Introducción a R

Diego J. Lizcano

2020-04-17

Índice general

Prólogo

Este libro es una pequeña guía y tutorial sobre como emplear el lenguaje estadistico `R`

Este libro ha sido escrito en R-Markdown empleando el paquete `bookdown`

Este libro está disponible en el repositorio Github: dlizcano/IntroR.

Capítulo 1

Introducción

Este libro es una pequeña guía para la aprender los elementos basicos de la estructura y programación de R.

R es un entorno y lenguaje de programación basado en objetos y enfocado en análisis estadístico y de datos. Vale la pena revisar su historia y evolución en la [pagina de wikipedia](https://es.wikipedia.org/wiki/R_(lenguaje_de_programación).

A lo largo del libro se presentarán códigos que el lector puede copiar y pegar en su consola de R para obtener los mismos resultados aquí del libro. Los códigos se destacan en una caja de color similar a la mostrada a continuación.

```
# R como calculadora
4 + 6
# mi primer objeto
a <- c(1, 5, 6)
# una operación con el objeto
5 * a
# una secuencia
1:10
```

Los resultados o salidas obtenidos de cualquier código se destacan con dos símbolos de númeral (##) al inicio de cada línea o renglón, esto quiere decir que todo lo que inicie con ## son resultados obtenidos y *NO* los debe copiar. Abajo se muestran los resultados obtenidos luego de correr el código anterior.

El usuario habitual de R no hace programación propiamente dicha, sino que utiliza R iterativa e interactivamente: ensaya, se equivoca y vuelve a probar. Solo cuando termina el ciclo y el resultado es satisfactorio, produce un resultado final. Que, usualmente, no es un programa para ejecutar sino, un reporte de resultados o un informe.

A diferencia de otros lenguajes de programación como Python, en R existen muchas y tal vez demasiadas maneras alternativas de hacer las cosas y eso es considerado como un problema por un programador. Y es un problema muy desconcertante para el principiante de R. No obstante, por motivos pedagógicos, el libro tratará de presentar una de las formas de resolver un determinado problema: la que el autor consideró más natural.

1.1 Requisitos

1. Disponer de una versión reciente de R y RStudio. Descargar la última versión.
2. Instalar el paquete `tidyverse`, el cual reune varios paquetes útiles de R:

```
# stable version on CRAN
install.packages("tidyverse")
# or development version on GitHub
# devtools::install_github('rstudio/bookdown')
```

1.2 Primeros pasos

Si se emplea RStudio (recomendado), lo más cómodo sería crear un proyecto nuevo mediante el menú *File > New Project > New Directory > proyecto1*.

Capítulo 2

Creando objetos simples

En R los objetos son la clave.

2.1 Vectores

Los vectores son colecciones de objetos de un solo tipo. La foma mas facil de crear un vector es escribir lo que estará dentro.

```
# usando la funcion concatenar
value_num1 <- c(3,4,2,6,20)
value_num2 <- c(5,10,15,20,25,30,35,50)
value_char <- c("koala","kangaroo","echidna")
logical_1 <- c(F,F,T,T)
logical_2 <- c(FALSE,FALSE,TRUE,TRUE)
vec_num <- 1:15

c(1:5,"a","b","c") # note como los numeros se convierten a texto (encerrado entre comillas)
```

```
## [1] "1" "2" "3" "4" "5" "a" "b" "c"
```

```
# concatenar dos vectores en uno solo
new_vec <- c(value_num1, value_num2)
```

Otra forma es usar funciones como rep y seq

```
# Creating Vectors: rep and seq functions
value <- rep(5,10)
value
```

```
##  [1] 5 5 5 5 5 5 5 5 5 5
```

```
seq(from=2,to=10,by=2)
```

```
## [1]  2  4  6  8 10
```

```
seq(from=2,to=10,length=5)
```

```
## [1]  2  4  6  8 10
```

2.2 Operaciones basicas con vectores

```
# Basic computation with numerical vectors
x <- rnorm(100) # extrae 100 valores al azar de la distribucion normal
y_exp <- 2^x + 1 # exponencial
y_cua <- x^2 + 1 # cuadratica
# grafiquemos
plot(x,y_exp) #graficar con y_cua
```

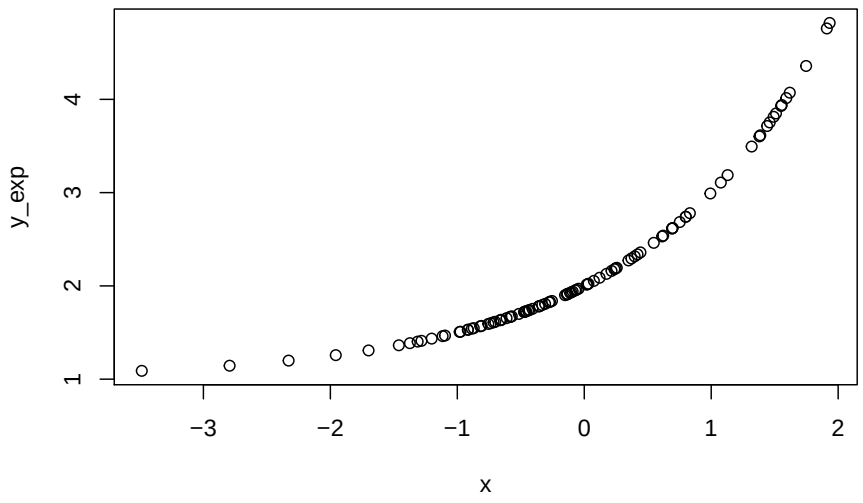


Figura 2.1: figura de una curva exponencial.

```
z <- (x-mean(x))/sd(x)  # see also 'scale'

mean(z) # la media
```

```
## [1] -1.373142e-18
```

```r
sd(y_exp) # la desviación
```

```
## [1] 0.8490138
```

2.3 Inspeccionando vectores

Las siguientes funciones, sirven para inspeccionar el contenido de un vector:

```r
length(x)
table(x)  # ¡muy importante! Cuenta cuantos valores de cada numero
summary(y)
head(x)
tail(x)
```

