Tema 5. Remuestreos en Modelos Lineales y Series Temporales

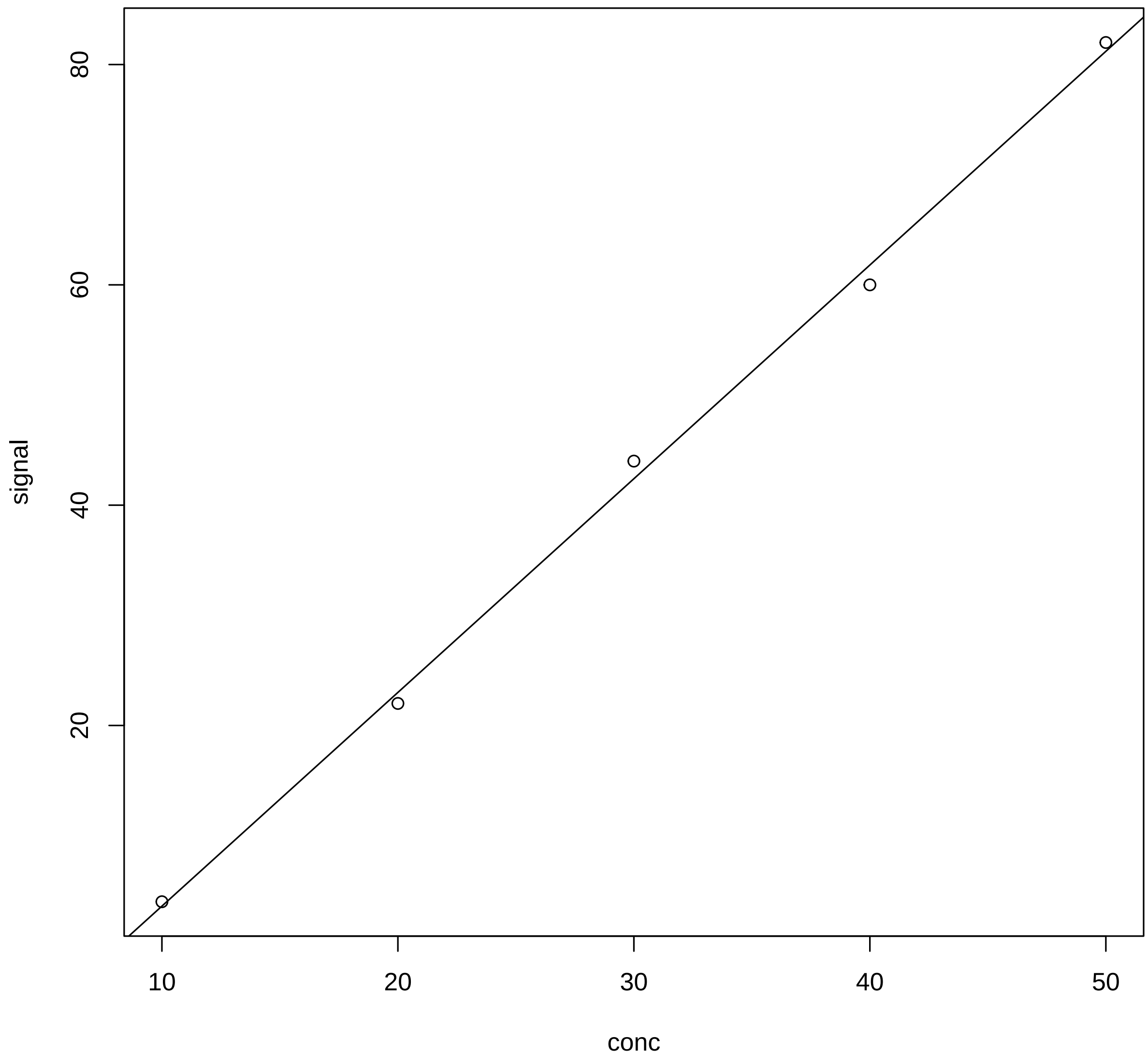
Introducción a la Regresión Lineal con R

Supongamos un ejemplo muy simple sobre dos vectores de datos:

```
conc = c(10, 20, 30, 40, 50)
signal = c(4, 22, 44, 60, 82)

lm.r = lm(signal ~ conc)
summary(lm.r)
```

```

Call:
lm(formula = signal ~ conc)

Residuals:
   1    2    3    4    5 
 0.4 -1.0  1.6 -1.8  0.8 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) -15.80000    1.66933  -9.465   0.0025 ** 
conc          1.94000    0.05033  38.544 3.84e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.592 on 3 degrees of freedom
Multiple R-squared:  0.998, Adjusted R-squared:  0.9973 
F-statistic:  1486 on 1 and 3 DF,  p-value: 3.842e-05
```

```
plot(conc, signal)
abline(lm.r)
```



Coeficientes y residuos del modelo

```
coef(lm.r)
```

```
(Intercept)        conc
     -15.80        1.94
```

```
resid(lm.r)
```

```
   1    2    3    4    5 
 0.4 -1.0  1.6 -1.8  0.8 
```

Valores predichos

```
fitted(lm.r)
```

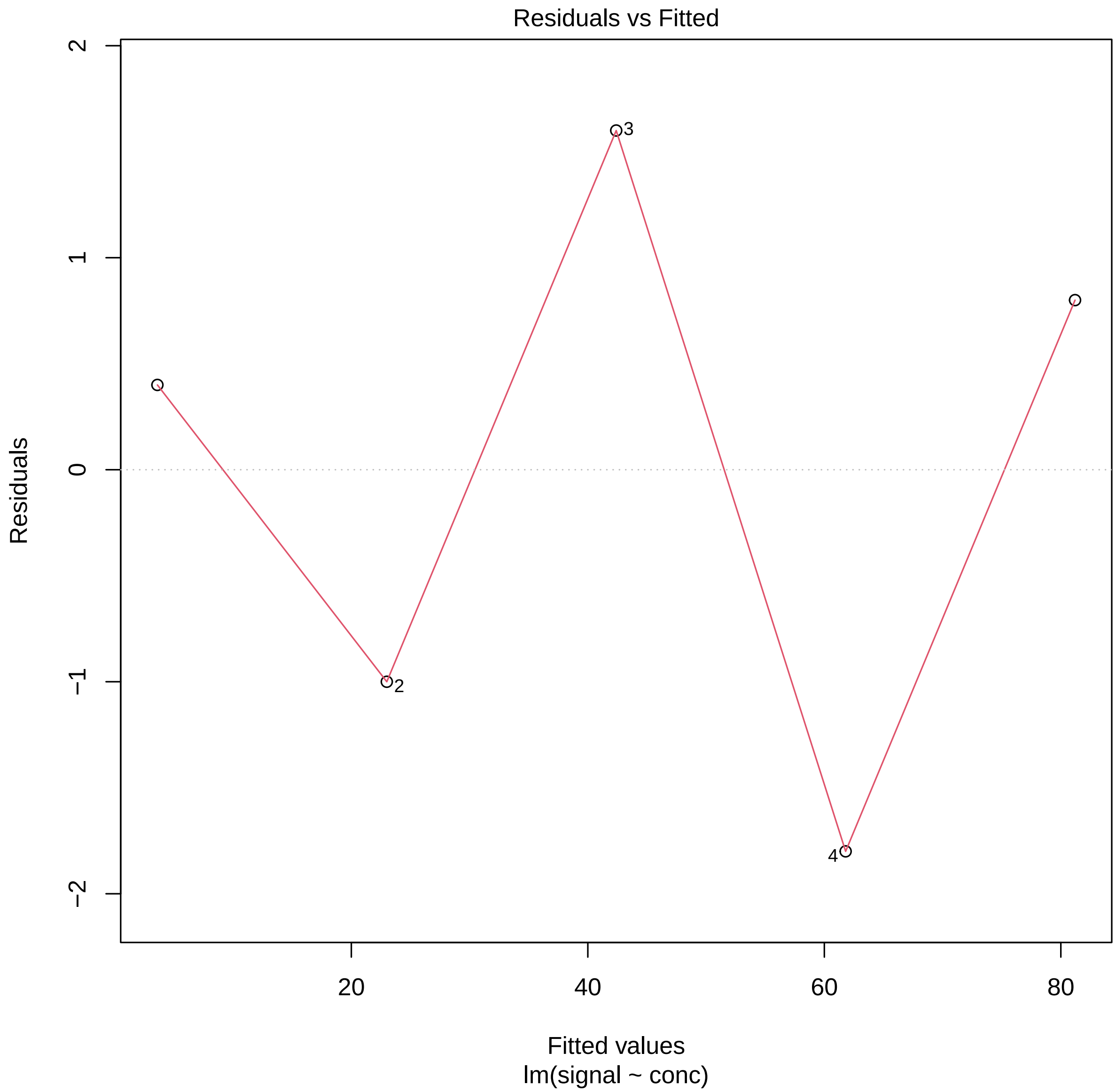
```
   1    2    3    4    5 
 3.6 23.0 42.4 61.8 81.2 
```

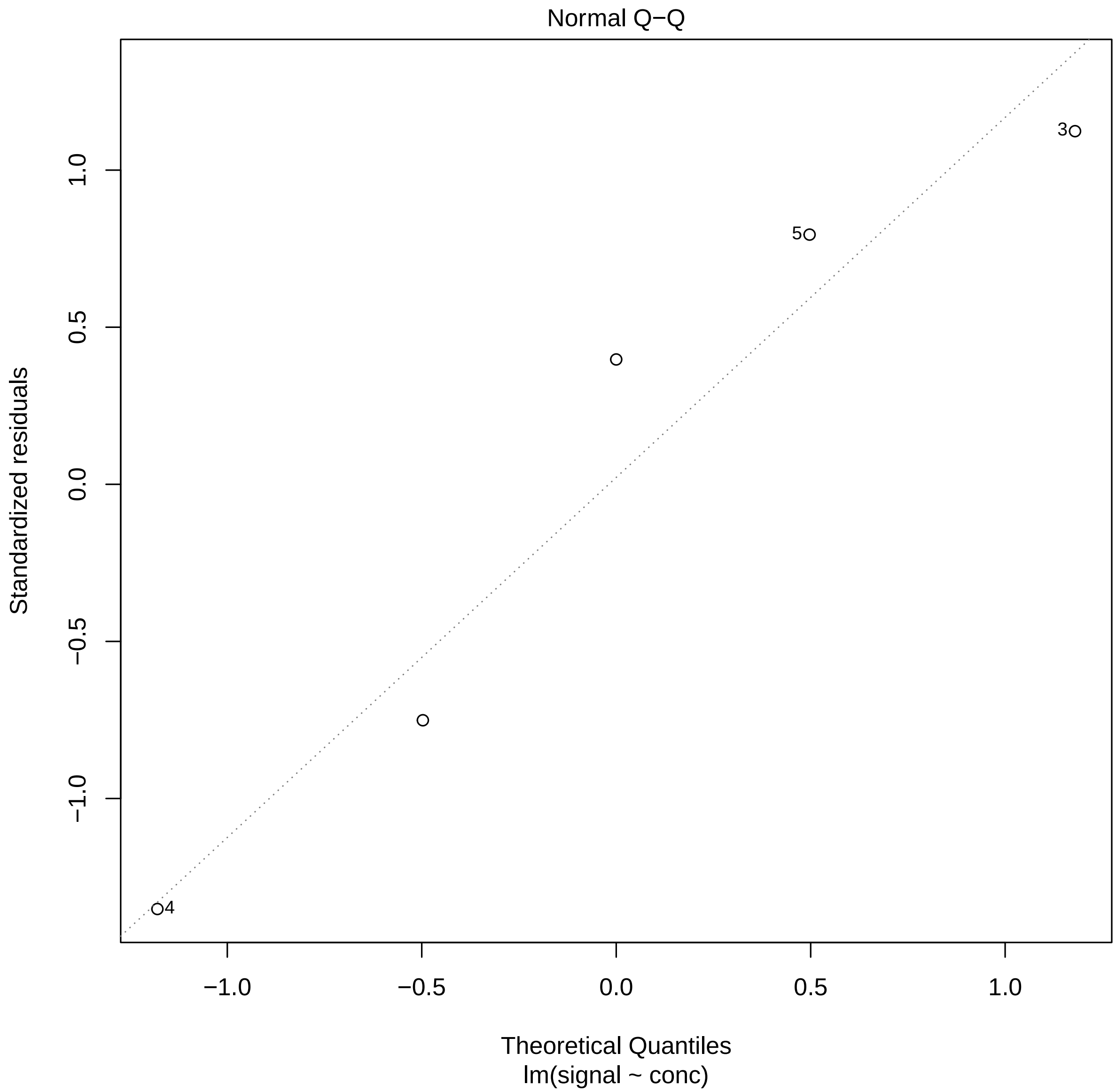
Intervalos de confianza para los parámetros

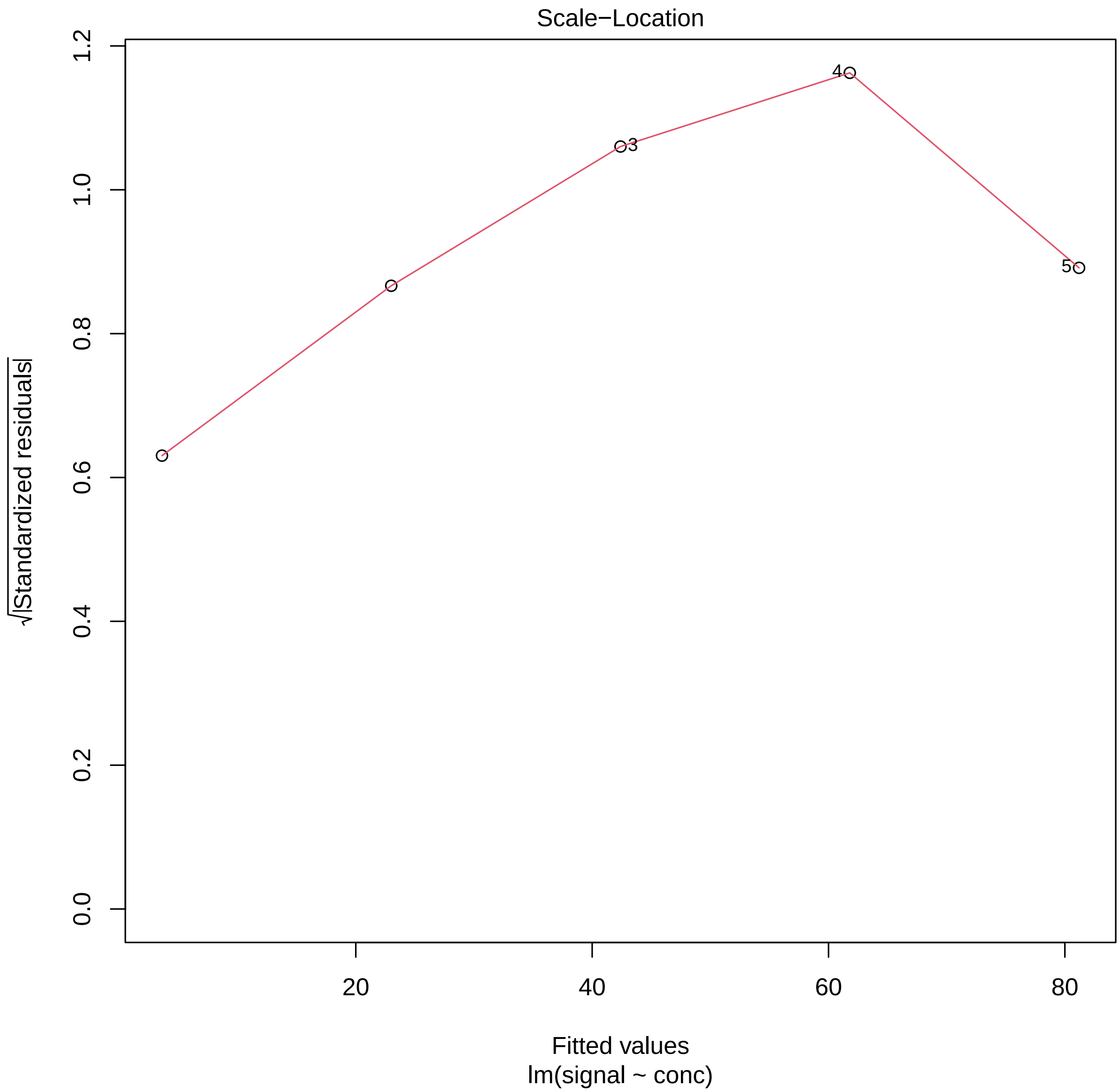
```
confint(lm.r)
```

```
                2.5 %    97.5 %
(Intercept) -21.11256 -10.48744
conc          1.77982   2.10018
```

