importantes pues, cada vez que busquen instalar paquetes, se les preguntará qué espejo quieren utilizar para la sesión en cuestión. Del espejo que selecciones, será del cuál R *bajará* el binario y la documentación.

Del CRAN es que se obtiene la última versión oficial de R. Diario se actualizan los espejos. Para más detalles consultar el FAQ.

Para contribuir un paquete en CRAN se deben seguir las instrucciones aquí.

4.2. Github

Git es un controlador de versiones muy popular para desarrollar software. Cuando se combina con GitHub se puede compartir el código con el resto de la comunidad. Éste controlador de versiones es el más popular entre los que contribuyen a R. Muchos problemas a los que uno se enfrenta alguien ya los desarrolló y no necesariamente publicó el paquete en CRAN. Para instalar algún paquete desde GitHub, se pueden seguir las instrucciones siguientes

```
install.packages("devtools")
devtools::install_github("username/packagename")
```

Donde `username` es el usuario de Github y `packagename` es el nombre del repositorio que contiene el paquete. Cuidado, no todo repositorio en GitHub es un paquete. Para más información ver el capítulo Git and GitHub en Wickham (2015).

4.3. Otras fuentes

Otros lugares en donde es común que se publiquen paquetes es en Bioconductor un projecto de software para la comprensión de datos del genoma humano.

5. Paquetes recomendados

Hay muchísimas librerías y lo recomendable es, dado un problema y un modelo para resolverlo, revisar si alguien ya implementó el método en algunas de las fuentes de paquetes mencionadas antes.

Para mantener orden en los paquetes descargados puede ser útil utilizar el Rinker y Kurkiewicz (2015) pues provee de herramientas para instalar paquetes en una forma un poco más sencilla que usando la función `install.packages`.

En particular, la función `p_load` permite instalar, cargar y actualizar uno o varios paquetes.

Si queremos instalar varios paquetes usando las herramientas del R básico (*base*) (R Core Team 2016) haríamos algo como (ejemplo tomado de Rinker y Kurkiewicz 2015, en la viñeta de intrducción al paquete):

```
packs <- c("XML", "devtools", "RCurl", "fakePackage", "SPSSemulate")
success <- suppressWarnings(sapply(packs, require, character.only = TRUE))
install.packages(names(success)[!success])
sapply(names(success)[!success], require, character.only = TRUE)
```

Con `pacman::p_load` la tarea se reduce a:

```
pacman::p_load(XML, devtools, RCurl, fakePackage, SPSSemulate)
```

> Nota como se puede llamar a una función por su nombre `p_load` una vez que ya cargamos el paquete en el cuál esa función está guardada con el comando `library(pacman)` o podemos llamarla directamente utilizando la convención `paquete::funcion`, en este caso, `pacman::p_load`.

Para instalar `pacman` escribe:

```
install.packages("pacman")
```

Algunos paquetes se encuentran en desarrollo. En particular, si se encuentran en `github` pueden descargarse usando la función `pacman::p_install_gh('usuario/repositorio')`.