Para seleccionar elementos de un vector se usa el corchete []. Pero, a diferencia de lo que ocurre con las tablas, como los vectores son objetos unidimensionales, no se usa coma. Obviamente, el corchete sigue admitiendo no solo los índices de los elementos que se quieren extraer, sino que además permite utilizar condiciones lógicas.

```r
x[1:33]
x[c(1,3)]
x[x > 1.5]
x[3:1]
x[-(1:2)]
x[-length(x)]
```

Capítulo 3

Matrices

Las matrices son el sigiente paso luego de los vectores. Las matrices son muy útiles en calculos algebraicos. Ls matrices se crean dimensionando un vector con la funcion dim o con la funcion matrix. Veamos algúnos ejemplos:

```
###################
# Creating Matrices: dim and matrix functions
value <- rnorm(12)
dim(value) <- c(4,3)
value
```

```
##            [,1]       [,2]       [,3]
## [1,] -2.2175846  1.0153142  1.5082344
## [2,] -0.8056742  0.8460231 -1.2255584
## [3,]  0.9610836 -0.4838860  0.3511315
## [4,]  1.5182528 -1.8672941 -0.4553212
```

```
dim(value) <- NULL
matrix(value,2,3)
```

```
##            [,1]      [,2]      [,3]
## [1,] -2.2175846 0.9610836 1.0153142
## [2,] -0.8056742 1.5182528 0.8460231
```

```
matrix(value,2,3,byrow=T)
```

```
##           [,1]       [,2]      [,3]
## [1,] -2.217585 -0.8056742 0.9610836
## [2,]  1.518253  1.0153142 0.8460231
```

```
# Creating Matrices: rbind and cbind functions
value <- matrix(rnorm(6),2,3,byrow=T)
value2 <- rbind(value,c(1,1,2))
```

```
value2
```

```
##             [,1]       [,2]       [,3]
## [1,]  0.6431504 -0.3967127  0.3401161
## [2,] -0.9090011 -0.1844483 -1.7982879
## [3,]  1.0000000  1.0000000  2.0000000
```

```
value3 <- cbind(value2,c(1,1,2))
```

Las matrices son objetos que parecen tablas sin serlo en forma estricta.

Capítulo 4

Tablas

Las tablas en R se conocen como Data Frames.

Muy frecuentemente, los datos se disponen en tablas: las hojas de cálculo, las bases de datos, los ficheros csv, etc. contienen, esencialmente, tablas. Además, casi todos los métodos estadísticos (p.e., la regresión lineal) operan sobre información organizada en tablas. Como consecuencia, gran parte del día a día del trabajo con R consiste en manipular tablas de datos para darles el formato necesario para acabar analizándolos estadística o gráficamente.

Las hojas de cálculo son herramientas que prácticamente todos hemos usado para manipular tablas de datos. Sin embargo R y R studio no permiten la flexibilidad de manejar tablas con el mouse. Asi que hay que aprender algunos comandos basicos para manipular e indexar.

4.1 Crear Data Frames

Para practicar y como ejemplo, usaremos una tabla que viene por defecto con R. Se trata de la tabla iris3. Esta tiene columnas que contienen cuatro características métricas de cada especie de flor: la longitud y la anchura de sus pétalos y sépalos; y la especie: setosa, versicolor o virgínica, a la que pertenecen. De momento y hasta que aprendamos cómo importar datos de fuentes externas, utilizaremos este y otros conjuntos de datos de ejemplo que incluye R.

Ya que es impráctico mostrar tablas enteras en la consola, sobre todo cuando son grandes. Para mostrar solo parte de ellas, hay algunas funciones útiles para inspeccionar tablas (y, como veremos más adelante, no solo tablas), esta son:

```
str (iris)        # "representación textual" del objeto
class (iris)      # "representación de la clase" del objeto
head (iris)    # primeras seis filas
```

```r
tail (iris)    # últimas seis filas
dim (iris)     # filas x columnas
colnames (iris)  # nombre de sus columnas
summary (iris)  # resumen estadístico de las columnas
```

4.2 Indexar data frames

Las tablas se indexan usando el corchete [] y este debe ir separado por una coma [,] donde el primer valor representa las filas y el segundo las columnas, siempre en ese orden. Por ejemplo:

```r
iris[1:5,] # muestra las 5 primeras filas
iris[,1:3] # muestra las 3 primeras columnas
iris[25,4] # muestra el dato de la fila 25 y columna 4

# los corchetes tambien admiten los nombres de las columnas
iris[, "Species"]

## Usando el $ en las tablas
# Pero se usa mas habitualmente (porque es más rápido y más legible) iris$Species en l
iris$Species
```

El corchete también permite seleccionar filas mediante condiciones lógicas. Por ejemplo seleccionemos de la tabla de iris las que corresponden a la especie setosa:

```r
iris[iris$Species == "setosa",]
```

4.2.1 Ejercicio:

Selecciona las filas de iris cuya longitud del pétalo sea mayor que 4.

4.3 Crear una columna nueva

Lo mas practico es usar el nombre de la tabla seguido de $ y el nombre de la nueva columna. Por ejemplo creemos la columna raiz cuadrada de la longitud del petalo

```r
iris$raiz_petalo <- sqrt (iris$Petal.Length) # sqrt es la raiz cuadrada.
```

Tengan en cuenta que:

- agregar una columna que ya existe la reemplaza,
- agregar una columna que no existe la crea y

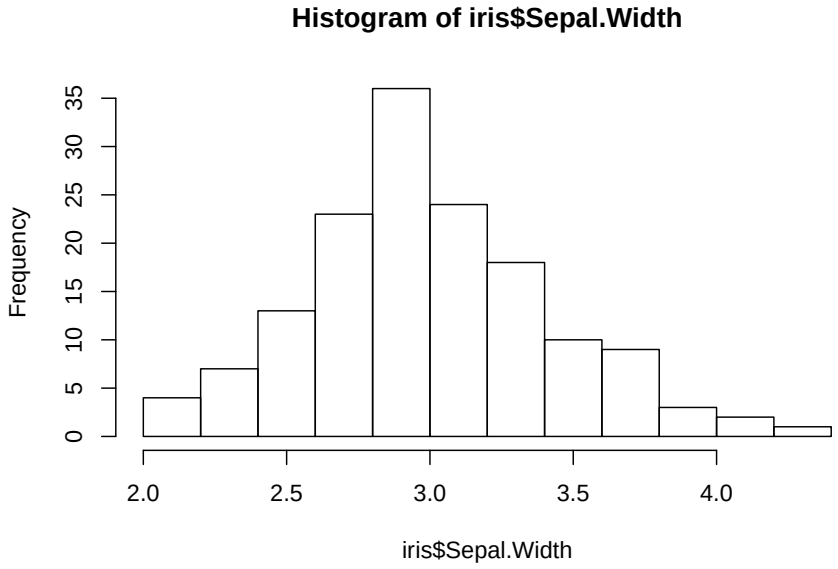
- asignar NULL a una columna existente la elimina.