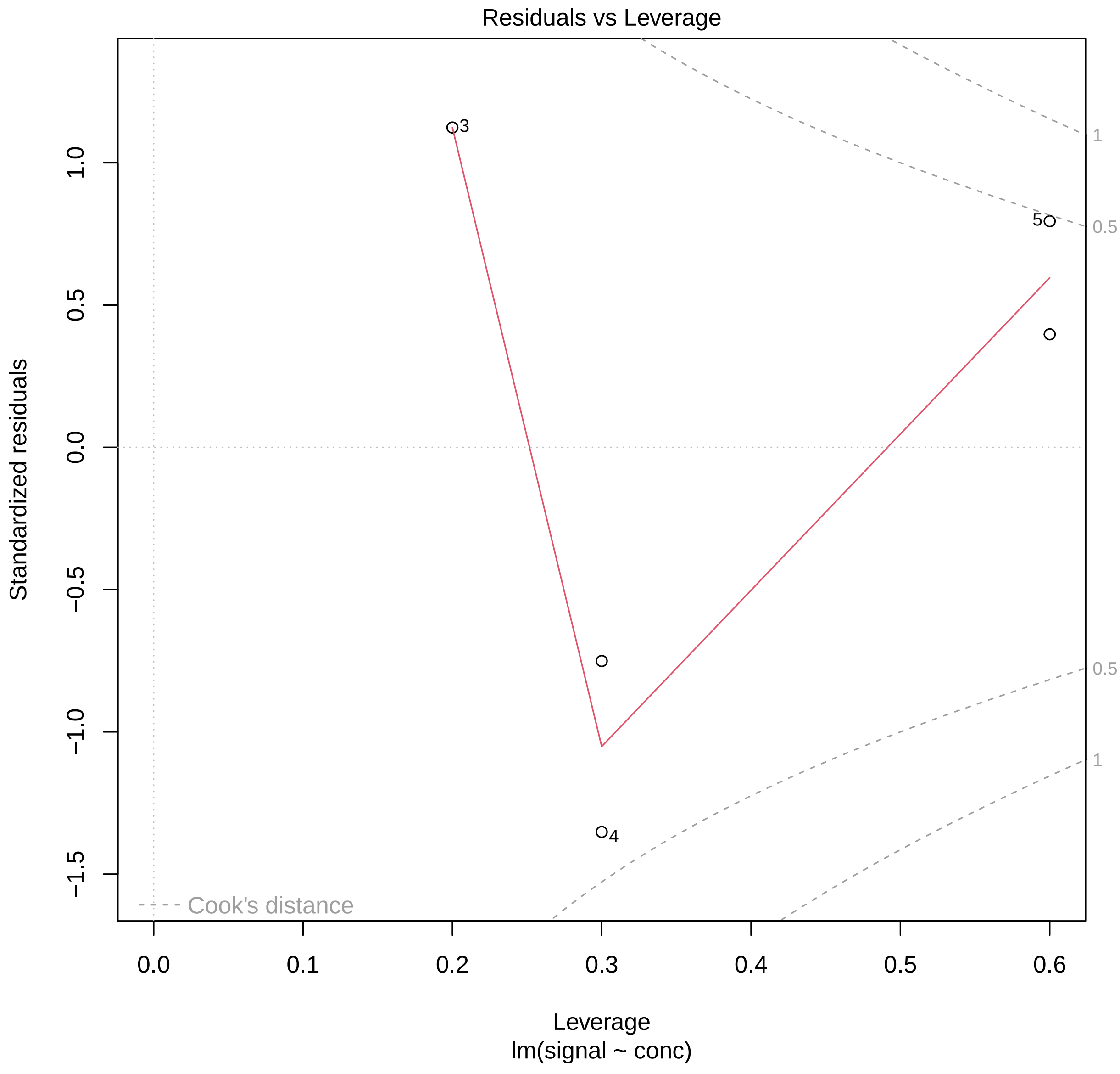
```
# layout(matrix(1:4,2,2))
plot(lm.r)
```


Residuals vs Fitted
Residuals
2
1
0
−1
−2
3
2
4
20
40
60
80
Fitted values
lm(signal ~ conc)


Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(signal ~ conc)
−1.0
−0.5
0.0
0.5
1.0
3
5
4



Scale−Location
1.2
1.0
0.8
0.6
0.4
0.2
0.0
√|Standardized residuals|
3
4
5
20
40
60
80
Fitted values
lm(signal ~ conc)



Predicción de nuevas observaciones

```
newconc = c(5, 15, 25, 35, 45)

predict(lm.r, data.frame(conc = newconc), level = 0.9, interval = "confidence")
```

```
   fit       lwr       upr
1 -6.1 -9.502218 -2.697782
2 13.3 10.858090 15.741910
3 32.7 30.923250 34.476750
```

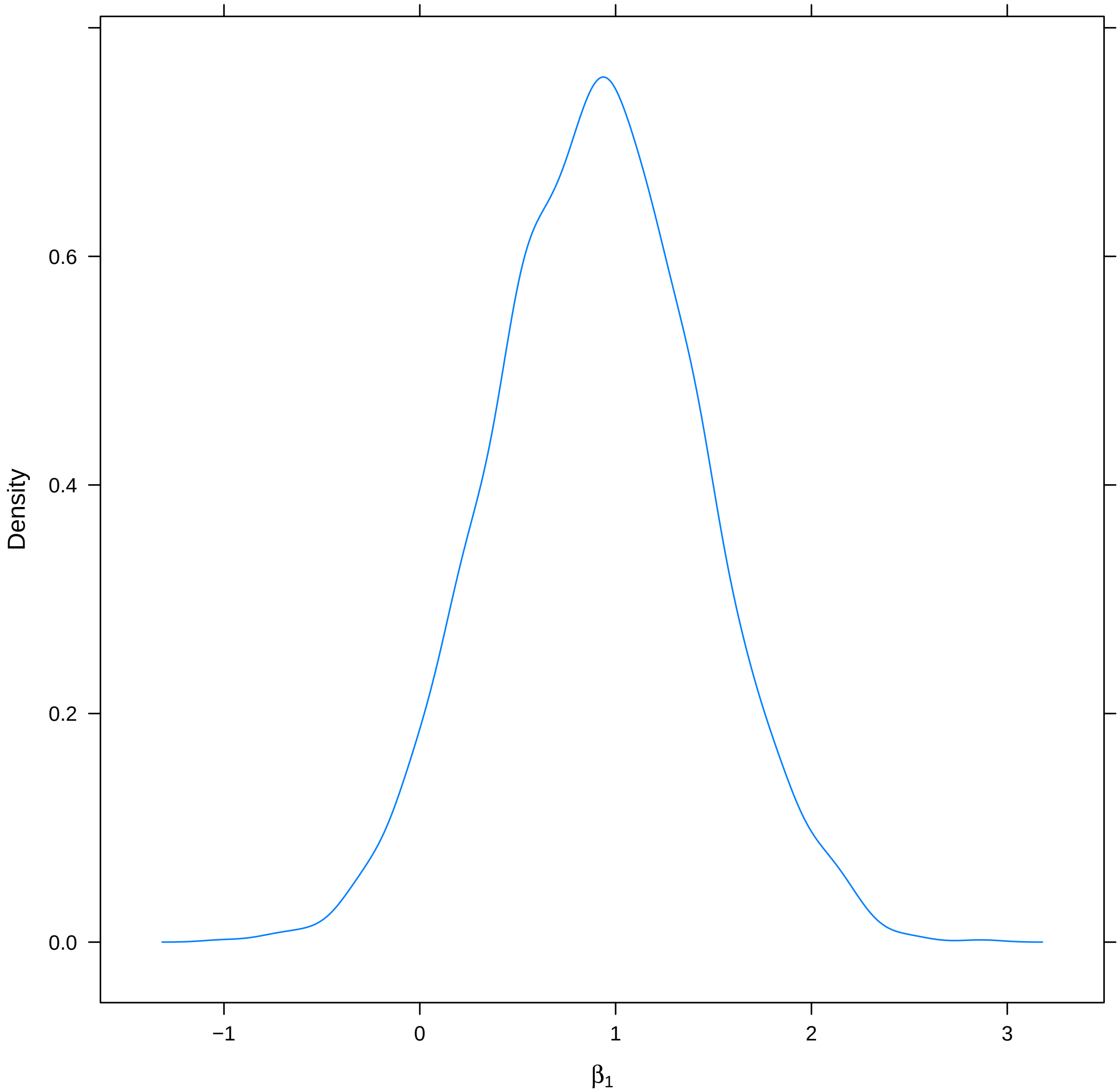
```
4 52.1 50.323250 53.876750
5 71.5 69.058090 73.941910
```

Ejemplo de Regresión bootstrap con residuos

Se simula un modelo de regresión lineal con errores distribuidos según una normal

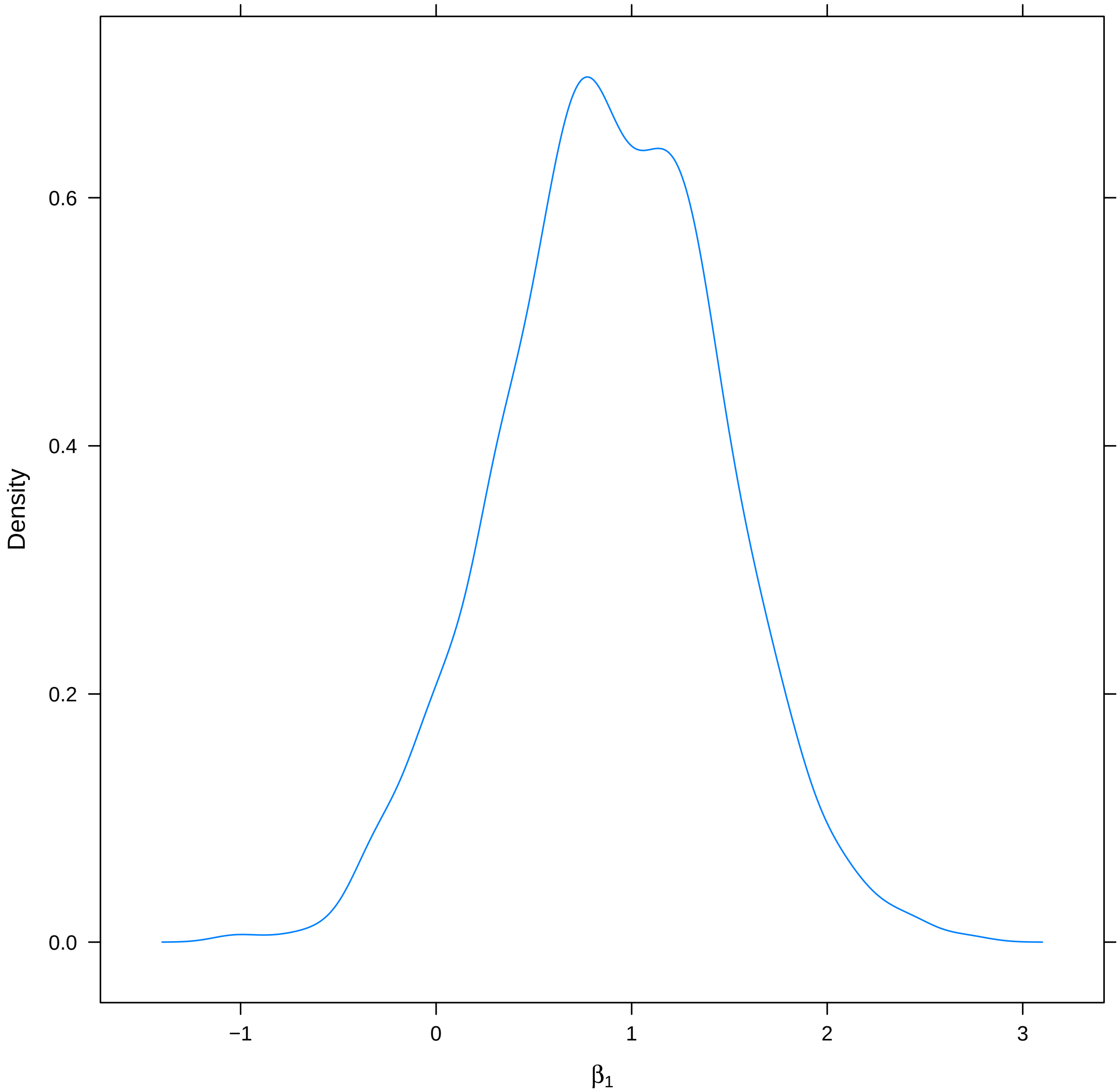
```
N = 15
sd = 1.5
x = rnorm(N)
y = 3 * x + sd * rnorm(N)^2
est = lm(y ~ x)

kk = residuals(est)
beta = coef(est)
```

```
bootResid = replicate(2000, {
    epsilon = sample(kk, replace = TRUE)
    coef(lm((cbind(1, x) %*% beta + epsilon) ~ x))[2]
})

library(latticeExtra)

densityplot(~bootResid, plot.points = FALSE, auto.key = list(columns = 2), xlab =
↪  expression(beta[1]))
```



