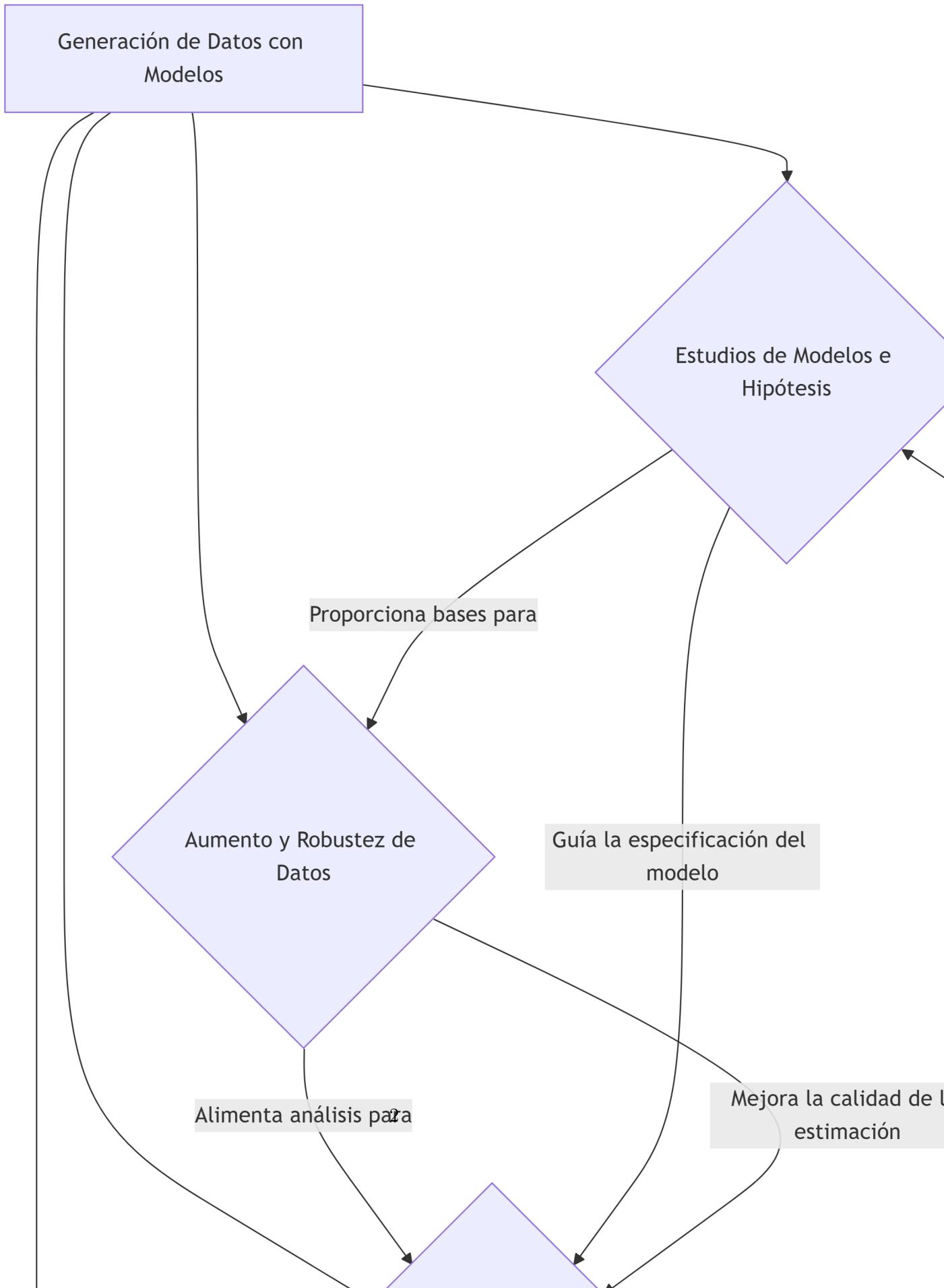
Unidad V: Generación de datos desde modelos de probabilidad

Sesión 1: Martes 10-06-2026

Simulación de datos desde modelos de probabilidad

Principio: Los datos iniciales me dan una idea del proceso aleatorio que los generó. Con este proceso puedo generar datos aumentados para extraer el máximo de información.

- **1 Estudios de modelos generando datos en base de hipótesis**
- **2 Aumento y robustez de datos**
- **3 Estudio de propiedades estadísticas**
- **4 Verificación y validación de métodos estadísticos**


Generación de Datos con Modelos
Estudios de Modelos e Hipótesis
Proporciona bases para
Aumento y Robustez de Datos
Guía la especificación del modelo
Alimenta análisis para
Mejora la calidad de l
estimación

La simulación de datos, anclada en modelos probabilísticos, es una piedra angular en la estadística moderna. Permite explorar fenómenos aleatorios sin información empírica, facilitando la formulación de hipótesis y el diseño experimental. Además, es crucial para aumentar y robustecer análisis con datos escasos, permitiendo un estudio profundo de la variabilidad y estabilidad de las estimaciones. La técnica también es indispensable para la estimación de propiedades de estadísticos, como las distribuciones de muestreo y los intervalos de confianza, mediante métodos como Monte Carlo. Finalmente, la simulación es vital para verificar y validar la robustez y precisión de los métodos estadísticos bajo diversas condiciones controladas.

1. Estudios de modelos generando datos en base de hipótesis

Este principio recomienda explorar fenómenos aleatorios sin datos reales, facilitando la formulación y prueba de hipótesis sobre su comportamiento y el diseño de experimentos futuros. Es esencial para obtener una comprensión inicial del sistema antes de cualquier recolección empírica.

2. Aumento y Robustez de Datos

La simulación es útil para enriquecer análisis con datos empíricos escasos, generando información a partir de modelos ajustados. Esto permite estudiar la variabilidad y estabilidad de las estimaciones y obtener conclusiones más robustas a partir de conjuntos de datos limitados.

3. Estudio de Propiedades Estadísticas

Si la hipótesis sobre mis datos fuesen ciertas ¿Qué propiedades tiene cada método estadístico? Este principio se centra en estimar propiedades de estadísticos con distribuciones desconocidas analíticamente. Mediante simulaciones como Monte Carlo, se pueden aproximar distribuciones de muestreo, calcular intervalos de confianza y determinar la potencia de pruebas, vital para la inferencia.

4. Verificación y Validación de Métodos

Si algunas de mis supuestos sobre el modelo no fuesen verdaderos, ¿cómo afectaria esto a mis métodos estadísticos? La simulación es clave para probar la validez y robustez de los métodos estadísticos. Permite verificar si un método produce resultados precisos y confiables bajo condiciones controladas y diversos escenarios, asegurando su aplicabilidad y exactitud.

Ejemplos

5. Ejemplo de exploración y Generación de Hipótesis

Generador de realizaciones de variables aleatorias de una dimensión

Este apartado, generamos datos de distribuciones de probabilidad univariadas comunes en R, lo que permite explorar hipótesis sobre la naturaleza de una sola variable.

1. Generando números uniformes (`runif()`)

La función `runif(n, min=0, max=1)` genera `n` números aleatorios de una distribución uniforme. Cada valor en el rango `[min, max]` tiene la misma probabilidad de ser generado.

Densidad asociada: dunif(x, min=0, max=1)

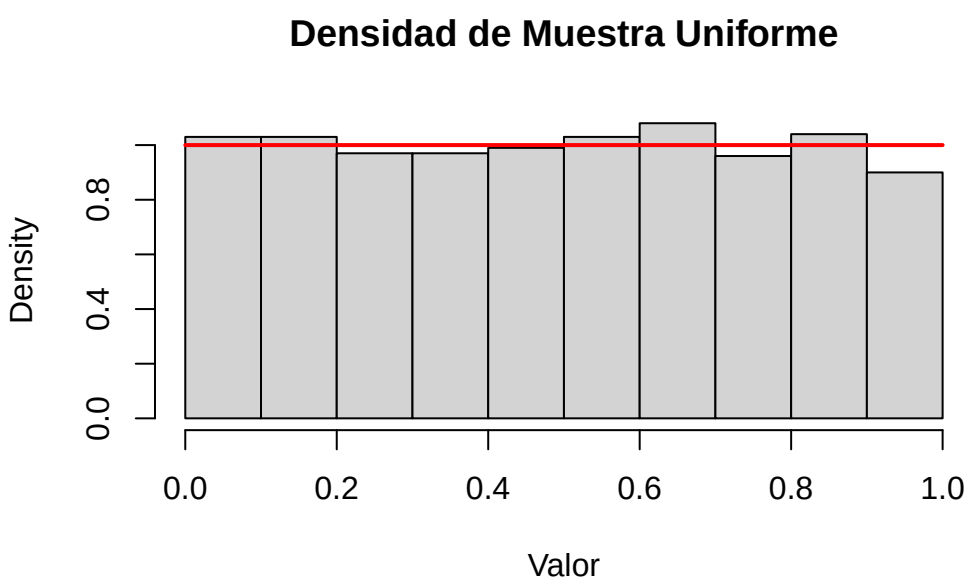
```r
# Generar 5 números uniformes entre 0 y 1
runif(5)
```

```
[1] 0.6012310 0.3829916 0.4272950 0.9811948 0.3319592
```

```r
# Generar 10 números uniformes entre 10 y 20
runif(10, min = 10, max = 20)
```

```
 [1] 11.38080 10.26917 16.36635 10.43846 13.97073 10.69871 17.62160 14.79145
 [9] 14.33785 13.98337
```

```r
# Visualizar la densidad de una muestra uniforme
hist(runif(1000, min = 0, max = 1), freq = FALSE,
     main = "Densidad de Muestra Uniforme", xlab = "Valor")
curve(dunif(x, min = 0, max = 1), add = TRUE, col = "red", lwd = 2)
```



2. Generando números binomiales (`rbinom()`)

La función `rbinom(n, size, prob)` genera `n` números aleatorios de una distribución binomial. Esta distribución describe el número de éxitos (`x`) en un número fijo de ensayos (`size`), donde cada ensayo tiene una probabilidad de éxito (`prob`).