4.4 Visualizar la tabla graficamente

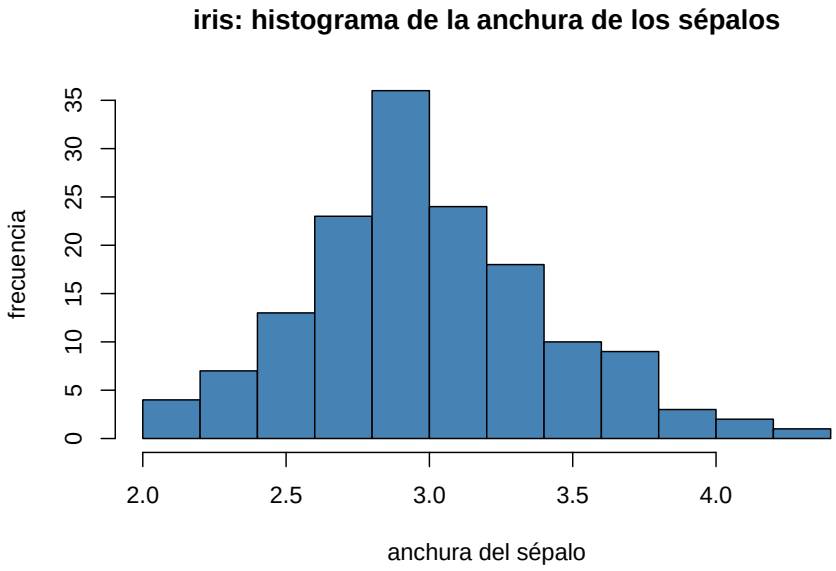
4.4.1 Como histograma

A veces es mucho más informativa una representación visual de los datos. La manera más rápida (y recomendada) de hacerse una idea de la distribución de los datos de una columna numérica es usando histogramas. En R, para representar el histograma de la columna Sepal.Width de iris se puede hacer:

```
hist(iris$Sepal.Width)
```



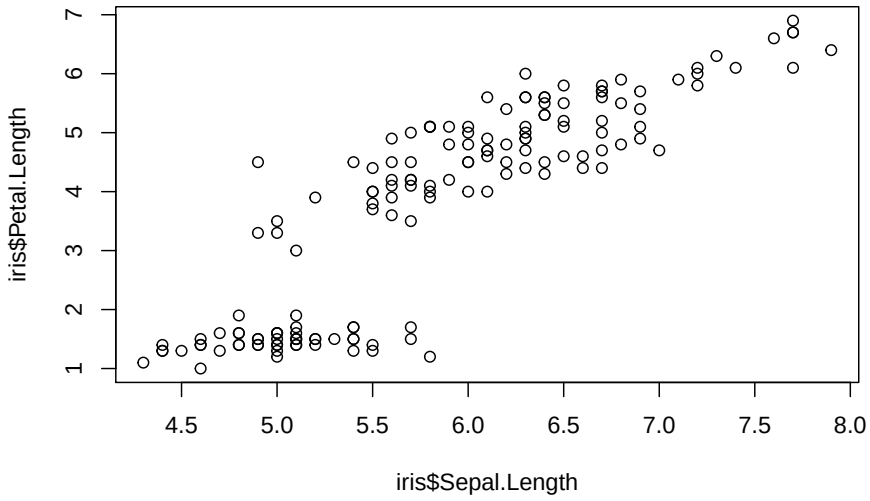

```
#### y para editar el grafico se especifican parametros adicionales
hist(iris$Sepal.Width, main = "iris: histograma de la anchura de los sépalos",
      xlab = "anchura del sépalo", ylab = "frecuencia",
      col = "steelblue")
```



4.4.2 Como dos variables numericas

Por ejemplo representemos la lomgitud del sepalo versus el petalo.

```
plot(iris$Sepal.Length, iris$Petal.Length)
```

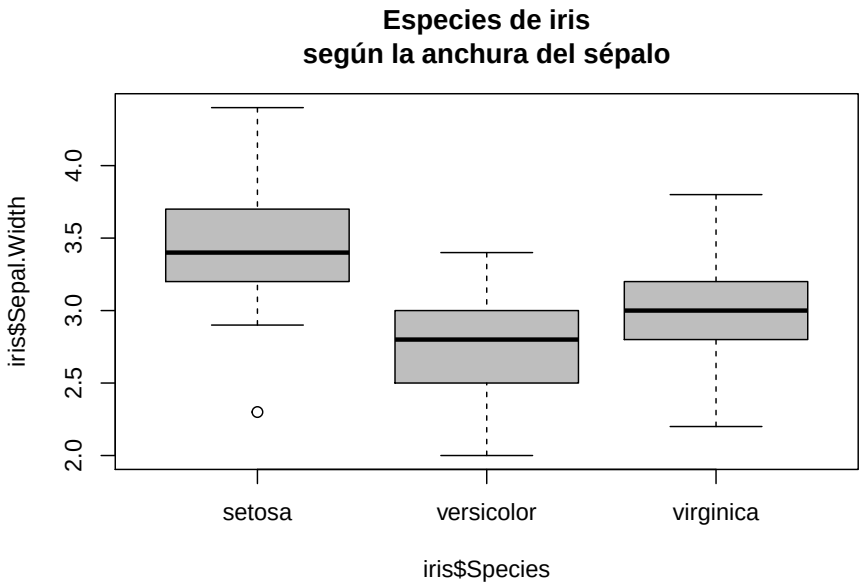


4.4.3 Diagrama de caja (boxplots)

Los diagramas de cajas (boxplot) estudian la distribución de una variable continua en función de una variable categórica. Están emparentados con los histogramas porque resumen la distribución de una variable continua. Para ello utilizan una representación todavía mas esquemática que la de un histograma: una caja y unos segmentos que acotan las regiones donde la variable continua concentra el grueso de las observaciones.