Se puede considerar otra opción simulando directamente desde el modelo de regresión estimado:

```
bootResid2 = replicate(2000, coef(lm(simulate(est)[, 1] ~ x))[2])

densityplot(~bootResid2, plot.points = FALSE, auto.key = list(columns = 2), xlab =
↪  expression(beta[1]))
```



Regresión bootstrap con la librería `simpleboot`

```
library(simpleboot)

lmodel = lm(y ~ x)
# Bootstrap con residuos
lboot2 = lm.boot(lmodel, R = 1000, rows = FALSE)
summary(lboot2)
```

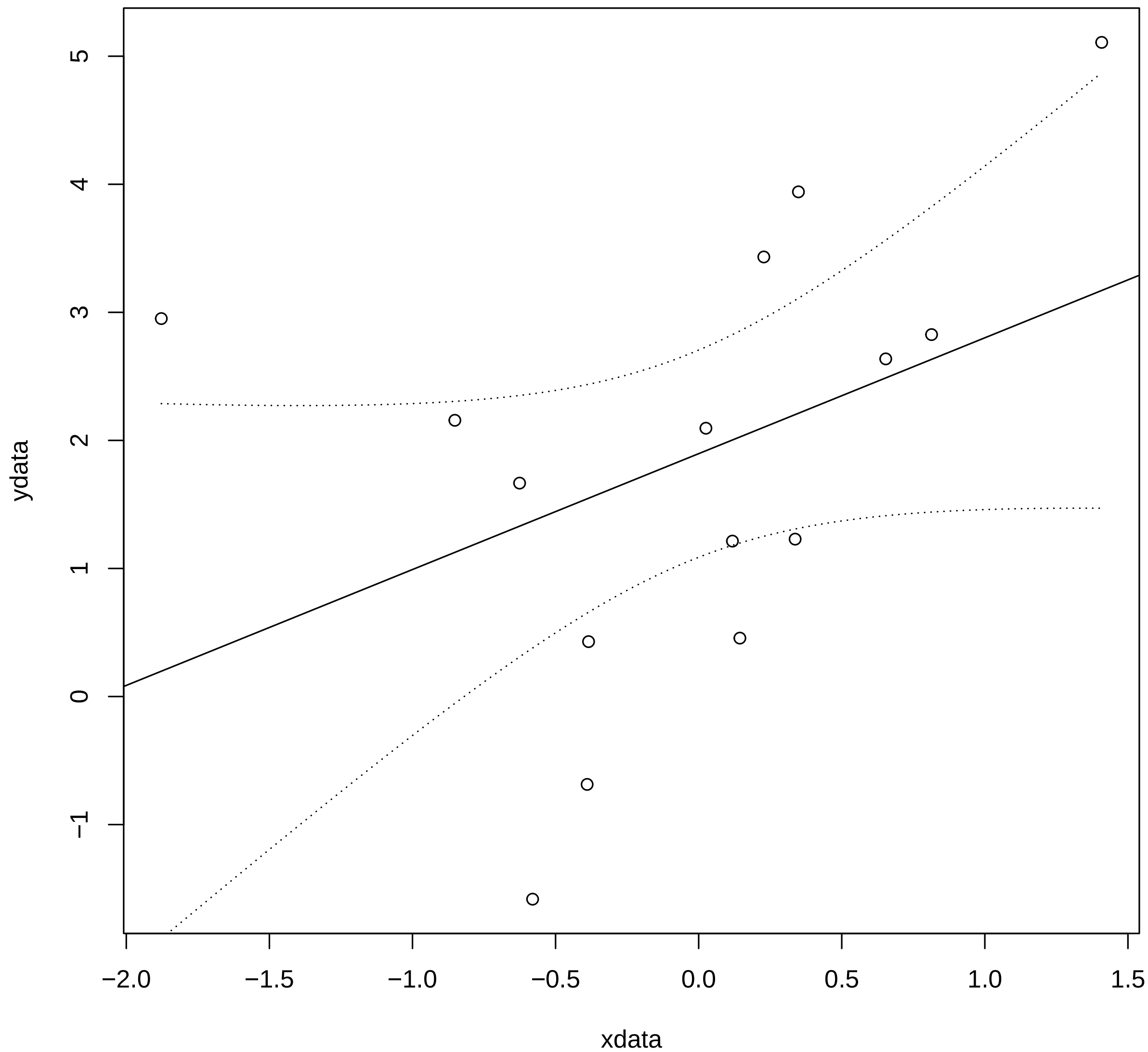
```
BOOTSTRAP OF LINEAR MODEL  (method = residuals)

Original Model Fit
------------------
Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
     1.8963       0.9048

Bootstrap SD's:
(Intercept)            x
  0.4044691    0.5208341
```

Gráfico de los datos y de la recta de regresión original junto con

+1.96 veces el error estándar bootstrap

−1.96 veces el error estándar bootstrap

```
plot(lboot2)
```

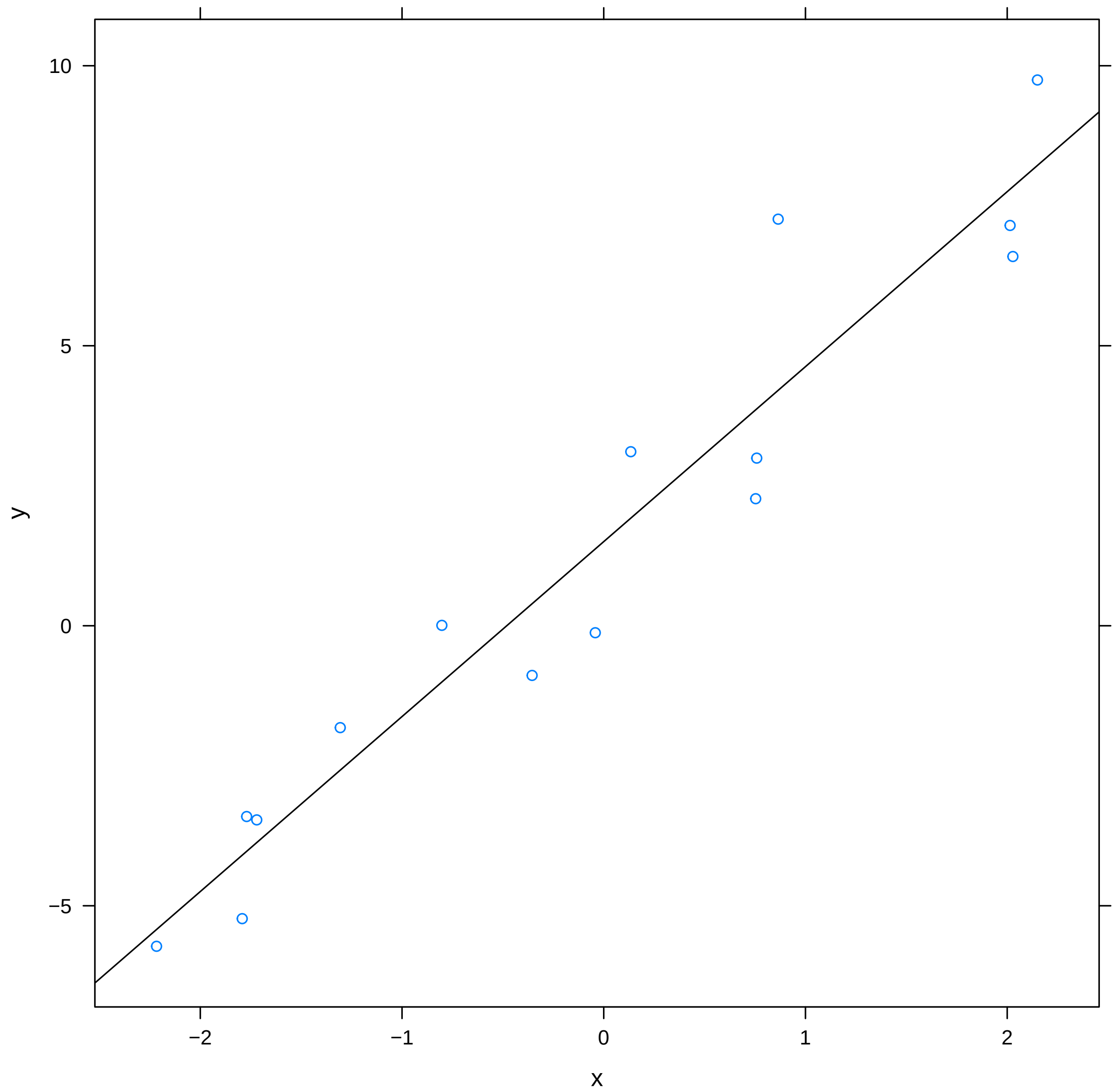


Bootstrap en regresión basado en pares de valores

Se simula un modelo de regresión lineal con errores distribuidos según una normal

```
set.seed(666)
N = 15
sd = 1.5
```

```
x = rnorm(N)
y = 3 * x + sd * rnorm(N)^2
est = lm(y ~ x)

library(latticeExtra)
xyplot(y ~ x) + layer(panel.abline(est))
```

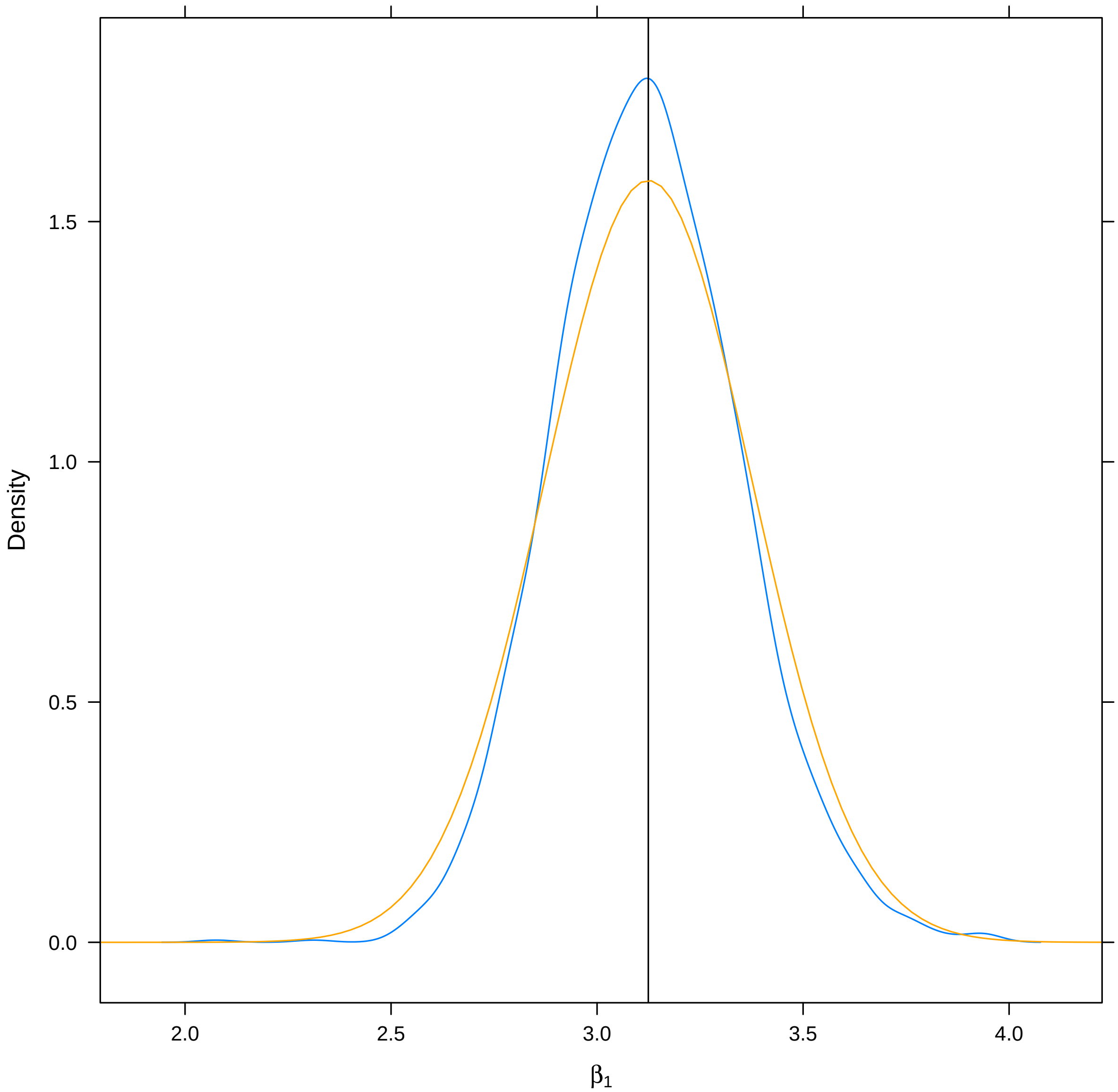



```
bootPair = replicate(2000, {
    ind = sample(1:N, replace = TRUE)
    coef(lm(y[ind] ~ x[ind]))[2]
```

```
})

# est tiene los valores de la recta de regresión original
betaEst = coef(est)[2]
sdBeta = sqrt(vcov(est)[2, 2])
```

Gráfico de la distribución del estadístico beta original y el estadístico beta remuestreado

```
densityplot(bootPair, plot.points = FALSE, xlab = expression(beta[1])) +
↪   layer(panel.abline(v = betaEst)) +
    layer(panel.mathdensity(args = list(mean = betaEst, sd = sdBeta), col = "orange",
        n = 100))
```



Si se compara el valor del error estándar bootstrap $\hat{\sigma}_\beta$ con respecto al error estándar del modelo de regresión original:

```
sd(bootPair)
```

```
[1] 0.2227081
```

```
sqrt(vcov(est)[2, 2])
```

```
[1] 0.2515871
```

Regresión bootstrap con la librería `simpleboot`

```r
N = 15
sd = 1.5
x = rnorm(N)
y = 3 * x + sd * rnorm(N)^2

library(simpleboot)
lmodel = lm(y ~ x)

lboot2 = lm.boot(lmodel, R = 1000)
summary(lboot2)
```