Densidad (función de masa de probabilidad) asociada: dbinom(x, size, prob)

Código de ejemplo:

```
# Simular el número de caras al lanzar una moneda 10 veces, repetido 5 veces
rbinom(n = 5, size = 10, prob = 0.5)

# Simular el número de estudiantes que aprueban de 20, si la prob de aprobar es 0.7
rbinom(n = 1, size = 20, prob = 0.7)

# Visualizar la densidad de una muestra binomial
sim_binomial <- rbinom(n = 1000, size = 10, prob = 0.5)
barplot(table(sim_binomial) / length(sim_binomial),
        main = "Densidad de Muestra Binomial", xlab = "Número de Éxitos", ylab = "Probabilida
points(x = 0:10, y = dbinom(0:10, size = 10, prob = 0.5), col = "red", pch = 16) # Puntos de
```

3. Generando números de Poisson (`rpois()`)

La función `rpois(n, lambda)` genera `n` números aleatorios de una distribución de Poisson, donde `lambda` es la tasa promedio de ocurrencias en un intervalo fijo de tiempo o espacio.

Densidad (función de masa de probabilidad) asociada: dpois(x, lambda)

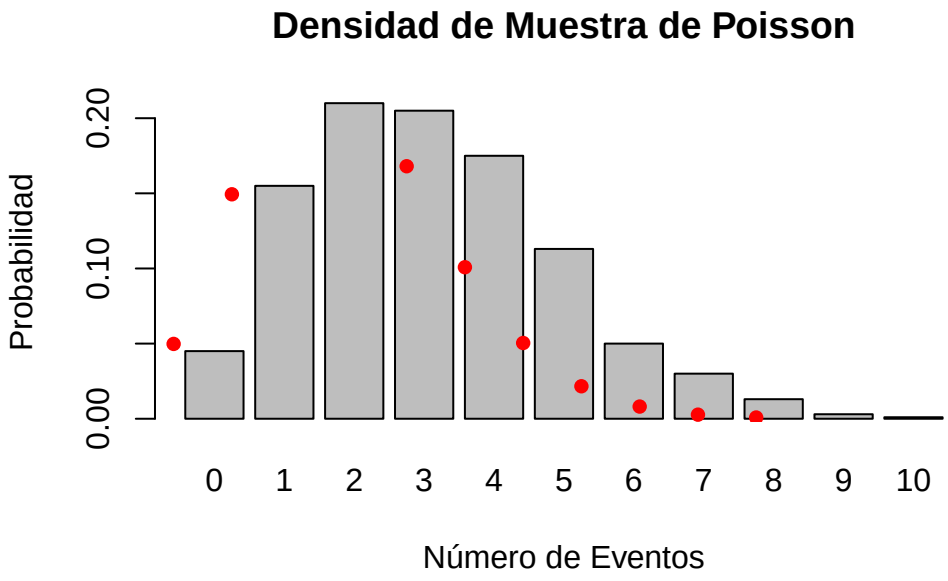
```
# Simular el número de llamadas recibidas en una hora, si el promedio es 5 llamadas/hora
rpois(n = 1, lambda = 5)
```

```
[1] 6
```

```
# Simular el número de defectos por metro de tela, si el promedio es 0.2 defectos/metro
rpois(n = 10, lambda = 0.2)
```

```
 [1] 0 0 0 0 0 0 1 0 0 0
```

```
# Visualizar la densidad de una muestra de Poisson
sim_poisson <- rpois(n = 1000, lambda = 3)
barplot(table(sim_poisson) / length(sim_poisson),
        main = "Densidad de Muestra de Poisson", xlab = "Número de Eventos", ylab = "Probabil
points(x = 0:max(sim_poisson), y = dpois(0:max(sim_poisson), lambda = 3), col = "red", pch =
```



4. Generando números normales (`rnorm()`)

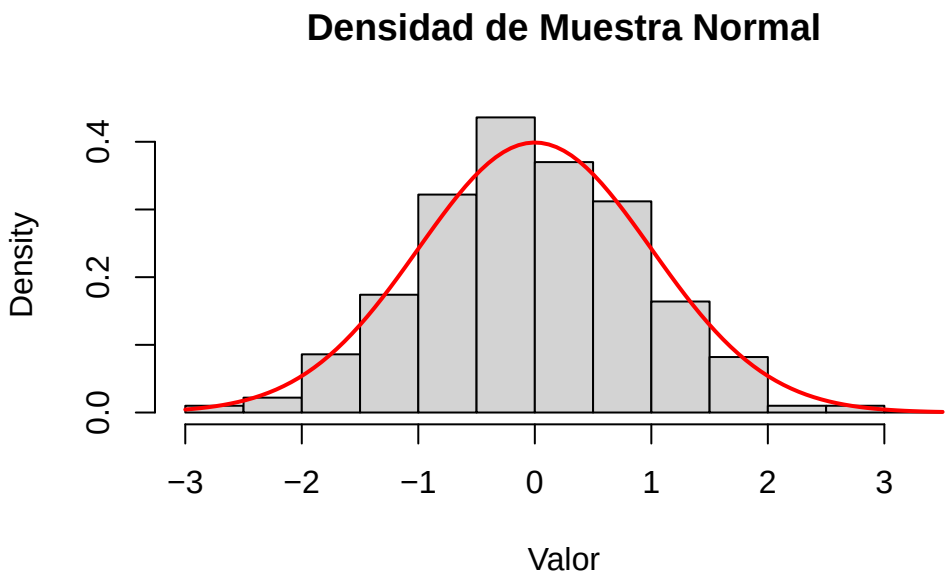
La función `rnorm(n, mean=0, sd=1)` genera `n` números aleatorios de una distribución normal (gaussiana) con una media (`mean`) y desviación estándar (`sd`) especificadas.

Densidad asociada: dnorm(x, mean=0, sd=1)

```
# Simular la altura de 10 personas, con media 170 cm y desviación estándar 5 cm
rnorm(n = 10, mean = 170, sd = 5)
```

```
 [1] 172.8953 171.6127 166.1689 173.8960 173.3208 178.8660 169.5844 177.5598
 [9] 174.2020 176.6325
```

```
# Generar 1000 valores de una distribución normal estándar y visualizar su densidad
hist(rnorm(1000, mean = 0, sd = 1), freq = FALSE,
     main = "Densidad de Muestra Normal", xlab = "Valor")
curve(dnorm(x, mean = 0, sd = 1), add = TRUE, col = "red", lwd = 2)
```



5. Generando números exponenciales (`rexp()`)

La función `rexp(n, rate=1)` genera `n` números aleatorios de una distribución exponencial, donde `rate` (tasa) es 1/media.

Densidad asociada: dexp(x, rate=1)

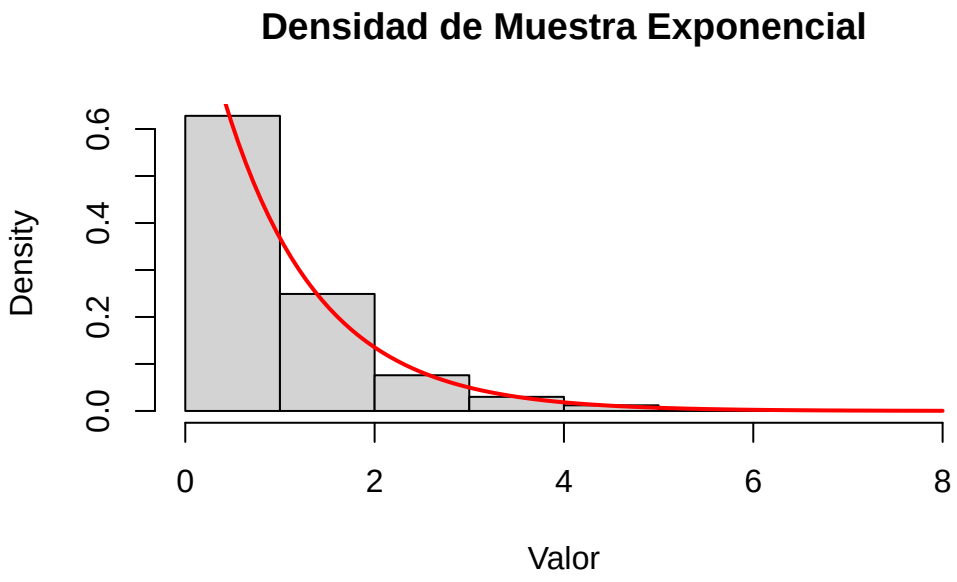
```r
# Simular el tiempo de espera hasta la siguiente llegada de un cliente, si la tasa es 0.5 lle
rexp(n = 1, rate = 0.5)
```

```
[1] 0.9358658
```

```r
# Generar 20 tiempos de falla de un componente con una tasa de 0.1 fallas/hora
rexp(n = 20, rate = 0.1)
```

```
 [1]  0.09694069  6.14840183  2.96622702  2.31219243  0.29807096  1.32220639
 [7] 13.06326649  2.74185439  0.22256181  2.21786757  4.30554105  0.99096433
[13]  9.54290739  1.21651232  0.33483242 31.23623296  4.65182108  0.74343163
[19]  0.04572504 17.23301317
```

```r
# Visualizar la densidad de una muestra exponencial
hist(rexp(1000, rate = 1), freq = FALSE,
     main = "Densidad de Muestra Exponencial", xlab = "Valor")
curve(dexp(x, rate = 1), add = TRUE, col = "red", lwd = 2)
```



6. Asegurando la reproducibilidad (`set.seed()`)

Para que tus simulaciones generen los mismos resultados cada vez que se ejecutan (esencial para la verificación de hipótesis y compartir código reproducible), se usa `set.seed()`.

```
# Sin set.seed(), cada ejecución dará números diferentes
rnorm(3)
```

```
[1]  0.6689621 -1.5391420 -0.3028142
```

```
rnorm(3)
```

```
[1] 0.84058543 0.23488447 0.08422145
```

```
# Con set.seed(), siempre obtendrás los mismos números
set.seed(42)
rnorm(3) # La primera ejecución con esta semilla
```

```
[1]  1.3709584 -0.5646982  0.3631284
```

```
set.seed(42)
rnorm(3) # La segunda ejecución con la misma semilla da los mismos resultados
```

```
[1]  1.3709584 -0.5646982  0.3631284
```

Generador de realizaciones de variables aleatorias de dos dimensiones

Simular datos bidimensionales nos permite explorar cómo dos variables podrían estar relacionadas o no. Esto es clave para probar ideas sobre cómo funcionan los sistemas.

