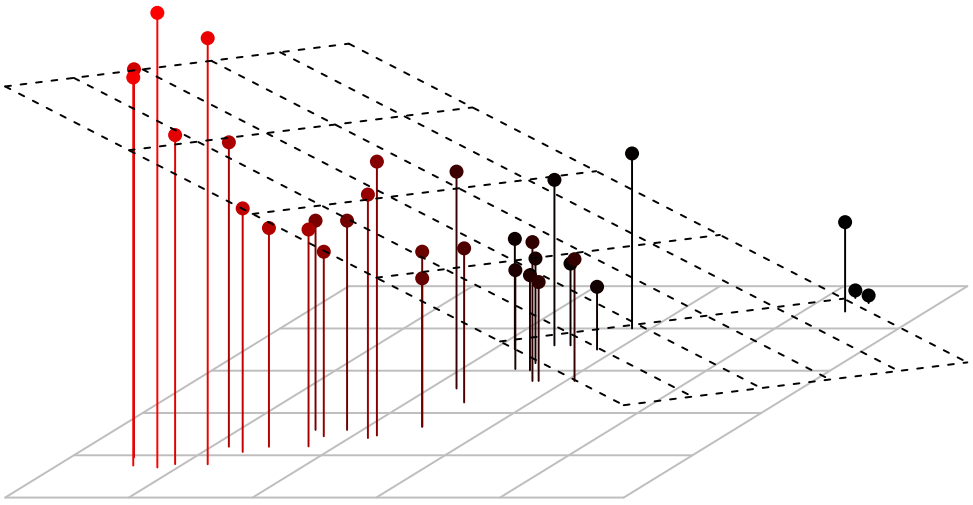
Notas de Econometría en R

Alvaro Carril

Versión 1.1.20171011



Contenidos

1 Introducción **2**
- 1.1 ¿Qué es R? . . . 2
- 1.2 ¿Por qué R? . . . 3
- 1.3 ¿Puedo usar este documento sin R? . . . 3
- 1.4 Obtener R . . . 4
- 1.5 Trabajando en R . . . 6
- 1.6 R en 5 minutos . . . 7
- 1.7 Usar R interactivamente y a través de scripts . . . 9
- 1.8 R base vs. tidyverse . . . 9

2 Regresión lineal simple **11**
- 2.1 Estimando los parámetros del modelo . . . 14
- 2.2 Mínimos Cuadrados Ordinarios . . . 18
- 2.3 Interpretando los parámetros estimados . . . 23
  - 2.3.1 La unidad de medida . . . 26
- 2.4 Distribución muestral de los estimadores . . . 29
- 2.5 Propiedades de los estimadores MCO . . . 34
  - 2.5.1 Estimadores insesgados . . . 34
- 2.6 Varianza muestral de los estimadores . . . 42
  - 2.6.1 Estimando la varianza del error . . . 46
  - 2.6.2 Error estándar (de los estimadores) . . . 46
  - 2.6.3 Extraer estadísticos de una regresión . . . 47
  - 2.6.4 Predicciones y residuos . . . 48

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.

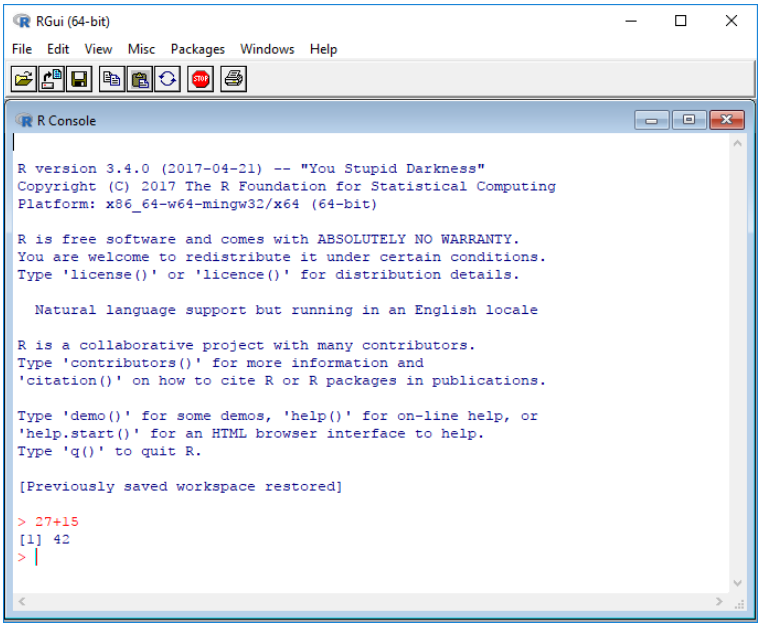
   [Ejemplo]

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 1.8) 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`).

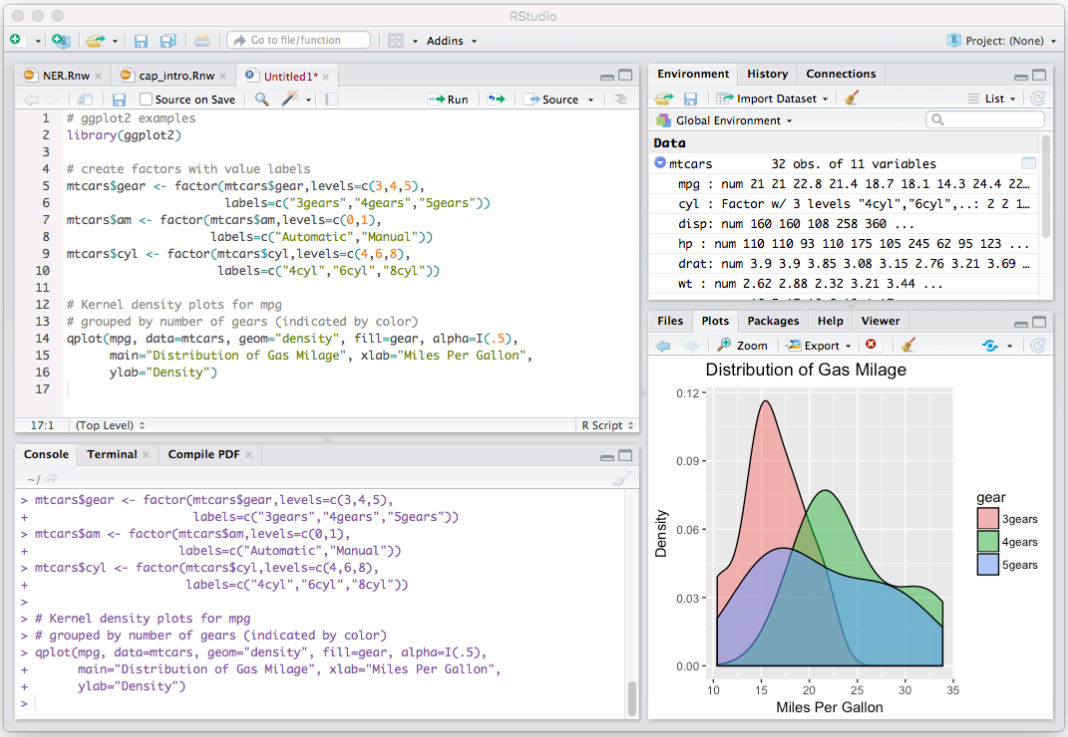
1.4 Obtener R

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.

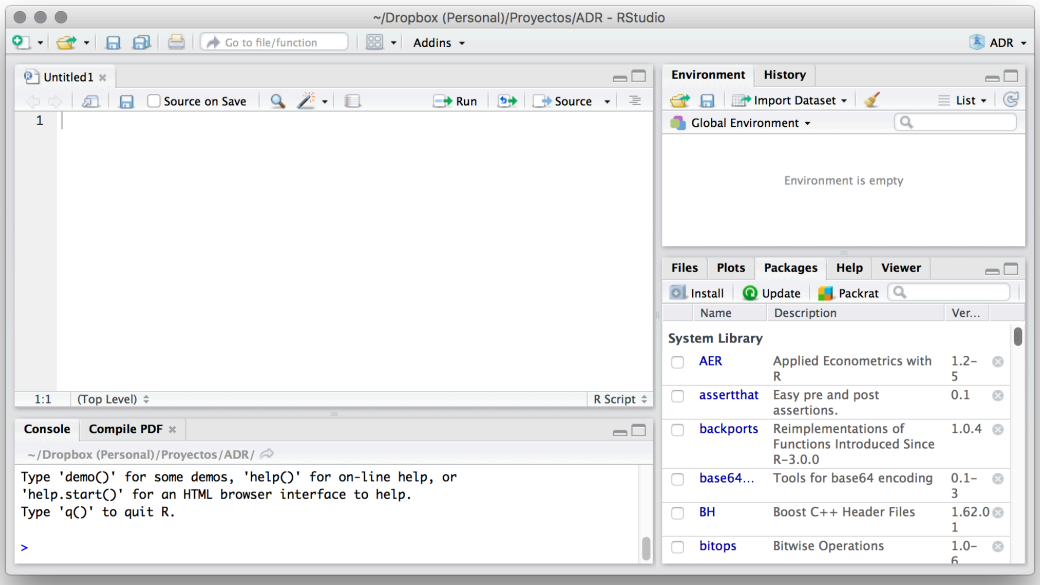


Figura 1.1: RStudio 1.0.136 en macOS 10.12.2

Daremos instrucciones a R a través de dos métodos:

1. Usando 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.

2. A través de un conjunto de comandos escritos en un ***script***, que es un archivo de extensión `.R` con varias instrucciones que serán ejecutadas 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, elige File ▸ New File ▸ 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 que hayamos creado, 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 42 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 cierta forma, este documento es un manual sobre qué funciones usar y cómo usarlas para manipular y analizar datos. 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 (`valores <- c(x,y)`) o usar la función directamente, sin guardar su resultado (`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.

1.7 Usar R interactivamente y a través de scripts

1.8 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:

- **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:

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

```r
# 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.

2 Regresión lineal simple

El modelo de regresión lineal es el caballo de batalla de la econometría aplicada. [Intro motivacional]

En este capítulo comenzaremos a usar R para cargar bases de datos, las que usaremos para estimar modelos lineales y crear gráficos. Estas herramientas nos ayudarán a ilustrar conceptos fundamentales del modelo de regresión lineal y los supuestos que lo sustentan. Nos apoyaremos de los siguientes paquetes, que deben ser cargados antes de correr los bloques que vendrán a continuación:

```
library(tidyverse)
library(haven)
library(reshape2)
```

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 1.8) 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`).

Las preguntas que abordaremos en este capítulo involucran dos variables, que llamaremos x e y. Nos interesará entonces "explicar y en términos de x". Por ejemplo, y podría ser la tasa de crímenes de varios barrios y x el número de parques construidos en esos barrios. O y podría ser el porcentaje de los votos obtenidos por distintos candidatos y x su gasto en la campaña electoral, etc. Si bien esto parece muy simple (y de ahí el nombre del capítulo),

los conceptos que veremos acá serán generalizados muy fácilmente a más variables.

Tomemos este último ejemplo y construyamos un caso concreto. La Figura 2.1 muestra datos de 173 campañas entre los dos candidatos finalistas en elecciones de congreso estadounidenses. Nos enfocamos en las variables de gasto y votos obtenidos por el "candidato A" (un nombre del que Kafka sin duda estaría orgulloso). El eje x corresponde al porcentaje gastado por este candidato (`shareA`), mientras que el eje y indica el porcentaje de los votos que obtuvo (`voteA`). El código para cargar los datos y crear el gráfico se muestra a continuación:

```
# Leer datos (Stata) y asignarlos al objetos 'votos'
votos <- read_dta("http://fmwww.bc.edu/ec-p/data/wooldridge/vote1.dta")
# Imprimir las primeras observaciones de las variables que nos interesan
select(votos, voteA, shareA)
```

```
#> # A tibble: 173 x 2
#>    voteA shareA
#>    <dbl>  <dbl>
#>  1    68  97.41
#>  2    62  60.88
#>  3    73  97.01
#>  4    69  92.40
#>  5    75  72.61
#>  6    69  96.38
#>  7    59  78.23
#>  8    71  98.81
#>  9    76  96.97
#> 10    73  87.44
#> # ... with 163 more rows
```

