```
## Warning in as.POSIXlt.POSIXct(Sys.time()): unknown timezone 'default/America/Santiago'
```

1 Introducción

En las últimas décadas la economía ha ido adoptando a los computadores para realizar econometría aplicada. Sin embargo, la manera en que se enseña econometría sigue siendo eminentemente teórica. La gran mayoría de los textos de econometría deriva los modelos como ejercicios de álgebra lineal, y más tarde (o nunca) intentan conectar esto con el trabajo aplicado, usando ejemplos específicos y pequeños extractos de código (si es que hay algo de código). Este libro intenta dar la vuelta a este paradigma, apoyándose en R desde un comienzo y en forma continua para el desarrollo de conceptos.

1.1 ¿Qué es R?

R es un ambiente de software y un lenguaje de programación interpretado para manipular datos, hacer análisis estadístico y visualizar datos. Es una implementación de S, un lenguaje de programación matemático orientado a objetos más antiguo. Es software libre y de código abierto, constantemente desarrollado y ampliado por colaboradores de múltiples disciplinas.

R es mucho más flexible que la mayoría de los paquetes estadísticos normalmente usados por economistas. Es un lenguaje de programación completamente desarrollado, no sólo un programa con tests y métodos pre-programados. Es una alternativa más flexible, poderosa y barata con respecto a otros paquetes estadísticos comunes, como Stata o SPSS.

1.2 ¿Por qué R?

Creo que R será el lenguaje *de facto* usado por economistas en el futuro próximo. Históricamente, economistas y econometristas han favorecido otros programas para realizar análisis estadístico: Gauss, Matlab, Eviews, SAS, SPSS y Stata, por nombrar algunos. Todos tienen sus ventajas, pero comparten una característica común: son programas comerciales, lo que en muchos casos implican preios prohibitivamente altos, ciclos de desarrollo relativamente largos y en general una falta de incorporación de lo que se está haciendo "en la frontera".

En respuesta a esto, muchas ciencias (de las que economía va a la saga) se han movido al uso de software libre para realizar análisis estadístico. Lenguajes como Python, Julia y R han avanzado enormemente en años recientes y hoy constituyen una fuerte alternativa frente a los paquetes comerciales, subsanando todos los incovenientes mencionados.

Si bien Stata sigue dominando el área, la migración hacia R ya está en progreso. Muchos cursos de econometría

1.3 ¿Puedo usar este documento sin R?

Si. El objetivo detrás de este apunte es aprender econometría primero y R después. El uso de R siempre es para *apoyar* una explicación, usándolo como herramienta de aplicación de las ideas expuestas. Siempre muestro el código de lo que está sucediendo "detrás del telón", pero lo hago para aumentar la transparencia de los ejemplos y permitir que sean replicados por el lector que así lo quiera. A quien no le interese, puede enfocarse directamente en los datos y resultados calculados (en los bloques de código) y en los gráficos creados (inmediatamente después de los bloques). He puesto especial cuidado en que el texto sea legible aún para quien quiera obviar el código. Por otro lado, para quien le interese ahondar *aún más* en el código, he creado unos bloques que explican el código que va siendo usado, en caso que los comentarios no sean suficientes.

De esta forma, me gusta pensar que este apunte tiene tres formas de ser leído:

1. Como un apunte de econometría en el que no se aprende nada de R, concentrándose en los textos y figuras pero saltando todo el código y sus explicaciones.
2. Como un apunte de econometría donde el código y sus comentarios ayudan a esclarecer los mecanismos detrás de los conceptos explicados.

3. Como un apunte de econometría y suscinta introducción a R, estudiando el código con sus explicaciones y replicando estos *scripts* en tu propio computador.