1. Variables Normales Correlacionadas

Generar dos variables normales con una relación específica.

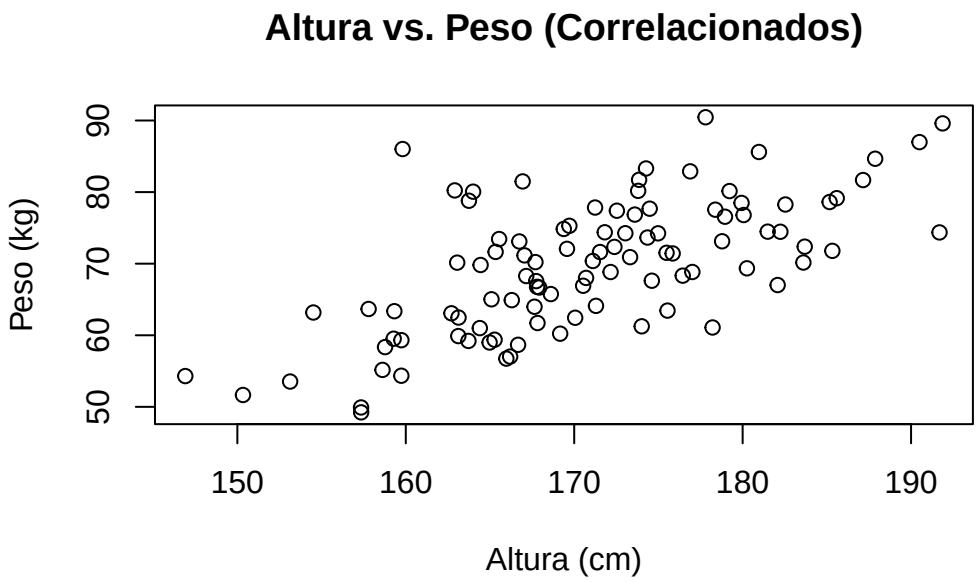
```r
set.seed(123) # Para reproducibilidad
n <- 100     # Cantidad de puntos
rho <- 0.7   # Correlación deseada

# Generar variables normales independientes
z1 <- rnorm(n)
z2 <- rnorm(n)

# Transformar para crear correlación (ej. altura y peso)
altura <- 170 + 10 * z1
peso <- 70 + 10 * (rho * z1 + sqrt(1 - rho^2) * z2)

# Ver datos y correlación
datos_simulados <- data.frame(Altura = altura, Peso = peso)
print(head(datos_simulados))
```

```
    Altura     Peso
1 164.3952 61.00335
2 167.6982 70.22327
3 185.5871 79.14923
4 170.7051 68.01161
5 171.2929 64.10910
6 187.1506 81.68389
```

```r
print(cor(datos_simulados$Altura, datos_simulados$Peso))
```

```
[1] 0.6592833
```

```r
# Visualizar la relación
plot(datos_simulados$Altura, datos_simulados$Peso,
     main = "Altura vs. Peso (Correlacionados)",
     xlab = "Altura (cm)", ylab = "Peso (kg)")
```



2. Relación Lineal con Ruido (Regresión Simple)

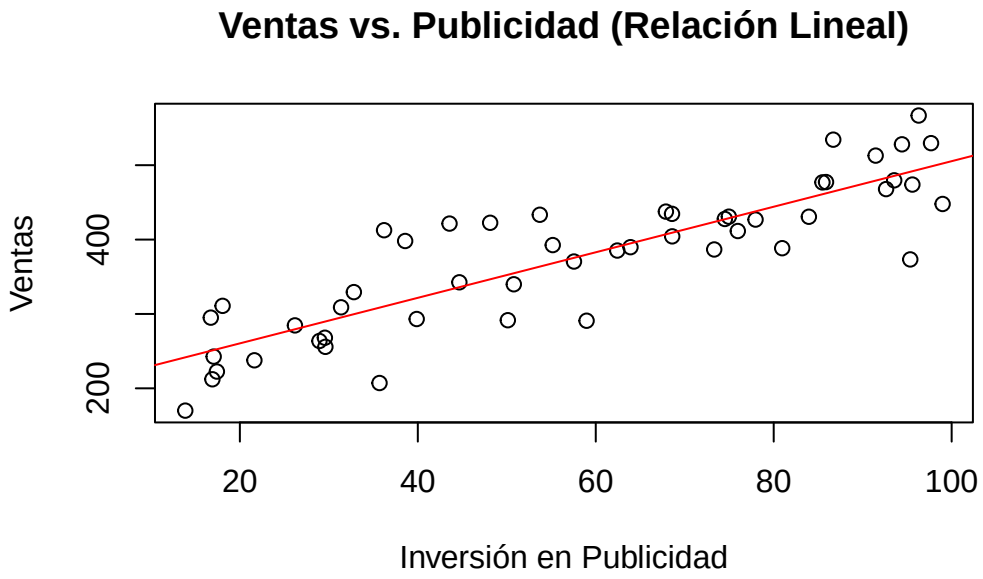
Simular una variable que depende linealmente de otra, con algo de aleatoriedad.

```r
set.seed(456) # Para reproducibilidad
n <- 50      # Cantidad de puntos

# Variable independiente (ej. publicidad)
publicidad <- runif(n, min = 10, max = 100)

# Variable dependiente (ej. ventas = 200 + 3*publicidad + ruido)
ventas <- 200 + 3 * publicidad + rnorm(n, mean = 0, sd = 50)

# Ver datos
datos_simulados_reg <- data.frame(Publicidad = publicidad, Ventas = ventas)
print(head(datos_simulados_reg))
```

```
  Publicidad   Ventas
1   18.05964 310.8932
2   28.94611 263.6956
3   75.96597 411.4787
4   86.69202 534.3030
```

```
5   80.95581 388.3985
6   39.87640 293.1895
```

```r
# Visualizar la relación
plot(datos_simulados_reg$Publicidad, datos_simulados_reg$Ventas,
     main = "Ventas vs. Publicidad (Relación Lineal)",
     xlab = "Inversión en Publicidad", ylab = "Ventas")
abline(lm(Ventas ~ Publicidad, data = datos_simulados_reg), col = "red")
```



3. Variables Independientes (sin relación)

Generar dos variables donde una no afecta a la otra.

```r
set.seed(789) # Para reproducibilidad
n <- 100      # Cantidad de puntos

# Variable 1 (ej. examen 1)
examen_1 <- rnorm(n, mean = 70, sd = 10)

# Variable 2 (ej. tiempo de traslado)
tiempo_traslado <- runif(n, min = 5, max = 60)
```

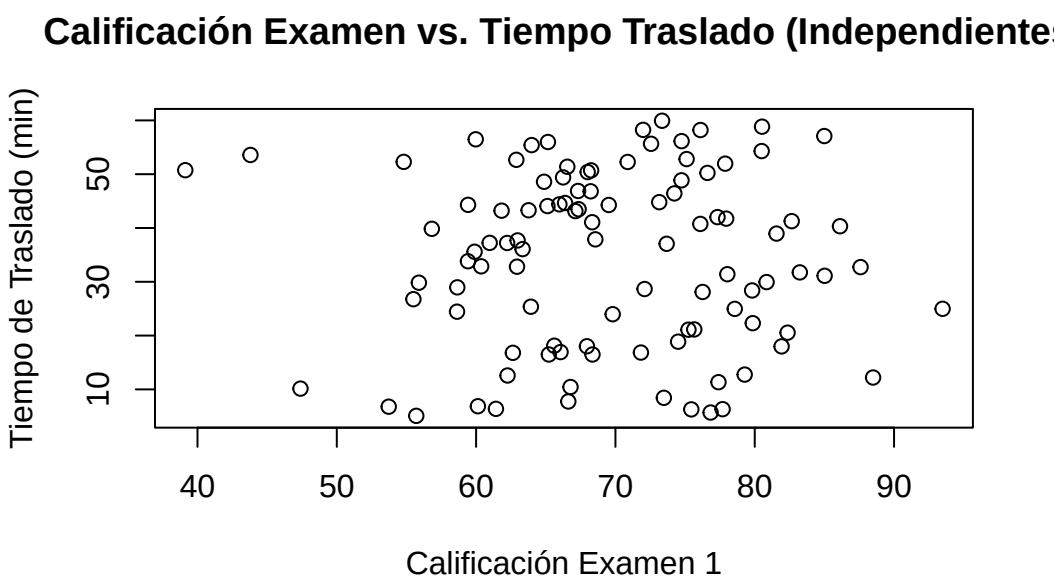
```r
# Ver datos y correlación (cercana a cero)
datos_independientes <- data.frame(Examen1 = examen_1, TiempoTraslado = tiempo_traslado)
print(head(datos_independientes))
```

```
   Examen1 TiempoTraslado
1 75.24097       21.09588
2 47.39232       10.14501
3 69.80320       23.97269
4 71.83140       16.84554
5 66.38649       44.64698
6 65.15516       55.98353
```

```r
print(cor(datos_independientes$Examen1, datos_independientes$TiempoTraslado))
```

```
[1] -0.01696665
```

```r
# Visualizar la ausencia de relación
plot(datos_independientes$Examen1, datos_independientes$TiempoTraslado,
     main = "Calificación Examen vs. Tiempo Traslado (Independientes)",
     xlab = "Calificación Examen 1", ylab = "Tiempo de Traslado (min)")
```