```
# Graficar ambas variables
ggplot(votos, aes(shareA, voteA)) + geom_point()
```

El código R

Usando `haven::read_dta()` podemos leer una base de datos de Stata (con extensión `*.dta`), la que asignamos al objeto `votos`. Luego usamos `dplyr::select()` para seleccionar algunas variables e imprimirlas. Finalmente usamos las funciones `ggplot2::ggplot()` y `ggplot2::geom_point()` para crear un gráfico de puntos.

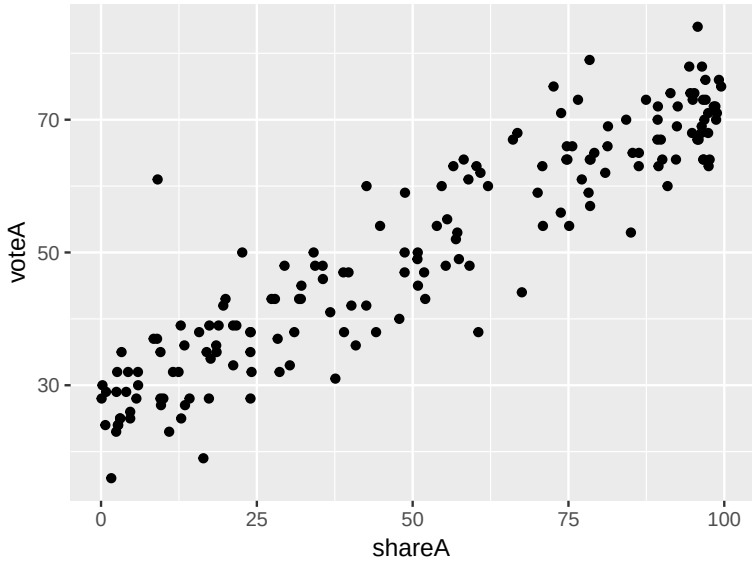


Figura 2.1: Relación entre porcentaje de gasto (`shareA`) y porcentaje de votos obtenidos (`voteA`)

Al intentar modelar la relación entre x e y surgen una serie de preguntas. Quizás la más inmediata es: "¿cuál es la relación funcional entre ambas variables?" Nuestro primer supuesto será que las variables se relacionan de manera lineal, es decir,

$$y = ax + b. \tag{2.1}$$

La ecuación [2.1] corresponde a la clásica función lineal de colegio, y su interpretación geométrica es la misma: a es un término que representa la pendiente de una recta, y b es su intercepto. Este supuesto de relación lineal parece cumplirse bastante bien para nuestro ejemplo de la Figura 2.1: no sabemos qué valores deberían tomar a y b, pero definitivamente pareciera existir alguna combinación de valores de a y b tal que una línea se ajuste "bien" a los datos.

Sin embargo, el modelo expresado en [2.1] asume que existe una relación lineal *perfecta* entre x e y, lo que es altamente restrictivo. En términos de la Figura 2.1, es como suponer que podríamos trazar una recta que pasara por *todos* los puntos, lo que evidentemente es imposible. Entonces surge naturalmente la necesidad de permitir que otros factores —distintos de x— afecten en cierta medida a y, de forma que nuestro modelo tenga un margen de error. Esto se logra agregando un **término de error**, denotado por μ, al modelo lineal.

Adicionalmente, por un simple tema de convención desde ahora en adelante llamaremos β_0 al parámetro del intercepto y β_1 al parámetro de la pendiente. Entonces podemos reescribir nuestro modelo, que ahora es

$$y = \beta_0 + \beta_1 x + \mu. \quad [2.2]$$

Esta ecuación define el **modelo de regresión lineal simple**. Entendamos bien qué quiere decir cada palabra de este título:

- **modelo**, porque es un intento de simplificar la realidad para enfocarnos en algunos aspectos que nos interesan
- **de**, una preposición en la gramática del español
- **regresión** es el término usado originalmente por Galton (1886), aunque conceptualmente no guarda mucha relación con lo que hacemos ahora[1] (pero el nombre es *cool*)
- **lineal**, porque estamos asumiendo que los aspectos de la realidad que nos interesan —las variables x e y— se relacionan de manera lineal en parámetros
- **simple**, porque estamos restringiendo el análisis a solamente dos variables (por el momento).

Muchos libros tienen nombres distintos para x e y. Yo uso la convención (bastante establecida) de referirme a y como la **variable dependiente** del modelo, y a x como la **variable independiente** del modelo. Estas son las variables que ya tenemos, y que intentaremos relacionar. Ya mencionamos que μ es el llamado **término de error**, que captura factores que afectan a y que no hemos incluido en el modelo. Ojo que μ no es igual a u, y se pronuncia «mu». Esto es sólo para enredar, claro.

En conjunto nos referiremos a β_0 y β_1 como los **parámetros** (o coeficientes) del modelo. Estos son simples números, pero como no los tenemos, intentaremos *estimarlos*. Encontrar una forma razonable de estimar estos parámetros será el objetivo de este capítulo (¡y más allá!). Emprendemos esta tarea a continuación.

2.1 Estimando los parámetros del modelo

Nuestro objetivo ahora es estimar los parámetros β_0 y β_1 del modelo de regresión simple explicitado en [2.2]. En términos geométricos, queremos encontrar una forma de estimar

[1] Stigler (1986) cuenta una historia más detallada al respecto.

el intercepto y la pendiente de una recta para que esta se ajuste "bien" a los datos.

El modelo expresado en [2.2] es nuestro supuesto de cómo se relacionan variables *poblacionales*. Sin embargo, nosotros (con alta probabilidad) no tendremos acceso a datos de la población completa, si no que a una *muestra*. Entonces tenemos que adaptar nuestra ecuación para que represente un **modelo muestral**. Logramos esto simplemente suponiendo que contamos con una muestra aleatoria de la población completa. Llamamos $(x_i, y_i) : i = 1, \ldots, n$ a una muestra aleatoria de tamaño n de la población, es decir, x_i e y_i son vectores con n observaciones. Entonces reescribimos el modelo en términos de esta muestra:

$$y_i = \beta_0 + \beta_1 x_i + \mu_i, \qquad [2.3]$$

para todo i, donde $i = 1, \ldots, n$ es un índice que identifica a las observaciones (filas) en los datos. Podríamos tener n personas, familias o empresas en nuestra muestra.

La ecuación [2.3], que sólo se diferencia de [2.2] por ese pequeño subíndice i, es la versión *muestral* del modelo lineal simple. Como hemos supuesto que tenemos una muestra aleatoria, todavía podemos pensar que la variable dependiente y_i es una función lineal de la variable independiente x_i, y que tenemos un término de error μ_i para capturar el efecto que otros factores tienen sobre y_i.

Por ejemplo, podríamos tener un vector y_i con los salarios de $n = 30$ personas, y modelarlo como una función lineal de los años de educación de esas personas, x_i. Resulta útil tener una base de datos concreta con estos vectores, por lo que usaremos una pequeña muestra de datos simulados.

```
# Leer datos CSV de internet
simdatos <- read_csv(
  "https://raw.githubusercontent.com/acarril/NER/master/datos/simdatos.csv")
# Imprimir las primeras observaciones de los datos
simdatos
```

```
#> # A tibble: 30 x 2
#>        y         x
#>    <int>     <dbl>
#>  1     1  4.199913
#>  2     1  7.510634
#>  3     1  2.125473
#>  4     2  8.988857
#>  5     2 10.243105
#>  6     2 11.296823
```

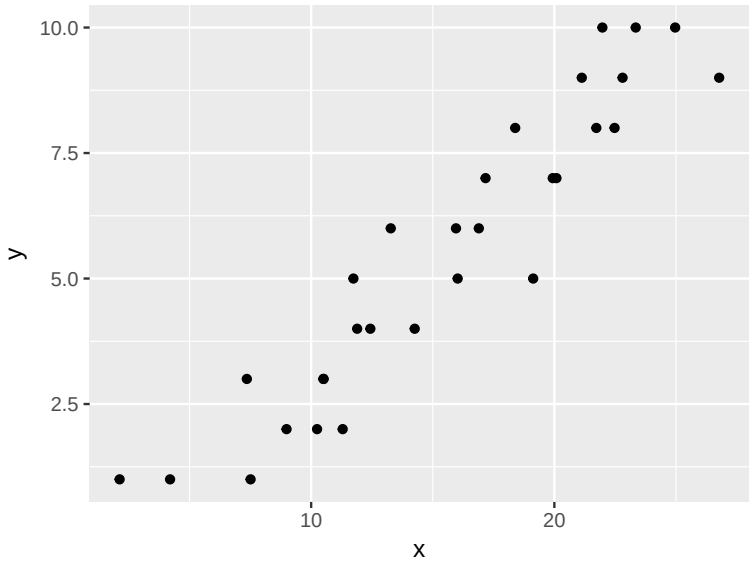
```
#>  7     3  7.356365
#>  8     3 10.505349
#>  9     3 10.511601
#> 10     4 12.434589
#> # ... with 20 more rows
```

El código R

`read_csv()` permite leer bases de datos en formato CSV. Lo usamos para leer una base en línea y luego asignamos esos datos al objeto `simdatos` (datos simulados). Luego escribimos el nombre del objeto (`simdatos`) para imprimirlo. Como es un *tibble* (una base de datos), R imprime solamente las primeras 10 observaciones. Esto es útil para hacernos una idea rápida de las variables de la base y los valores que éstas toman.

En estos datos la variable `y` contiene salarios por hora (en miles de pesos) y la variable `x` contiene años de educación. Cada fila representa una observación (en este caso, una persona) distinta. Siempre es recomendable graficar los datos, ya que un gráfico entrega mucha información y revela patrones que son difíciles de ver en los datos. Creamos un gráfico simple con **ggplot2**:

```
ggplot(simdatos, aes(x, y)) + geom_point()
```



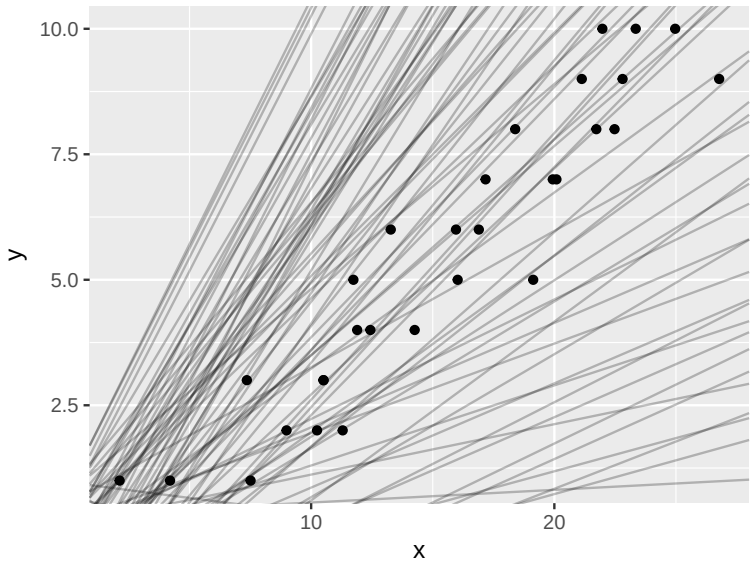
El código R

ggplot2 es un paquete muy poderoso para crear gráficos de distintos tipos. Su principal función es `ggplot()`, que permite definir los datos y las variables que usaremos (dentro de `aes()`). Luego usamos **funciones geom** para definir el tipo de gráfico; en este caso `geom_point()` crea un gráfico de puntos.

Vemos que existe una evidente relación lineal entre x_i e y_i: a mayor nivel de educación parece haber mayor nivel de ingreso. Esto es bueno para nosotros, ya que queremos ajustar los datos al modelo lineal que escribimos en [2.3]. Cabe preguntarse ahora cuál es la manera óptima de elegir los parámetros que determinan dicha relación lineal, es decir, ¿cómo podemos elegir β_0 (el intercepto) y β_1 (la pendiente) para que nuestra predicción de y_i sea lo mejor posible?

Es claro que podríamos elegir entre una infinita variedad de combinaciones de β_0 y β_1 para modelar los datos. Por ejemplo, el código de abajo simula 150 líneas con interceptos y pendientes "razonables", los que se grafican a continuación.