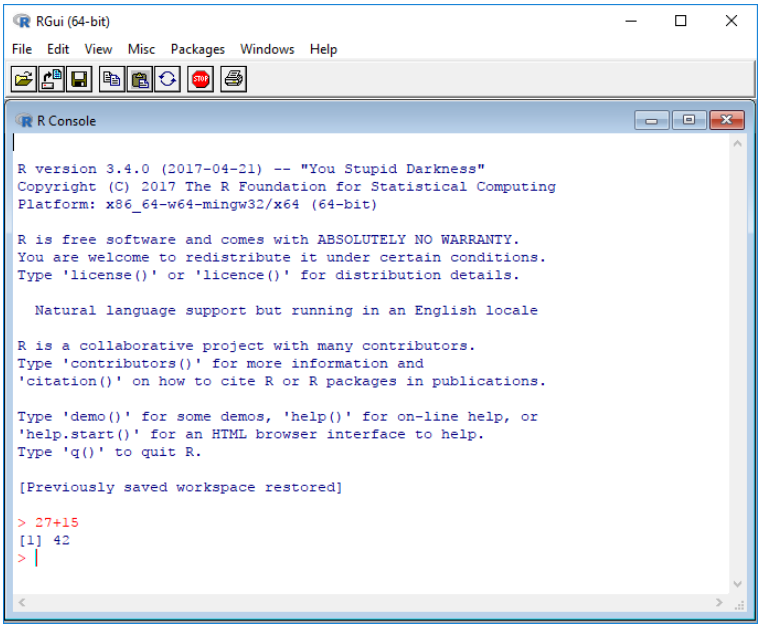
   [Ejemplo]

El código R

`library()` es una función que nos permite cargar paquetes de R, los que contienen más funciones y datos. Usaremos `library()` al comienzo de nuestro código, para mantener el orden. **tidyverse** contiene una gran cantidad de paquetes (ver Sección 1.7) y lo usaremos en todos los capítulos. **haven** nos permite usar la función `read_dta()` para leer una base de datos de Stata (con extensión `*.dta`).