A continuación, hay una lista de paquetes que se recomienda descargar o revisar para tener a la mano herramientas diversas útiles para el trabajo del científico de datos. La lista no es comprensiva pues hay un gran número de paquetes útiles.

```
# Para cargar datos al ambiente de trabajo (data load)
pacman::p_load(RODBC, RMySQL, RPostgreSQL, RSQLite, foreign, Rpostgres, haven
               , readr)
pacman::p_install_gh("hadley/readxl")
pacman::p_install_gh("rstats-db/RPostgres")

# Para manipular datos (data manipulation)
pacman::p_load(plyr, dplyr, data.table, tidyr, stringr, lubridate, gsubfn)

# Para visualizar datos (data visualization)
pacman::p_load(ggplot2, graphics, ggvis)
pacman::p_install_gh("RcppCore/Rcpp")
pacman::p_install_gh("rstats-db/DBI")
pacman::p_install_gh('ramnathv/htmlwidgets')
pacman::p_install_gh('rstudio/leaflet')
pacman::p_install_gh('bwlewis/rthreejs')
pacman::p_install_gh('htmlwidgets/sparkline')
pacman::p_load(dygraphs, DT, DiagrammeR, networkD3, googleVis)

# Para modelar (data modelling)
pacman::p_load(car, mgcv, lme4, nlme, randomForest, multcomp, vcd, glmnet, survival, caret)

# Para generar reportes (reports)
pacman::p_load(shiny, xtable, knitr, rmarkdown)

# Para trabajar con datos espaciales (spatial data)
pacman::p_load(sp, maptools, maps, ggmap, rgdal)

# Para trabajo con series de tiempo (time series)
pacman::p_load(zoo, quantmod)

# Para escribir código de alto rendimiento en R (High performance R code)
pacman::p_load(Rcpp, parallel)

# Trabajar con la web
pacman::p_load(XML, jsonlite, httr)

# Para escribir paquetes en R
pacman::p_load(devtools, testthat, roxygen2)
```

6. Scripting

R es un intérprete. Utiliza un ambiente basado en línea de comandos. Por ende, es necesario escribir la secuencia de comandos que se desea realizar a diferencia de otras herramientas en donde es posible utlizar el mouse o menús.

Aunque los comandos pueden ser ejecutados directamente en consola una única vez, también es posible guardarlos en archivos conocidos como *scripts*. Típicamente, utilizamos la extensión **.R** o **.r**. En RStudio (RStudio Team 2016), `CTRL + SHIFT + N` abre inmediatamente un nuevo editor en el panel superior izquierdo.

En RStudio, por ejemplo, se puede *ir editando* el script y corriendo los comandos línea por línea con `CTRL + ENTER`. Esto también aplica para *correr* una selección del texto editable[2].

Es posible también correr todo el script

```
source("foo.R")
```

O con el atajo `CTRL + SHIFT + S` en RStudio.

Para enlistar algunos shortcuts comunes en RStudio presiona `ALT + SHIFT + K`.

De la misma manera, si utilizas `Emacs + ESS` (Rossini y col. 2004), existen múltiples atajos de teclado para realizar todo mucho más eficientemente. Estudiarlos no es tiempo perdido.

7. Ayuda y documentación

`R` tiene mucha documentación. Dado que es imposible recordar todas las funciones o cómo utilizar todo lo que ya está hecho, es necesario aprender a leerla. Desde la consola se puede accesar a la misma.

Para ayuda general,

```
help.start()
```

Para la **ayuda de una función en especifico**, por ejemplo, si se quiere graficar algo y sabemos que existe la funcion `plot` podemos consultar fácilmente la ayuda.

```
help(plot)
# o tecleando directamente
?plot
```

El segundo ejemplo se puede extender para buscar esa función en todos los paquetes que tengo instalados en mi ambiente al escribir `??plot`.

A veces, es útil ver el **cuerpo de una función**. Esta tarea no necesariamente es trivial. Para funciones generadas por el usuario, usa

```
xx <- function(x) x^2
body(xx)
```

```
## x^2
```

```
# o simplemente imprimir el objeto en donde guardamos la función
xx
```

```
## function(x) x^2
```

También funciona para algunas funciones de paquete, por ejemplo `rename`:

[2] RStudio tiene muchos atajos de teclado que facilitan el trabajo.

```
library(plyr)
body(rename)
```

```
## {
##     names(x) <- revalue(names(x), replace, warn_missing = warn_missing)
##     duplicated_names <- names(x)[duplicated(names(x))]
##     if (warn_duplicated && (length(duplicated_names) > 0L)) {
##         duplicated_names_message <- paste0("`", duplicated_names,
##             "`", collapse = ", ")
##         warning("The plyr::rename operation has created duplicates for the ",
##             "following name(s): (", duplicated_names_message,
##             ")", call. = FALSE)
##     }
##     x
## }
```