```
set.seed(314)
modelos <- tibble(
  beta1 = runif(150, -3, 1),
  beta2 = runif(150, -1, 1)
  )
ggplot(simdatos, aes(x, y)) +
  geom_point() +
  geom_abline(
    aes(intercept = beta1, slope = beta2),
    data = modelos, alpha = 1/4
  )
```



Si bien hay muchas líneas que claramente no se ajustan bien a los datos, varias otras sí lo hacen, y resulta difícil determinar a simple vista cuál es "la mejor" (asumiendo que tal línea existe). Es evidente que necesitamos un criterio riguroso para elegir los parámetros β_0 y β_1 de manera óptima. Esto es lo que veremos a continuación.

2.2 Mínimos Cuadrados Ordinarios

El método de Mínimos Cuadrados Ordinarios (MCO) es una respuesta directa a la pregunta que planteamos recién: ¿qué criterio conviene usar para elegir el modelo lineal que se ajuste "mejor" a los datos? Intuitivamente, la idea es minimizar el error de ajuste entre los datos reales contenidos en y_i y los datos predichos por nuestra elección de β_0 y β_1, que denotaremos con $\hat{y}_i$.

Ahora formalizaremos esta intuición, y para esto necesitamos introducir un poco de notación nueva. Recordemos que estamos trabajando con [2.3], que es la versión muestral del modelo lineal simple (reimpresa a continuación para comodidad del lector):

$$y_i = \beta_0 + \beta_1 x_i + \mu_i.$$

Dada una muestra de datos, llamaremos $\hat{\beta}_0$ y $\hat{\beta}_1$ a nuestra estimación de los coeficientes. Entonces la **línea de regresión**, denotada por $\hat{y}_i$, está definida como

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i. \qquad [2.4]$$

Algebraicamente, $\hat{y}_i$ es el vector que resulta de multiplicar el vector x_i por el escalar $\hat{\beta}_1$ y luego sumarle $\hat{\beta}_0$. Geométricamente, $\hat{y}_i$ es simplemente una recta que resulta de aplicar una transformación lineal a x_i, donde el intercepto y la pendiente dependen de los valores que hayamos elegido para $\hat{\beta}_0$ y $\hat{\beta}_1$ en [2.4]. La Figura 2.2 muestra una línea de regresión en azul.

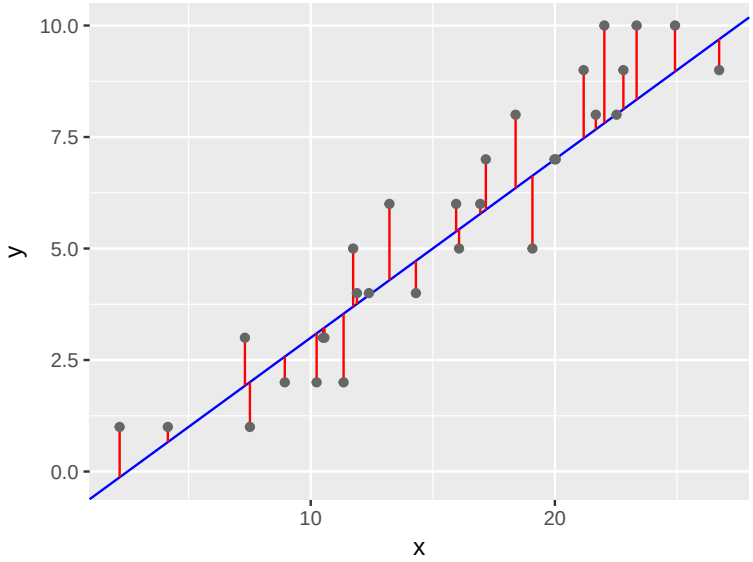


Figura 2.2: Línea de regresión $\hat{y}_i$ en azul y residuo $\hat{\mu}_i$ en rojo

Ahora, si tomamos la diferencia entre y_i (los datos) y $\hat{y}_i$ (nuestra predicción) obtenemos $\hat{\mu}_i$, el **residuo** de nuestra estimación:

$$\begin{aligned} \hat{\mu}_i &= y_i - \hat{y}_i \\ \hat{\mu}_i &= y_i - \hat{\beta}_0 + \hat{\beta}_1 x_i. \end{aligned} \qquad [2.5]$$

Gráficamente el residuo corresponde a la distancia vertical entre cada punto (los datos y_i) y la recta (la predicción $\hat{y}_i$), dibujado con líneas rojas en la Figura 2.2. Es decir que para cada observación i tenemos una medida de la magnitud del error que una elección particular de $\hat{\beta}_0$ y $\hat{\beta}_1$ producen.

Armados con esta notación podemos volver a la pregunta central: ¿Cómo elegimos de manera óptima los parámetros? El método de MCO consiste en elegir $\hat{\beta}_0$ y $\hat{\beta}_1$ en [2.5] tal que se minimice la suma de los residuos al cuadrado.[2] Geométricamente esto es equivalente a minimizar la suma de las distancias verticales entre todos los puntos y la línea de

[2] ¿Por qué al cuadrado? Porque se asume que es irrelevante si el error es una subestimación o una so-

regresión (las líneas rojas de la Figura 2.2). Matemáticamente el problema es

$$\begin{aligned} &\min_{\hat{\beta}_0,\hat{\beta}_1} \sum_{i=1}^{n} \hat{\mu}_i^2 \\ \Leftrightarrow &\min_{\hat{\beta}_0,\hat{\beta}_1} \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2. \end{aligned} \tag{2.6}$$

Al resolver esta minimización obtendremos los estimadores MCO.[3] Estos corresponden a la elección óptima de $\hat{\beta}_0$ y $\hat{\beta}_1$ que hemos estado buscando:

$$\begin{aligned} \hat{\beta}_1 &= \sum_{i=1}^{n} \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \\ &= \frac{\text{Cov}(x,y)}{\text{Var}(x)} \end{aligned} \tag{2.7}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}, \tag{2.8}$$

donde $\bar{x}$ e $\bar{y}$ son los promedios de x_i e y_i, $\text{Var}(x)$ es la varianza muestral de x y $\text{Cov}(x,y)$ es la covarianza muestral de x e y.

¡Hemos logrado nuestro objetivo! Lo anterior significa que para encontrar los estimadores MCO y obtener la línea de regresión que mejor se ajuste a los datos solo tenemos que calcular cuatro cosas: $\bar{y}$, $\bar{x}$, $\text{Cov}(x,y)$ y $\text{Var}(x,y)$. ¡Hagámoslo ahora!

```
# Indicar base de datos a usar
attach(simdatos)
# Promedios de 'x' e 'y'
x.barra <- mean(x)
y.barra <- mean(y)
# Varianza de 'x' y covarianza entre 'x' e 'y'
var.x <- var(x)
cov.xy <- cov(x,y)
# Calcular parámetros MCO
beta1 <- cov.xy/var.x
beta0 <- y.barra - beta1*x.barra
```

breestimación, de forma que lo único que importa es la magnitud. ¿Por qué no se usa el valor absoluto entonces? Buena pregunta: `https://stats.stackexchange.com/q/46019/91358`. Básicamente, porque es algebraicamente más simple manejar valores cuadráticos, y no hace ninguna diferencia en el resultado final.

[3] La derivación paso a paso puede encontrarse, por ejemplo, en Wooldridge (2016, cap. 2).

```r
# 'Desactivar' base de datos
detach(simdatos)
# Imprimir parámetros calculados
beta0
```

```
#> [1] -1.185369
```

```r
beta1
```

```
#> [1] 0.4311958
```

El código R

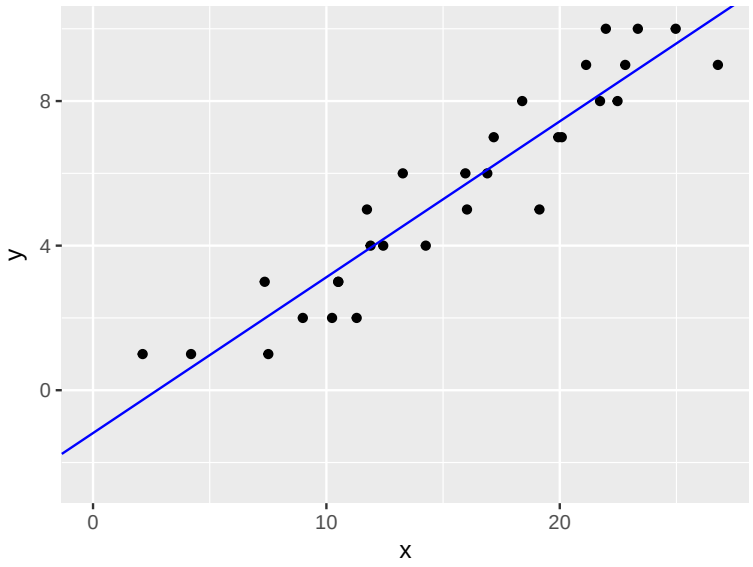
La función `attach()` es útil para indicar que vamos a usar una base de datos en particular para todas las funciones que siguen. Entonces cuando calculamos el promedio de x con `mean(x)`, R sabe que se trata de la variable x en el marco `simdatos` (y no de `votos`, por ejemplo). Luego usamos `detach()` para "desactivar" el uso de la base que indicamos previamente.

Vemos que es fácil calcular los estimadores MCO, y para este ejemplo particular obtuvimos que $\hat{\beta}_0 = -1.19$ y $\hat{\beta}_1 = 0.43$ (redondeando). Con esto podemos completar la línea de regresión definida en [2.4], usando nuestras estimaciones:

$$\hat{y}_i = -1.19 + 0.43 x_i.$$

Esto está bien, pero no es demasiado práctico para visualizar lo que hemos logrado. Una alternativa mejor es usar estos parámetros para graficar $\hat{y}_i$ sobre los datos:

```r
# Graficar línea de regresión MCO
ggplot(simdatos, aes(x,y)) +
  expand_limits(x = 0, y = -2.5) +
  geom_point() +
  geom_abline(
    intercept = beta0, slope = beta1,
    color = "blue", show.legend = FALSE )
```



Finalmente, si bien resulta útil calcular explícitamente los estimadores para entender mejor qué es lo que estamos haciendo, esto no es demasiado práctico: estamos calculando y guardando promedios, varianzas y covarianzas que no nos interesan. En realidad cualquier programa estadístico (decente) tiene algún método para estimar los parámetros de un modelo lineal de manera más compacta, y R no es la excepción:

```
# Estimar directamente los parámetros de un modelo lineal
lm(y ~ x, data = simdatos)

#>
#> Call:
#> lm(formula = y ~ x, data = simdatos)
#>
#> Coefficients:
#> (Intercept)            x
#>     -1.1854       0.4312
```

El código R

La función `lm()` viene de *linear model*, y permite calcular rápidamente los parámetros de un modelo lineal. El primer argumento de la función es una fórmula de R, que no es exactamente lo mismo que una fórmula en el sentido usual de la palabra. Por el momento basta entender que a la izquierda de ~ indicamos la variable dependiente, mientras que a la derecha indicamos las variables independientes. Entonces `lm()` toma una fórmula como `y~x` y la traduce automáticamente a algo como `y = beta0 + beta1 * x`.

Hemos calculado los parámetros del modelo, y sabemos lo que éstos quieren decir en términos matemáticos (minimizan el la suma de los errores cuadráticos) y geométricos (son el intercepto y la pendiente de la línea de "mejor" ajuste). ¿Pero pueden decirnos algo de los datos que usamos para calcularlos? ¿Qué significa, en términos de la relación entre educación e ingreso, que $\hat{\beta}_0 = -1.19$ y $\hat{\beta}_1 = 0.43$? Saber interpretar correctamente los parámetros de una regresión es un arte, y comenzamos a explorarlo a continuación.

2.3 Interpretando los parámetros estimados

Una vez obtenidos los parámetros del modelo, nos interesa poder interpretarlos de manera útil. ¿Qué es lo que quiere decir que hayamos obtenido ciertos valores de $\hat{\beta}_0$ y $\hat{\beta}_1$, más allá de ser el intercepto y la pendiente de una ecuación? Para explorar este tema dejaremos de usar datos simulados y pasaremos a datos reales. En particular usaremos `wage2`, que es una de las bases usadas en Wooldridge (2016) y contiene información de años de educación e ingresos para 935 personas.[4]

```
ingresos <- read_dta("http://fmwww.bc.edu/ec-p/data/wooldridge/wage2.dta")
ingresos <- select(ingresos, wage, educ)
```

El código R

Usamos `haven::read_dta()` para leer bases de datos de Stata, como ya hicimos previamente. En la segunda línea, `select()` nos permite seleccionar las variables `wage` y `educ` de la base `ingresos` que acabamos de leer. Asignamos el resultado de `select()` al mismo objeto para sobreescribirlo con solamente las variables que nos interesan.

[4] Esta base es un subconjunto de los datos del estudio de Blackburn y Neumark (1992).

Sólo por conveniencia, podemos escribir la línea de regresión definida en [2.4] usando los nombres de las variables que nos interesan: salario mensual en dólares (`wage`) y años de educación (`educ`). Luego estimamos los coeficientes del modelo con `lm()`, esta vez definiendo una fórmula donde `wage` es la variable dependiente y `educ` es la variable independiente:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$
$$\Leftrightarrow \widehat{wage} = \hat{\beta}_0 + \hat{\beta}_1 educ.$$

```
lm(wage ~ educ, data = ingresos)

#>
#> Call:
#> lm(formula = wage ~ educ, data = ingresos)
#>
#> Coefficients:
#> (Intercept)         educ
#>      146.95        60.21

ggplot(ingresos, aes(educ, wage)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```

El código R

`geom_smooth` es una función geom que podemos usar en conjunto con `ggplot()` para graficar medias condicionales. Al usar la opción `method = "lm"` indicamos que queremos graficar un modelo lineal entre las variables de los ejes. El uso de `se=FALSE` es para desactivar los intervalos de confianza, tema del que hablaremos más adelante.