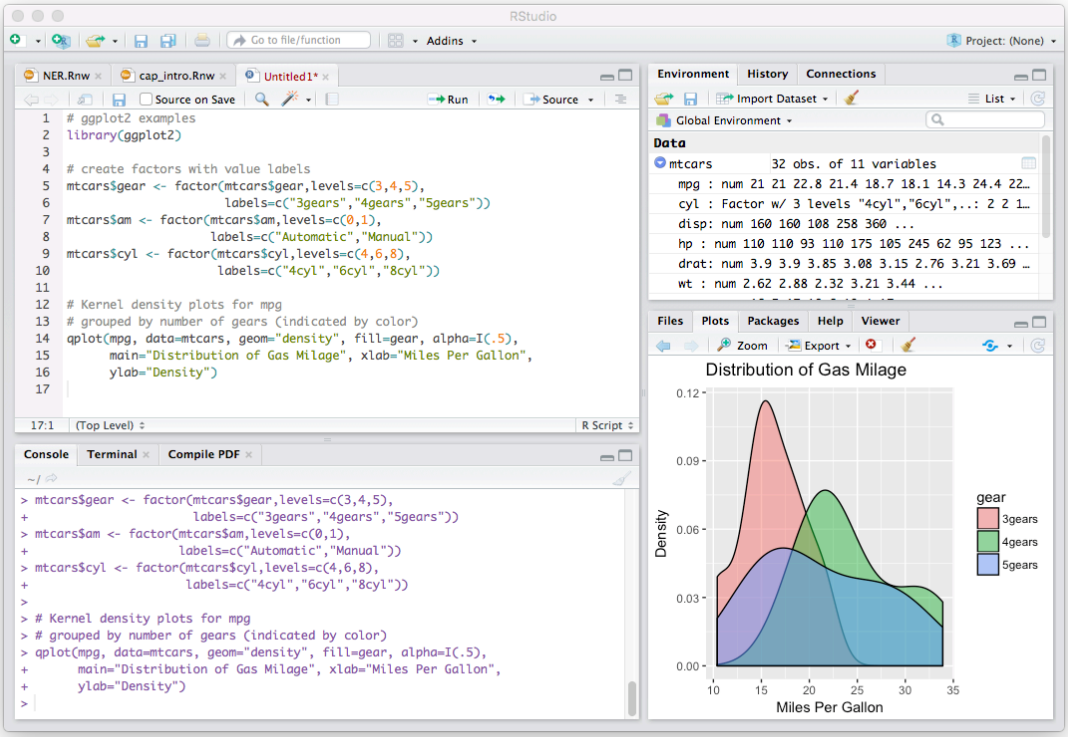
1.4 Obtener R

La instalación base de R puede obtenerse directamente de `https://cloud.r-project.org/` (en Windows tienes que elegir la opción *base*). Una vez instalado puedes usar RGui para interactuar con R. Sin embargo, esta opción no es demasiado sexy:



Mi recomendación es usar RStudio, un IDE (*integrated development environment*) que nos ayudará a trabajar mucho más cómodamente con R. RStudio incluye una consola, un editor de código con resaltado de sintaxis, explorador de objetos y una larga lista de cosas que puede que no entiendas aún, pero que seguro harán tu vida más fácil. Con toda seguridad se *ve* mejor que RGui. Puedes descargar RStudio de `https://www.rstudio.com/`, y también es gratis (asegúrate de haber instalado R antes de instalar RStudio).

Si usas Mac, algunas funciones de R necesitarán de un Servidor X11, el cual ya no es incluido por Apple. Este paso es opcional, pero si quieres prevenirte de no recibir errores inesperados en el uso de alguna función más adelante, puedes descargar XQuartz de `https://www.xquartz.org/` e instalarlo ahora en tu sistema.



1.5 Trabajando en R

R es un ambiente de software y un lenguaje de programación interpretado para hacer análisis estadístico. En este libro interactuaremos con R principalmente a través de RStudio, que es un ambiente de desarrollo integrado (IDE, por sus siglas en inglés) para programar en R. La interfaz de RStudio se muestra en la Figura 1.1.

