Capítulo 16: Números aleatorios y distribuciones de probabilidad

En R podemos trabajar con las distribuciones de probabilidad usuales, ya sean continuas o discretas. Para ello, podemos recurrir a las siguientes 4 funciones específicas de R :

Función	Significado	Significado
p	probability	Calcula probabilidades acumuladas (f.p.a)
q	quantile	Calcula cuantiles (percentiles)
		**Sólo uso gráfico en el caso continuo*
d	density	Calcula probabilidades puntuales (f.m.p/f.d.p)
r	random	Genera datos aleatorios según una distribución específica¿

Todas las funciones son muy útiles para el análisis estadístico y resolución de problemas, pero para el caso de este curso, nos centraremos en **r** de random para generar números aleatorios.

En R, generar números aleatorios es una tarea común que se puede realizar de diversas formas utilizando funciones predefinidas. Vamos a listar todas las funciones que hay:

1. sample(). Muestreo aleatorio.
2. runif(). Números aleatorios de una distribución uniforme.
3. rbinom(). Números aleatorios de una distribución binomial.
4. rpois(). Números aleatorios de una distribución Poisson.
5. rexp(). Números aleatorios de una distribución exponencial.
6. rhyper(). Números aleatorios de una distribución hipergeométrica.
7. rnbinom(). Números aleatorios de una distribución binomial negativa.
8. rnorm(). Números aleatorios de una distribución normal.
9. rgamma(). Números aleatorios de una distribución gamma.
10. rchisq(). Números aleatorios de una distribución chi-cuadrada.
11. rbeta(). Números aleatorios de una distribución beta.
12. rt(). Números aleatorios de una distribución t de Student.
13. rf(). Números aleatorios de una distribución F de Fisher.
14. rlogis(). Números aleatorios de una distribución logística.
15. rcauchy(). Números aleatorios de una distribución de Cauchy.
16. rlaplace(). Números aleatorios de una distribución Laplace (requiere la librería extraDistr).
17. rinvgauss(). Números aleatorios de una distribución inversa gaussiana (requiere la librería statmod).
18. rvonmises(). Números aleatorios de una distribución de Von Mises (requiere la librería CircStats o circular).
19. rlgamma(). Números aleatorios de una distribución log-gamma (requiere extraDistr).
20. rpareto(). Números aleatorios de una distribución Pareto (requiere extraDistr).

Para proposito de este curso, nada más se explicaran 5 formas de generar números aleatorios:

1. runif(n, min, max): Números uniformes entre min y max.
2. rnorm(n, mean, sd): Números normales con media mean y desviación estándar sd.
3. sample(x, size): Muestra de tamaño size de los elementos de x.
4. rbinom(n, size, prob): Números binomiales con size ensayos y probabilidad de éxito prob.
5. rexp(n, rate): Números exponenciales con tasa(rate) λ .

1. Generar números aleatorios uniformes

Para generar números aleatorios con una distribución uniforme, se usa la función `runif()`. Véase el siguiente ejemplo:

Genere 5 números aleatorios entre el 0 y 1:

```
num_al1 <- runif(5)
num_al1
```

```
## [1] 0.14762021 0.47743809 0.77219369 0.06419312 0.04860934
```

Note como la función `runif()` por defecto me da números (en forma de vector) entre 0 y 1, pero al consultar con el manual interno de R usando la función `help()` me da que tiene más parametros que me van a ayudar a cambiar los límites de mis números aleatorios. Véase el siguiente ejemplo:

Genere 5 números aleatorios entre el 10 y 20

```
num_al2 <- runif(5, min = 10, max = 20)
num_al2
```

```
## [1] 18.85970 16.08917 16.41123 16.48099 14.44914
```