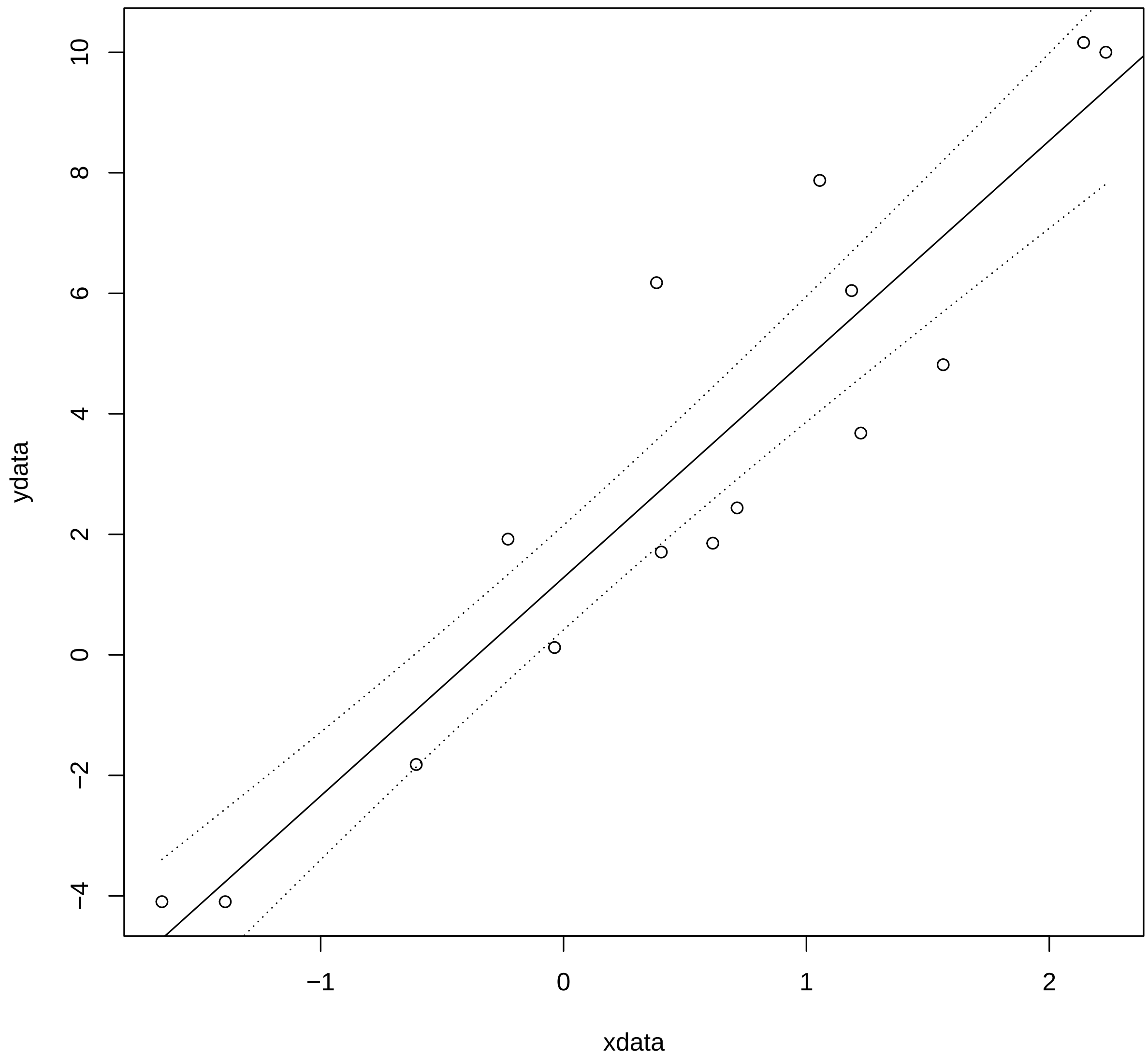
```
BOOTSTRAP OF LINEAR MODEL  (method = rows)

Original Model Fit
------------------
Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
      1.283        3.625

Bootstrap SD's:
(Intercept)            x
  0.4340727    0.2949170
```

Gráfico de los datos y de la recta de regresión original junto con

$+1.96$ veces el error estándar bootstrap

-1.96 veces el error estándar bootstrap

```r
plot(lboot2)
```



Regresión bootstrap con la librería `boot`

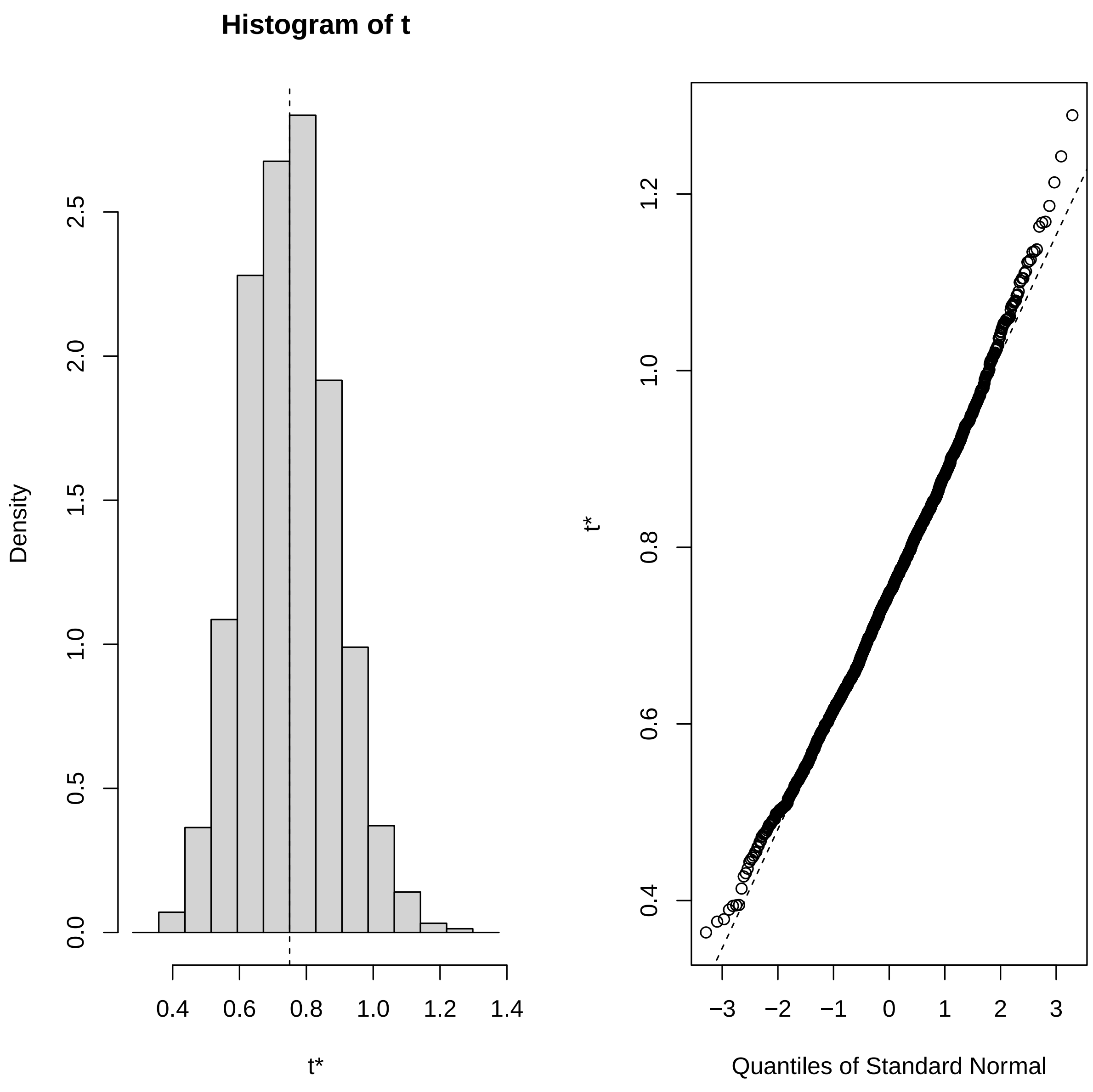
```
N = 50
sd = 0.5
x = rnorm(N)
y = 10 * x + sd * rnorm(N)^2
datos = data.frame(y, x)
```

```r
# Regresion basada en parejas o filas
boot.reg = function(data, i) {
    mod = lm(y ~ x, data = data[i, ])
    coef(mod)
}
```

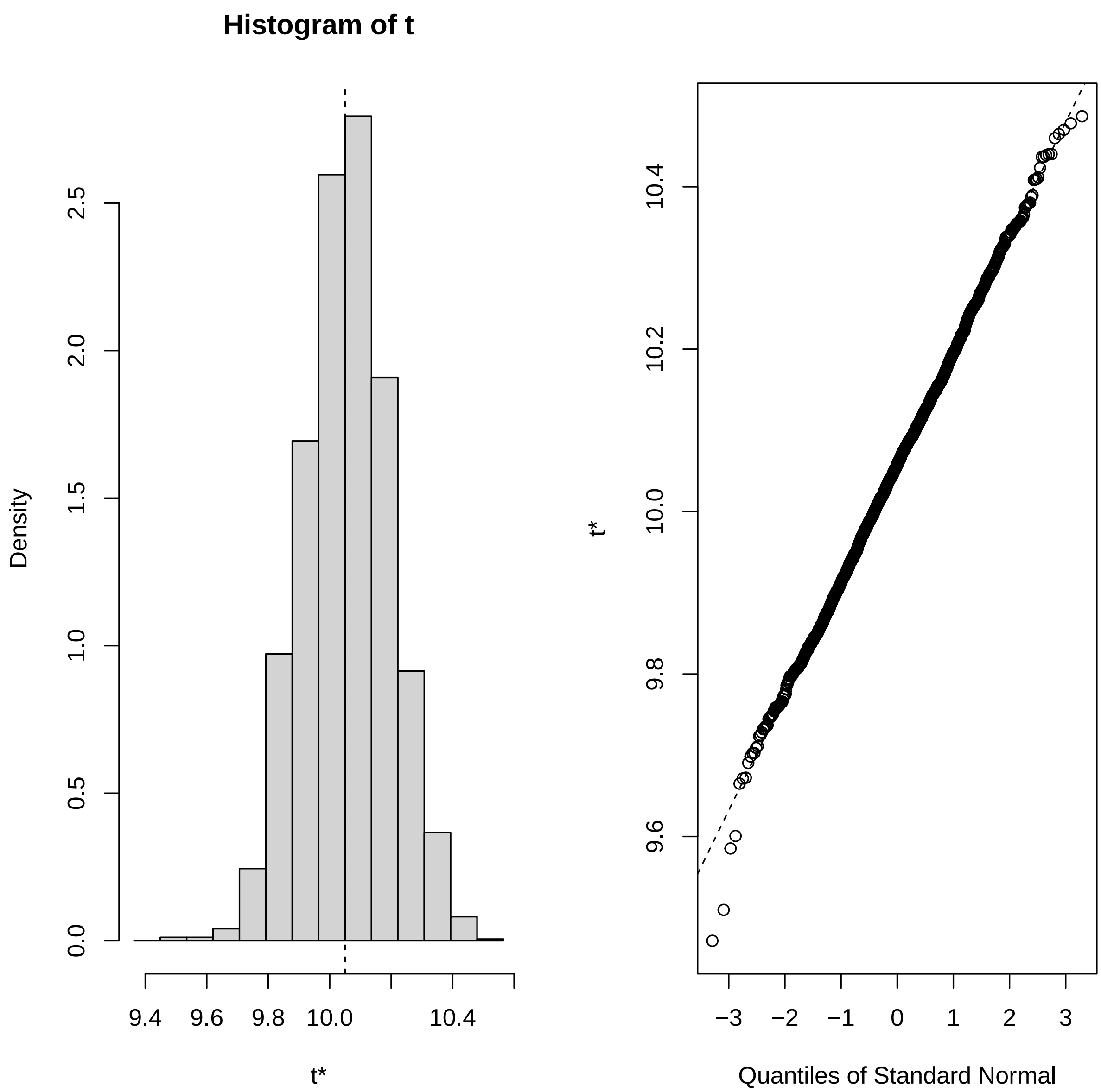
```r
library(boot)

boot.1 = boot(data = datos, statistic = boot.reg, R = 2000)
boot.1
```

```

ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = datos, statistic = boot.reg, R = 2000)


Bootstrap Statistics :
      original        bias    std. error
t1*  0.7500493 -0.0002547164   0.1345602
t2* 10.0499865  0.0058431609   0.1412231
```

```r
plot(boot.1, index = 1, nclass = 15)
```




```r
plot(boot.1, index = 2, nclass = 15)
```



Alternativamente

```
library(ggplot2)
qqnorm(boot.1$t[, 1])
qqline(boot.1$t[, 1], col = "darkgreen")

boot1 = as.data.frame(boot.1$t[, 1])
colnames(boot1) = "beta0"
ggplot() + geom_histogram(data = boot1, aes(beta0), bins = 15, color = "darkblue", fill =
↪  "lightblue") +
    labs(x = expression(beta[0])) + theme_bw()
```

```
detach("package:ggplot2")
```

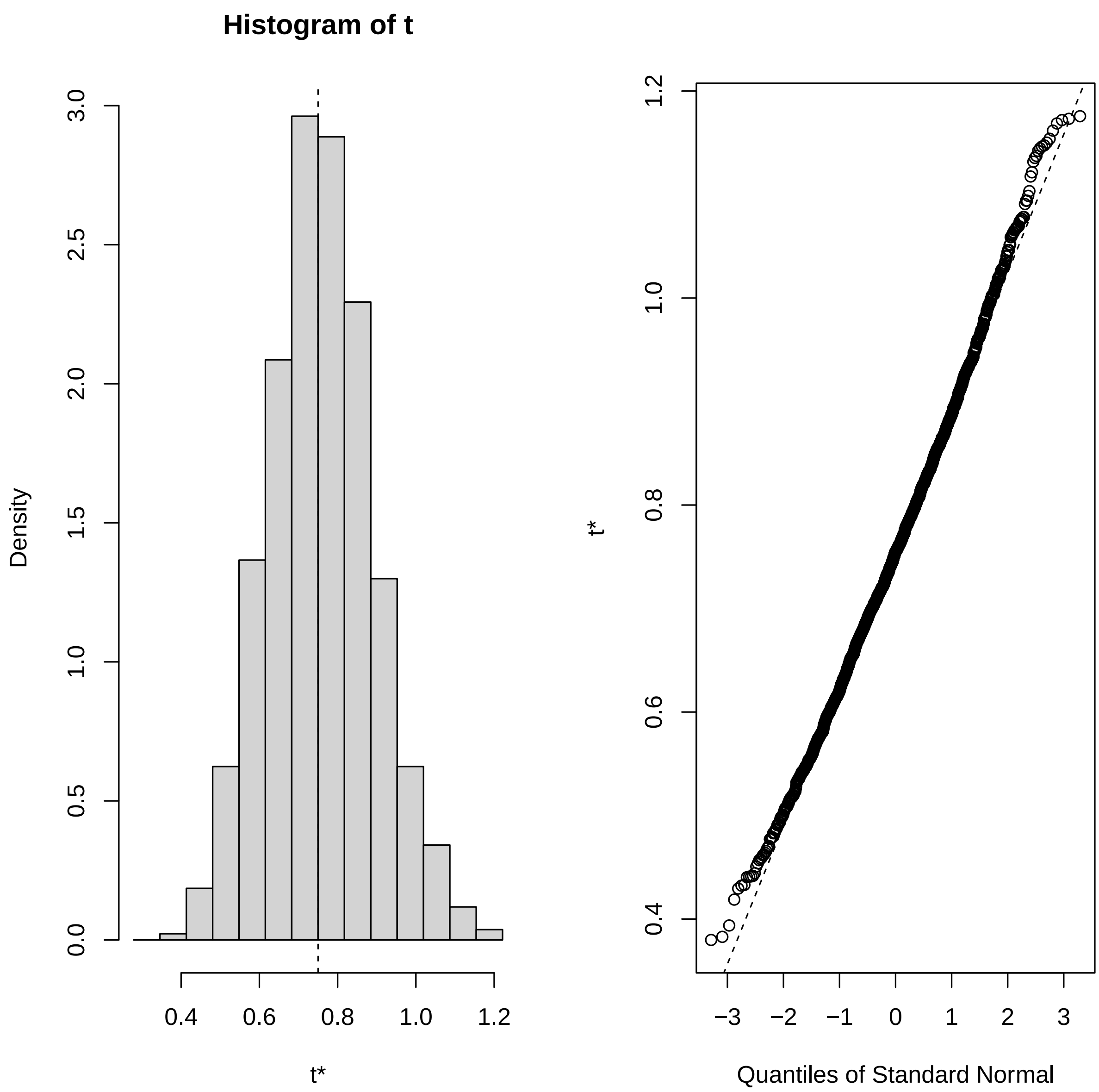
Regresión basada en residuos

```
boot.reg2 = function(losdatos, i) {
    modelo = lm(y ~ x, data = losdatos)
    yhat = fitted(modelo)
    e = resid(modelo)
    y.star = yhat + e[i]
    modelB = lm(y.star ~ x)
    coef(modelB)
}
```