El intercepto del modelo lineal (β_0) es el valor predicho de y cuando $x = 0$. En términos de nuestro ejemplo, si una persona tiene 0 años de educación el modelo predice un ingreso mensual promedio de 146.95 dólares. Es importante notar que en algunos contextos no tiene demasiado sentido pensar en $x = 0$ (¿cuántas personas con 0 años de educación existirán? ¿cuántas hay en los datos?). En esas situaciones $\hat{\beta}_0$ no tiene mucho valor en sí mismo, y es labor del investigador discriminar los casos dónde esto ocurra. Exploraremos este punto con mayor profundidad en otro ejemplo más adelante.

El coeficiente estimado para la pendiente ($\hat{\beta}_1$) nos dice la cantidad en la que cambia $\hat{y}$

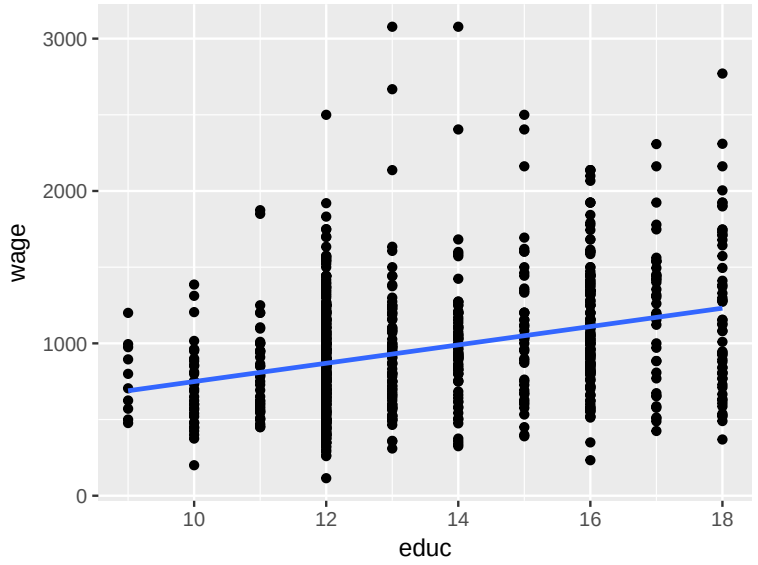


Figura 2.3: Modelo de regresión lineal relacionando años de educación (`educ`) y salario (`wage`)

cuando x aumenta en una unidad. En otras palabras,

$$\Delta\hat{y} = \hat{\beta}_1 \cdot \Delta x, \tag{2.9}$$

donde el operador Δ ("delta") significa simplemente "el cambio en ...".

El aumento de x puede ir asociado tanto a un aumento como a una disminución de $\hat{y}$. Si $\hat{\beta}_1 > 0$ entonces un aumento de una unidad de x está asociado a un aumento de $\hat{\beta}_1$ en $\hat{y}$; si $\hat{\beta}_1 < 0$ entonces un aumento de una unidad de x está asociado a una disminución de $-\hat{\beta}_1$ en $\hat{y}$. Por ejemplo, en la estimación graficada en la Figura 2.3 hemos obtenido que $\hat{\beta}_1 = 60.21$. Esto puede interpretarse como que un aumento de un año de educación (x aumenta en una unidad) está asociado a un aumento promedio de (casi) 60 dólares en el salario. Podemos asegurar que un aumento de x se relaciona con un aumento en $\hat{y}$ porque el signo del coeficiente estimado es positivo; si el coeficiente estimado hubiese sido negativo, entonces diríamos que un año adicional de educación *reduce* el salario promedio (¿tendría sentido esto?).

Hemos sido cuidadosos en no atribuirle una interpretación causal a la variable independiente. En otras palabras, evitamos decir que un año adicional de educación *provoca* un aumento promedio de 60 dólares en el ingreso. La pregunta de causalidad es importante, y un tema muy estudiado en econometría. Sin embargo, sólo podremos hablar de causalidad

más adelante, cuando hayamos estudiado las propiedades estadísticas de los estimadores de MCO e impongamos supuestos más fuertes sobre la población.

2.3.1 La unidad de medida

Habrás notado que la unidad de medida de las variables es clave a la hora de interpretar los coeficientes estimados. En el ejemplo anterior x son años de educación, y por lo tanto $\hat{\beta}_1 = 60.21$ implica que un año adicional de educación está asociado a un aumento de 60.21 dólares en el salario mensual promedio, porque la unidad de y son dólares mensuales.

Veamos ahora cómo cambian los parámetros estimados cuando cambia la unidad de medida de las variables. En concreto, usaremos los datos `mtcars`, que contiene 32 observaciones de modelos de auto. Nos concentramos en una variable que indica su rendimiento en millas por galón (`mpg`) y otra con su peso en miles de libras (`wt`).

```r
# Calcular estadística descriptiva para las variables 'mpg' y 'wt'
summary(select(mtcars, mpg, wt))
```

```
#>       mpg             wt
#>  Min.   :10.40   Min.   :1.513
#>  1st Qu.:15.43   1st Qu.:2.581
#>  Median :19.20   Median :3.325
#>  Mean   :20.09   Mean   :3.217
#>  3rd Qu.:22.80   3rd Qu.:3.610
#>  Max.   :33.90   Max.   :5.424
```

```r
# Estimar el modelo lineal
lm(mpg ~ wt, data = mtcars)
```

```
#>
#> Call:
#> lm(formula = mpg ~ wt, data = mtcars)
#>
#> Coefficients:
#> (Intercept)           wt
#>      37.285       -5.344
```

El código R

`summary()` es una de las funciones más útiles de R base, ya que permite producir un resumen con información útil de distintos objetos: bases de datos, modelos, etc. En este caso la usamos para producir estadística descriptiva de las variables elegidas con `select()`, para evitar usar la función sobre todas las variables de `mtcars`.

Al estimar este modelo obtenemos que $\hat{\beta}_1 = -5.34$. Esto significa que un aumento de una unidad en `wt` está asociado a una disminución promedio de 5.34 en `mpg`.

OK, ¿y?

Como no sé mucho de autos ni del sistema métrico imperial, no tengo idea si esto es mucho o poco. A pesar que entiendo cómo interpretar el coeficiente en términos algebraicos, la unidad de medida de las variables hace que esta información no sea tan útil (para mi).

Afortunadamente es muy fácil transformar las variables y luego volver a estimar el modelo con ellas. Crearemos las variables `rendimiento_kml` con el rendimiento en kilómetros por litro y `peso_kg` con el peso en kilógramos de cada auto:

```
# Crear marco 'autos' con las variables que nos interesan
autos <- as_tibble(select(rownames_to_column(mtcars, var = "modelo"),
                          modelo, mpg, wt))
# Crear dos variables nuevas de rendimiento y peso
autos <- mutate(autos,
                rendimiento_kml =  mpg * 0.425144,
                peso_kg = wt * 453.592)
# Estimar el modelo con estas nuevas variables
lm(rendimiento_kml ~ peso_kg, data = autos)

#>
#> Call:
#> lm(formula = rendimiento_kml ~ peso_kg, data = autos)
#>
#> Coefficients:
#> (Intercept)      peso_kg
#>   15.851548    -0.005009
```

Esta transformación todavía presenta un pequeño problema. Como la unidad de medida de `peso_kg` son kilos, $\hat{\beta}_1 = -0.005$ nos dice que por cada kilo adicional el modelo predice una reducción de 0.005 en el rendimiento promedio. Claramente pensar en un kilo adi-

cional en algo tan pesado como un auto no tiene mucho sentido, por lo que estimaremos el modelo con una variable que indique el peso en unidades de 100 kilos:

```
# Agregar variable con peso en unidades de 100kg
autos <- autos %>% mutate(peso_100kg = peso_kg / 100)
# Estimar modelo con esta nueva variable independiente
lm(rendimiento_kml ~ peso_100kg, data = autos)

#>
#> Call:
#> lm(formula = rendimiento_kml ~ peso_100kg, data = autos)
#>
#> Coefficients:
#> (Intercept)   peso_100kg
#>     15.8515      -0.5009
```

El código R

Este bloque introduce el uso del **operador de pipa**, `%>%`. Creado por Milton Bache en su paquete **magrittr**, el operador de pipa sirve para pasar un objeto como primer argumento de una función. Entonces en lugar de escribir `f(x,y)` podemos escribir de manera equivalente `x %>% f(y)`. Esto permite "desanidar" funciones, ya que el operador es recursivo: operaciones como $f(g(x))$ pueden expresarse como `x %>% g %>% f`, lo que las hace más legibles.

Ahora podemos interpretar el modelo más fácilmente, teniendo unidades adecuadas. El coeficiente asociado a la variable independiente se puede interpretar como que el modelo predice una reducción promedio de 0.5 km/l por cada 100 kilos adicionales que pese un auto. Esto definitivamente parece razonable, y los coeficientes entregan información útil directamente. Es importante notar que este valor de $\hat{\beta}_1$ usando `peso_100kg` es igual al calculado usando `peso_kg`, pero multiplicado por 100. Al realizar una transformación lineal sobre la variable, transformamos de manera inversa el coeficiente asociado.

Por otro lado, el parámetro del intercepto indica que el modelo predice un rendimiento promedio de 15.85 para un auto de 0 kilos. Este corresponde a un claro caso donde el intercepto no tiene sentido en sí mismo, ya que es imposible un auto de 0 kilos.

Recordemos que la línea de regresión de este modelo es

$$\hat{y}_i = 15.85 - 0.5 \cdot x_i.$$

Podemos usar la ecuación de la línea de regresión para calcular el valor predicho $\hat{y}_i$ dado