Pero en la resolución de un problema más complejo puede que queramos verificar que nuestra respuesta es correcta, pero como son números aleatorios, cada que se corre un codigo para generar números aleatorios la respuesta casi siempre es diferente, entonces para ese tipo de situaciones, y otras más que puedan surgir, existe la función `set.seed()`. Esta función nos va a que siempre se creen los mismos números aleatorios cada que corramos un codigo que genere números aleatorios. Véase el siguiente ejemplo:

```
set.seed(123)
num_al3 <- runif(5, min = 10, max = 20)
num_al3
```

```
## [1] 12.87578 17.88305 14.08977 18.83017 19.40467
```

En su uso más básico, la función `set.seed()` recibe un parámetro numérico llamado semilla que hace que las funciones sean repetibles. No importa cuándo o dónde se corra la función pseudo-aleatoria, si tienen la misma semilla antes, obtendremos el mismo resultado. En el contexto de generación de números aleatorios, "semilla" (o seed en inglés) hace referencia a un valor inicial que sirve como punto de partida para el algoritmo que genera los números pseudoaleatorios.

*Nota: puede que lleguen a salir diferentes numeros y es normal, el problema puede deberse a:

- Versión de R: Si las versiones de R son diferentes, es posible que los algoritmos subyacentes para generar números aleatorios también lo sean, lo que podría producir resultados distintos.
- Paquetes específicos: Si estás utilizando funciones de generación de números aleatorios de un paquete externo, la versión del paquete también debe coincidir.
- Plataforma: Aunque poco común, en algunos casos extremos (como diferencias entre sistemas operativos), puede haber pequeñas discrepancias si los métodos subyacentes difieren.

Ahora, si quisieramos convertirlo a una función discreta o para redondear cualquier tipo de números usamos `round()`.

```
set.seed(123)
(round(num_al3))
```

```
## [1] 13 18 14 19 19
```

2. Generar números aleatorios normales

Para generar números aleatorios con una distribución normal, se usa la función `rnorm()`. Ejemplo:

Generar 5 números aleatorios con media 0 y desviación estándar 1:

```
set.seed(123)
(random_normal <- rnorm(5))
```

```
## [1] -0.56047565 -0.23017749  1.55870831  0.07050839  0.12928774
```

Si revisamos la media aritmética, con la función `mean()`, y desviación estándar, con la función `sd()`, de este experimento podemos confirmar nuestros argumentos iniciales (con cierto margen de error por el tamaño de la muestra).

```
vector_normal_estandar <- rnorm(300)
# media cercana a cero
mean(vector_normal_estandar)
```

```
## [1] 0.01822809
```

```
#desviación estándar cercana a 1
sd(vector_normal_estandar)
```

```
## [1] 0.947009
```

La media y la desviación estándar usan las siguientes formulas:

$$\bar{x} = \frac{1}{n}\sum_{i=0}^{n} x_i \qquad s = \sqrt{\frac{1}{n-1}[\sum_{i=0}^{n}(x_i - \bar{x})^2]}$$

Media Desviación estándar

Podemos hacer lo mismo escogiendo la media y la desviación estándar:

Generar 5 números aleatorios con media 10 y desviación estándar 2:

```
num_al4 <- rnorm(5, mean = 10, sd = 2)
num_al4
```

```
## [1] 10.662358  5.971579 10.423961 12.473350 14.075148
```

3. Generar número aleatorios

Para generar números enteros aleatorios sin ninguna distribución en especifico se usa la función `sample()` para realiza un muestreo aleatorio (con o sin reemplazo) de un conjunto de valores proporcionado por el usuario. Ejemplo:

Generar 5 números enteros aleatorios entre 1 y 100:

```
set.seed(123)
num_al5 <- sample(1:100, 5)
num_al5
```

```
## [1] 31 79 51 14 67
```

4. Generar números aleatorios binomiales

Para generar números aleatorios con una distribución binomial, se usa la función `rbinom()`.Ejemplo:

Generar 5 números aleatorios con 10 ensayos y probabilidad de éxito 0.5:

```
set.seed(123)
num_al6<- rbinom(5, size = 10, prob = 0.5)
num_al6
```

```
## [1] 4 6 5 7 7
```