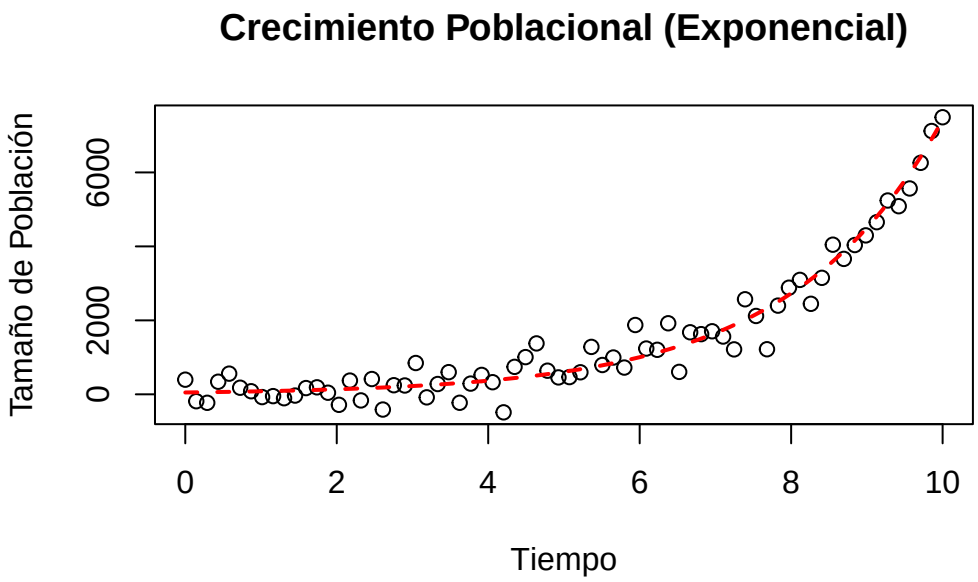
4. Relación No Lineal (Exponencial)

Simular una variable que crece o decrece exponencialmente con respecto a otra.

```r
set.seed(1011) # Para reproducibilidad
n <- 70      # Cantidad de puntos

# Variable independiente (ej. tiempo)
tiempo <- seq(0, 10, length.out = n)

# Variable dependiente: crecimiento exponencial con ruido
poblacion <- 50 * exp(0.5 * tiempo) + rnorm(n, mean = 0, sd = 400)

# Ver datos
datos_exponenciales <- data.frame(Tiempo = tiempo, Poblacion = poblacion)
print(head(datos_exponenciales))
```

```
     Tiempo Poblacion
1 0.0000000  396.1956
2 0.1449275 -186.4092
3 0.2898551 -228.9739
4 0.4347826  340.7638
5 0.5797101  559.2761
6 0.7246377  178.4095
```

```r
# Visualizar la relación exponencial
plot(datos_exponenciales$Tiempo, datos_exponenciales$Poblacion,
     main = "Crecimiento Poblacional (Exponencial)",
     xlab = "Tiempo", ylab = "Tamaño de Población")
curve(50 * exp(0.5 * x), add = TRUE, col = "red", lwd = 2, lty = 2) # Curva teórica
```



Simulación de cadenas de Markov

Una **cadena de Markov** es un proceso estocástico que describe una secuencia de eventos donde la probabilidad de cada evento futuro depende únicamente del estado actual, y no de la secuencia de eventos que lo precedieron. Matemáticamente, esto se conoce como la **propiedad de Markov**. Se define por un conjunto de estados y una **matriz de probabilidades de transición** P, donde Pij es la probabilidad de pasar del estado i al estado j.

$$\{X_1, X_2, X_3, \dots, X_n\} = \{X_n\}$$

$$P(X_{n+1} = j | X_n = i, X_{n-1} = s, \dots, X_1 = f) = P(X_{n+1} = j | X_n = i) = P_{ij}$$

Simulación de Cadenas de Markov: Ejemplos con R

Las cadenas de Markov modelan sistemas que cambian de estado discretamente a lo largo del tiempo, donde el próximo estado solo depende del estado actual. Esto es útil para explorar la dinámica de sistemas con transiciones probabilísticas.

1. Clima Simple: Soleado o Lluvioso

Simular la secuencia de estados del clima (soleado o lluvioso) día a día, basándose en la probabilidad de que cambie o permanezca igual.

	Soleado	Lluvioso
Soleado	0.8	0.2
Lluvioso	0.3	0.7

```r
set.seed(101) # Para reproducibilidad

# Definir los posibles estados del clima
estados <- c("Soleado", "Lluvioso")

# Matriz de probabilidades de transición
# P(Soleado -> Soleado) = 0.8, P(Soleado -> Lluvioso) = 0.2
# P(Lluvioso -> Soleado) = 0.3, P(Lluvioso -> Lluvioso) = 0.7
matriz_transicion <- matrix(c(0.8, 0.2,  # Desde "Soleado"
                              0.3, 0.7), # Desde "Lluvioso"
                            nrow = 2, byrow = TRUE,
                            dimnames = list(estados, estados))

n_dias <- 30 # Simular 30 días
estado_actual <- "Soleado" # Empezamos con un día soleado
secuencia_clima <- character(n_dias)
secuencia_clima[1] <- estado_actual

# Simular la cadena de Markov día a día
for (i in 2:n_dias) {
  # Obtener las probabilidades de transición desde el estado actual
  prob_siguientes <- matriz_transicion[estado_actual, ]

  # Muestrear el siguiente estado basado en esas probabilidades
  estado_actual <- sample(estados, size = 1, prob = prob_siguientes)
  secuencia_clima[i] <- estado_actual
}

print("Secuencia de Clima Simulada:")
```

```
[1] "Secuencia de Clima Simulada:"
```

```r
print(secuencia_clima)
```

```
 [1] "Soleado"  "Soleado"  "Soleado"  "Soleado"  "Soleado"  "Soleado"
 [7] "Soleado"  "Soleado"  "Soleado"  "Soleado"  "Soleado"  "Lluvioso"
```

```
 [13] "Soleado"  "Soleado"  "Lluvioso" "Lluvioso" "Lluvioso" "Soleado"
 [19] "Soleado"  "Soleado"  "Soleado"  "Soleado"  "Lluvioso" "Lluvioso"
 [25] "Lluvioso" "Soleado"  "Soleado"  "Soleado"  "Soleado"  "Soleado"
```

```r
# Frecuencia de estados en la simulación
print("Frecuencia de estados:")
```

```
[1] "Frecuencia de estados:"
```

```r
print(table(secuencia_clima))
```

```
secuencia_clima
Lluvioso  Soleado
       7       23
```

2. Movimiento de un Cliente entre Secciones de una Tienda

Simular el recorrido de un cliente por diferentes secciones de una tienda (e.g., Entrada, Ropa, Comida, Salida), con probabilidades de moverse de una a otra.

```r
set.seed(202) # Para reproducibilidad

# Definir las secciones de la tienda
secciones <- c("Entrada", "Ropa", "Comida", "Salida")

# Matriz de probabilidades de transición
# Ejemplo:
# De Entrada: 50% Ropa, 30% Comida, 20% Salida
# De Ropa: 10% Entrada, 40% Ropa, 30% Comida, 20% Salida
# De Comida: 0% Entrada, 20% Ropa, 50% Comida, 30% Salida
# De Salida: 100% Salida (estado absorbente)
matriz_tienda <- matrix(c(0.0, 0.5, 0.3, 0.2, # Desde Entrada
                          0.1, 0.4, 0.3, 0.2, # Desde Ropa
                          0.0, 0.2, 0.5, 0.3, # Desde Comida
                          0.0, 0.0, 0.0, 1.0), # Desde Salida
                        nrow = 4, byrow = TRUE,
                        dimnames = list(secciones, secciones))

n_pasos_max <- 15 # Simular hasta 15 movimientos
trayectoria_cliente <- character(n_pasos_max)
estado_actual <- "Entrada" # El cliente siempre empieza en la Entrada
```

```r
trayectoria_cliente[1] <- estado_actual

# Simular la trayectoria
for (i in 2:n_pasos_max) {
  if (estado_actual == "Salida") { # Si ya salió, se queda en "Salida"
    trayectoria_cliente[i] <- "Salida"
    next
  }

  prob_siguientes <- matriz_tienda[estado_actual, ]
  estado_actual <- sample(secciones, size = 1, prob = prob_siguientes)
  trayectoria_cliente[i] <- estado_actual
}

print("Trayectoria del Cliente Simulada:")
```

```
[1] "Trayectoria del Cliente Simulada:"
```

```r
print(trayectoria_cliente)
```

```
 [1] "Entrada" "Ropa"    "Comida"  "Comida"  "Comida"  "Comida"  "Salida"
 [8] "Salida"  "Salida"  "Salida"  "Salida"  "Salida"  "Salida"  "Salida"
[15] "Salida"
```

```r
# Contar cuántas veces estuvo en cada sección
print("Visitas por sección:")
```

```
[1] "Visitas por sección:"
```

```r
print(table(trayectoria_cliente))
```

```
trayectoria_cliente
 Comida Entrada    Ropa  Salida
      4       1       1       9
```

Métodos de aceptación y rechazo

Este método simula valores de distribuciones complejas (como las bimodales) usando una distribución más simple. Se generan propuestas de la simple y se aceptan o rechazan según una regla de probabilidad, permitiendo muestrear formas inusuales.