un nivel de x_i. Por ejemplo, nuestra estimación predice que un auto de 1600 kilos tendrá un rendimiento promedio de $15.85 - 0.5 \cdot 16 = 7.84$ km/l. Esto se ajusta bastante a los datos reales, como puede apreciarse en la Figura 2.4.

```
# Graficar la línea de regresión sobre los datos
autos %>% ggplot(aes(peso_100kg, rendimiento_kml)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```

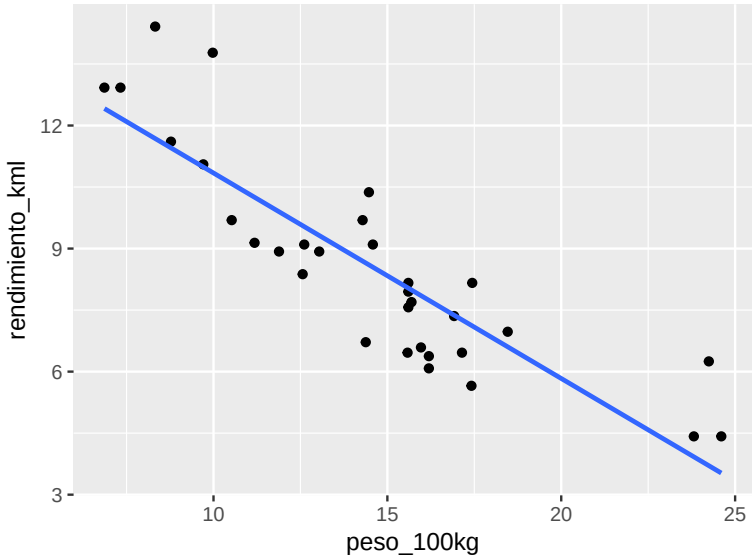


Figura 2.4: Modelo de regresión lineal relacionando el peso de un auto (en unidades de 100 kilos) y su rendimiento (en kilómetros por litro)

2.4 Distribución muestral de los estimadores

Es importante tener en cuenta que los estimadores MCO son eso: estimadores de un parámetro poblacional real. Podemos pensar en MCO como una máquina en donde entra una muestra y salen estimadores, tal como se representa en la Figura 2.5. Las estimaciones que realicemos dependen de la muestra que usemos, de forma que distintas muestran producirán distintas estimaciones de los parámetros.

No es necesario creer en mi palabra. A continuación simulamos 500 muestreos aleatorios de 1000 observaciones cada uno. Nos concentraremos únicamente en el parámetro de la

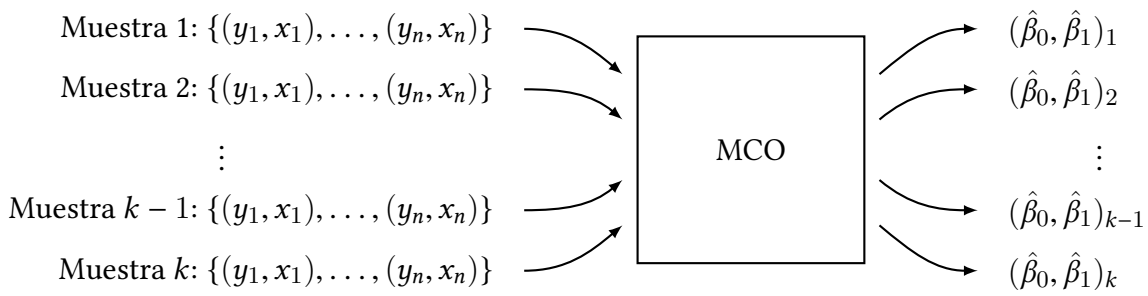
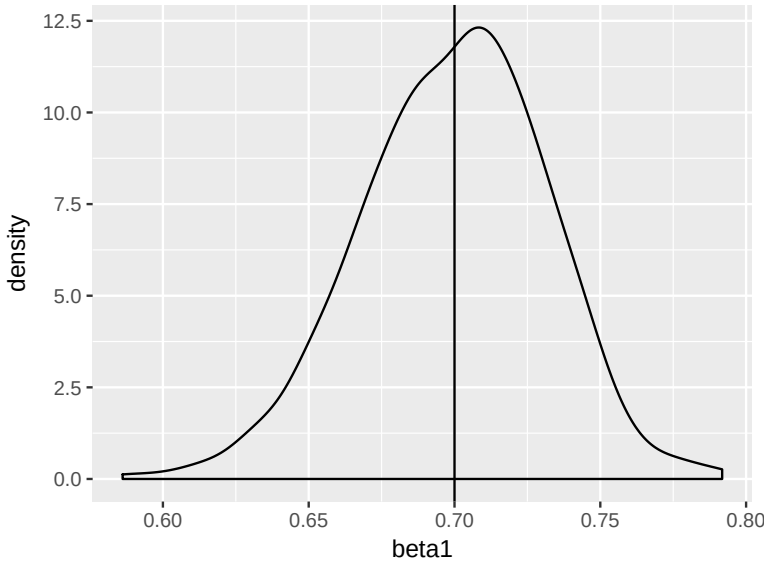


Figura 2.5: MCO es como una caja donde entran muestras y salen parámetros estimados

pendiente, β_1. Fijamos el verdadero valor de dicho parámetro en $\beta_1 = 0.7$ (en el código `b <- 0.7`). Al simular los datos tomados de una población normal, usando los parámetros que hemos definido, estimamos el parámetro k veces y obtenemos k valores para $\hat{\beta}_1$, los que vamos guardando en el vector `beta1`. Finalmente hacemos un gráfico de densidad con los valores de dicho vector.

```r
set.seed(1102) # fijar la semilla para resultados replicables

# Parámetros poblacionales ("verdaderos")
a <- 0.2 # verdadero valor del intercepto
b <- 0.7 # verdadero valor de la pendiente

# Parámetros de la simulación
k <- 500 # número de simulaciones
n <- 1000 # tamaño muestral

# Comenzar simulación
beta1 <- numeric(length = k) # vector vacío para guardar pendientes estimadas
for(i in 1:k) { # loop de 500 simulaciones
  x <- rnorm(n, mean=2, sd=1) # variable independiente
  u <- rnorm(n, mean=0, sd=1) # término de error
  y <- a + b*x + u # variable dependiente
  estimacion <- lm(y ~ x) # estimar parámetros por MCO
  beta1[i] <- coef(estimacion)[2] # guardar pendiente (2do coef) estimada
}

# Graficar densidad de 500 pendientes estimadas
```

```r
ggplot(tibble(beta1), aes(beta1)) +
  geom_vline(xintercept = b) +
  geom_density()
```



Lo que este gráfico nos muestra es que a pesar de que existe un único valor poblacional para uno de los parámetros estimados (β_1 en este caso), al estimar una regresión por MCO con distintas muestras obtendremos distintos valores estimados para dicho parámetro. En un caso real no conoceremos el valor poblacional del parámetro, ni podremos tomar 500 muestras distintas... ¡tendremos suerte si obtenemos 1 buena muestra! Por eso, es esencial tener presente esta distribución de $\hat{\beta}$ al estimar un modelo.

Habrás notado que al tomar 500 muestras y estimar $\hat{\beta}_1$, los valores estimados se concentran en torno al verdadero valor de β_1, que es 0.7 en este ejemplo. ¡Esto es bueno... y no es casualidad! En realidad es una propiedad de los estimadores MCO. A continuación formalizaremos esta y otras propiedades, con el fin de entender qué debe ocurrir para que se cumplan.

Hasta ahora hemos aprendido a calcular e interpretar los parámetros de un modelo lineal simple. Sin embargo, recordemos que nuestra estimación de $\hat{\beta}_0$ y $\hat{\beta}_1$ proviene de una muestra aleatoria de la población. En esta sección analizaremos las propiedades estadísticas de $\hat{\beta}_0$ y $\hat{\beta}_1$, lo que nos dará herramientas para analizar qué tanto se acercan nuestras estimaciones a los verdaderos parámetros poblacionales.

Primero nos detendremos a entender el problema en sí: ¿en qué se diferencian los parámetros estimados ($\hat{\beta}_0$ y $\hat{\beta}_1$) de los verdaderos parámetros poblacionales (β_0 y β_1)?

Cuando planteamos el modelo de regresión lineal simple (ecuación [2.2]), supusimos que las variables x e y se relacionaban por medio de una función lineal, y que

Si bien en estricto rigor suponemos que el "tamaño" de la población es infinito, fijamos $N = 1.000.000$ para efectos de nuestra población simulada. Luego definimos un error `u` con media 0 (es uno de nuestros supuestos) y desviación estándar `sigma` $= 5$. Finalmente generamos `y` como una función lineal de `x`, con intercepto 2 y pendiente 10, e incluyendo nuestro error `u`$\sim \mathcal{N}(0, 5)$.[5]

```r
# Parámetros iniciales
set.seed(03072017)
N <- 1000000
sigma <- 5

# Variables poblacionales
x <- runif(N)
u <- rnorm(N, 0, sigma)
y <- 2 + 10*x + u
poblacion <- data.frame(y,x)
```

Al haber creado nosotros mismos los datos poblacionales, estamos seguros de los verdaderos valores de los parámetros: $\beta_0 = 2$ y $\beta_1 = 10$. La única desviación que se dará en nuestra simulación es debido a que la población no es infinita (aunque se le acerca bastante).

```r
# Estimar parámetros poblacionales
lm(y~x, data = poblacion)

#>
#> Call:
#> lm(formula = y ~ x, data = poblacion)
#>
#> Coefficients:
#> (Intercept)            x
#>       1.985       10.020
```

[5] No es necesario asumir que $\mu \sim \mathcal{N}$, ya que bastaría con que $\mathrm{E}(\mu) = 0$. Es un error común pensar que MCO requiere errores con distribución normal.

Verificamos que en nuestra población simulada los parámetros son ... A continuación extraemos una muestra aleatoria de 100 observaciones ($n = 100$) de la población. Esto es lo que nosotros suponemos que hacemos cuando tenemos, por ejemplo, una base de datos con puntajes SIMCE para de 100 estudiantes. Luego estimamos los coeficientes $\hat{\beta}_0$ y $\hat{\beta}_1$ para esta muestra en particular.

```r
# Extraer una muestra aleatoria de la población y estimar sus parámetros
muestra <- poblacion[sample(1:N, 100),]
lm(y~x, data = muestra)

#>
#> Call:
#> lm(formula = y ~ x, data = muestra)
#>
#> Coefficients:
#> (Intercept)            x
#>       3.101        8.085

coefficients(modelo_muestral)

#> (Intercept)           x
#>    3.101083    8.084905
```

Observamos que los coeficientes estimados en esta muestra particular son bastante distintos de los reales: $\hat{\beta}_0 = 3.1$ y $\hat{\beta}_1 = 8.08$. Este fenómeno no es exclusivo a los estimadores, y en realidad ocurre para cualquier tipo de estadística que es calculada para la muestra de una población.

Por ejemplo, supón una población (un poco ridícula) de 4 personas, y observamos sus pesos en kilogramos:

$$P = \{74, 62, 65, 71\}.$$

Si tomamos muestras de 2 personas repetidas veces, obtendremos distintas estimaciones del promedio poblacional. Algunos promedios muestrales serán iguales al promedio poblacional, mientras que otros no. Lo importante es que al tomar varias muestras y guardar registro de los promedios muestrales,

ninguna muestra de menos de 2 observaciones nos entregará este promedio. Al extraer repetidas muestras obtendremos una distribución muestral: algunos promedios de nuestra muestra se repetirán más que otros.

2.5 Propiedades de los estimadores MCO

Hasta ahora hemos aprendido a calcular e interpretar los parámetros de un modelo lineal simple. En la Sección 2.4 enfatizamos que los estimadores MCO ($\hat{\beta}_0$ y $\hat{\beta}_1$, o en general $\hat{\beta}$) tienen una distribución de probabilidad. Intuitivamente, la razón por la que esto ocurre es porque nuestra estimación de $\hat{\beta}_0$ y $\hat{\beta}_1$ depende de la muestra aleatoria particular que tengamos de la población; si tomáramos distintas muestras, obtendríamos distintos valores para $\hat{\beta}$. Entonces nuestra estimación de $\hat{\beta}$ es solo una realización de infinidad de posibilidades, como se representa en la Figura 2.6.

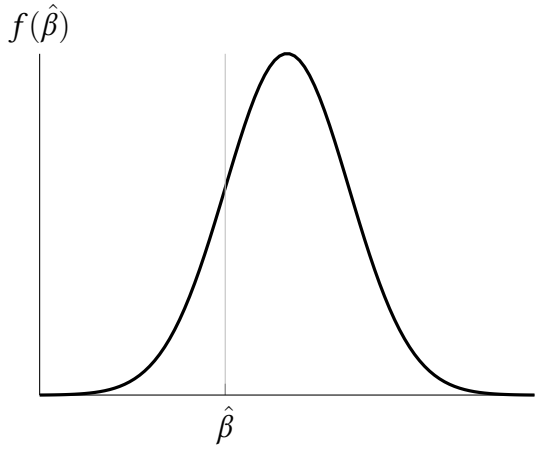


Figura 2.6: La realización de $\hat{\beta}$ que nosotros observamos proviene de una función de probabilidad

2.5.1 Estimadores insesgados

Teniendo en cuenta que los los estimadores tienen una función de densidad de probabilidad, parece natural querer que esta probabilidad al menos esté centrada en torno al verdadero parámetro β, es decir, que

$$\mathrm{E}(\hat{\beta}) = \beta. \qquad [2.10]$$

A un estimador que cumple con esta propiedad se le llama **estimador insesgado**. La Figura 2.7 muestra la distribución de un estimador insesgado en azul, mientras que la distribución de un estimador sesgado está graficada en rojo. Notar que ambos tienen

igual varianza, pero sólo el primero está centrado en torno al verdadero valor de β, y por lo tanto es insesgado.

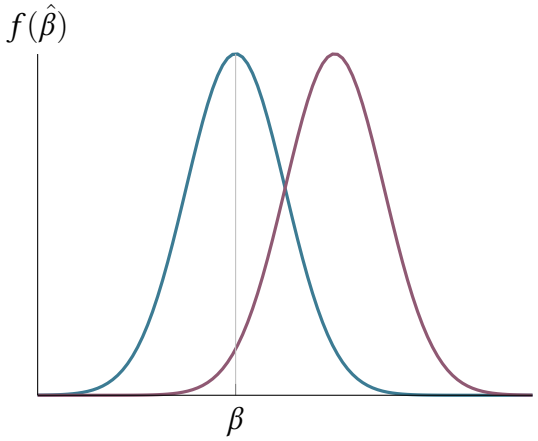


Figura 2.7: Estimador insesgado (azul) vs. estimador sesgado (rojo)

Para asegurar que los estimadores MCO que hemos obtenido son insesgados deben cumplirse cuatro supuestos. Hemos enunciado estos supuestos a medida que construimos los estimadores, pero a continuación los enunciaremos de forma más explícita.

RLS.1: Lineal en parámetros Esto corresponde al supuesto fundamental del modelo, donde asumimos que las variables poblacionales se relacionan siguiendo la siguiente forma funcional:

$$y = \beta_0 + \beta_1 x + \mu. \qquad \text{[2.2, revisitada]}$$

Nuestro objetivo es estimar β_0 y β_1 suponiendo que los datos se relacionan de forma lineal en parámetros y que x, y y μ son variables aleatorias.

RLS.2: Muestreo aleatorio Suponemos que tenemos una muestra aleatoria de tamaño n de la población total, de forma que en términos muestrales el modelo lineal es

$$y_i = \beta_0 + \beta_1 x_i + \mu_i, \quad i = 1, \ldots, n. \qquad \text{[2.3, revisitada]}$$

Sí, es lo mismo que arriba, pero con el subíndice. En términos de notación la diferencia es sutil; lo importante es entender lo que esta diferencia significa.

Por ejemplo, supongamos que queremos estimar el efecto del peso de un auto sobre su rendimiento en carretera. Es decir que el modelo a estimar es

$$rendimiento_i = \beta_0 + \beta_1 peso_i + \mu_i,$$

donde nos interesa en particular la estimación de β_1.

Para efectos de este ejemplo supondremos que los datos de la población completa de automóviles del país se encuentra en la base `mpg`. Usaremos la cilindrada del auto (en litros) como aproximación de su peso (`displ`), y su rendimiento en carretera (en millas por galón) como variable independiente (`hwy`). Estimar el coeficiente de interés es muy simple:

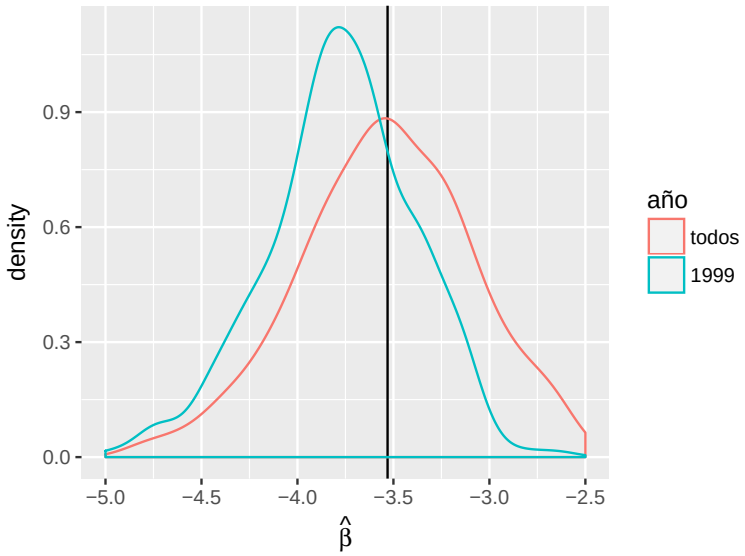
```
# Estimamos modelo "poblacional"
lm(hwy ~ displ, data = mpg)