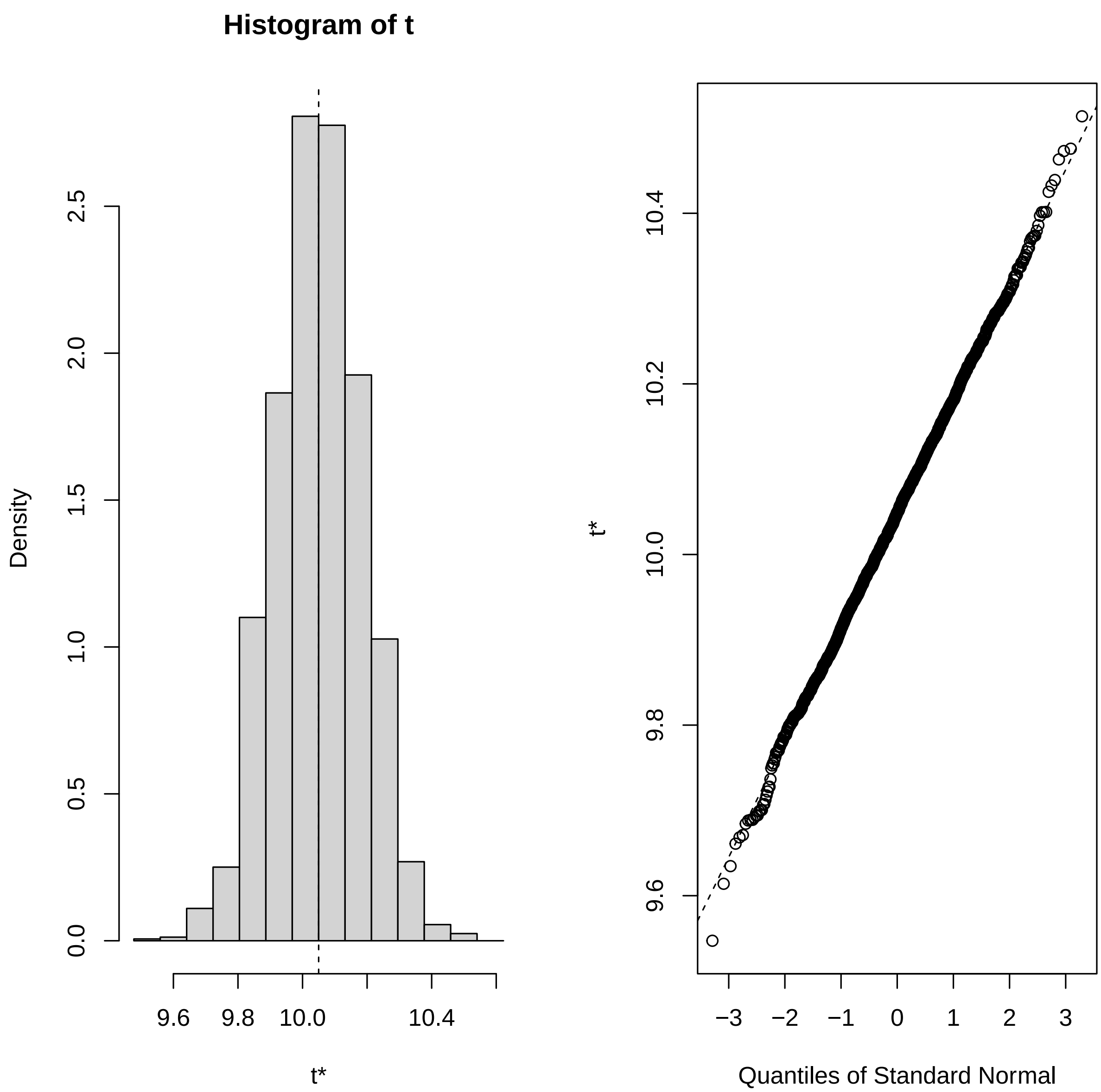
```
boot.2 = boot(data = datos, statistic = boot.reg2, R = 2000)
boot.2
```

```

ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = datos, statistic = boot.reg2, R = 2000)


Bootstrap Statistics :
      original        bias    std. error
t1*  0.7500493  0.007248614    0.133633
t2* 10.0499865 -0.001892554    0.134344
```

```
plot(boot.2, index = 1, nclass = 15)
```




```
plot(boot.2, index = 2, nclass = 15)
```



Regresión bootstrap con la librería car

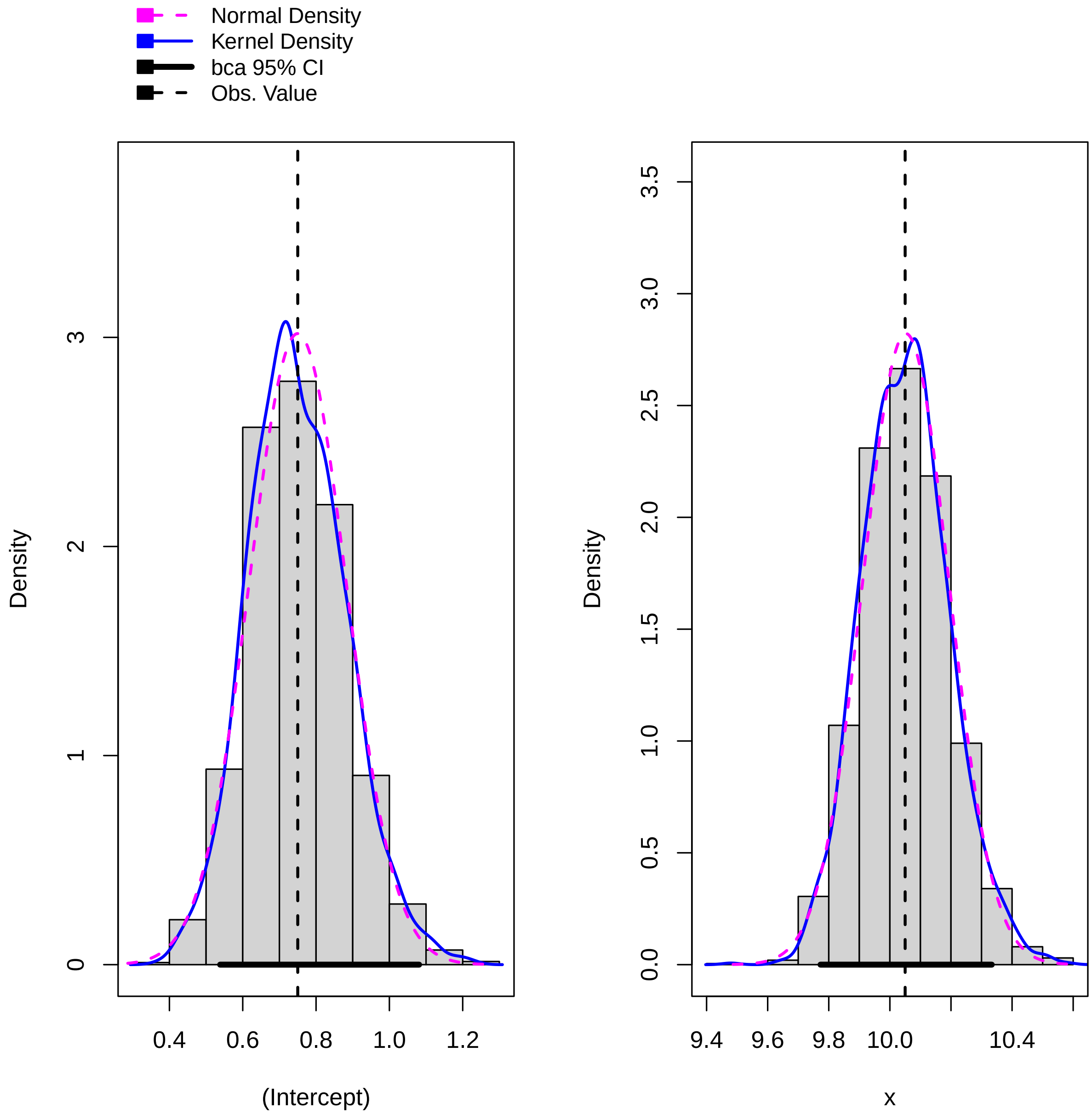
```
library(car)

modeloB = lm(y ~ x, datos)
betahat.boot = Boot(modeloB, R = 2000)
summary(betahat.boot)  # default summary
```

```
Number of bootstrap replications R = 2000
            original    bootBias  bootSE  bootMed
(Intercept)  0.75005 -0.00012534 0.13218  0.73906
x           10.04999  0.00226288 0.14138 10.05161
```

```
confint(betahat.boot)
```

```
Bootstrap bca confidence intervals

                2.5 %    97.5 %
(Intercept) 0.5383728  1.081026
x           9.7729504 10.333456
```

```
hist(betahat.boot)
```



Con **residuos**:

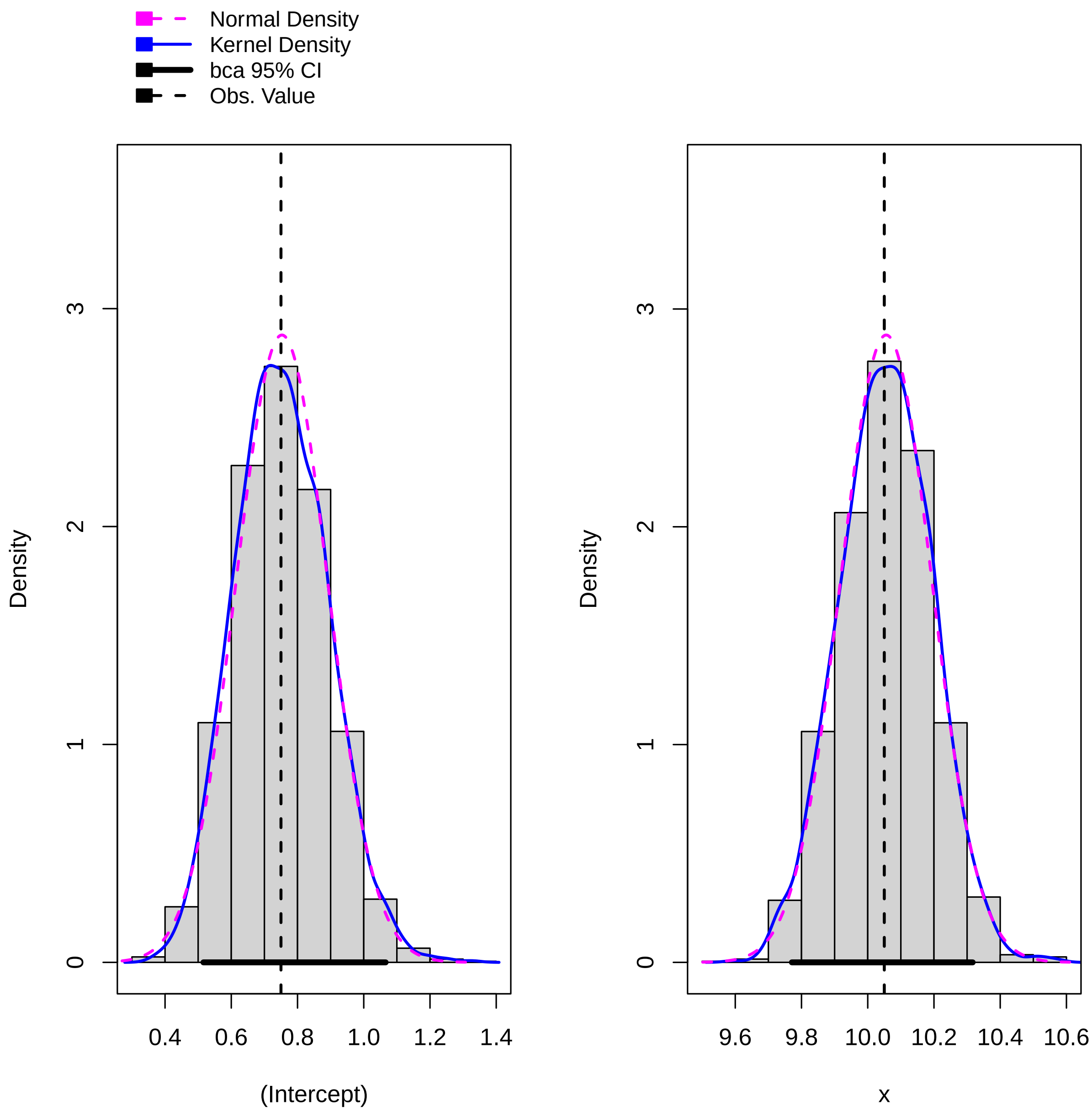
```
betahat.boot2 = Boot(modeloB, method = "residual", R = 2000)
summary(betahat.boot2)  # default summary
```

```

Number of bootstrap replications R = 2000
             original  bootBias   bootSE  bootMed
(Intercept)  0.75005 0.0028486 0.13858  0.74811
x           10.04999 0.0057248 0.13850 10.05723
```

```
confint(betahat.boot2)
```

```
Bootstrap bca confidence intervals

                2.5 %    97.5 %
(Intercept) 0.5159492  1.066047
x           9.7709649 10.316876
```

```
hist(betahat.boot2)
```



ANOVA unifactorial con Bootstrap

- Una manera cómoda de aplicar bootstrap en técnicas ANOVA es mediante la librería `WRS2`
- Esta librería permite trabajar también con *medias recortadas* (trimming means) y funciona bien en el caso de heterocedasticidad y falta de normalidad.
- Se aplica un ANOVA unifactorial asumiendo normalidad

```
library(WRS2)
# help(viagra)

summary(aov(libido ~ dose, data = viagra))
```

```
            Df Sum Sq Mean Sq F value Pr(>F)
dose         2  20.13  10.067   5.119 0.0247 *
Residuals   12  23.60   1.967
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Se aplica un remuestreo bootstrap

```
t1waybt(libido ~ dose, tr = 0, nboot = 1000, data = viagra)
```

```
Call:
t1waybt(formula = libido ~ dose, data = viagra, tr = 0, nboot = 1000)

Effective number of bootstrap samples was 981.