Por ejemplo, podemos estudiar la distribución de la anchura del sépalo en iris en función de la especie usando diagramas de cajas así:

```
boxplot(iris$Sepal.Width ~ iris$Species, col = "gray",
        main = "Especies de iris\nsegún la anchura del sépalo")
```



La notación y ~ x es muy común en R y significa que vas a hacer algo con y en función de x; en este caso, algo es un diagrama de cajas. Cuando construyamos modelos, querremos entender la variable objetivo y en función de una o más variables predictoras y volveremos a hacer uso de esa notación.

Capítulo 5

Creación y e indexación de listas

Las tablas son contenedores de información estructurada: las columnas son del mismo tipo, todas tienen la misma longitud, etc. Gran parte de los datos con los que se trabaja habitualmente son estructurados, palabra que, en la jerga, significa que admiten una representación tabular.

Sin embargo, cada vez es más habitual trabajar directamente con información desestructurada. Particularmente, en ciencia de datos. Eso justifica el uso de las listas, que pueden definirse como contenedores genéricos de información desestructurada.

Las listas son objetos muy versatiles que mezclan varias cosas y sirven como contenedores genericos de dstos e informacón. Estas son una ventaja de R, sobre otros lenguajes como Python que no las tienen. Pero pueden llegar a ser objetos complejos dificiles de entender y visualizar.

```
a <- c(1,2,3,4,5,6,7) # vector numerico
b <- c("Casa", "Carro", "Beca") # vector alfanumerico

florecitas <- iris[3:25,2:5] #data frame

lista1 <- list(a, b, florecitas)
```

las listas disponen de un operador para extraer elementos, los dobles corchetes, [[]], que funcionan de manera parecida y analoga a $.

```
lista1[[2]] # extrae el objeto b
```

```
## [1] "Casa"  "Carro" "Beca"
```

```
lista1[[3]] [,4] # extrae la columna 4 del objeto 3
```

```
##  [1] setosa setosa setosa setosa
##  [5] setosa setosa setosa setosa
##  [9] setosa setosa setosa setosa
## [13] setosa setosa setosa setosa
## [17] setosa setosa setosa setosa
## [21] setosa setosa setosa
## 3 Levels: setosa ... virginica
```

Capítulo 6

Creación de Funciones

Esta sección, por su importancia, pertenece propiamente a la sección de programación. La creación de funciones en lo que sigue del curso es fundemental y, presenta la gran versatilidad de un lenguaje de programación.

Creemos una función que opera sobre un vector que vamos a llamar x

```
media <- function(x){      # inicio de la funcion
 longitud <- length(x)
 suma <- sum(x)
 return (suma / longitud) # devuelve el resultado de la media
}                          # final de la funcion


# creemos un vector
vector1 <- rnorm (100) # 100 datos al azar de la distribucion normal

# apliquemos la funcion al vector
media(vector1)
```

```
## [1] -0.03054159
```

Capítulo 7

Creación de bucles

En la programación en R es habitual construir loops o bucles dentro de los cuales se va modificando el valor de una expresión. Los bucles más habituales en R comienzan con for. Su sintaxis es

for (var in vector){ # expresión que se repite }

Ejemplo: construyamos un bucle que repite la impresion de un nombre 10 veces

```
for (i in 1:10){      # variable i en un vector de 1 a 10
  print ("Carlos")   # expresión que se repite
}                     # final del bucle
```

```
## [1] "Carlos"
## [1] "Carlos"
## [1] "Carlos"
## [1] "Carlos"
## [1] "Carlos"
## [1] "Carlos"
## [1] "Carlos"
## [1] "Carlos"
## [1] "Carlos"
## [1] "Carlos"
```

Hagamos un bucle mas interesante que simula datos de pesencia ausencia obtenidos al azar (con probabilidad 0.5) de la distibucion binomial, para un estudio de ocupacion con 15 sitios y cuatro visitas repetidas a cada sitio.

```
sitios  <- 15
visitas <- 4
datos <- matrix(NA, 15,4) # matriz vacia donde vamos a poner los datos

for (i in 1:sitios){      # variable i en un vector de 1 a 10
```

```
  y <- rbinom(visitas, 1, 0.5) # 0.5 es la probabilidad
  datos [i,] <- y
}

datos
```

```
##       [,1] [,2] [,3] [,4]
##  [1,]    0    0    1    1
##  [2,]    1    1    0    1
##  [3,]    1    1    0    1
##  [4,]    1    0    1    1
##  [5,]    1    0    0    1
##  [6,]    1    1    0    1
##  [7,]    0    1    0    1
##  [8,]    1    0    1    0
##  [9,]    0    1    0    0
## [10,]    1    1    0    1
## [11,]    1    0    1    1
## [12,]    1    0    1    0
## [13,]    0    0    0    1
## [14,]    1    0    0    1
## [15,]    1    0    1    0
```

7.0.1 Ejercicio:

Crear una matriz de datos simulados, de un estudio donde se cuentan renacuajos en 20 sitios con cinco visitas repetidas a cada sitio.

