# Análisis Estadístico con R

true

12 de abril de 2021

## Contents



Tomado de Kross et al. (2020)

## Simulación

Una de las grandes ventajas de utilizar un lenguaje de programación estadística como R es su vasta colección de herramientas para la simulación de números aleatorios.

Esta sesión asume la familiaridad con algunas distribuciones de probabilidad comunes, pero estos temas solo se discuten con respecto a la generación de números aleatorios. Incluso si no tienes experiencia previa con estos conceptos, deberías ser capaz de completar la sesión y entender las ideas principales.

- La primera función que usaremos para generar números aleatorios es `sample()`. Usa `?sample` para ver la ayuda.
- Vamos a simular el lanzamiento de cuatro dados de seis caras: `sample(1:6, 4, replace = TRUE)`
- Ahora repita el comando para ver cómo tu resultado difiere. (La probabilidad de obtener el mismo resultado exacto es $(\frac{1}{6})^4 = 0.00077$, que es bastante pequeña!)

  `+ `sample(1:6, 4, replace = TRUE)` da instrucciones a R para seleccionar cuatro número al azar entr`
- Ahora, obtén una muestra de 10 números entre 1 y 20, SIN REEMPLAZO. Para hacerlo, simplementre elimina el argumento `replace`.

  `+ Dado el último comando muestrea sin reemplazo, ningún número aparece más de una vez en la salida.`
- `LETTERS` es una variable predefinida en `R` que contiene un vector de las 26 letras del alfabeto en Inglés. Imprímela.
- La función de `sample()` también se puede utilizar para permutar, o reorganizar, los elementos de un vector. Por ejemplo, ejecuta `sample(LETTERS)` para permutar las 26 letras del alfabeto.

  `+ Esto es idéntico a tomar una muestra de tamaño 26 de `LETTERS`, sin reemplazo. Cuando no se espec`

Hasta el momento, el código sería:

```
? sample
sample(1:6, 4, replace = TRUE)
sample(1:20, 10)
LETTERS
sample(LETTERS)
```

- Ahora, supongamos que queremos simular 100 lanzamientos de una moneda de dos caras *injusta*. Esta moneda particular tiene un 0.3 de probabilidad de salir `"sello"` y un 0,7 probabilidad de salir `'cara'`.
- Sea 0 el valor que representa *sello* y 1 el valor que representa *cara*. Utilice `sample()` para extraer una muestra de tamaño 100 a partir del vector `c(0,1)`, con el reemplazo. Ya que la moneda es *injusta*, debemos dar probabilidades específicas a los valores 0 (sello) y 1 (cara) con un cuarto argumento, `prob = c(0.3, 0.7)`. Asigna el resultado de una nueva variable llamada `flips`.
- Imprime los valores de `flips`
- Ya que fijamos la probabilidad de que salga cara sea 0.7, es de esperar que aproximadamente 70 de las monedas salga 1. Cuenta el número real de 1s contenido en lanzamientos utilizando la función `sum()`.
- El lanzamiento de una moneda es un resultado binario (0 ó 1) y estamos haciendo 100 ensayos independientes (lanzamientos), por lo que puede utilizar 'rbinom() para simular una variable aleatoria binomial. Mira la ayuda de esa función.

El código sería:

```
flips <- sample(c(0,1), 100, replace = TRUE, prob = c(0.3, 0.7))
sum(flips)
```

Cada distribución de probabilidad en R tiene una función`r***` (`r` de *random/aleatorio*), una función `d***` (*densidad*), una función `p***` (*probabilidad*), y una función `q***` (*cuantil*). Por ahora estamos interesados en las funciones `r***`, pero exploren los demás por su cuenta.

- Una variable aleatoria binomial representa el número de "éxitos" (cara) en un número determinado de *ensayos* independientes (lanzamientos). Por lo tanto, podemos generar una sola variable aleatoria que representa el número de cabezas en 100 lanzamientos de nuestra moneda *injusta* usando *rbinom (1, size = 100, prob = 0.7)*. Tenga en cuenta que sólo se especifica la probabilidad de *éxito* y no la probabilidad de *fracaso*. Inténtalo ahora.
- De manera equivalente, si queremos ver todos los 0s y 1s, podemos pedir 100 observaciones, cada una de tamaño 1, con probabilidad de éxito de 0.7. Inténtalo, asignando el resultado a una variable llamada nueva `flips2`. Es decir:

```
flips2 <- rbinom (n = 100, size = 1, prob = 0.7)
```

- Imprime los resultados de `flips2`
- Ahora suma los resultados. El resultado debería ser al rededor de 70.
- Similar a `rbinom()`, podemos utilizar `R` para simular números aleatorios de muchas otras distribuciones de probabilidad. Mira la ayuda de `rnorm()`.
- La distribución normal estándar tiene media 0 y desviación estándar 1. Como se puede ver en la sección *Use* en la ayuda, los valores por defecto para los argumentos `mean` y `sd` son 0 y 1, respectivamente. Por lo tanto, `rnorm(10)` va a generar 10 números aleatorios de una distribución normal estándar. Inténtalo.
- Ahora, vuelve a hacer lo mismo, pero con media 100 y desviación estándar 25.

Por último, ¿qué pasa si queremos simular 100 *grupos* de números al azar, cada uno con 5 valores generados a partir de una distribución de Poisson con media 10? Empecemos un grupo de 5 números, entonces y luego veremos cómo repetir la operación de 100 veces de una manera conveniente y compacta.

- Genera 5 valores aleatorios de una distribución de Poisson con media 10. Echa un vistazo a la ayuda si necesitas ayuda.
- Ahora usa `replicate(100, rpois(5, 10))` para realizar esta operación 100 veces. Almacena el resultado en una nueva variable llamada `my_pois`.
- Iprime el resultado. Luego, realiza un histograma de la variable

- `replicate()` crea una matriz, cada columna contiene 5 números aleatorios generados a partir de una distribución de Poisson con media 10. Ahora podemos encontrar la media de cada columna en `my_pois` usando la función `colMeans()`. Almacena el resultado en una variable llamada `cm`.
- Miremos la distribución de las medias por columna con un histograma.

  `+ Parece que nuestros valores se distribuyen casi normalmente, ¿verdad? Ese es el teorema del límit`
- Todas las distribuciones de probabilidad estándar están incorporados en R, incluyendo exponencial (`rexp()`), Chi-cuadrado (`rchisq()`), gamma (`rgamma()`), etc.

La simulación es prácticamente un campo propio y sólo hemos rozado la superficie de lo que es posible. Los animo a explorar estas y otras funciones más por su cuenta.

# Referencias

Kross, Sean, Nick Carchedi, Bill Bauer, and Gina Grdina. 2020. *Swirl: Learn r, in r*. https://CRAN.R-project.org/package=swirl.