Test statistic: 4.3205
p-value: 0.05607
Variance explained: 0.645
Effect size: 0.803
```

`Effect size` se refiere a la cantidad de variabilidad explicada por cada término del modelo, que puede ser uno o más parámetros.

Se puede considerar también un ANOVA bifactorial

```r
# help(goggles)

t2way(attractiveness ~ gender * alcohol, data = goggles, tr = 0)
```

```
Call:
t2way(formula = attractiveness ~ gender * alcohol, data = goggles,
    tr = 0)

                 value p.value
gender          2.0323   0.164
alcohol        40.0983   0.001
gender:alcohol 24.4083   0.001
```

Se puede usar también un análisis *post-hoc*

```r
mcp2atm(attractiveness ~ gender * alcohol, data = goggles, tr = 0)
```

```
Call:
mcp2atm(formula = attractiveness ~ gender * alcohol, data = goggles,
    tr = 0)

                  psihat  ci.lower  ci.upper p-value
gender1           11.250  -4.82883  27.32883 0.16374
alcohol1          -1.875 -18.53329  14.78329 0.77361
alcohol2          34.375  18.65382  50.09618 0.00001
alcohol3          36.250  18.82376  53.67624 0.00002
gender1:alcohol1  -1.875 -18.53329  14.78329 0.77361
gender1:alcohol2 -28.125 -43.84618 -12.40382 0.00014
gender1:alcohol3 -26.250 -43.67624  -8.82376 0.00081
```

ANOVA unifactorial con Bootstrap basado en residuos

Otra alternativa es aplicar el bootstrap basado en modelos con la librería `boot`.

Se generan unos datos artificiales

```r
Nj = c(41, 37, 42, 40)
Ntot = sum(Nj)
muJ = rep(c(-1, 0, 1, 2), Nj)
MisDatos = data.frame(IV = factor(rep(LETTERS[1:4], Nj)), DV = rnorm(Ntot, muJ, 6))

head(MisDatos)
```

```
  IV        DV
1  A  3.548670
2  A  8.380972
3  A -7.731933
4  A -2.691104
5  A -3.248257
6  A  9.474212
```

```
with(MisDatos, tapply(DV, IV, mean))
```

```
         A          B          C          D
-0.4842300  1.4492215  1.2595738  0.9744863
```

```
with(MisDatos, tapply(DV, IV, var))
```

```
       A        B        C        D
29.12334 33.75782 42.60869 44.93353
```

```
with(MisDatos, tapply(DV, IV, length))
```

```
 A  B  C  D
41 37 42 40
```

Un ANOVA clásico obtiene

```
(anoriginal = anova(lm(DV ~ IV, data = MisDatos)))
```

```
Analysis of Variance Table

Response: DV
           Df Sum Sq Mean Sq F value Pr(>F)
IV          3   93.3  31.089  0.8249  0.482
Residuals 156 5879.6  37.690
```

```
(Fbase = anoriginal["IV", "F value"])
```

```
[1] 0.8248801
```

```
(pBase = anoriginal["IV", "Pr(>F)"])
```

```
[1] 0.4820048
```

Aplicando la librería `boot`

```
mediaglobal = mean(MisDatos$DV)
E = MisDatos$DV - mediaglobal  ## residuos

Boot.Anova = function(dat, i) {
    media.star = mediaglobal + E[i]
    anBS = anova(lm(media.star ~ IV, data = dat))
    return(anBS["IV", "F value"])
}

library(boot)
booAnova = boot(MisDatos, statistic = Boot.Anova, R = 1000)
booAnova
```

```


ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = MisDatos, statistic = Boot.Anova, R = 1000)


Bootstrap Statistics :
     original    bias    std. error
t1* 0.8248801 0.1769524   0.8437987
```

```
Fstar = booAnova$t
Fmayor = (Fstar > Fbase)
```

P-valor remuestreado

```
(pValBS = (sum(Fmayor)/length(Fmayor)))
```

```
[1] 0.468
```

Alternativamente se puede remuestrear de manera directa:

```
meanstar = with(MisDatos, tapply(DV, IV, mean))
cuentas = with(MisDatos, tapply(DV, IV, length))

grpA = MisDatos$DV[MisDatos$IV == "A"] - meanstar[1]
grpB = MisDatos$DV[MisDatos$IV == "B"] - meanstar[2]
grpC = MisDatos$DV[MisDatos$IV == "C"] - meanstar[3]
grpD = MisDatos$DV[MisDatos$IV == "D"] - meanstar[4]

simIV = MisDatos$IV
R = 1000
```

Tenemos una distribución F bootstrapeada en `Fstar` basada en medias de grupos iguales (la hipótesis nula), pero no se asumen normalidad ni homogeneidad

```
Fstar = numeric(R)

for (i in 1:R) {
    groupA = sample(grpA, size = cuentas[1], replace = T)
    groupB = sample(grpB, size = cuentas[2], replace = T)
    groupC = sample(grpC, size = cuentas[3], replace = T)
    groupD = sample(grpD, size = cuentas[4], replace = T)

    simDV = c(groupA, groupB, groupC, groupD)
    simdata = data.frame(simDV, simIV)
    Fstar[i] = oneway.test(simDV ~ simIV, data = simdata)$statistic
}
```

```
quantile(Fstar, 0.95)
```

```
    95%
2.83843
```

```
Fbase = anoriginal["IV", "F value"]  # anoriginal[1,5]
Fmayor = (Fstar > Fbase)
```

P-valor remuestreado

```
(pValBS = (sum(Fmayor)/length(Fmayor)))
```

```
[1] 0.449
```

Aplicación del bootstrap a series temporales

Se considera un modelo de serie AR(1) simple

$$y_t = \beta y_{t-1} + \varepsilon_t$$

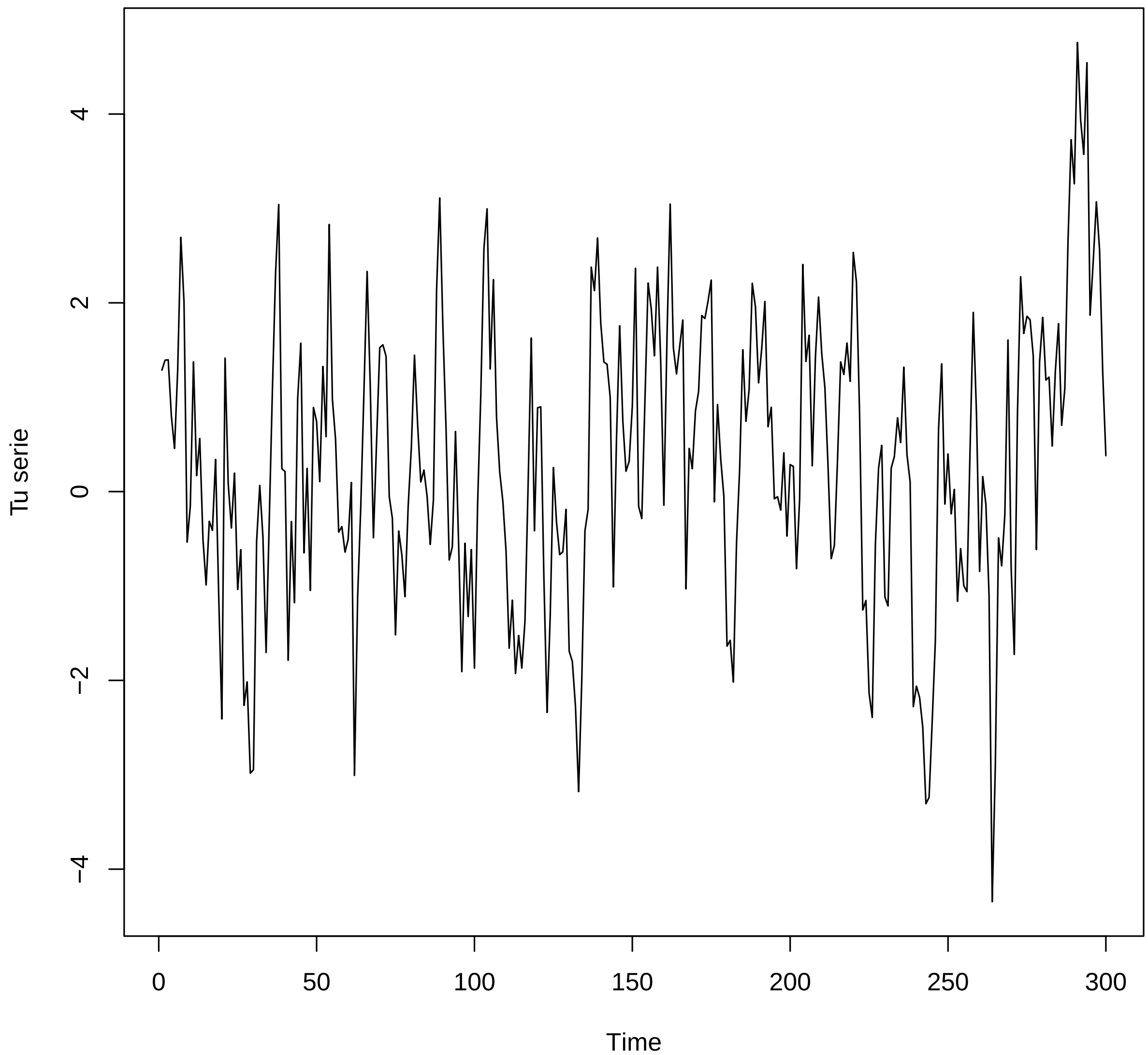
donde $\varepsilon_t \sim N(0.1)$.

Se simulan unos datos

```
N = 300
epsilon = rnorm(N)
y = epsilon
for (i in 2:N) {
    y[i] = y[i - 1] * 0.7 + epsilon[i]
}

plot.ts(ts(y), t = "l", ylab = "Tu serie")
```

O bien usando el comando de R

```
N = 300
# arima.sim(n=N, list(ar=0.7), innov=epsilon)

y = arima.sim(n = N, list(ar = 0.7))
plot.ts(y, t = "l", ylab = "Tu serie")
```



Se puede estimar β por máxima verosimilitud, usando el comando `arima`

```
(est.arima = arima(y, order = c(1, 0, 0), include.mean = FALSE))
```

```

Call:
arima(x = y, order = c(1, 0, 0), include.mean = FALSE)

Coefficients:
         ar1
```

```
      0.7078
s.e.  0.0405

sigma^2 estimated as 1.152:  log likelihood = -447.24,  aic = 898.48
```

Bootstrap en series temporales con residuos

Para construir la muestra bootstrap se usan los residuos estimados:

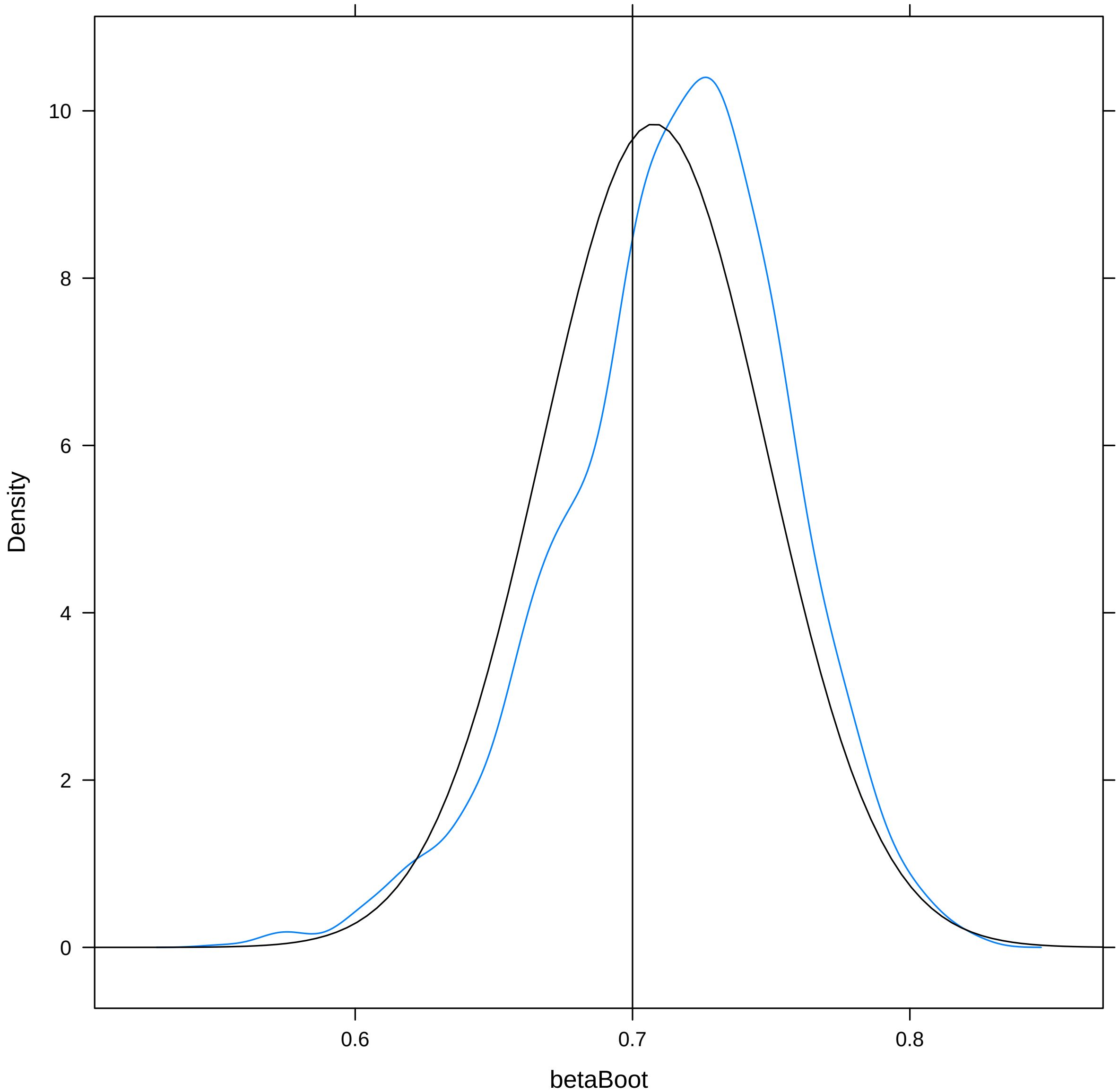
```
kk = residuals(est.arima)
betaB = coef(est.arima)

betaBoot = replicate(2000, {
    epsilon = sample(kk, size = N, replace = TRUE)
    eso = arima.sim(n = N, list(ar = betaB), innov = epsilon)
    coef(arima(eso, order = c(1, 0, 0), include.mean = FALSE))
})
```

Se compara el estimador bootstrap del error estándar, con el obtenido de la serie original por máxima verosimilitud.