Veamos una simulación más compleja. Un equipo de futból, con 11 jugadores, sabe que la probabilidad de que uno de ellos se lesione cada partido es de 5%. Podemos simular el número de lesiones en el calendario de 10 partidos con la función `rbinom()`.

```
# rbinom(número de partidos, número de jugadores, probabilidad de lesión)
set.seed(46)
rbinom(10, 11, 0.05)
```

```
##  [1] 0 0 1 0 0 1 1 1 1 2
```

En este caso vemos que se lesionó 1 en el tercer partido, 1 en el sexto, etc. Si queremos obtener el total de lesionados podemos utilizar operaciones de vectores.

```
set.seed(46)
sum(rbinom(10, 11, 0.05))
```

```
## [1] 7
```

5. Generar números aleatorios exponenciales

Para generar números aleatorios con una distribución exponencial, se usa la función `rexp()`. Ejemplo:

Generar 5 números aleatorios con tasa (rate) 1:

```
set.seed(123)
num_al6 <- rexp(5, rate = 1)
num_al6
```

```
## [1] 0.84345726 0.57661027 1.32905487 0.03157736 0.05621098
```

Para el proposito de este curso solo se utilizó la función **r** seguido de la abreviación de la distriobución para R, pero para ejercicios más avanzados se van a utilizar las otras funciones (p,q,d), por ejemplo:

Ejemplo:

Consideremos una variable aleatoria X con distribución normal, media igual a 50 y varianza igual a 25.

1. Calcular la probabilidad de que X sea menor o igual a 48. Es decir, $\mathbb{P}(X \leq 48)$.

```
pnorm(48, mean = 50, sd = sqrt(25))
```

```
## [1] 0.3445783
```

Notemos que necesitamos la desviación típica en vez de la varianza.

2. Calcular la probabilidad de que X sea mayor a 48. Esto es $\mathbb{P}(X > 48)$.

```
# Note el tercer argumento.
pnorm(48, mean = 50, sd = sqrt(25), lower.tail = FALSE)
```

```
## [1] 0.6554217
```

3. Calcular la probabilidad de que X sea mayor o igual a 45 y menor que 55; es decir $\mathbb{P}(45 \leq X < 55)$.

Notemos que: $\mathbb{P}(45 \leq X < 55) = \mathbb{P}(X < 55) - \mathbb{P}(X \leq 45)$

```
pnorm(55, 50, sqrt(25)) - pnorm(45, 50, sqrt(25))
```

```
## [1] 0.6826895
```

4. Cuál es el valor de X que deja a un 90% bajo él? $\mathbb{P}(X \leq x_0) = 0.90$.

```
qnorm(0.90, mean = 50, sd = sqrt(25))
```

```
## [1] 56.40776
```

5. Generemos un conjunto de 10 datos que sigan una distribución normal de media 50 y varianza 25:

```
set.seed(123) #Utilice set.seed para comparar los resultados
rnorm(10, mean = 50, sd = sqrt(25))
```

```
##  [1] 47.19762 48.84911 57.79354 50.35254 50.64644 58.57532 52.30458 43.67469
##  [9] 46.56574 47.77169
```