Acepta una muestra propuesta x si:

$$U < M \cdot g(x) f(x)$$

Donde:

- U: Es un número aleatorio generado de una distribución uniforme entre 0 y 1 ($U \sim Unif(0,1)$).
- f(x): Es la función de densidad de probabilidad (FDP) de la **distribución objetivo** de la que quieres muestrear.
- g(x): Es la FDP de la **distribución propuesta** (o envolvente), de la cual es fácil generar muestras.
- M: Es una constante tal que M g(x) f(x) para todo x. Es decir, M g(x) debe ser una "envolvente" de f(x).

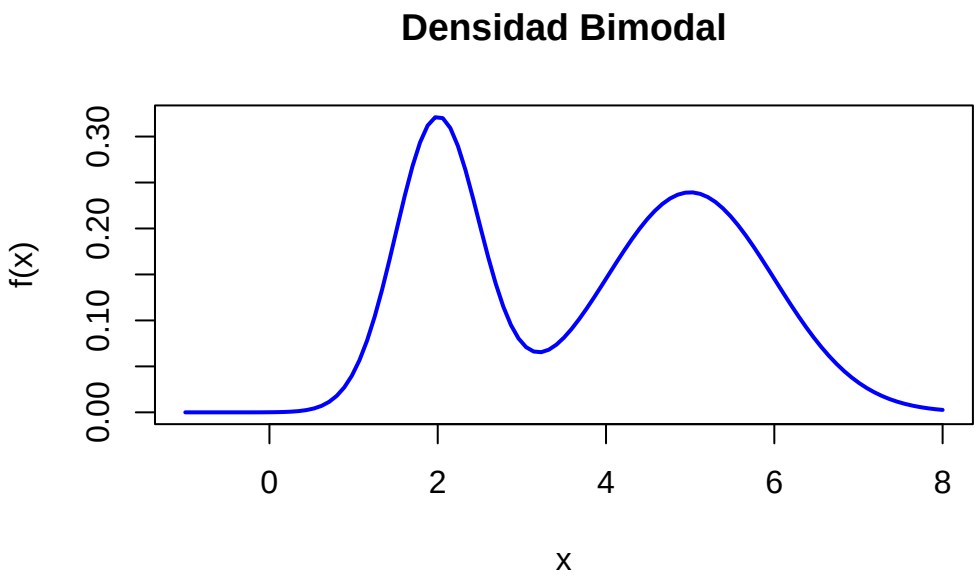
El **muestreo de aceptación/rechazo** genera números aleatorios de una distribución objetivo f(x) (compleja) usando una distribución propuesta g(x) (fácil de muestrear) y una constante M tal que M g(x) f(x). Se acepta una muestra propuesta x (generada de g(x)) si un número uniforme aleatorio U (entre 0 y 1) es menor que la razón f(x)/(M g(x)), de lo contrario se rechaza, asegurando que los puntos aceptados provengan efectivamente de la distribución f(x).

Hipótesis: Queremos muestrear una mezcla de dos distribuciones normales (una densidad bimodal), con la siguiente función de densidad:

$$f(x) = w1 \ \cdot N(x \mid \mu 1 \ , \sigma 1 \ ) + w2 \ \cdot N(x \mid \mu 2 \ , \sigma 2 \ ) donde w1 \ + w2 \ = 1.$$

En nuestro ejemplo, $f(x) = 0.4 \cdot N(x \mid \mu 1 \ = 2, \sigma 1 \ = 0.5) + 0.6 \cdot N(x \mid \mu 2 \ = 5, \sigma 2 \ = 1)$.

```r
# función de densidad bimodal
f_x <- function(x) {
  0.4 * dnorm(x, mean = 2, sd = 0.5) + 0.6 * dnorm(x, mean = 5, sd = 1)
}

#  la curva de densidad
curve(f_x(x), from = -1, to = 8,
      main = "Densidad Bimodal",
      xlab = "x", ylab = "f(x)",
      col = "blue", lwd = 2)
```




```r
set.seed(1233) # Para reproducibilidad

# 1. Densidad objetivo (la que queremos muestrear: bimodal)
target_density <- function(x) {
  0.4 * dnorm(x, mean = 2, sd = 0.5) + 0.6 * dnorm(x, mean = 5, sd = 1)
}

# 2. Densidad propuesta (una normal simple que la "envuelve")
proposal_density <- function(x) {
  dnorm(x, mean = 3.5, sd = 2)
}

# 3. Encontrar M (constante para escalar la propuesta)
x_range <- seq(-2, 8, length.out = 1000)
M <- max(target_density(x_range) / proposal_density(x_range))
cat("Valor de M:", M, "\n") # M debe ser > 1
```

```
Valor de M: 2.173388
```

```r
# 4. Generar muestras por Aceptación-Rechazo
n_muestras_deseadas <- 5000
muestras_aceptadas <- numeric(n_muestras_deseadas)
```

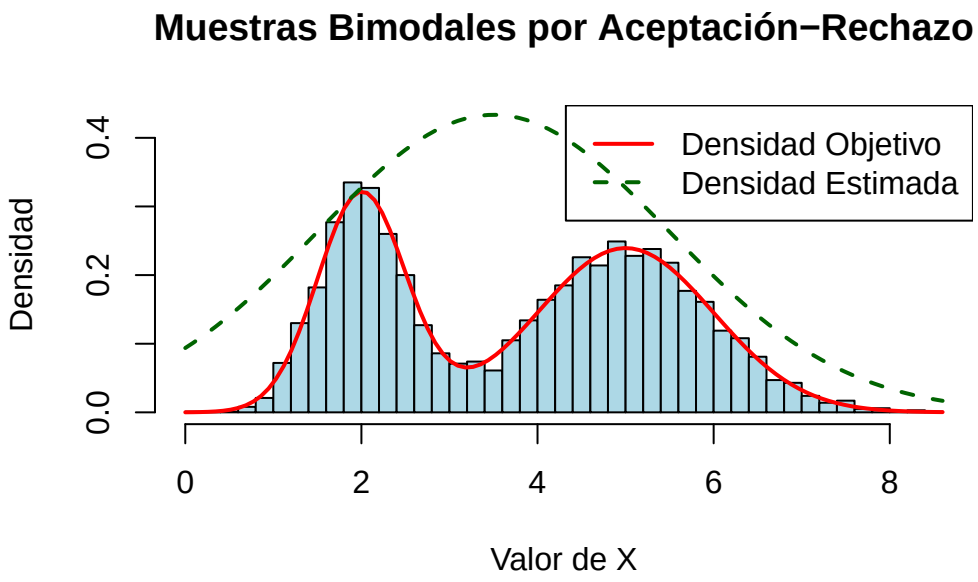
```r
conteo_aceptadas <- 0
conteo_propuestas <- 0

while (conteo_aceptadas < n_muestras_deseadas) {
  propuesta_x <- rnorm(1, mean = 3.5, sd = 2) # Generar de la propuesta
  u <- runif(1) # Valor para decisión

  # Aceptar si la propuesta está "debajo" de la densidad objetivo ajustada
  if (proposal_density(propuesta_x) > 0 && u < (target_density(propuesta_x) / (M * proposal_
    conteo_aceptadas <- conteo_aceptadas + 1
    muestras_aceptadas[conteo_aceptadas] <- propuesta_x
  }
  conteo_propuestas <- conteo_propuestas + 1 # Contar todas las propuestas
}

cat("Propuestas generadas:", conteo_propuestas, "\n")
```

```
Propuestas generadas: 10982
```

```r
cat("Eficiencia:", n_muestras_deseadas / conteo_propuestas, "\n")
```

```
Eficiencia: 0.4552905
```

```r
# 5. Visualizar resultados
hist(muestras_aceptadas, breaks = 50, freq = FALSE,
     main = "Muestras Bimodales por Aceptación-Rechazo",
     xlab = "Valor de X", ylab = "Densidad", col = "lightblue",
     ylim = c(0,0.43))

# Añadir curvas para comparación
curve(target_density(x), add = TRUE, col = "red", lwd = 2) # Objetivo
curve(M * proposal_density(x), add = TRUE, col = "darkgreen", lty = 2, lwd = 2) # Propuesta e

legend("topright", legend = c("Densidad Objetivo", "Densidad Estimada"),
       col = c("red", "darkgreen"), lty = c(1, 2), lwd = 2)
```



Sesión 2: Sabado 14-06-2026

6. Ejemplo del aumento y robustez de los datos

Estimación de la Varianza y Construcción de Intervalos de Confianza via Bootstrapping

El bootstrapping remuestra los datos originales para estimar la variabilidad de una estadística compleja, como el AUC, o para construir intervalos de confianza robustos cuando las fórmulas analíticas son difíciles de obtener o las suposiciones no se cumplen. Permite entender la fiabilidad de una métrica de rendimiento de un modelo sin depender de teorías distributivas restrictivas.