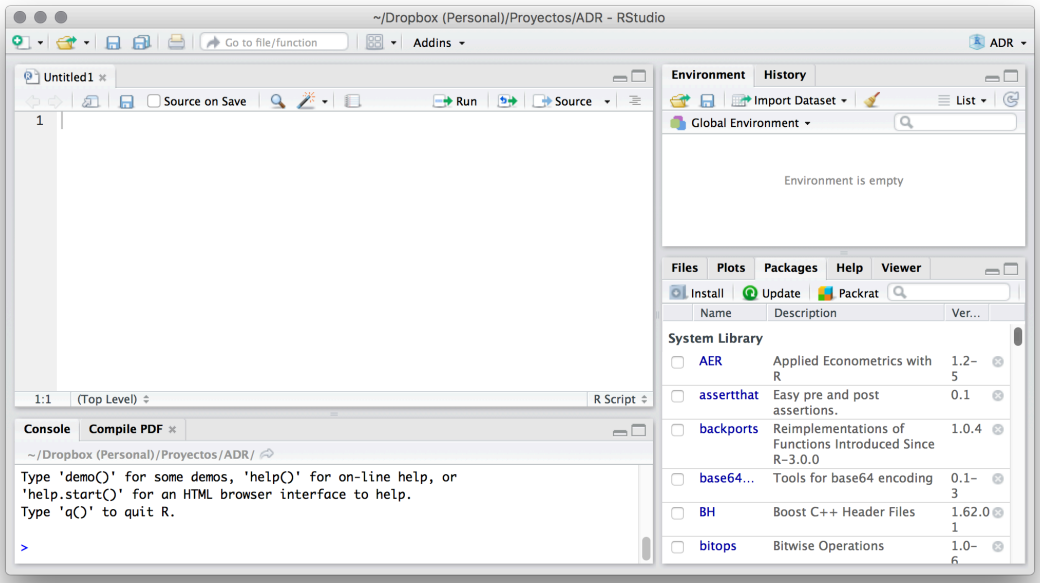


Figura 1.1: RStudio 1.0.136 en macOS 10.12.2

Daremos instrucciones a R a través de dos métodos. El más inmediato es a través la **Consola**, donde ejecutaremos comandos de a uno, escribiéndolos en el *command prompt* o símbolo de sistema (>) del panel *Console*, en la esquina inferior izquierda. El segundo método será a través de un conjunto de comandos escritos en un ***script***, que es un archivo de extensión `.R` con varias instrucciones para que R ejecute de manera no interactiva. El documento "Untitled1" que deberías ver en tu ventana de RStudio es para escribir y guardar scripts (si no lo vez, Archivo, Nuevo, R Script). Por el momento puedes usar la consola para ir siguiendo lo que vayamos haciendo.

1.6 R en 5 minutos

Revisaremos ahora brevemente los conceptos más fundamentales de R: crear objetos mediante asignaciones y manipularlos mediante funciones. No te preocupes si no sabes para qué sirve cada una de las funciones; lo importante aquí es puedas entender los conceptos y que puedas replicar los ejemplos.

R es un lenguaje de programación orientado a objetos, lo que significa que mucho del trabajo consiste en crearlos y manipularlos. Un **objeto** es un elemento con nombre al que le hemos asignado un valor en el sentido amplio de la palabra: los valores pueden ser datos, funciones, gráficos o resultados, entre otros. Entonces una de las operaciones más comunes es la asignación de un valor a un objeto, la que se realiza con el **operador de asignación** `<-`, y tiene la siguiente sintaxis general:

```
objeto <- valor
```

Por ejemplo, podemos crear el objeto `x` asignándole el valor 10. Para hacerlo escribe la siguiente línea en tu Consola y presiona Enter ⟷ para ejecutarla:

```
x <- 10
```

En RStudio, un buen atajo para memorizar rápido es usar Alt - para insertar el operación de asignación. Si has tenido éxito, deberías ver el objeto `x` asignado con valor 10 en tu panel de *Environment*, o Ambiente (arriba a la derecha). En adelante no diré explícitamente que hay que ejecutar cada línea con Enter ⟷, ya que se sobreentiende (espero).

Una asignación casi siempre es silenciosa, es decir, no se mostrará su resultado. Para **examinar un objeto** simplemente escribimos su nombre:

```
x

#> [1] 10
```

Una manera de mostrar inmediatamente el resultado de una asignación es encerrando la expresión completa entre paréntesis. Por ejemplo, definimos un segundo objeto escribiendo

```
(y <- 2*3)

#> [1] 6
```

El símbolo `[1]` que antecede a los resultados anteriores indica que se trata del primer resultado. Por ejemplo, si ejecutas el código `1:42` para obtener una secuencia de 40 enteros

deberías algo como lo siguiente (dependiendo del ancho de tu Consola):

```
1:42

#>  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
#> [24] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
```

Para manipular objetos usaremos **funciones**, que permiten realizar una gran variedad de operaciones con distintos tipos de objetos. Por ejemplo, podemos combinar `x` e `y` en un solo objeto usando la función `c()`:

```
valores <- c(x,y)
valores

#> [1] 10  6
```

Existe una cantidad enorme de funciones para manipular objetos. De hecho, este libro es en esencia un manual sobre qué funciones usar y cómo usarlas para lograr analizar datos de manera efectiva. Comenzaremos de forma sencilla, calculando el promedio de los valores `x` e `y` usando la función `mean()`:

```
mean(valores)

#> [1] 8
```

Notar que podemos asignar el resultado de una función a un objeto (como en `valores <- c(x,y)`) o usar la función directamente, sin guardar su resultado (como en `mean(valores)`). Normalmente asignaremos el resultado de la función si es que queremos usar dicho resultado para otra operación, como en el primer caso.