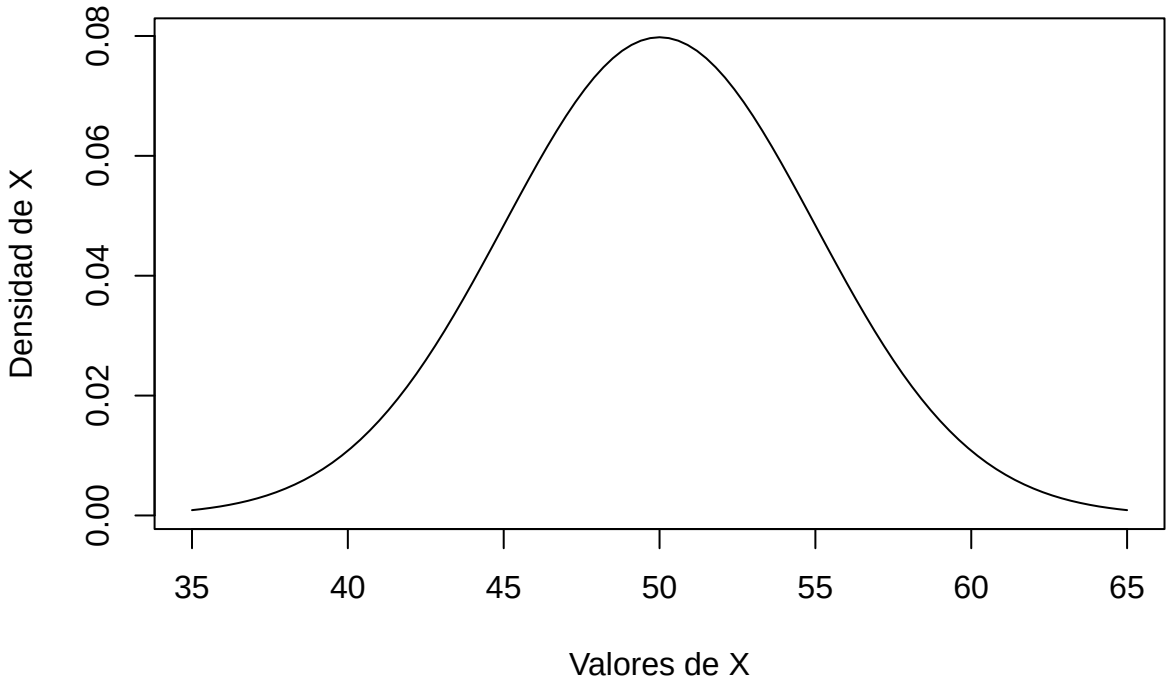
6. La función d, en este caso dnorm() no tiene mucho uso cuando nos referimos a distribuciones continuas. Es más conveniente usar la función p en estos casos. Por ejemplo, para calcular la probabilidad de que X se encuentre entre 35 y 55 usamos el código:

```
pnorm(55, mean = 50, sd = sqrt(25)) - pnorm(35, mean = 50, sd = sqrt(25))
```

```
## [1] 0.8399948
```

7. Sin embargo, podemos usar dnorm() para construir el gráfico de la distribución de probabilidad de X con la ayuda del comando curve():

```
curve(dnorm(x, mean = 50, sd = sqrt(25)), xlim = c(35,65), xlab = "Valores de X", ylab = "Densidad de X")
```



Notar como curve es una manera de graficar, pero R ofrece otras herramientas que veremos más adelante.

Estos problemas, aunque puedan parecer avanzados, son únicamente ilustrativos y están diseñados para mostrar la capacidad de este lenguaje en la resolución eficiente de este tipo de cuestiones, sin necesidad de recurrir a tablas de probabilidad.

Ejercicio con probabiliad:

Los siguientes ejercicios muestran la importancia de R para simular eventos con probabilidad y así responder a preguntas con un análisis estadístico más sofisticado. Sin embargo, a la hora de leer el problema vamos a ver que se necesitan entender funciónes que en este curso no se ven, pero si en cursos de R enfocados a la probabilidad, por lo que no es necesario que respondan a los ejercicios, pero si ver la solución.

Nota: el "Curso Intermedio de R con Probabilidad y Estadística" son los lunes y miércoles de 14:00-16:00h, para iniciar el miércoles 5 de febrero en el salón PF101 (Periférico). El curso será de 4 sesiones. Los cursos son abiertos a toda persona, estudiante o docente. No hay necesidad de registrarse. No hay prerrequisitos.