```
c(sd(betaBoot), sqrt(vcov(est.arima)))
```

```
[1] 0.04095439 0.04052548
```

```
library(latticeExtra)

sdBeta = sqrt(vcov(est.arima))
densityplot(~betaBoot, plot.points = FALSE) + layer(panel.abline(v = 0.7)) +
↪  layer(panel.mathdensity(args = list(mean = betaB,
    sd = sdBeta), col = "black", n = 100))
```



Supongamos ahora que se supone de manera errónea que el proceso es un AR(2)

$$y_t = \beta_1 y_{t-1} + \beta_2 y_{t-2} + \varepsilon_t$$

Se estiman entonces los parámetros, suponiendo que es un AR(2).

```
est2 = arima(y, order = c(2, 0, 0), include.mean = FALSE)
est2
```

```
Call:
arima(x = y, order = c(2, 0, 0), include.mean = FALSE)

Coefficients:
         ar1      ar2
      0.7257  -0.0253
s.e.  0.0576   0.0577

sigma^2 estimated as 1.151:  log likelihood = -447.15,  aic = 900.29
```

Se puede estudiar la precisión de β_2

```
kk = residuals(est2)
(betaB = coef(est2))
```

```
        ar1         ar2
 0.72568356 -0.02525758
```

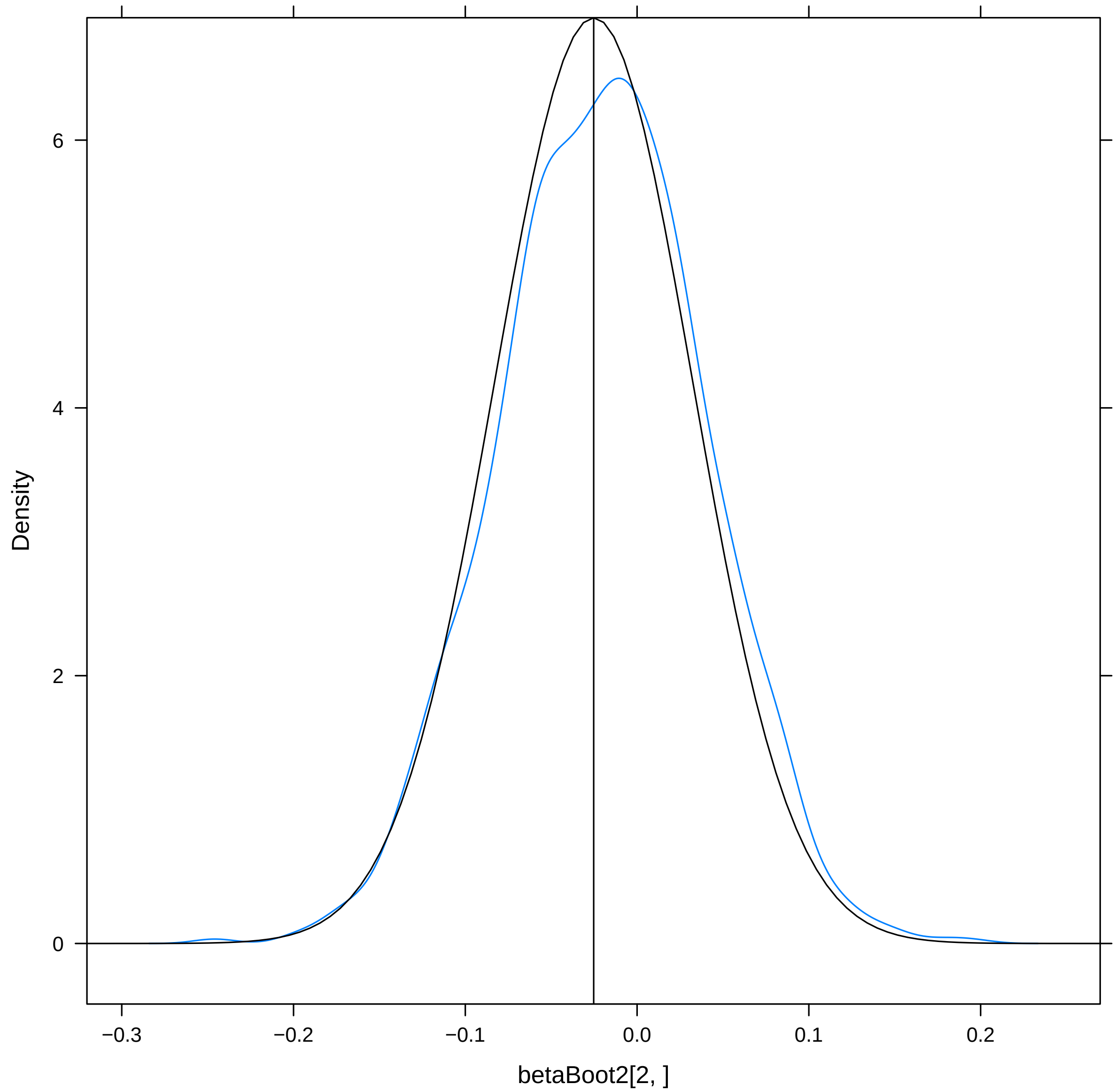
```
betaBoot2 = replicate(2000, {
    epsilon = sample(kk, N, replace = TRUE)
    eso = arima.sim(n = N, list(ar = betaB), innov = epsilon)
    coef(arima(eso, order = c(2, 0, 0), include.mean = FALSE))
})
```

Se puede estudiar y comparar también la desviación estándar de β_2

```
c(sd(betaBoot2[2, ]), sqrt(vcov(est2)[2, 2]))
```

```
[1] 0.05994506 0.05770196
```

```
seB = sqrt(diag(vcov(est2)))

library(latticeExtra)

densityplot(betaBoot2[2, ], plot.points = FALSE) + layer(panel.abline(v = betaB[2])) +
    layer(panel.mathdensity(args = list(mean = betaB[2], sd = seB[2]), col = "black",
        n = 100))
```

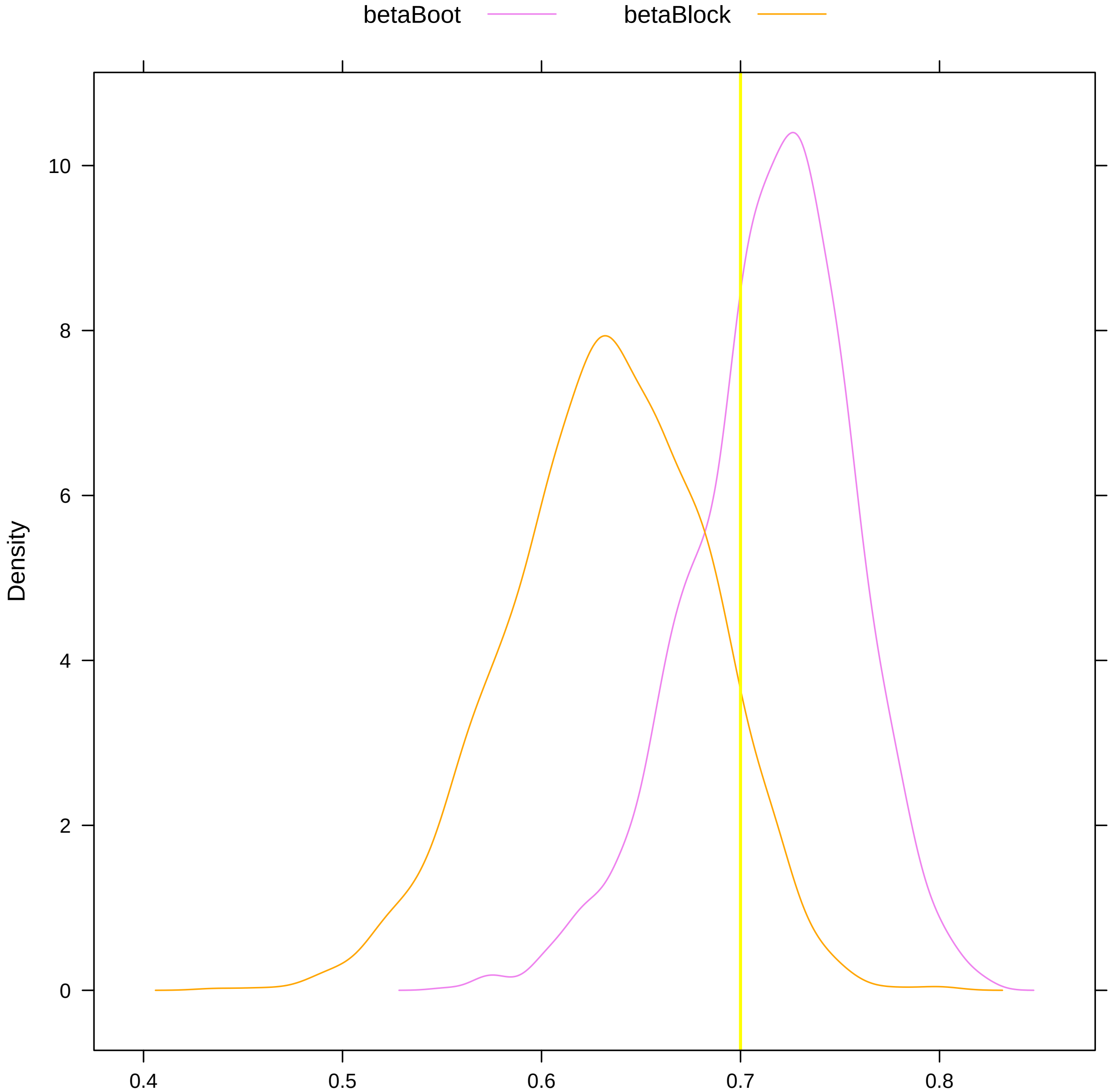

6
4
2
0
Density
−0.3
−0.2
−0.1
0.0
0.1
0.2
betaBoot2[2,]

Bloques móviles (*moving blocks*)

```
N = 300
blockLen = 10
blockNum = N/blockLen

betaBlock = replicate(2000, {
    start = sample(1:(N - blockLen + 1), size = blockNum, replace = TRUE)
    blockedIndices = c(sapply(start, function(x) seq(x, x + blockLen - 1)))
    eso = y[blockedIndices]
    coef(arima(eso, order = c(1, 0, 0), include.mean = FALSE))
})

c(sd(betaBlock), sqrt(vcov(est.arima)))
```

```
[1] 0.04998887 0.04052548
```

```
densityplot(~betaBoot + betaBlock, xlab = "", plot.points = FALSE, auto.key =
↪  list(columns = 2),
    par.settings = list(superpose.line = list(col = c("violet", "orange"))), panel =
    ↪  function(...) {
        panel.densityplot(...)
        panel.abline(v = 0.7, col.line = "yellow", lwd = 2)
    })
```


betaBoot
betaBlock
Density
10
8
6
4
2
0
0.4
0.5
0.6
0.7
0.8

Bootstrap con `tsboot`

Se puede usar el comando `tsboot` de la librería `boot`

```
library(boot)

N = 300
epsilon = rnorm(N)

# Simulas un AR(1)
y = arima.sim(n = N, list(ar = 0.6), innov = epsilon)

bootf = function(miserie) {
    fit = ar(miserie, order.max = 1)  # modelo AR(1)
    return(fit$ar)
}
```

`tsboot` con bloques móviles

Por ejemplo, bootstrap por bloques cada uno con longitud 10:

```
boot2 = tsboot(y, bootf, R = 2000, l = 10, sim = "fixed")

teta.star2 = as.vector(boot2$t)
summary(teta.star2)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.3429  0.4808  0.5188  0.5160  0.5544  0.6747 
```

IC del percentil:

```
quantile(teta.star2, probs = c(0.025, 0.975))
```

```
     2.5%     97.5% 
0.4037745 0.6133298 
```

Se puede considerar un bootstrap estacionario con longitud media de bloque 10:

```r
boot3 = tsboot(y, bootf, R = 2000, l = 10, sim = "geom")

teta.star3 = as.vector(boot3$t)
summary(teta.star3)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.2948  0.4794  0.5168  0.5163  0.5550  0.6737
```

```r
quantile(teta.star3, probs = c(0.025, 0.975))
```

```
     2.5%     97.5%
0.3999446 0.6269048
```

`tsboot` con remuestreo de residuos

Ahora, para el remuestreo basado en modelos, necesitamos el modelo original.

```r
ar1 = ar(y, order.max = 1)  # Ajustas un AR(1)
armodel = list(order = c(ar1$order, 0, 0), ar = ar1$ar)

ar.res = ar1$resid[!is.na(ar1$resid)]
ar.res = ar.res - mean(ar.res)
```

Funciones para incluir en el comando `tsboot`:

```r
bootf = function(miserie){
   fit = ar(miserie, order.max=1)     # modelo AR(1)
   return(fit$ar)
}

bootsim = function(resid, n.sim, ran.args){

  # Generación de réplicas de series con arima.sim
  rg1 = function(n, resid){
     sample(resid, n, replace=TRUE)
  }

  ts.orig = ran.args$ts
  ts.mod = ran.args$model
```

```
  simulaciones = mean(ts.orig) + ts(arima.sim(model=ts.mod,

                  n=n.sim, rand.gen=rg1, res=as.vector(resid)))

  return(simulaciones)
}
```

```
boot2 = tsboot(ar.res, bootf, R = 1000, sim = "model", n.sim = length(y), orig.t = FALSE,
    ran.gen = bootsim, ran.args = list(ts = y, model = armodel))
```

Obtención de resultados

```
teta.star = as.vector(boot2$t)
summary(teta.star)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.4136  0.5558  0.5877  0.5849  0.6175  0.7105
```

IC del percentil:

```
quantile(teta.star, probs = c(0.025, 0.975))
```

```
     2.5%     97.5%
0.4845405 0.6699598
```

Modelos GLM con bootstrap

Se pueden considerar modelos lineales generalizados

```r
library(boot)

# help(remission)

head(remission)
```

```
   LI m r
1 0.4 1 0
2 0.4 1 0
3 0.5 1 0
4 0.5 1 0
5 0.6 1 0
6 0.6 1 0
```

```r
modelori = glm(r ~ LI, family = "binomial", data = remission)

summary(modelori)
```

```

Call:
glm(formula = r ~ LI, family = "binomial", data = remission)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.9448  -0.6465  -0.4947   0.6571   1.6971

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   -3.777      1.379  -2.740  0.00615 **
LI             2.897      1.187   2.441  0.01464 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 34.372  on 26  degrees of freedom
Residual deviance: 26.073  on 25  degrees of freedom
AIC: 30.073

Number of Fisher Scoring iterations: 4
```

```r
library(boot)