Finalmente, podemos escribir **comentarios** usando el símbolo `#`. Cuando R encuentre este símbolo, todo lo que siga después en la misma línea será ignorado. Por ejemplo,

```
# Podemos hacer composición de funciones!
mean(c(0, 1, 1, 2, 3, 5, 8, 13))  # Secuencia de Fibonacci (1ros términos)

#> [1] 4.125
```

Espero que hayas podido seguir los ejemplos que hemos desarrollado recién. Si aún no tienes todo claro, tranquilo; a medida que avancemos iremos reforzando estos conceptos. Sin embargo, si por alguna razón no pudiste ejecutar el código, te sugiero resolver el problema antes de continuar.

Usar R interactivamente y a través de scripts

Trabajando con datos

Objetos y asignaciones

R es un lenguaje interpretado, lo que significa que ejecuta nuestras instrucciones directamente, sin compilar un programa previo. Podemos usar R interactivamente a través de la consola:

```
1+2

#> [1] 3
```

La mayoría de las operaciones y funciones en R no guardan el resultado de su ejecución. Por ejemplo, el resultado anterior (`3`) es calculado pero no puede ser reutilizado sin volverse a calcular. Para lograr esto tenemos que asignar el resultado de la operación a un objeto:

```
x <- 1+2
```

1.7 R base vs. tidyverse

Una de las críticas más comunes que se hace a R es que hay muchas (¿demasiadas?) maneras de lograr un mismo resultado. Por ejemplo, R base tiene un sistema propio para crear gráficos, pero también existe **ggplot2** para lograr el mismo objetivo. Esta multiplicidad de métodos puede hacer que R sea más difícil de aprender que lo necesario.

R es un lenguaje antiguo, y muchas de sus funcionalidades básicas —lo que llamamos **R base**— operan bajo paradigmas anticuados. Uno de los más grandes aportes a la programación en R del útltimo tiempo fue realizado por Hadley Wickham, principal autor del **tidyverse**. Este es un conjunto de paquetes que modernizan cómo usamos R para manipular datos, y están diseñados para trabajar bien entre ellos.

En un artículo para R-bloggers[1], David Robinson resume los dos principales "curriculums" de aprendizaje de R:

[1] `https://www.r-bloggers.com/teach-the-tidyverse-to-beginners/`

- **R base primero**: enseñar elementos de sintaxis como `$` y `[[]]`, loops, condicionales, tipos de datos y funciones base como `tapply()`. Este enfoque no se concentra en un sólo marco de datos.
- **tidyverse primero**: comenzar usando **dplyr** para manipular marcos de datos y **ggplot2** para crear gráficos. Luego introducir rápidamente el uso de **tidyr** y **purrr**. Usar el operador `%>%` casi inmediatamente, pero dejar el uso de `$` y `[[]]` para más adelante. Este enfoque se concentra en un sólo marco de datos.

Mi opinión (y la de muchos otros) es que aprender a usar las herramientas del **tidyverse** es más fácil y más productivo. La filosofía detrás del **tidyverse** es similar a la de Python: "Debería haber una —y preferiblemente sólo una— forma obvia de lograr algo". Esta filosofía es buena al aprender un lenguaje de programación, ya que es más consistente y evita confusiones. Es por esto que en este documento prefiero usar herramientas del **tidyverse** cada vez que sea posible.

Por ejemplo, en R base existen al menos tres formas de crear una variable nueva a partir de otra existente. Por otro lado, con el **tidyverse** hay una sóla forma de lograr esto, que además (a mi) me parece más legible:

```
# Agregar una variable con R base
mtcars$libras <- mt$wt * 1000
mtcars[["libras"]] <- mtcars[["wt"]] / 1000
mtcars[, "libras"] <- mtcars[, "wt"] / 1000

# Agregar una variable con el tidyverse
mtcars <- mtcars %>% mutate(libras = wt / 1000)
```

Sin embargo, también hay elementos de R base que son importantes de aprender; de hecho, muchas cosas en R no son posibles sin ellos. Iremos introduciendo estos elementos a medida que nos sean útil para el tema en cuestión.