Ejercicio 1: Suponga que se lanzan repetidamente dos dados y se cuenta la suma de las caras hacia arriba en cada lanzamiento. Determine la probabilidad de los siguientes eventos:

a) La suma 3 sale antes que la suma 7.

b) La suma 4 sale antes que la suma 7.

```
#======== Problema de Dados ============================
cat("\n>>> Problema Gana la suma 3(4) sobre la suma 7 (Wackerly et al., 2008).\n")
```

```
##
## >>> Problema Gana la suma 3(4) sobre la suma 7 (Wackerly et al., 2008).
```

```
cat("[Probabilidad de ganar]\n")
```

```
## [Probabilidad de ganar]
```

```
#============================
N <- 20000
K <- 4
output <- rep(NA,N)
n <- rep(1,N)
extract <- function() sample(seq(2,12),1,prob=c(1,2,3,4,5,6,5,4,3,2,1)/36)
tab <- rep(0,12)
tab[7] <- -1
tab[K] <- +1
for(i in seq(N)) {
k <- 1
fin <- FALSE
while(!fin) {
k <- k+1
out <- tab[extract()]
fin <- ifelse(out==0,FALSE,TRUE)
}
output[i] <- out
n[i] <- k
}
cat("\nNúmero de juegos simulados =",N,"\n")
```

```
##
## Número de juegos simulados = 20000
```

```
cat("Prob{ Sale primero el",K,"que el 7 } =",sum(output>0)/N,"\n")
```

```
## Prob{ Sale primero el 4 que el 7 } = 0.32825
```

```
cat("Prob{ Sale primero el 7 que el",K,"} =",sum(output<0)/N,"\n")
```

```
## Prob{ Sale primero el 7 que el 4 } = 0.67175
```

```
cat("Número promedio de extracciones=",nbar <- sum(n)/N,"\n")
```

```
## Número promedio de extracciones= 4.9863
```

```
cat("Desviación estándar de extracciones=",round(sqrt(sum(n^2)/N-nbar^2),4),"\n")
```

```
## Desviación estándar de extracciones= 3.4733
```

```
#rm(list=ls())
```

Ejercicio 2:

Descripción del juego Craps. Un jugador lanza un par de dados y anota la suma de puntos de las caras hacia arriba. Si en el primer lanzamiento el jugar suma un 7 o un 11, el jugador gana el juego. Si, en el primer lanzamiento el jugador obtiene una suma de 2, 3 o 12, el jugador pierde. Si el jugador lanza cualquier otra suma (4,5,6,8,9 ó 10), la suma se convierte en comod´ın. Ahora bien, si el jugador no gana ni pierde en el primer lanzamiento, entonces lanza repetidamente el par de dados hasta que sale el comodín ó sale el 7. El jugador gana si sale su comod´ın antes que el 7. ¿Cuál es la probabilidad de que un jugador gane el juego Craps?

```
N <- 20000
output = rep(NA,N)
n <- rep(1,N)
extract <- function() sample(seq(2,12),1,prob=c(1,2,3,4,5,6,5,4,3,2,1)/36)
for(i in seq(N)) {
comodin <- extract()
if(any(comodin==c(7,11))) {
output[i] <- +1
} else if(any(comodin==c(2,3,12))) {
output[i] <- -1
} else {
k <- 1
fin <- FALSE
tab <- rep(0,12)
tab[7] <- -1
tab[comodin] <- +1
while(!fin) {
k <- k+1
out <- tab[extract()]
fin <- ifelse(out==0,FALSE,TRUE)
}
output[i] <- out
n[i] <- k
}
}
#print(output)
#print(n)
cat("\nNúmero de juegos simulados =",N,"\n")
```

```
##
## Número de juegos simulados = 20000
```

```
cat("Prob{ ganar} =",sum(output>0)/N,"\n")
```

```
## Prob{ ganar} = 0.48935
```

```
cat("Prob{perder} =",sum(output<0)/N,"\n")
```

```
## Prob{perder} = 0.51065
```

```
cat("Número promedio de extracciones=",nbar <- sum(n)/N,"\n")
```

```
## Número promedio de extracciones= 3.4066
```

```r
cat("Desviación estándar de extracciones=",round(sqrt(sum(n^2)/N-nbar^2),4),"\n")
```

```
## Desviación estándar de extracciones= 3.0486
```

```r
#rm(list=ls())
```