7.1 Convirtamolo en función

```
genera_datos <- function(sitios=15, visitas=4, probabilidad=0.5) {
  datos <- matrix(NA, sitios, visitas) # matriz vacia donde vamos a poner los datos

  for (i in 1:sitios){     # variable i
    y <- rbinom(visitas, 1, probabilidad) # dados guardados en y
    datos [i,] <- y       # y pasa a la fila i de la tabla de datos
  }
  return(datos) # muestra los datos al finalizar el bucle
}
```

```
# llamemos la funcion
genera_datos() #con los valores por defecto
```

```
##       [,1] [,2] [,3] [,4]
##  [1,]    1    0    0    0
##  [2,]    1    1    1    0
##  [3,]    0    1    1    0
##  [4,]    0    0    1    0
##  [5,]    0    1    1    0
##  [6,]    1    0    1    0
##  [7,]    0    1    1    0
##  [8,]    0    0    1    0
##  [9,]    1    1    0    1
## [10,]    1    1    0    0
## [11,]    0    1    1    0
## [12,]    0    0    0    1
## [13,]    1    0    0    0
## [14,]    1    0    0    1
## [15,]    1    0    0    0
```

```
genera_datos(30, 6, 0.5) # con 30 sitios, 6 visitas repetidas y probabilidad 0.5
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6]
##  [1,]    0    1    0    0    1    1
##  [2,]    0    0    0    1    0    0
##  [3,]    0    1    1    1    1    1
##  [4,]    1    0    1    0    0    0
##  [5,]    1    1    1    1    1    1
##  [6,]    0    1    1    1    0    1
##  [7,]    0    1    0    0    0    1
##  [8,]    0    0    0    0    1    1
##  [9,]    1    0    0    1    0    1
## [10,]    1    0    1    1    0    1
## [11,]    0    0    0    1    1    0
## [12,]    0    1    0    1    1    0
## [13,]    0    1    1    0    0    0
## [14,]    1    1    1    0    0    1
## [15,]    0    1    1    0    0    0
## [16,]    0    1    1    0    1    1
## [17,]    1    1    1    0    1    1
## [18,]    0    0    1    0    1    1
## [19,]    0    1    0    0    1    1
## [20,]    1    1    0    1    0    0
## [21,]    0    1    0    0    0    1
## [22,]    0    1    0    0    1    0
## [23,]    1    0    1    0    0    1
```

```
## [24,]    1    1    1    0    1    1
## [25,]    1    1    0    1    1    1
## [26,]    0    1    0    1    0    1
## [27,]    0    0    1    1    1    0
## [28,]    1    0    0    1    1    1
## [29,]    0    1    0    0    1    1
## [30,]    1    0    1    0    1    0
```

Capítulo 8

Modelos en R

Los modelos son una simplificación y abstracción de un sistema real que nos permite entender un proceso y/o sus resultados. Adicionalmente permiten entender como interactuan o se afectan los parametros si variamos algo dentro del modelo.

8.1 Modelo de la media

Aunque este primer modelo no es un modelo como tal, el siguiente código nos permite entender como el tamaño de la muestra afecta la distribución de la media. Para esto usaremos el set de datos iris de R y la columna largo del pétalo. Lo que haremos será extraer al azar un numero (n) de valores del vector largo del pétalo y calcular la media cien veces. Con estos valores construiremos un histograma.

Haremos una comparación grafica de la forma del histograma con la media (original) del vector largo del pétalo, el cual representaremos con color azul.

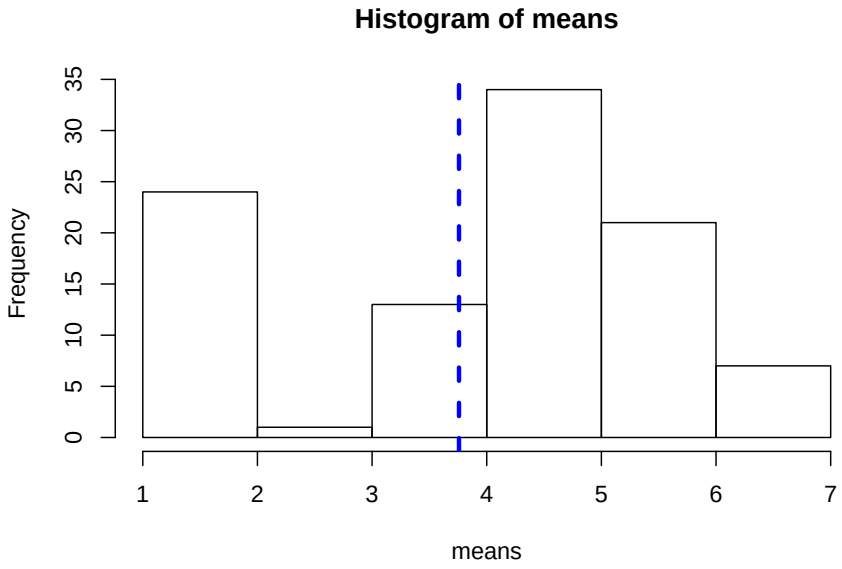
```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
iter <- 100 # Numero de iteraciones
n <- 1 # numero de datos que muestreo. Variar hasta 150
means <- rep(NA, iter) # aca almaceno las medias de cada iteracion
```

```r
for (i in 1:iter){
  d <- sample(iris$Petal.Length, n) # sample toma n (1) datos
  means[i] <- mean(d) # saca la media y guarda en means
}

hist(means)
abline(v=mean(iris$Petal.Length), lty=2, lwd=3, col="blue")
```



Ahora repitamos el mismo código 10 veces para ver como varia el histograma, ya que estamos seleccioando valores al azar.

Como siguiente paso cambiemos el valor de n a dos (n <- 2), luego a tres, luego cuatro, cinco… etc.

8.1.1 Ejercicio 1

Discuta como varia la forma del histograma en la medida en que el valor de n aumenta.

8.1.2 Ejercicio 2

Convierta el código en una funcion de forma tal que pueda variar el valor de n para producir las graficas en una sola linea.

Capítulo 9

Modelo de regresión lineal

Recordemos el algebra del modelo de regresión lineal:

$$y = \alpha + \beta x + \epsilon \tag{9.1}$$

Donde:

α y β son parámetros del modelo y ϵ el error del modelo. α corresponde al intercepto β corresponde al coeficiente de x (pendiente). cuando $\beta = 0$, no hay relación significativa entre las variables x y y.