#>
#> Call:
#> lm(formula = hwy ~ displ, data = mpg)
#>
#> Coefficients:
#> (Intercept)        displ
#>      35.698       -3.531

# Guardamos el beta poblacional para usarlo en gráfico
beta_pob <- coefficients(lm(hwy ~ displ, data = mpg))[2]
```

Como supusimos que el total de observaciones en `mpg` corresponde a la población completa, para efectos de este ejemplo el parámetro poblacional "verdadero" es $\beta_1 = -3.53$.

Intentaremos ahora estimar dicho parámetro usando dos muestra tomadas de los datos poblacionales. Una muestra será completamente aleatoria, mientras que la otra sólo se extraerá del subconjunto de automóviles del año 1999. Ambas muestras serán del mismo tamaño (50 observaciones), y repetiremos este muestreo la misma cantidad de veces para ambos tipos de muestra (300 muestreos).

```
# Parámetros iniciales
set.seed(1102)
k <- 300

# Guardar matriz de coeficientes bajo distintas muestras
betas <- matrix(data=NA, nrow=k, ncol=2)
```

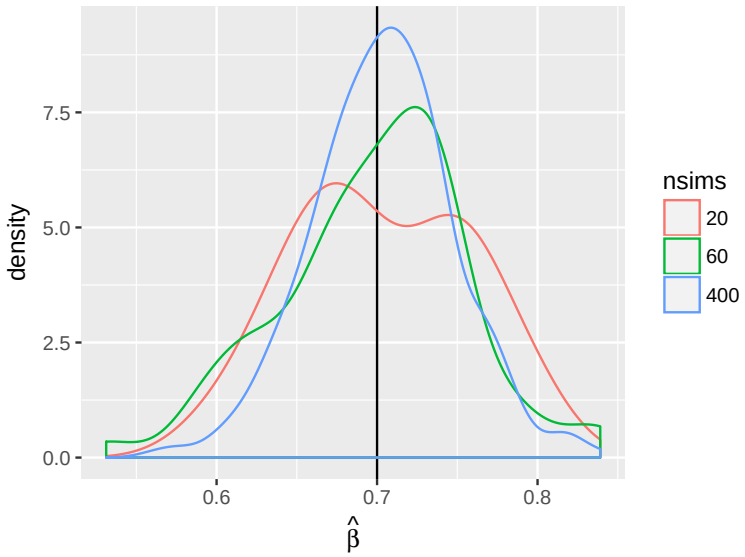
```r
for(i in 1:k) {
  autos <- sample_n(mpg, size = 50)
  autos99 <- sample_n(filter(mpg, year==1999), size = 50)
  betas[i,1] <- coef(lm(hwy ~ displ, data = autos))[2]
  betas[i,2] <- coef(lm(hwy ~ displ, data = autos99))[2]
}

# Transformar matriz a base de datos usable para graficar
betas <- melt(betas)
betas <- mutate(betas, año = factor(Var2, labels = c("todos","1999")))

# Graficar densidad de coefs. estimados por tipo de muestra
ggplot(betas, aes(value)) +
  geom_vline(xintercept = beta_pob) +
  geom_density(aes(color = año)) +
  scale_x_continuous(name = expression(hat(beta)), limits = c(-5, -2.5))
```



Al tomar una muestra aleatoria de la población, la distribución de $\hat{\beta}_1$ se centra en torno al verdadero valor de β_1. Por otro lado, al tomar una muestra que no es aleatoria (en el ejemplo la muestra es condicional al año), la distribución de $\hat{\beta}_1$ no estará centrada en torno al verdadero valor del parámetro, es decir, es un estimador sesgado. En este caso particular hay un sesgo a la baja, ya que es de esperar que si sólo seleccionamos autos

relativamente antiguos, estos serán en promedio menos eficientes.

```r
# Cargar paquete para melt() y fijar semilla
set.seed(1102)

# Definir función para guardar pendientes simuladas
simbetas <- function(k, a=0.2, b=0.7, n=500) {
  beta1 <- numeric(length = k)
  for(i in 1:k) {
    x <- rnorm(n, mean=2, sd=1)
    u <- rnorm(n, mean=0, sd=1)
    y <- a + b*x + u
    estimacion <- lm(y ~ x)
    beta1[i] <- coef(estimacion)[2]
  }
  beta1
}

# Aplicar la función con distintos números de simulaciones
nsims <- c(20,60,400)
betas <- lapply(nsims, simbetas)

# Transformar datos para graficarlos
betas <- melt(betas)
betas <- mutate(betas, nsims = factor(L1, labels = nsims))

# Graficar densidad de coeficientes estimados por n. de sims
ggplot(betas, aes(value)) +
  geom_vline(xintercept = .7) +
  geom_density(aes(color = nsims)) +
  labs(x = expression(hat(beta)))
```



RLS.3: Variación en las variables independientes Necesitamos que la (o las) variables independientes del modelo tengan algo de variación. Un poco más formalmente,

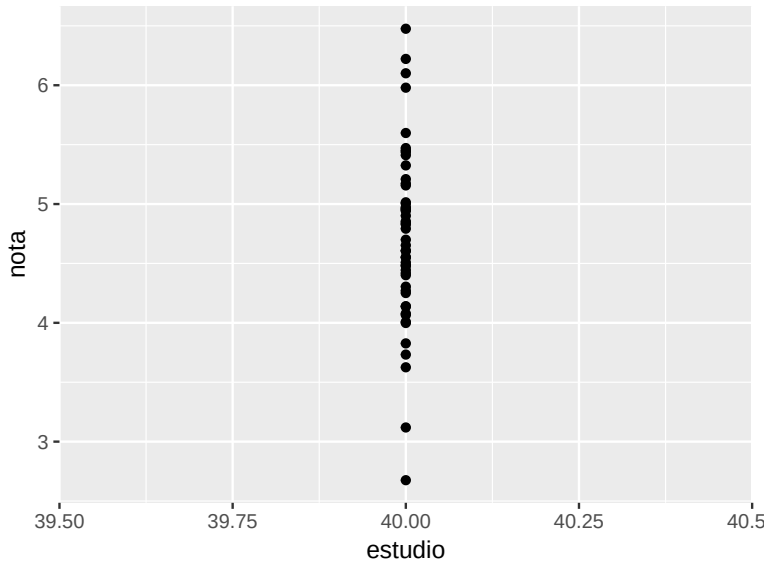
$$\{x_i, i = 1, \ldots, n\} \text{ no son todos iguales.} \qquad [2.11]$$

Este supuesto es fácil, pero necesario matemáticamente. Por ejemplo, supongamos que queremos estimar un modelo como

$$nota_i = \beta_0 + \beta_1 estudio_i + \mu_i,$$

donde $estudio_i$ son los minutos estudiados por el alumno i y $nota_i$ es su nota en una prueba. Para estimar este modelo es necesario que exista variabilidad en la cantidad de horas estudiadas por los alumnos de la muestra. Si todos estudiaron exactamente la misma cantidad de minutos, entonces será imposible determinar la relación entre minutos estudiados y la nota obtenida. Este hecho también es obvio geométricamente:

```
data <- tibble(
  nota = rnorm(50, 4.6, 1),
  estudio = rep(40, 50))
ggplot(data, aes(estudio, nota)) +
  geom_point()
```



¿Cuál es la la línea que mejor se ajusta a los datos? [6]

RLS.4: Media condicional cero El error u debe tener un valor esperado igual a cero, condicional a los valores de la variable independiente:

$$E(\mu|x) = 0. \tag{2.12}$$

En una muestra aleatoria esto significa que

$$E(\mu_i|x_i) = 0 \quad \forall\ i = 1, \ldots, n.$$

En palabras, el valor esperado de la media de los términos de error debe ser cero, condicional a la variable independiente. En otras palabras, la distribución del error tiene media 0 y no depende de x. Esto significa que no puede haber relación entre las variables independientes y el término de error.

¿Cómo llegamos a esta conclusión? Ya habíamos supuesto que la esperanza del error es cero, es decir, que $\mathrm{E}(\mu) = 0$. Mencionamos que no perdemos nada haciendo esto, ya que siempre podemos acomodar el intercepto de manera de asegurarnos que $\mathrm{E}(\mu) = 0$.

[6] Y recuerda que una función $f : X \to Y$ corresponde a cualquier subconjunto S del conjunto $X \times Y$ que satisface que si $(x, y) \in S$ y $(x, y') \in S$, entonces $y = y'$. O, en otras palabras, una función válida asigna a cada elemento de X exactamente un elemento en Y. O, en otras palabras, una función no puede ser una línea vertical.

Entonces este no es un supuesto demasiado fuerte. El punto clave es que necesitamos que μ y x sean independientes, ya que sólo de esa forma $\mathrm{E}(\mu|x) = E(\mu)$ (que ya supusimos es igual a 0). Por otro lado, si existe una correlación entre μ y x esto implicará que $\mathrm{E}(\mu|x) \neq 0$, y habremos violado el supuesto.

Para contemplar otra forma de entender este supuesto es conveniente escribirlo en términos de la esperanza condicional de y:

$$E(y|x) = \beta_0 + \beta_1 x.$$

La equivalencia debiese ser evidente: si esperamos que en promedio el error sea 0, entonces debemos esperar que en promedio y sea $\beta_0 + \beta_1 x$. Visto de esta forma, RLS.4 nos indica que y sólo depende linealmente en x, y de ninguna otra variable. Entonces es fácil anticipar que violaremos este supuesto cuando omitamos alguna variable relevante en nuestro modelo, o cuando la forma funcional esté mal especificada. Exploraremos estos casos en detalle más adelante.

El supuesto de media condicional cero es probablemente el supuesto más "importante" a la hora de determinar si una regresión lineal entregará parámetros insesgados. Esto ocurre porque es imposible determinar si el supuesto efectivamente se cumple o no, ya que depende de que se cumplan condiciones que no son directamente comprobables. Volveremos varias veces sobre este supuesto a lo largo del libro.

Finalmente, usando los supuestos RLS.1 a RLS.4 se puede demostrar que los estimadores MCO son insesgados, es decir, que[7]

$$\mathrm{E}(\hat{\beta}_0) = \beta_0 \text{ y } \mathrm{E}(\hat{\beta}_1) = \beta_1. \qquad [2.13]$$

En general, si cualquiera de los RLS.1 a RLS.4 no se cumplen, entonces perderemos la propiedad de insesgamiento. RLS.1 requiere que y y x estén relacionados linealmente. Hay que recordar, sin embargo, que la relación es solamente lineal en *parámetros*, no en variables, por lo que podemos transformar x e y para capturar relaciones no lineales más interesantes. RLS.2 asume que nuestros datos corresponden a una muestra aleatoria de la población, lo que no siempre se cumple en datos de corte transversal. Por ejemplo, es posible que solamente tengamos datos de micro y pequeñas empresas, o que solamente observemos jefas de hogar de los quintiles más pobres.

RLS.3 es bastante débil, y con mucha seguridad se cumplirá en la gran mayoría de datos. Sin embargo, RSL.4 representa un problema potencial. Si no se cumple que la media condicional del error es 0, esto es equivalente a decir que existe alguna correlación entre x

[7] La demostración completa corresponde al Teorema 2.1 de Wooldridge (2016, cap. 2).

y μ. Una razón por la que esto puede ocurrir es porque dejamos de incluir una variable relevante en nuestro modelo, de forma que esta afecta a x por medio de μ.

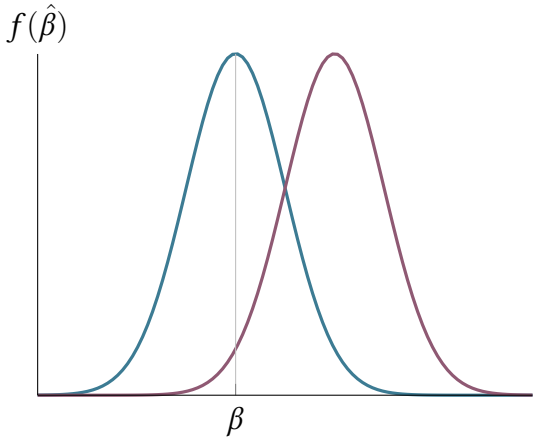


Figura 2.8: Estimador insesgado (azul) vs. estimador sesgado (rojo)

Además de requerir que el estimador se centre en torno al verdadero valor del parámetro, también queremos saber qué tan lejos podemos esperar que $\hat{\beta}$ se aleje de β en promedio. Esto nos permitirá elegir el estimador más eficiente, es decir, de entre todos los estimadores insesgados elegir aquél que tenga la menor varianza. La Figura 2.9 muestra la distribución de dos estimadores insesgados. Sin embargo, el estimador representado por la curva roja tiene relativamente mayor varianza que el representado por la azul.

Nuestro objetivo es obtener ambas propiedades para nuestros estimadores MCO. Sin embargo, estas propiedades sólo existirán en la medida que se cumplan algunos supuestos. En esta sección veremos cuáles son estos supuestos y qué propiedades entregan.

Es importante tener en cuenta que las propiedades estadísticas no tienen nada que ver con una muestra en particular, si no que es una propiedad que los estimadores cumplirán cuando se realice un muestreo aleatorio repetidamente.

2.6 Varianza muestral de los estimadores

Además de asegurarnos que la distribución muestral de $\hat{\beta}$ esté centrada alrededor de β —es decir, que $\hat{\beta}$ sea insesgado—, es importante saber qué tan dispersa es esta distribución de $\hat{\beta}$. La Figura 2.9 muestra el caso de dos estimadores insesgados donde uno tiene una

distribución con mayor varianza que el otro. Parece lógico querer elegir, de entre todos los estimadores insesgados, aquel con menor varianza (ie. el azul).

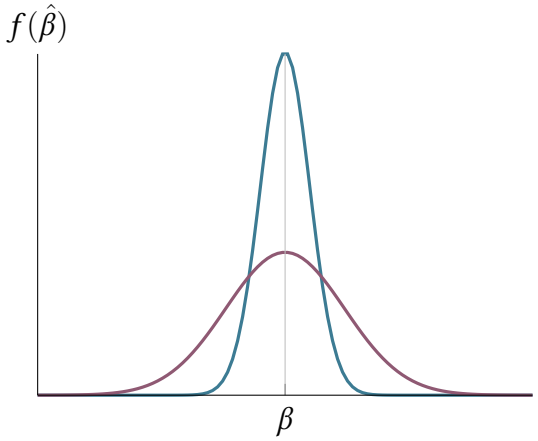


Figura 2.9: Densidades de estimadores insesgados con distinta varianza

Para poder calcular la varianza de los estimadores MCO es necesario[8] agregar un último supuesto:

RLS.5: Homocedasticidad Asumimos que el error μ tiene la misma varianza condicional para todos los valores de la variable independiente. Es decir,

$$\text{Var}(\mu|x) = \sigma^2. \tag{2.14}$$

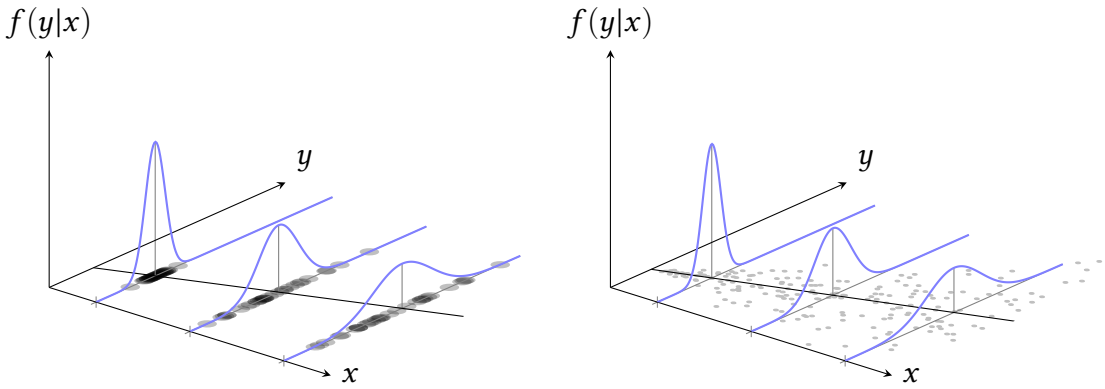
Para entender bien este supuesto es conveniente

Cuando $\text{Var}(\mu|x)$ depende de x (cambia con x) decimos que el término de error presenta **heterocedasticidad**, o varianza no constante. La heterocedasticidad es un problema que es más difícil de pronunciar que de entender: ocurre cuando la dispersión de una variable dependiente no es constante para distintos valores de la variable independiente.

La Figura 2.10 muestra datos heterocedásticos para el caso de una regresión bivariada, tanto para una variable dependiente discreta como continua. En el eje $f(y|x)$ se grafica la densidad del término de error, la que en este ejemplo claramente claramente disminuye a medida que x es mayor.

Podemos generar fácilmente un conjunto de datos que presenten heterocedasticidad:

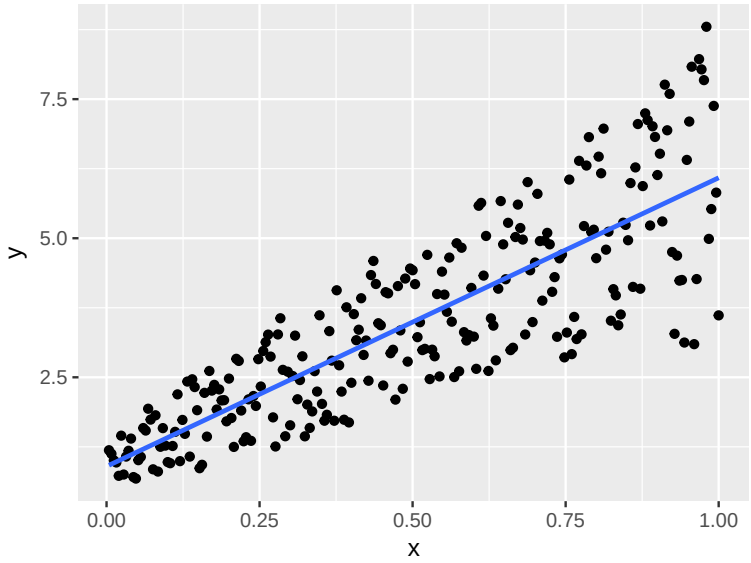
[8] En realidad no es *estrictamente* necesario, pero simplifica mucho los cálculos.



(a) Variable dependiente discreta (b) Variable dependiente continua

Figura 2.10: Heterocedasticidad

```r
n <- 250                              # Número de observaciones
x <- (1:n)/n                          # Generamos 'x' con valores entre 0 y 1
u <- rnorm(n, sd=1)                   # Valores aleatorios de una normal
i <- order(runif(n, max=dnorm(u)))    # Crear índice ordenado para el error
y <- 1 + 5 * x + u[rev(i)]            # Generamos 'y' con error ordenado por índice
data <- as_tibble(x, y)               # Crear marco de datos con variables
ggplot(data, aes(x, y)) +             # Graficar datos y regresión lineal
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```



Usando los supuestos SLR.1 a SLR.5 podemos demostrar que

$$\text{Var}(\hat{\beta}_0) = \frac{\sigma^2/n \sum_{i=1}^{n} x_i^2}{\sum_{i=1}^{n}(x_i - \overline{x})^2} \tag{2.15}$$

$$\text{Var}(\hat{\beta}_1) = \frac{\sigma^2}{\sum_{i=1}^{n}(x_i - \overline{x})^2}. \tag{2.16}$$

Usualmente nos interesará $\text{Var}(\hat{\beta}_1)$, ya que es la varianza del parámetro que acompaña a la variable independiente. Vemos que esta varianza depende positivamente de la varianza del error σ^2. Esto tiene sentido, ya que a mayor varianza de los factores no observables que afectan y, es más difícil estimar con precisión β_1. Por otro lado, $\text{Var}(\hat{\beta}_1)$ depende negativamente de la variación total de x, representada por $SST_x = \sum_{i=1}^{n}(x_i - \overline{x})^2$. Esto es así porque entre más dispersos estén los valores de la variable independiente, más fácil será detectar la relación entre $\text{E}(y|x)$ y x. Además, a medida que el tamaño muestral aumente, también lo hará la variación total en x_i, de forma que una muestra más grande siempre resultará en una menor varianza de $\hat{\beta}_1$.

En presencia de heterocedasticidad una estimación por MCO seguirá entregando coeficientes consistentes e insesgados. Sin embargo, no podremos estimar correctamente la matriz de varianzas-covarianzas, lo que producirá que los errores estándar de los coeficientes estén sesgados. Esto conduce a errores en tests de inferencia, como (por ejemplo) al determinar si un coeficiente es significativo.

2.6.1 Estimando la varianza del error

Usando los supuesos de regresión lineal simple obtenemos la varianza de $\hat{\beta}_0$ y $\hat{\beta}_1$ indicadas en [2.15] y [2.16]. El problema es que estas ecuaciones contienen requieren que conozcamos la varianza del error σ^2, la que casi siempre es desconocida. Entonces nuestro objetivo ahora es usar nuestros datos para estimar σ^2, y así poder estimar la varianza de los parámetros.

Usando los supuestos RLS.1 a RLS.5 es posible demostrar que

$$\mathrm{E}(\hat{\sigma}^2) = \sigma^2. \tag{2.17}$$

Entonces podríamos definir definir el estimador $1/n \sum_{i=1}^{n} \hat{\mu}_i^2$ para . Lamentablemente este estimador es sesgado, ya que no toma en cuenta dos restricciones que deben satisfacer los residuos de MCO:

$$\sum_{i=1}^{n} \hat{\mu}_i = 0$$

$$\sum_{i=1}^{n} x_i \hat{\mu}_i = 0$$

Estas restricciones vienen de las condiciones de primer orden del problema de minimización de MCO planteado en [2.6]. Una manera de entender cómo influyen estas restricciones al estimador planteado recién es pensando que si conociéramos $n - 2$ residuos, siempre podríamos obtener los últimos 2 usando estas restricciones (piensa por qué). Esto implica que si bien tenemos n grados de libertad para los errores, existen $n - 2$ grados de libertad para los residuos de MCO. Entonces el estimador insesgado de σ^2 corrige el estimador propuesto recién por este cambio en grados de libertad:

$$\hat{\sigma}^2 = \frac{1}{n-2} \sum_{i=1}^{n} \hat{\mu}_i^2. \tag{2.18}$$

Este estimador a veces es denotado por S^2. Una vez que obtenemos $\hat{\sigma}^2$ podemos usarlo en las ecuaciones [2.15] y [2.16] para calcular $\mathrm{Var}(\hat{\beta}_0)$ y $\mathrm{Var}(\hat{\beta}_1)$.