Ejemplo: Intervalo de Confianza para el Área Bajo la Curva ROC (AUC) de un Modelo Predictivo de Enfermedad.

```
# Cargar paquetes
library(pROC)
```

```
Warning: package 'pROC' was built under R version 4.4.3
```

```
Type 'citation("pROC")' for a citation.
```

```
Adjuntando el paquete: 'pROC'

The following objects are masked from 'package:stats':

    cov, smooth, var
```

```r
library(boot)

# Datos de ejemplo 'aSAH' (predicción de enfermedad)
data(aSAH)

# Ajustar modelo logístico simple
modelo_logistico <- glm(outcome ~ s100b, data = aSAH, family = binomial)
aSAH$prob_predicha <- predict(modelo_logistico, type = "response")

# Función para calcular AUC (para bootstrapping)
calcular_auc <- function(data, indices) {
  d <- data[indices, ]
  if (length(unique(d$outcome)) < 2) return(NA) # Evitar errores si falta una clase
  roc_obj <- roc(response = d$outcome, predictor = d$prob_predicha, quiet = TRUE)
  return(auc(roc_obj))
}

# Realizar bootstrapping (2000 remuestreos)
set.seed(123) # Para reproducibilidad
boot_results <- boot(data = aSAH, statistic = calcular_auc, R = 2000)

# Mostrar resultados e intervalo de confianza BCa
print(boot_results)
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = aSAH, statistic = calcular_auc, R = 2000)


Bootstrap Statistics :
     original        bias    std. error
t1* 0.7313686 -0.0005761782  0.05208095
```

```r
boot_ci <- boot.ci(boot_results, type = "bca")
print(boot_ci)
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 2000 bootstrap replicates

CALL :
boot.ci(boot.out = boot_results, type = "bca")

Intervals :
Level       BCa
95%   ( 0.6205,  0.8198 )
Calculations and Intervals on Original Scale
```

Evaluación Robusta del Rendimiento de Modelos via Cross-Validation

La validación cruzada divide los datos en múltiples subconjuntos para entrenar y probar un modelo repetidamente, ofreciendo una estimación más confiable de su rendimiento en datos nuevos y ayudando a seleccionar los mejores hiperparámetros. Es esencial para evitar el sobreajuste y obtener una medida realista de la capacidad de generalización del modelo, especialmente con datasets limitados o con muchas variables.

Ejemplo: Estimación del Error de Predicción de un Modelo de Regresión Penalizada (Lasso/Ridge) en Genómica.

```r
# Cargar paquetes
library(glmnet)
```

```
Warning: package 'glmnet' was built under R version 4.4.3

Cargando paquete requerido: Matrix

Loaded glmnet 4.1-9
```

```r
library(caret)
```

```
Warning: package 'caret' was built under R version 4.4.3

Cargando paquete requerido: ggplot2
```

```
Warning: package 'ggplot2' was built under R version 4.4.3
```

```
Cargando paquete requerido: lattice
```

```

Adjuntando el paquete: 'lattice'
```

```
The following object is masked from 'package:boot':

    melanoma
```

```r
# Datos de ejemplo 'diabetes' (predicción de enfermedad)
data(diabetes, package = "lars")
X <- as.matrix(diabetes$x)
Y <- diabetes$y

# Configuración para validación cruzada (10-fold CV)
ctrl <- trainControl(method = "cv", number = 10)

# Entrenar modelo Lasso con CV para encontrar el mejor lambda
set.seed(456) # Para reproducibilidad
lasso_model_cv <- train(x = X, y = Y,
                        method = "glmnet",
                        trControl = ctrl,
                        tuneGrid = expand.grid(alpha = 1, # Lasso
                                               lambda = seq(0.01, 1, length = 100)))

# Mostrar resultados y mejor lambda/RMSE
print(lasso_model_cv)
```

```
glmnet

442 samples
  1 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 398, 397, 398, 398, 397, 398, ...
Resampling results across tuning parameters:

  lambda  RMSE      Rsquared   MAE
```

0.01	54.28518	0.5074562	44.25058
0.02	54.27817	0.5075839	44.24399
0.03	54.26812	0.5077692	44.23326
0.04	54.25921	0.5079371	44.22512
0.05	54.25109	0.5080936	44.21860
0.06	54.24457	0.5082234	44.21261
0.07	54.23982	0.5083235	44.20682
0.08	54.23706	0.5083794	44.20225
0.09	54.23456	0.5084247	44.19939
0.10	54.23608	0.5083977	44.20010
0.11	54.23945	0.5083482	44.20349
0.12	54.24268	0.5082990	44.20833
0.13	54.24676	0.5082449	44.21319
0.14	54.25351	0.5081396	44.22040
0.15	54.26071	0.5080262	44.22602
0.16	54.26486	0.5079654	44.22809
0.17	54.26804	0.5079147	44.22873
0.18	54.27076	0.5078736	44.22874
0.19	54.27326	0.5078370	44.22815
0.20	54.27503	0.5078181	44.22676
0.21	54.27631	0.5078086	44.22663
0.22	54.27754	0.5078007	44.22857
0.23	54.27390	0.5078764	44.22721
0.24	54.26772	0.5079984	44.22362
0.25	54.26168	0.5081180	44.22009
0.26	54.25535	0.5082447	44.21614
0.27	54.24925	0.5083680	44.21232
0.28	54.24322	0.5084887	44.20771
0.29	54.23736	0.5086067	44.20311
0.30	54.23172	0.5087210	44.19867
0.31	54.22665	0.5088191	44.19476
0.32	54.22271	0.5088942	44.19174
0.33	54.21931	0.5089582	44.18915
0.34	54.21666	0.5090061	44.18731
0.35	54.21472	0.5090470	44.18652
0.36	54.21299	0.5090837	44.18588
0.37	54.21186	0.5091082	44.18568
0.38	54.21076	0.5091324	44.18549
0.39	54.20967	0.5091566	44.18528
0.40	54.20857	0.5091808	44.18504
0.41	54.20747	0.5092049	44.18479
0.42	54.20641	0.5092285	44.18454
0.43	54.20541	0.5092513	44.18437

0.44	54.20445	0.5092740	44.18451
0.45	54.20351	0.5092965	44.18477
0.46	54.20259	0.5093186	44.18503
0.47	54.20169	0.5093406	44.18525
0.48	54.20088	0.5093612	44.18549
0.49	54.20021	0.5093795	44.18601
0.50	54.19955	0.5093975	44.18670
0.51	54.19893	0.5094149	44.18739
0.52	54.19838	0.5094314	44.18828
0.53	54.19790	0.5094465	44.18938
0.54	54.19746	0.5094611	44.19049
0.55	54.19705	0.5094754	44.19160
0.56	54.19667	0.5094892	44.19273
0.57	54.19630	0.5095031	44.19383
0.58	54.19599	0.5095162	44.19488
0.59	54.19570	0.5095289	44.19594
0.60	54.19544	0.5095411	44.19698
0.61	54.19518	0.5095532	44.19799
0.62	54.19490	0.5095651	44.19890
0.63	54.19467	0.5095760	44.19986
0.64	54.19446	0.5095869	44.20079
0.65	54.19428	0.5095974	44.20190
0.66	54.19412	0.5096076	44.20303
0.67	54.19387	0.5096190	44.20405
0.68	54.19343	0.5096332	44.20484
0.69	54.19301	0.5096470	44.20560
0.70	54.19270	0.5096583	44.20656
0.71	54.19244	0.5096686	44.20757
0.72	54.19221	0.5096785	44.20859
0.73	54.19201	0.5096880	44.20961
0.74	54.19179	0.5096977	44.21063
0.75	54.19159	0.5097074	44.21163
0.76	54.19142	0.5097164	44.21265
0.77	54.19130	0.5097244	44.21370
0.78	54.19121	0.5097321	44.21476
0.79	54.19096	0.5097422	44.21579
0.80	54.19067	0.5097530	44.21681
0.81	54.19041	0.5097633	44.21783
0.82	54.19017	0.5097733	44.21886
0.83	54.18998	0.5097828	44.21994
0.84	54.18983	0.5097920	44.22109
0.85	54.18970	0.5098008	44.22224
0.86	54.18960	0.5098093	44.22338

```
  0.87    54.18951  0.5098176  44.22452
  0.88    54.18946  0.5098255  44.22566
  0.89    54.18939  0.5098339  44.22679
  0.90    54.18932  0.5098432  44.22789
  0.91    54.18931  0.5098516  44.22907
  0.92    54.18940  0.5098585  44.23041
  0.93    54.18952  0.5098651  44.23174
  0.94    54.18974  0.5098694  44.23322
  0.95    54.19002  0.5098720  44.23479
  0.96    54.19033  0.5098743  44.23637
  0.97    54.19067  0.5098761  44.23795
  0.98    54.19119  0.5098754  44.23969
  0.99    54.19184  0.5098727  44.24157
  1.00    54.19251  0.5098702  44.24325

Tuning parameter 'alpha' was held constant at a value of 1
RMSE was used to select the optimal model using the smallest value.
The final values used for the model were alpha = 1 and lambda = 0.91.
```

```r
cat("\nMejor lambda por CV:", lasso_model_cv$bestTune$lambda, "\n")
```

```
Mejor lambda por CV: 0.91
```

Estudio de las Propiedades de Estimadores en Muestras Pequeñas via Simulación Monte Carlo

La simulación Monte Carlo permite evaluar el comportamiento de los estimadores (como su sesgo o tasa de error Tipo I) en escenarios específicos donde no hay soluciones analíticas simples, como con distribuciones de datos no estándar o tamaños de muestra pequeños. Consiste en repetir el proceso de muestreo y estimación miles de veces para observar las propiedades empíricas del estimador.