9.1 Regresión lineal simple

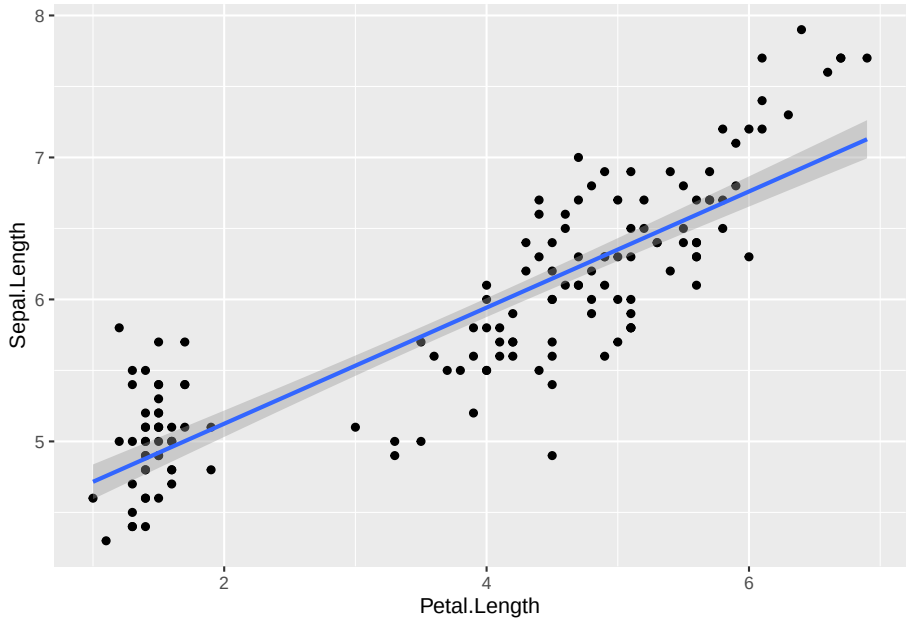
Usaremos el set de datos iris para realizar una regresión lineal entre la longitud del pétalo y la longitud del sépalo.

```
model1 <- lm(Sepal.Length~Petal.Length, data=iris)
summary(model1)
```

```
##
## Call:
## lm(formula = Sepal.Length ~ Petal.Length, data = iris)
##
## Residuals:
##      Min       1Q   Median       3Q
## -1.24675 -0.29657 -0.01515  0.27676
##      Max
##  1.00269
##
## Coefficients:
```

```
##              Estimate Std. Error
## (Intercept)   4.30660    0.07839
## Petal.Length  0.40892    0.01889
##              t value Pr(>|t|)
## (Intercept)    54.94   <2e-16 ***
## Petal.Length   21.65   <2e-16 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05
##   '.' 0.1 ' ' 1
##
## Residual standard error: 0.4071 on 148 degrees of freedom
## Multiple R-squared:   0.76,  Adjusted R-squared:  0.7583
## F-statistic: 468.6 on 1 and 148 DF,  p-value: < 2.2e-16
```

```
library(ggplot2)
ggplot(model1, aes(x=Petal.Length, y=Sepal.Length)) + geom_point() +
  geom_smooth(method = "lm") # try geom_smooth()
```



9.1.1 Ejercicio:

Discuta e interprete los coeficientes del modelo.

Ahora utilizaremos el modelo que hemos calculado para predecir cómo se comporta la longitud del sépalo cuando la longitud del pétalo esta entre 2 y 4.

```
newdato <- data.frame (Petal.Length = seq (2, 4, by = 0.1))
predict(model1, newdata = newdato) # predice sepalo cuando petalo es de 4 a 7
```

```
##        1        2        3        4
## 5.124448 5.165340 5.206232 5.247125
##        5        6        7        8
## 5.288017 5.328909 5.369801 5.410694
##        9       10       11       12
## 5.451586 5.492478 5.533370 5.574262
##       13       14       15       16
## 5.615155 5.656047 5.696939 5.737831
##       17       18       19       20
## 5.778724 5.819616 5.860508 5.901400
##       21
## 5.942293
```

9.1.2 Ejercicio:

Use el modelo para predecir el sépalo cuando el pétalo tiene un valor de 7 o mas

9.2 Regresión lineal con 2 o más predictores

Ahora haremos las cosas un poco más complejas (sin que sean difíciles) para entender que sucede cuando hay dos o más predictores de la forma:

$$y = \alpha + \beta_1 x + \beta_2 x + \epsilon, \tag{9.2}$$

Usaremos mas (+) para combinar efectos. Dos puntos (:) para interacciones A:B; y asterisco para efectos e interacciones, ej A*B = A+B+A:B

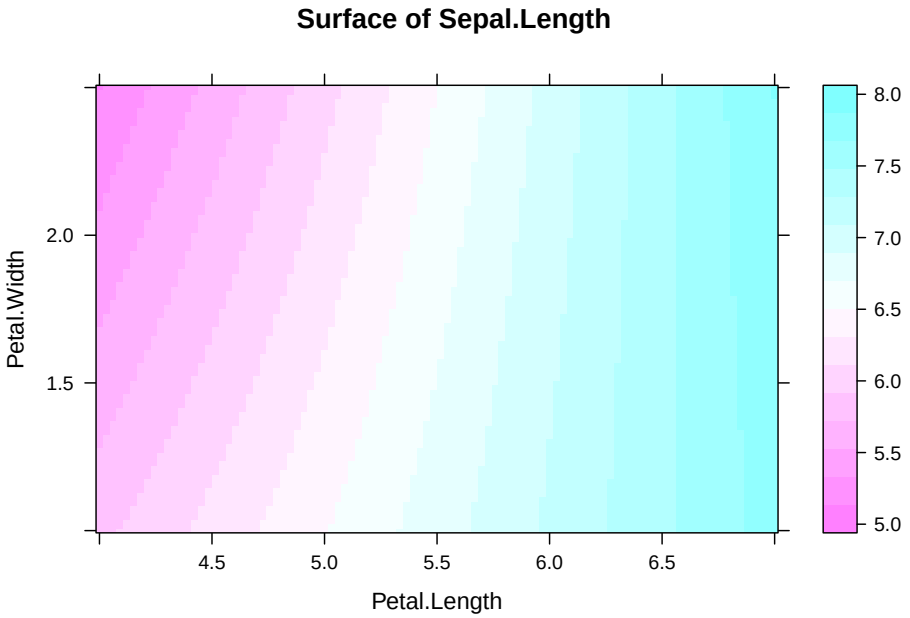
```
model1a <- lm(Sepal.Length~Petal.Length * Petal.Width, data=iris)
summary(model1a)
```

```
##
## Call:
## lm(formula = Sepal.Length ~ Petal.Length * Petal.Width, data = iris)
##
## Residuals:
##      Min       1Q   Median       3Q
## -1.00058 -0.25209  0.00766  0.21640
##      Max
##  0.89542
```

```
##
## Coefficients:
##                            Estimate
## (Intercept)                 4.57717
## Petal.Length                0.44168
## Petal.Width                -1.23932
## Petal.Length:Petal.Width    0.18859
##                            Std. Error
## (Intercept)                   0.11195
## Petal.Length                  0.06551
## Petal.Width                   0.21937
## Petal.Length:Petal.Width      0.03357
##                            t value
## (Intercept)                 40.885
## Petal.Length                 6.742
## Petal.Width                 -5.649
## Petal.Length:Petal.Width     5.617
##                            Pr(>|t|)
## (Intercept)                 < 2e-16 ***
## Petal.Length               3.38e-10 ***
## Petal.Width                8.16e-08 ***
## Petal.Length:Petal.Width   9.50e-08 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05
##   '.' 0.1 ' ' 1
##
## Residual standard error: 0.3667 on 146 degrees of freedom
## Multiple R-squared:  0.8078, Adjusted R-squared:  0.8039
## F-statistic: 204.5 on 3 and 146 DF,  p-value: < 2.2e-16
```

```
library(lattice)
newdato<-expand.grid(list(Petal.Length = seq(4, 7, length.out=100),
                          Petal.Width=seq(1, 2.5, length.out=100)))
newdato$Sepal.Length<-predict(model1a, newdata = newdato) # predice sepalo con petalo

levelplot(Sepal.Length~Petal.Length + Petal.Width, data=newdato,
  xlab = "Petal.Length", ylab = "Petal.Width",
  main = "Surface of Sepal.Length")
```