2.6.2 Error estándar (de los estimadores)

Parece natural pensar que un estimador para la desviación estándar del error es

$$\hat{\sigma} = \sqrt{\hat{\sigma}^2}. \tag{2.19}$$

A este término se le conoce como el **error estándar de la regresión**. Lamentablemente $\hat{\sigma}$ no es un estimador insesgado de σ, pero es consistente: a medida que la cantidad de datos aumenta, el estimador converge en probabilidad al verdadero parámetro σ. Esta situación está representada en la Figura 2.11.

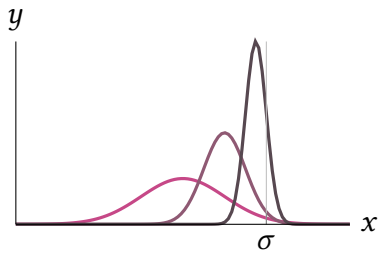


Figura 2.11: Estimador sesgado consistente

La estimación $\hat{\sigma}$ es interesante en sí misma porque representa la desviación estándar de y una vez que hemos eliminado los efectos de x. Sin embargo, el principal uso de $\hat{\sigma}$ es para estimar la desviación estándar de $\hat{\beta}_1$. Dado que $\text{sd}(\hat{\beta}_1) = \sigma/\sqrt{SST_x}$, el estimador natural para esta desviación estándar es

$$\text{se}(\hat{\beta}_1) = \frac{\hat{\sigma}}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}}. \qquad [2.20]$$

A este término se le conoce como el **error estándar** de $\hat{\beta}_1$. El error estándar de un estimador nos dá una idea de cuán dispersa es nuestra estimación. Debido a esto, el error estándar juega un rol importante en econometría, y lo usaremos para construir tests estadísticos e intervalos de confianza para distintos métodos que veremos más adelante.

2.6.3 Extraer estadísticos de una regresión

Para guardar el modelo asignamos `lm()` a un objeto:

```
modelo <- lm(wage ~ educ, data = ingresos)
```

El objeto `modelo` no solo incluye los coeficientes estimados, si no que una serie de otros elementos que son muy útiles —veremos ahora por qué. En estricto rigor `modelo` es lo que R llama una lista, y contiene una serie de otros objetos. Podemos imprimir los nombres de estos objetos escribiendo

```
names(modelo)
```

```
#>  [1] "coefficients"  "residuals"     "effects"       "rank"
#>  [5] "fitted.values" "assign"        "qr"            "df.residual"
#>  [9] "xlevels"       "call"          "terms"         "model"
```

Para acceder a los objetos guardados usamos la notación de `$`, como es usual en R. Por ejemplo, para imprimir los coeficientes del modelo escribimos

```
modelo$coefficients
```

```
#> (Intercept)        educ
#>   146.95244    60.21428
```

También existen funciones para acceder directamente a los objetos. Por ejemplo,

```
coefficients(modelo)
```

```
#> (Intercept)        educ
#>   146.95244    60.21428
```

En cualquier caso, el resultado es un vector cuyos elementos tienen nombres: el intercepto (β_0) es `(Intercept)` y el nombre del primer coeficiente (β_1) es el nombre de la variable x, es decir, `educ`. Usando esto podemos acceder a elementos específicos del vector usando sus nombres o sus índices, como se mencionó en [ref]:

```
coef(modelo)[1]
```

```
#> (Intercept)
#>    146.9524
```

```
coef(modelo)["educ"]
```

```
#>     educ
#> 60.21428
```

2.6.4 Predicciones y residuos

La predicción de un modelo, a veces llamada valores ajustados, es simplemente el vector $\hat{y}_i$. Teniendo los valores ajustados podemos calcular también los residuos del modelo, μ_i:

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i \quad [2.21]$$

$$\hat{\mu}_i = y_i - \hat{y}_i \quad [2.22]$$

Podemos calcular fácilmente ambos vectores con los elementos que tenemos:

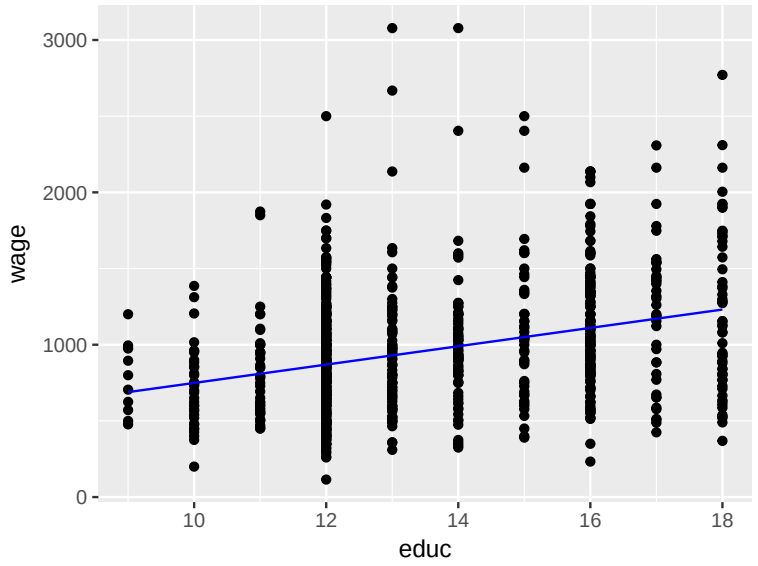
```
ingresos$prediccion <- coef(modelo)[1] + coef(modelo)[2] * ingresos$educ
ingresos$residuos <- ingresos$wage - ingresos$prediccion
```

Alternativamente, podemos calcular los valores predichos y los residuos del modelo usando funciones específicas para ello:

```
modelo.sim <- lm(y ~ x, data = simdatos)
simdatos$prediccion <- fitted(modelo.sim)
simdatos$residuos <- resid(modelo.sim)
```

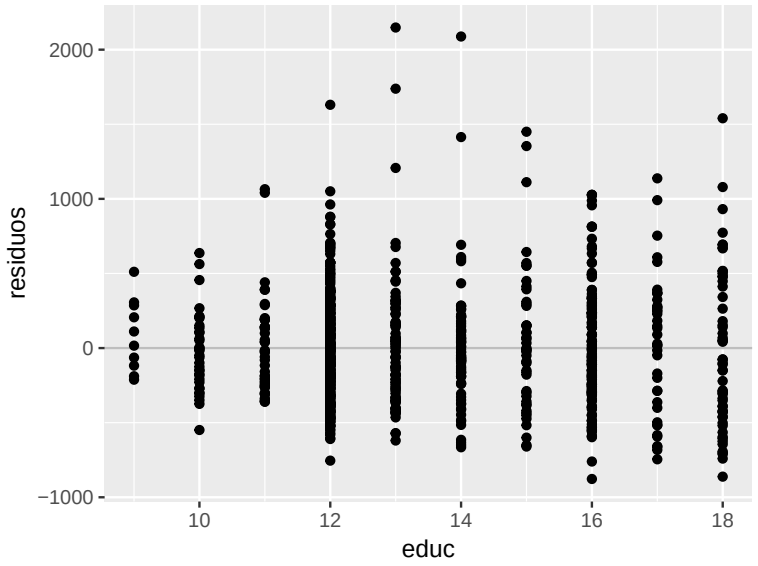
Podemos graficar nuestra predicción sobre los datos para inspeccionar visualmente cómo se ve. Para esto volvemos a usar **ggplot2**:

```
ggplot(ingresos, aes(educ, wage)) +
  geom_point() +
  geom_line(aes(y = prediccion), color = "blue")
```

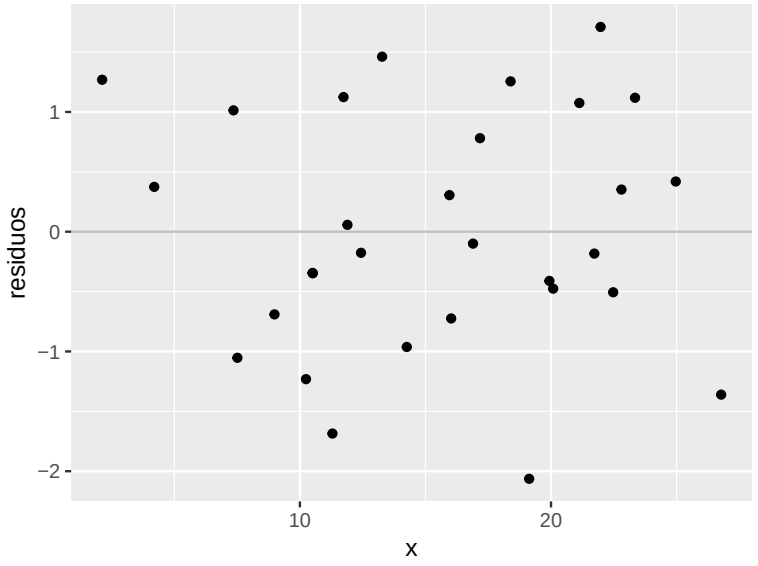


También resulta interesante analizar los residuos, que son "la otra cara de la moneda". Si bien los valores predichos nos muestran el patrón que el modelo ha sido capaz de capturar, los residuos nos mostrarán lo que el modelo ha omitido. Recordar que los residuos son simplemente la diferencia entre el valor predicho y el real, como se indica en [2.5]. Graficamos los residuos escribiendo

```
ggplot(ingresos, aes(educ, residuos)) +
  geom_hline(yintercept = 0, color = "gray") +
  geom_point()
```




```
ggplot(simdatos, aes(x, residuos)) +
  geom_hline(yintercept = 0, color = "gray") +
  geom_point()
```



Además de estos elementos directamente accesibles guardados en `modelo`, existen una serie de funciones que podemos aplicar al objeto para realizar otros cálculos o resumir información útil. Por ejemplo, es muy común usar `summary()` para obtener un resumen de información importante del modelo estimado:

```
summary(modelo)

#>
#> Call:
#> lm(formula = wage ~ educ, data = ingresos)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -877.38 -268.63  -38.38  207.05 2148.26
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)  146.952     77.715   1.891   0.0589 .
#> educ          60.214      5.695  10.573   <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 382.3 on 933 degrees of freedom
#> Multiple R-squared:  0.107,Adjusted R-squared:  0.106
#> F-statistic: 111.8 on 1 and 933 DF,  p-value: < 2.2e-16
```

Problemas

Ejercicio 1.
Crea un estimador que sea insesgado pero no sea de mínima varianza. Usa datos simulados para contrastarlo con el estimador MCO.

Ejercicio 2.
The second exercise in chapter one.

Bibliografía

Angrist, Joshua D. y Jörn-Steffen Pischke (2009). *Mostly Harmless Econometrics: An Empiricist's Companion*. 1 edition. Princeton: Princeton University Press. 392 págs.

– (2017). "Undergraduate Econometrics Instruction: Through Our Classes, Darkly". En: *Journal of Economic Perspectives* 31.2, págs. 125-144.

Blackburn, McKinley y David Neumark (1992). "Unobserved Ability, Efficiency Wages, and Interindustry Wage Differentials". En: *The Quarterly Journal of Economics* 107.4, págs. 1421-1436.

Galton, Francis (1886). "Regression Towards Mediocrity in Hereditary Stature". En: *The Journal of the Anthropological Institute of Great Britain and Ireland* 15, págs. 246-263.

Koenker, Roger (1981). "A note on studentizing a test for heteroscedasticity". En: *Journal of Econometrics* 17.1, págs. 107-112.

Long, J. Scott y Laurie H. Ervin (2000). "Using Heteroscedasticity Consistent Standard Errors in the Linear Regression Model". En: *The American Statistician* 54.3, págs. 217-224.

Stigler, Stephen M. (1986). *The History of Statistics: The Measurement of Uncertainty Before 1900*. Google-Books-ID: M7yvkERHIIMC. Harvard University Press. 436 págs.

Wooldridge, Jeffrey M. (2016). *Introductory econometrics: a modern approach*. 6th ed. OCLC: 935880069. Boston, MA: Cengage Learning. 789 págs.

Índice alfabético

comentarios, 8
Consola, 6

dplyr, 9

error estándar, 47
error estándar de la regresión, 47
estimador insesgado, 34

funciones, 8
funciones geom, 17

ggplot2, 9, 16, 17, 49

haven, 4, 11
heterocedasticidad, 43
homocedasticidad, 43

línea de regresión, 18

magrittr, 28
modelo de regresión lineal simple, 14
modelo muestral, 15

objeto, 7
operador de asignación, 7
operador de pipa, 28

parámetros, 14
purrr, 9

R base, 9
residuo, 19

script, 6

tidyr, 9
tidyverse, 4, 9, 11
término de error, 13, 14

variable dependiente, 14
variable independiente, 14