Ejemplo: Tasa de Error Tipo I de una Prueba de Hipótesis para la Media de una Distribución Altamente Asimétrica (e.g., Datos de Costos Sanitarios).

```r
# Parámetros de simulación
n_sims <- 5000       # Número de simulaciones
tamano_muestra <- 30 # Tamaño de cada muestra
alfa <- 0.05         # Nivel de significancia
```

```r
# Parámetros de la distribución log-normal (H0: media = 100)
sdlog_val <- 0.8
meanlog_val <- log(100) - (sdlog_val^2 / 2)

# Simulación Monte Carlo
p_valores <- numeric(n_sims)
set.seed(789) # Para reproducibilidad

for (i in 1:n_sims) {
  # Generar datos log-normales bajo H0
  datos_simulados <- rlnorm(n = tamano_muestra, meanlog = meanlog_val, sdlog = sdlog_val)

  # Realizar prueba t y guardar p-valor
  t_test_result <- t.test(datos_simulados, mu = 100)
  p_valores[i] <- t_test_result$p.value
}

# Calcular Tasa de Error Tipo I empírica
tasa_error_tipo_I <- sum(p_valores < alfa) / n_sims

cat("\nAlfa nominal:", alfa, "\n")
```

```
Alfa nominal: 0.05
```

```r
cat("Tasa de Error Tipo I empírica (datos asimétricos):", tasa_error_tipo_I, "\n")
```

```
Tasa de Error Tipo I empírica (datos asimétricos): 0.0862
```

Manejo de Datos Faltantes Imputation Multiple:

Cuando un dataset es limitado y presenta valores perdidos, se puede utilizar la **imputación múltiple**. Esta técnica implica ajustar un modelo estadístico (basado en los datos observados) para predecir y generar múltiples conjuntos de valores plausibles para los datos faltantes.

La imputación múltiple rellena los datos faltantes generando varias versiones completas del conjunto de datos, rellenando los valores perdidos de forma probabilística. Esto permite analizar los datos sin sesgos significativos por la pérdida de información y combinar los resultados para una inferencia más robusta, incorporando la incertidumbre de la imputación. Es crucial para estudios con patrones de datos faltantes complejos o no aleatorios.

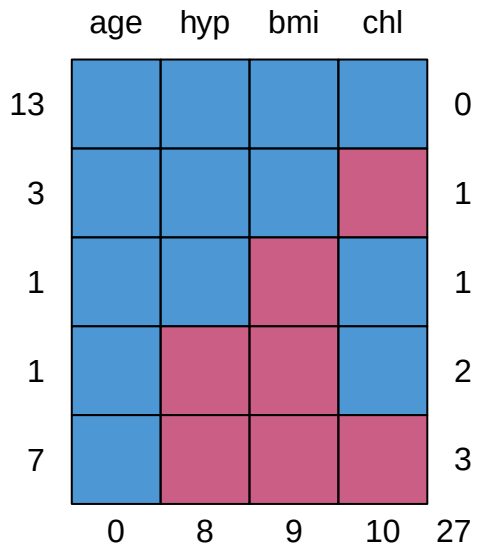
Ejemplo: Análisis de Impacto de Factores de Salud en la Presión Sanguínea en un Estudio con Datos Faltantes.

```
# Cargar paquete
library(mice)
```

```
Warning: package 'mice' was built under R version 4.4.3
```

```
Adjuntando el paquete: 'mice'

The following object is masked from 'package:stats':

    filter

The following objects are masked from 'package:base':

    cbind, rbind
```

```
# Usar datos de ejemplo 'nhanes' con NA
data(nhanes)
md.pattern(nhanes) # Patrón de faltantes
```



```
   age hyp bmi chl
13   1   1   1   1  0
3    1   1   1   0  1
1    1   1   0   1  1
1    1   0   0   1  2
7    1   0   0   0  3
     0   8   9  10 27
```

```r
# Realizar imputación múltiple (5 datasets)
set.seed(42) # Reproducibilidad
imputed_data <- mice(nhanes, m = 5, method = "pmm", printFlag = FALSE)

# Ajustar modelo lineal en cada dataset imputado
model_fits <- with(imputed_data, lm(chl ~ bmi + hyp))

# Combinar los resultados de los modelos
pooled_results <- pool(model_fits)

# 4. Mostrar resumen combinado
print(summary(pooled_results))
```

```
         term  estimate std.error statistic       df   p.value
1 (Intercept) 78.905502 63.678726  1.239119 17.01323 0.2321244
2         bmi  2.771842  2.199206  1.260383 16.43887 0.2251289
3         hyp 33.118378 21.408734  1.546956 12.50266 0.1468033
```

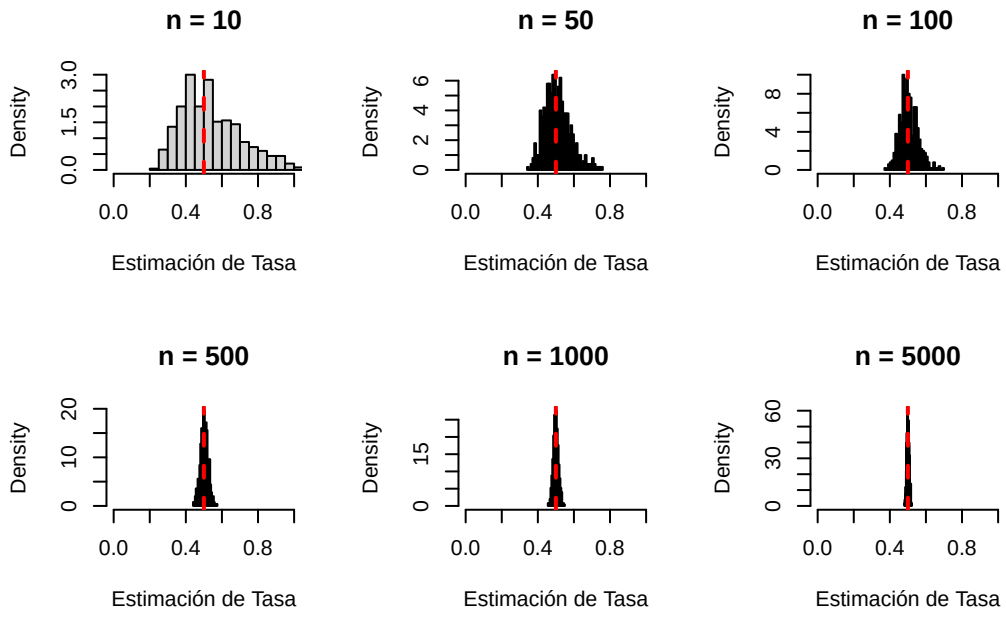
7. Ejemplo de Estudio de Propiedades Estadísticas

Este principio utiliza la simulación para **resolver el problema** de comprender y verificar el comportamiento asintótico de los estimadores y métodos inferenciales. La simulación nos da una visión empírica vital de su fiabilidad en escenarios prácticos.

Convergencia de un Estimador (Consistencia): Evaluar la Fiabilidad del Estimador de Tasa de Fallas de Componentes

En ingeniería, estimar la tasa de fallas (λ) de componentes es crucial para la planificación del mantenimiento. El Estimador de Máxima Verosimilitud (MLE), $1/\bar{x}$, es comúnmente usado para datos de tiempo hasta la falla (distribución exponencial). La siguiente simulación verificar empíricamente si este estimador es consistente, es decir, si sus estimaciones se acercan al valor real de la tasa de fallas a medida que se recolectan más datos.

```r
# Parámetros
true_lambda <- 0.5 # Tasa real
sample_sizes <- c(10, 50, 100, 500, 1000, 5000) # Tamaños de muestra
n_runs_per_size <- 500 # Repeticiones

estimates_list <- vector("list", length(sample_sizes))
set.seed(123) # Para reproducibilidad

# Simular y estimar
for (s_idx in seq_along(sample_sizes)) {
  n <- sample_sizes[s_idx]
  current_estimates <- numeric(n_runs_per_size)
  for (i in 1:n_runs_per_size) {
    data_exp <- rexp(n, rate = true_lambda)
    current_estimates[i] <- 1 / mean(data_exp) # MLE lambda
  }
  estimates_list[[s_idx]] <- current_estimates
}

# Visualizar convergencia
par(mfrow = c(2, 3))
for (s_idx in seq_along(sample_sizes)) {
  n <- sample_sizes[s_idx]
  hist(estimates_list[[s_idx]], breaks = 30, freq = FALSE,
       main = paste("n =", n), xlab = "Estimación de Tasa", xlim = c(0, 2 * true_lambda))
  abline(v = true_lambda, col = "red", lty = 2, lwd = 2) # Tasa verdadera
}
```




```
par(mfrow = c(1, 1))

# Medias de las estimaciones
cat("Medias de estimaciones por tamaño de muestra:\n")
```

```
Medias de estimaciones por tamaño de muestra:
```

```
print(sapply(estimates_list, mean))
```

```
[1] 0.5470512 0.5090266 0.5066188 0.5019865 0.4999208 0.5002957
```

```
cat("Tasa verdadera:", true_lambda, "\n")
```

```
Tasa verdadera: 0.5
```

Propiedades de un Intervalo de Confianza: Verificar la Fiabilidad de un IC para el Consumo Promedio en Marketing

Una empresa de marketing quiere estimar el consumo promedio de un producto en un nuevo segmento de clientes y desea saber si sus intervalos de confianza del 95% realmente capturan

la media poblacional el 95% de las veces. La simulación puede verificar empíricamente la tasa de cobertura de su intervalo de confianza (usando la prueba t, con varianza desconocida), asegurando que sus afirmaciones sobre la precisión de la estimación del consumo promedio son válidas y fiables.

```r
# Parámetros
true_mu <- 10 # Media verdadera
true_sigma <- 5 # Desviación estándar verdadera
sample_size <- 20 # Tamaño de muestra
confidence_level <- 0.95 # Nivel de confianza
n_sims <- 10000 # Simulaciones

contains_true_mu <- logical(n_sims)
set.seed(456) # Para reproducibilidad

# Simular y calcular ICs
for (i in 1:n_sims) {
  sample_data <- rnorm(n = sample_size, mean = true_mu, sd = true_sigma)
  t_test_result <- t.test(sample_data, conf.level = confidence_level)

  # Verificar si IC contiene la media verdadera
  contains_true_mu[i] <- (true_mu >= t_test_result$conf.int[1] && true_mu <= t_test_result$c
}

# Tasa de cobertura empírica
coverage_rate <- sum(contains_true_mu) / n_sims

cat("Nivel de confianza nominal:", confidence_level, "\n")
```

```
Nivel de confianza nominal: 0.95
```

```r
cat("Tasa de cobertura empírica:", coverage_rate, "\n")
```

```
Tasa de cobertura empírica: 0.9515
```