9.2.1 Ejercicio:

Cambie el rango de la predicción de 2 a 8 Cambie en el modelo la interacion de las covariables a +

Capítulo 10

Modelos de distribución

En esta sección introduciremos algunos elementos matemáticos que no deben intimidarnos! Si bien las ecuaciones son aparentemente complejas el principio que hay detrás es muy elemental y se pude dominar mejor viendo los videos sugeridos.

10.1 Binomial

Recordemos la ecuación de la distribución Binomial:

$$P(x) = \frac{n!}{k!\,(n-k)!} p^k q^{n-k} = \left(\begin{array}{c} n \\ k \end{array} \right) p^k q^{n-k} \tag{10.1}$$

Donde: n es el número de ensayos. k es el número de éxitos. p es la probabilidad de exíto en un ensayo. $q = 1 - p$ es la probabilidad de fracasar en un ensayo.

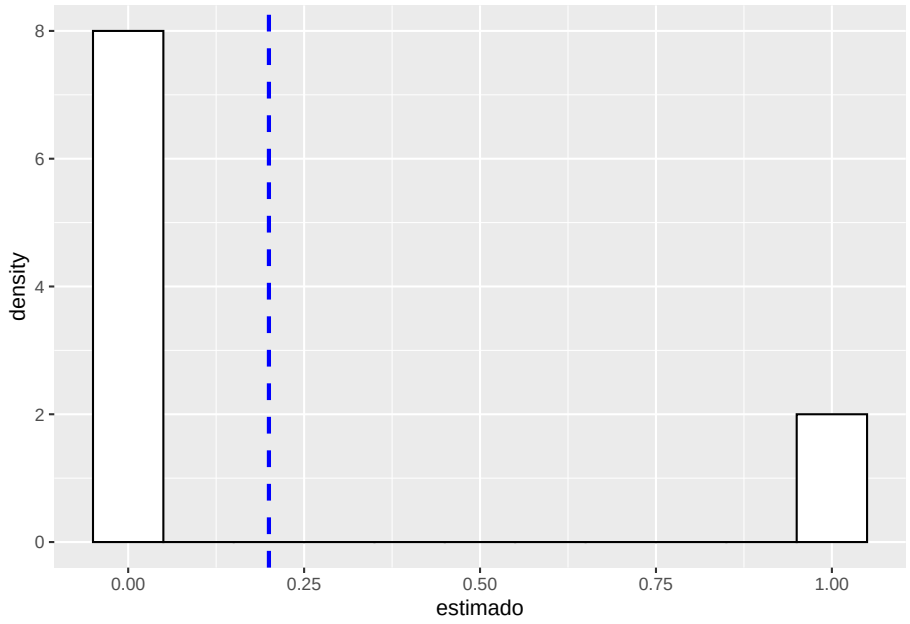
Por favor *no se asuste* por la aparente complejidad de la ecuación. Para simplificar la interpretación usaremos en lugar de la ecuación, el álgebra del modelo. Si desea aprender un poco mas de la distribución binomial le recomiendo el video que ha preparado Khan Academy. con un interesante acento mexicano.

$$P(x) \sim \mathbf{Bin}(n, p) \tag{10.2}$$

$P(x)$ es la probabilidad de un valor especifico de x, el cual se distribuye de forma binomial Bin con los parámetros: n que corresponde al número de ensayos y p a la probabilidad.

A continuación, vamos a generar un set de n datos con una probabilidad dada p. Estos datos son almacenados en un dataframe y luego visualizados como una distribución de frecuencias. La línea azul representa el promedio de esos datos.

```
n<-10 # numero de datos
p<- 0.5 # probabilidad (~proporcion de unos)
# Generemos datos con esa informacion
daber<-data.frame(estimado=rbinom(n, 1, p))
# Grafiquemos
library(ggplot2)
ggplot(daber, aes(x=estimado)) +
    geom_histogram(aes(y=..density..), # Histograma y densidad
                   binwidth=.1, # Ancho del bin
                   colour="black", fill="white") +
        geom_vline(aes(xintercept=mean(estimado, na.rm=T)),
          color="blue", linetype="dashed", size=1) # media en azul
```



10.1.1 Ejercicio:

Varié primero el número de datos y luego la probabilidad haciéndola cercana a cero y luego cercana a uno. Vea como cambia el promedio con el número de datos y la probabilidad.

10.2 Poisson

Recordemos la ecuación de la distribución Binomial:

$$P\left(x\right) = \frac{e^{-\mu}\mu^{x}}{x!} \tag{10.3}$$

Donde:

μ es el número medio de ocurrencia (éxitos) en un intervalo en particular. e es la constante 2.71828 (base of the Naperian logarithmic system). x es el numero de ocurrencias (éxitos). $P(x)$ es la probabilidad de un valor especifico de x.

De igual forma que en la distribución binomial, no nos dejemos intimidar por la complejidad de la ecuación y usemos el algebra del modelo que es mucho más sencilla. Si desea entender un poco mejor el proceso Poisson vean el video que preparó Khan Academy.