Para plot, en cambio, al usar la función `body` se ve:

```
body(plot)
```

```
## UseMethod("plot")
```

Esto es porque `plot` es una función genérica (S3) que tiene métodos para distintas clases de objetos. En esos casos, primero debemos usar la función `methods` para enlistar los métodos que tiene esa función.

```
methods(plot)
```

```
##  [1] plot.acf*           plot.data.frame*    plot.decomposed.ts*
##  [4] plot.default        plot.dendrogram*    plot.density*
##  [7] plot.ecdf           plot.factor*        plot.formula*
## [10] plot.function       plot.hclust*        plot.histogram*
## [13] plot.HoltWinters*   plot.isoreg*        plot.lm*
## [16] plot.medpolish*     plot.mlm*           plot.ppr*
## [19] plot.prcomp*        plot.princomp*      plot.profile.nls*
## [22] plot.raster*        plot.spec*          plot.stepfun
## [25] plot.stl*           plot.table*         plot.ts
## [28] plot.tskernel*      plot.TukeyHSD*
## see '?methods' for accessing help and source code
```

Si tiene asteriscos, significa que la función para ese método en particular no viene directamente del espacio de nombres del paquete pero, de cualquier forma, lo podemos pedir usando la `función getAnywhere` para cualquiera de los métodos que se desplegaron:

```
getAnywhere(plot.density)
```

```
## A single object matching 'plot.density' was found
## It was found in the following places
##   registered S3 method for plot from namespace stats
##   namespace:stats
## with value
##
## function (x, main = NULL, xlab = NULL, ylab = "Density", type = "l",
##     zero.line = TRUE, ...)
## {
##     if (is.null(xlab))
##         xlab <- paste("N =", x$n, "  Bandwidth =", formatC(x$bw))
##     if (is.null(main))
```

```
##         main <- deparse(x$call)
##     plot.default(x, main = main, xlab = xlab, ylab = ylab, type = type,
##         ...)
##     if (zero.line)
##         abline(h = 0, lwd = 0.1, col = "gray")
##     invisible(NULL)
## }
## <bytecode: 0x214dad8>
## <environment: namespace:stats>
```

Nota como el método `plot.density` viene del paquete `stats` [3].

La documentación normalmente se acompaña de **ejemplos**. Para *correr* los ejemplos sin necesidad de copiar y pegar, prueba

```
example(plot)
```

Para búsquedas más comprensivas, se puede buscar de otras maneras:

```
apropos("foo") # Enlista todas las funciones que contengan la cadena "foo"
RSiteSearch("foo") # Busca por la cadena "foo" en todos los manuales de ayuda
# y listas de distribución.
```

8. Optimizando

Es común que muy pronto nos encontremos con limitaciones al poder de cómputo y rapidez con el que R procesa los datos. Hay operaciones intensivas como, por ejemplo, la inversión de matrices (qr) o el análisis por componentes principales (svd). Incluso una selección de variables (*back/forward selection*) usando una simple regresión lineal sobre múltiples regresores puede llevar un tiempo de cómputo de horas/días o no terminar.

Una de las manera más rápidas de mejorar el performance de `R` es instalando las librerías de álgebra lineal que puede utilizar el software para hacer las operaciones más rápido.

Para mucho (demasiado) detalle al respecto, referirse a la comparación de performance en Eddelbuettel (2010) o al paquete del mismo autor Eddelbuettel (2016).

Para la parte práctica de todo esto, referirse a este blog para instalar las librerías apropiadas para BLAS y Lapack (Klamer 2014). Para una comparación bastante práctica de las diferentes versiones de esas librerías, ver aquí (Nguyen 2014).

[3] Hay otro tipo de funciones en las accesar al código fuente no se pueda con los métodos descritos. Para ello, es útil revisar la sección .ºld-school object-oriented programming in R"(Adler 2010, p.131-133) o las secciones dedicadas a los objetos S3 y S4 en Wickham (2014).