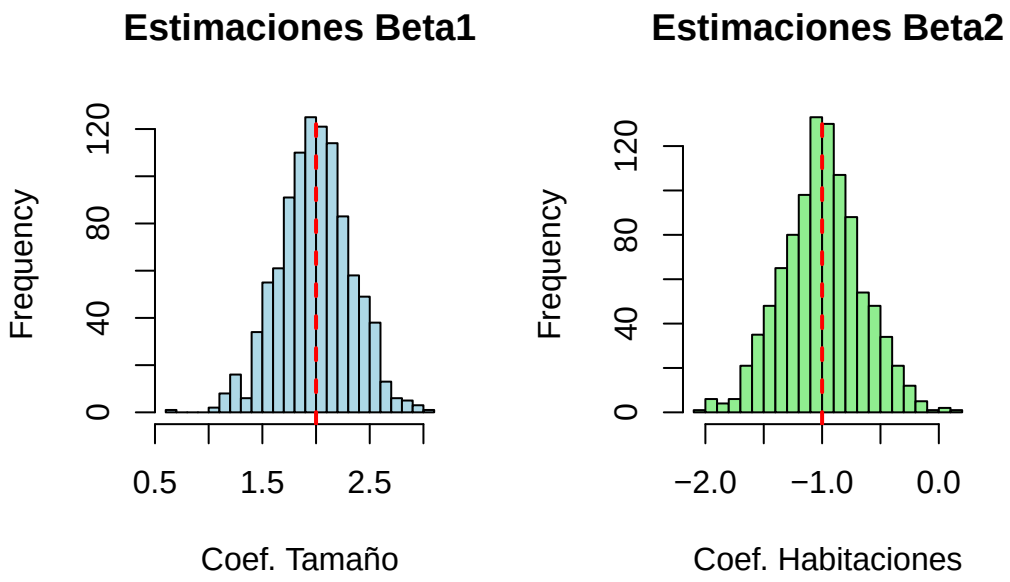
8. Ejemplo de Verificación y Validación de Métodos

Estudio de Sensibilidad de un Modelo Lineal: Evaluar la Robustez de Predicciones de Precios Inmobiliarios bajo Colinealidad

Al predecir precios de casas, variables como tamaño de la casa y número de habitaciones pueden estar altamente correlacionadas (colinealidad). La simulación nos ayuda a entender

y cuantificar cómo esta colinealidad puede hacer que las contribuciones individuales de estas variables al precio estimado sean inestables o contradictorias en diferentes muestras, incluso si el modelo general predice bien. Esto ayuda a los analistas inmobiliarios a comprender las limitaciones y la sensibilidad de sus modelos estándar.

```
# Parámetros
n_obs <- 50 # Casas
n_sims <- 1000 # Simulaciones
beta0_true <- 5 # Intercepto real
beta1_true <- 2 # Coef. real X1
beta2_true <- -1 # Coef. real X2
sigma_error <- 1 # Ruido

correlation_x1_x2 <- 0.9 # Colinealidad

beta1_estimates <- numeric(n_sims)
beta2_estimates <- numeric(n_sims)
set.seed(789) # Para reproducibilidad

# Simular y ajustar modelos
for (i in 1:n_sims) {
  X1 <- rnorm(n_obs)
  Z <- rnorm(n_obs)
  X2 <- correlation_x1_x2 * X1 + sqrt(1 - correlation_x1_x2^2) * Z # X2 correlacionado

  Y <- beta0_true + beta1_true * X1 + beta2_true * X2 + rnorm(n_obs, sd = sigma_error) # Pre
  model <- lm(Y ~ X1 + X2)

  beta1_estimates[i] <- coef(model)["X1"]
  beta2_estimates[i] <- coef(model)["X2"]
}

# Visualizar variabilidad
par(mfrow = c(1, 2))
hist(beta1_estimates, breaks = 30, main = "Estimaciones Beta1", xlab = "Coef. Tamaño", col =
abline(v = beta1_true, col = "red", lty = 2, lwd = 2)

hist(beta2_estimates, breaks = 30, main = "Estimaciones Beta2", xlab = "Coef. Habitaciones",
abline(v = beta2_true, col = "red", lty = 2, lwd = 2)
```




```
par(mfrow = c(1, 1))

cat("SD Beta1:", sd(beta1_estimates), "\n")
```

```
SD Beta1: 0.3332102
```

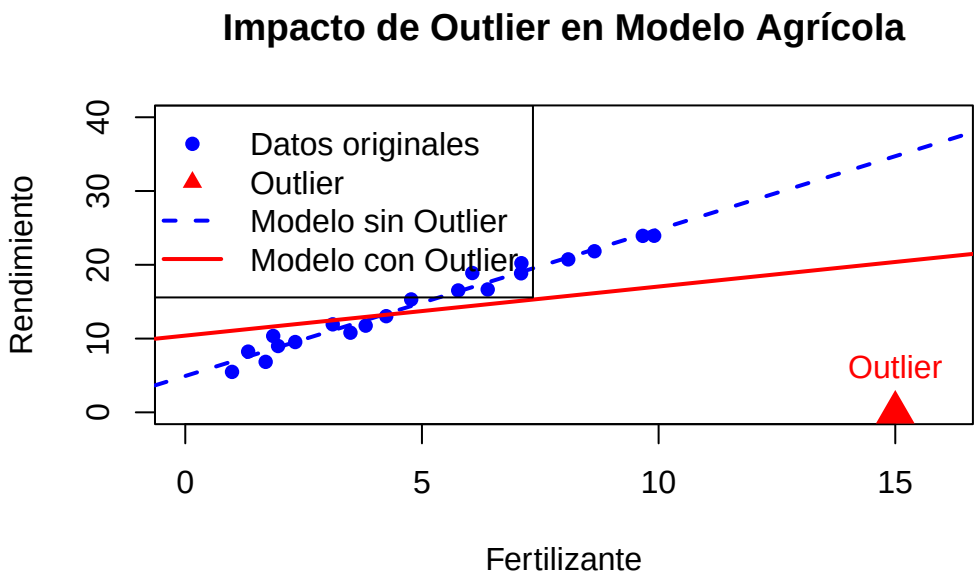
```
cat("SD Beta2:", sd(beta2_estimates), "\n")
```

```
SD Beta2: 0.3396312
```

Estudio de Puntos Influyentes: Validar la Fiabilidad de un Modelo de Rendimiento Agrícola ante Errores de Medición

En agricultura, se usa regresión para predecir el rendimiento de cultivos (Y) basado en fertilizantes (X). Un error de medición accidental al registrar el fertilizante o el rendimiento de una parcela (un outlier) puede distorsionar el modelo. La simulación demuestra cuán vulnerable es la regresión estándar a estos errores. Al introducir un punto anómalo, se valida la necesidad de métodos robustos o de detección de outliers para asegurar que las recomendaciones agrícolas no se basen en estimaciones sesgadas.

```r
# Parámetros base
n_points <- 20 # Parcelas
true_beta0 <- 5
true_beta1 <- 2
error_sd <- 1

# Generar datos "limpios"
set.seed(987) # Reproducibilidad
X <- runif(n_points, min = 0, max = 10) # Fertilizante
Y <- true_beta0 + true_beta1 * X + rnorm(n_points, sd = error_sd) # Rendimiento

# Crear outlier influyente (error de medición)
outlier_X <- 15
outlier_Y <- 0

# 1. Ajustar modelo sin outlier
data_clean <- data.frame(X, Y)
model_clean <- lm(Y ~ X, data = data_clean)

# 2. Ajustar modelo con outlier
X_outlier <- c(X, outlier_X)
Y_outlier <- c(Y, outlier_Y)
data_with_outlier <- data.frame(X = X_outlier, Y = Y_outlier)
model_with_outlier <- lm(Y ~ X, data = data_with_outlier)

# 3. Visualizar impacto
plot(X, Y, main = "Impacto de Outlier en Modelo Agrícola",
     xlab = "Fertilizante", ylab = "Rendimiento",
     col = "blue", pch = 16, xlim = c(0, 16), ylim = c(0, 40))
points(outlier_X, outlier_Y, col = "red", pch = 17, cex = 2) # Outlier
text(outlier_X, outlier_Y + 2, "Outlier", col = "red", pos = 3)

abline(model_clean, col = "blue", lty = 2, lwd = 2) # Línea sin outlier
abline(model_with_outlier, col = "red", lty = 1, lwd = 2) # Línea con outlier

legend("topleft", legend = c("Datos originales", "Outlier", "Modelo sin Outlier", "Modelo co
       col = c("blue", "red", "blue", "red"),
       pch = c(16, 17, NA, NA), lty = c(NA, NA, 2, 1), lwd = 2)
```




```
cat("Coeficientes SIN outlier:\n")
```

```
Coeficientes SIN outlier:
```

```
print(coef(model_clean))
```

```
(Intercept)           X
   4.926217    1.986220
```

```
cat("\nCoeficientes CON outlier:\n")
```

```

Coeficientes CON outlier:
```

```
print(coef(model_with_outlier))
```

```
(Intercept)           X
 10.3993867   0.6654753
```