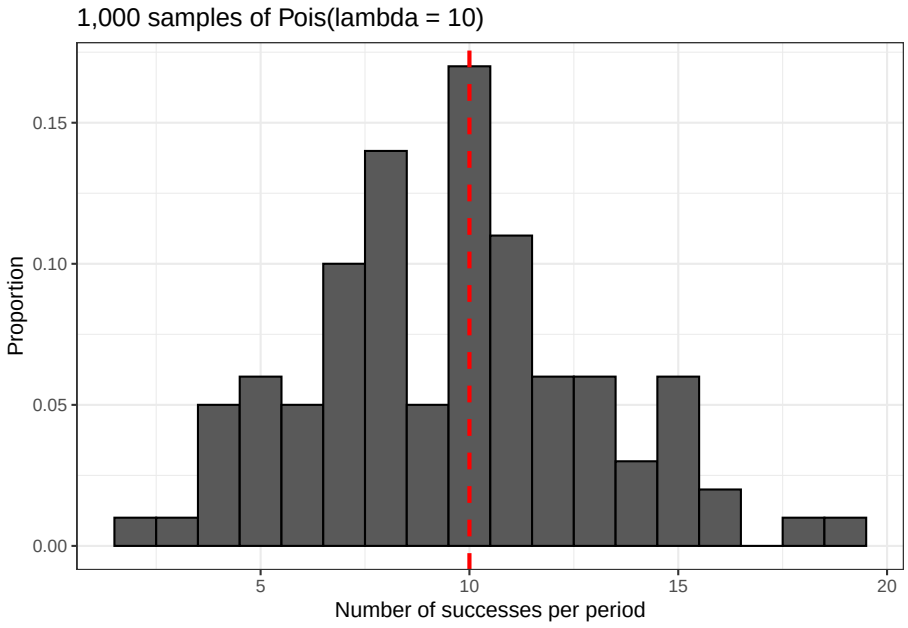
$$P\left(x\right) \sim \textbf{Pois}(\lambda) \tag{10.4}$$

Donde:

$P(x)$ es la probabilidad de un valor especifico de x, el cual se distribuye de acuerdo a la distribución Poisson ($Pois$) y que equivaldría a un valor específico del conteo. λ es la media de la distribución.

```
n <- 100
lambda <- 10
poisson_data <- data.frame('data' = rpois(n, lambda))


library(tidyverse)
poisson_data %>% ggplot() +
  geom_histogram(aes(x = data,
                     y = stat(count / sum(count))),
                     color = 'black',
                 binwidth = 1) +
  geom_vline(xintercept = lambda,
             size = 1,
             linetype = 'dashed',
             color = 'red') +
  theme_bw() +
  labs(x = 'Number of successes per period',
       y = 'Proportion',
       title = '1,000 samples of Pois(lambda = 10)')
```



10.2.1 Ejercicio:

Use el modelo para variar n y lambda… vea que sucede.

10.3 Normal (Gausiana)

La ecuación de la distribución Binomial es:

$$P\left(x\right) = \frac{e^{-(x-\mu)^2/(2\sigma^2)}}{\sigma\sqrt{2\pi}} \tag{10.5}$$

Donde:

$P(x)$ es la probabilidad de un valor especifico de x, μ es la media de la distribución. σ es la desviación estandard π es 3.14159 e es 2.71828

De igual forma que en las distribuciónes anteriores, no nos dejemos intimidar por la complejidad de la ecuación y usemos el algebra del modelo que es mucho más sencilla.

$$P\left(x\right) \sim \mathcal{N}(\mu,\, \sigma^2) \tag{10.6}$$

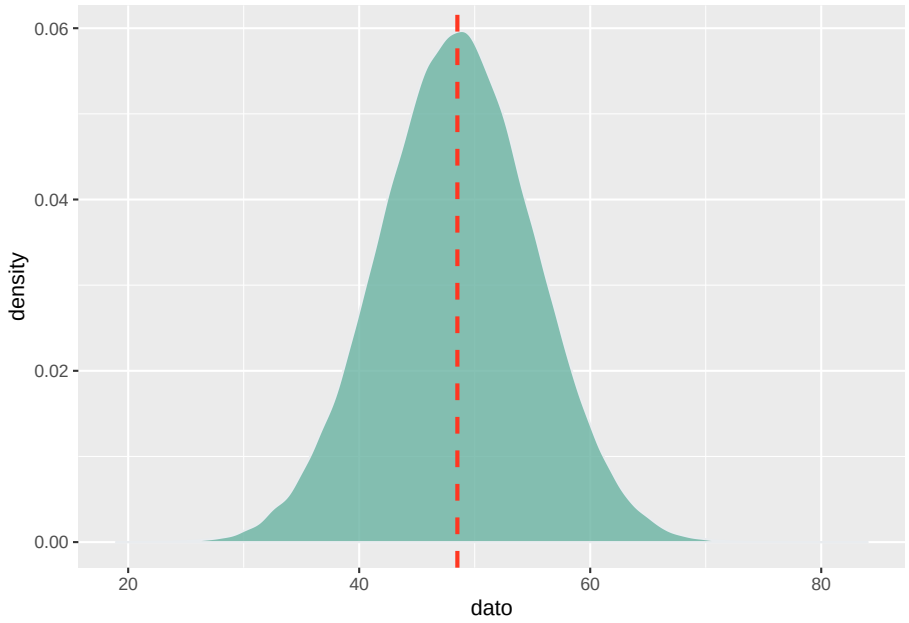
$P(x)$ es la probabilidad de un valor especifico de x, por lo general un valor continuo. μ es la media de la distribución. σ es la desviación estandard

Veamola graficamente:

```
n <- 100000
mean <- 48.5
sd <- 6.7

# Create a sample of 50 numbers which are normally distributed.
y <- data.frame("dato"=rnorm(n, mean, sd))
# Plot the histogram for this sample.
# library(lattice)
# densityplot(y$dato, main = "Normal DIstribution")

ggplot( aes(x=dato), data = y) +
    geom_density(fill="#69b3a2", color="#e9ecef", alpha=0.8) +
    geom_vline(xintercept = mean, size = 1, colour = "#FF3721",
               linetype = "dashed")
```



10.3.1 Ejercicio:

Use el modelo para variar n y la media y la desviación, vea que sucede y discuta.