# help(remission)

model.boot = function(data, indices) {
    sub.data = data[indices, ]
    model = glm(r ~ LI, family = "binomial", data = sub.data)
    coef(model)
}

glm.boot = boot(remission, model.boot, R = 5000)
glm.boot
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = remission, statistic = model.boot, R = 5000)


Bootstrap Statistics :
     original    bias    std. error
t1* -3.777140 -3.168310    25.71728
t2*  2.897264  3.234061    26.49930
```

Los correspondientes intervalos de confianza son

```r
boot.ci(glm.boot, index = 1, type = "bca")
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 5000 bootstrap replicates

CALL :
boot.ci(boot.out = glm.boot, type = "bca", index = 1)

Intervals :
Level       BCa
95%   (-7.866, -1.198 )
Calculations and Intervals on Original Scale
```

```r
boot.ci(glm.boot, index = 2, type = "bca")
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 5000 bootstrap replicates

CALL :
```

```
boot.ci(boot.out = glm.boot, type = "bca", index = 2)

Intervals :
Level       BCa
95%   ( 0.575,  8.184 )
Calculations and Intervals on Original Scale
```

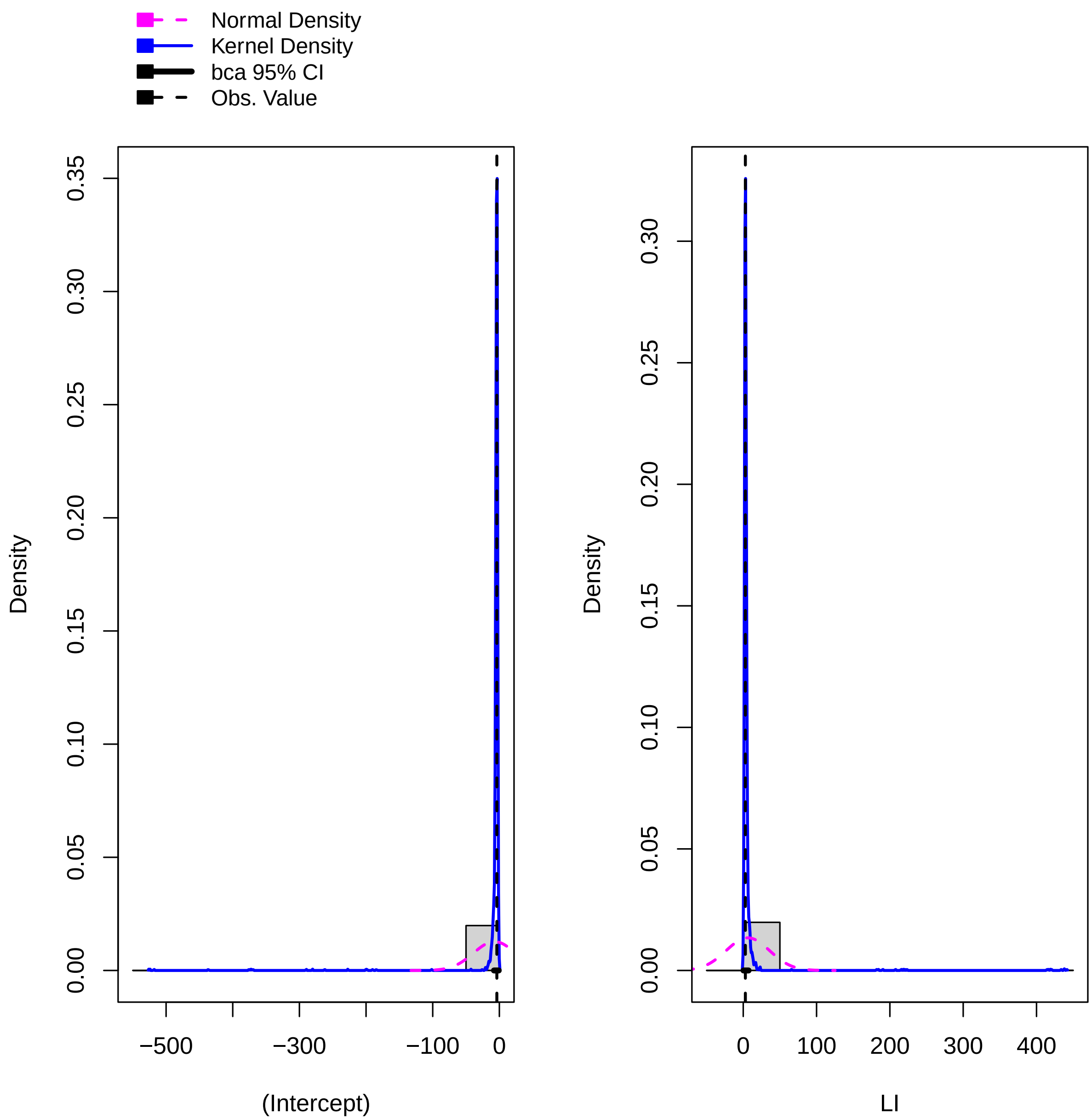
```
library(car)

betahat.boot = Boot(modelori, R = 2000)
summary(betahat.boot)
```

```

Number of bootstrap replications R = 2000
            original bootBias bootSE bootMed
(Intercept)  -3.7771  -3.5712 31.290 -4.1375
LI            2.8973   3.4213 29.744  3.1973
```

```
confint(betahat.boot)
```

```
Bootstrap bca confidence intervals

                 2.5 %     97.5 %
(Intercept) -8.0654702 -0.9303116
LI           0.3538417  7.3117510
```

```
hist(betahat.boot)
```



Ejemplo con programa en paralelo

```
# load credit data
library(caret)
library(dplyr)

data(GermanCredit)
data = GermanCredit %>%
    rename(credit = Class)
```

```r
logit = glm(credit ~ Amount + Age + Duration + Personal.Male.Single + Purpose.UsedCar +
    Property.RealEstate, data = data, family = binomial(link = "logit"))

summary(logit)
```

```

Call:
glm(formula = credit ~ Amount + Age + Duration + Personal.Male.Single +
    Purpose.UsedCar + Property.RealEstate, family = binomial(link = "logit"),
    data = data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.2441  -1.1775   0.6573   0.8441   1.7508

Coefficients:
                       Estimate Std. Error z value Pr(>|z|)
(Intercept)           7.901e-01  2.841e-01   2.781  0.00541 **
Amount               -5.627e-05  3.263e-05  -1.724  0.08463 .
Age                   1.372e-02  6.873e-03   1.996  0.04593 *
Duration             -3.251e-02  7.530e-03  -4.317 1.58e-05 ***
Personal.Male.Single  4.640e-01  1.513e-01   3.068  0.00215 **
Purpose.UsedCar       1.210e+00  2.912e-01   4.155 3.25e-05 ***
Property.RealEstate   4.897e-01  1.767e-01   2.772  0.00557 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1221.7  on 999  degrees of freedom
Residual deviance: 1131.9  on 993  degrees of freedom
AIC: 1145.9

Number of Fisher Scoring iterations: 4
```

```r
library(boot)
library(parallel)

R = 1000  # Número de réplicas
n = nrow(data)  # Tamaño muestral
k = length(coef(logit))  # Número de coeficientes

myLogitCoef = function(data, indices, formula) {
    d = data[indices, ]
    fit = glm(formula, data = d, family = binomial(link = "logit"))
    return(coef(fit))
}
```

Usas un clúster en paralelo con 4 núcleos:

```
# help(makeCluster) help(clusterExport)

cl = makeCluster(4)
clusterExport(cl, "myLogitCoef")

coef.boot = boot(data = data, statistic = myLogitCoef, R = 1000, formula = logit$formula,
    parallel = "snow", ncpus = 4, cl = cl)
stopCluster(cl)
```

```
coef.boot
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = data, statistic = myLogitCoef, R = 1000, formula = logit$formula,
    parallel = "snow", ncpus = 4, cl = cl)


Bootstrap Statistics :
         original        bias     std. error
t1*  7.900806e-01  1.098052e-02 0.2924274015
t2* -5.626745e-05  9.717326e-07 0.0000365674
t3*  1.371852e-02  1.644983e-05 0.0069641746
t4* -3.251194e-02 -5.872808e-04 0.0078611728
t5*  4.640478e-01  7.212130e-03 0.1547658044
t6*  1.209960e+00  4.123192e-02 0.2932644138
t7*  4.897382e-01  2.251051e-03 0.1796891071
```

```
boot.ci(coef.boot, index = 1, type = c("bca", "perc"))
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = coef.boot, type = c("bca", "perc"), index = 1)

Intervals :
Level     Percentile            BCa
95%   ( 0.2152,  1.3602 )   ( 0.1887,  1.3528 )
Calculations and Intervals on Original Scale
```

```r
boot.ci(coef.boot, index = 2, type = c("bca", "perc"))
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = coef.boot, type = c("bca", "perc"), index = 2)

Intervals :
Level     Percentile            BCa
95%   (-0.0001,  0.0000 )   (-0.0001,  0.0000 )
Calculations and Intervals on Original Scale
```

```r
boot.ci(coef.boot, index = 3, type = c("bca", "perc"))
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = coef.boot, type = c("bca", "perc"), index = 3)

Intervals :
Level     Percentile            BCa
95%   ( 0.0002,  0.0275 )   ( 0.0002,  0.0280 )
Calculations and Intervals on Original Scale
```

```r
boot.ci(coef.boot, index = 4, type = c("bca", "perc"))
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = coef.boot, type = c("bca", "perc"), index = 4)

Intervals :
Level     Percentile            BCa
95%   (-0.0490, -0.0186 )   (-0.0483, -0.0183 )
Calculations and Intervals on Original Scale
```

```r
boot.ci(coef.boot, index = 5, type = c("bca", "perc"))
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = coef.boot, type = c("bca", "perc"), index = 5)

Intervals :
Level     Percentile            BCa
```

```
95%   ( 0.1665,  0.7732 )   ( 0.1616,  0.7669 )
Calculations and Intervals on Original Scale
```

```
boot.ci(coef.boot, index = 6, type = c("bca", "perc"))
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = coef.boot, type = c("bca", "perc"), index = 6)

Intervals :
Level     Percentile            BCa
95%   ( 0.702,  1.846 )   ( 0.642,  1.773 )
Calculations and Intervals on Original Scale
```

```
boot.ci(coef.boot, index = 7, type = c("bca", "perc"))
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = coef.boot, type = c("bca", "perc"), index = 7)

Intervals :
Level     Percentile            BCa
95%   ( 0.1543,  0.8487 )   ( 0.1545,  0.8493 )
Calculations and Intervals on Original Scale
```

```
library(car)

data(GermanCredit)

elogit = glm(Class ~ Amount + Age + Duration + Personal.Male.Single + Purpose.UsedCar +
    Property.RealEstate, data = GermanCredit, family = binomial(link = "logit"))


betahat.boot = Boot(elogit, R = 2000)
summary(betahat.boot)
```

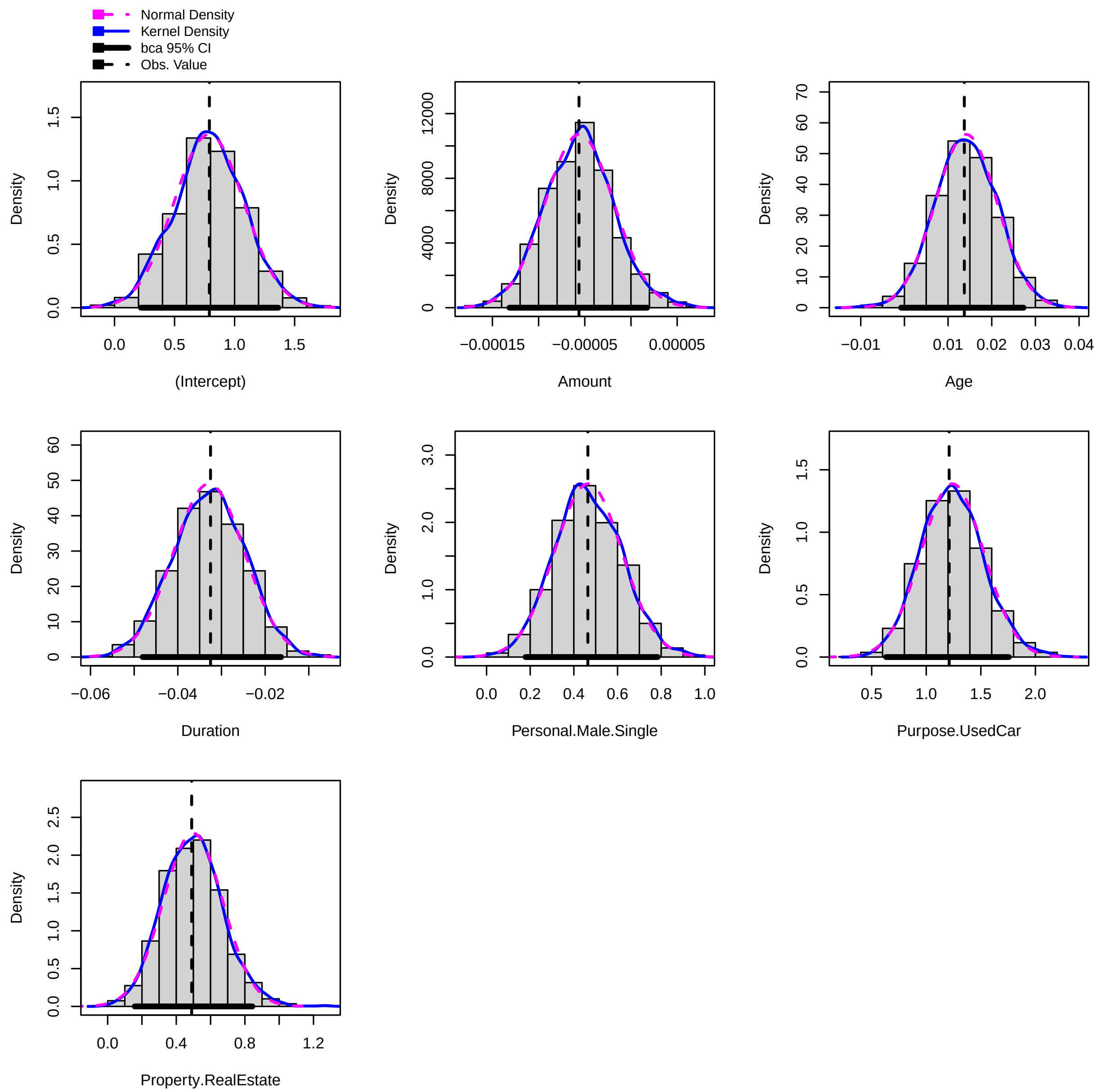
```

Number of bootstrap replications R = 2000
                        original    bootBias     bootSE     bootMed
(Intercept)           7.9008e-01 -3.4012e-03 0.29211102  7.8680e-01
Amount               -5.6267e-05  4.4815e-07 0.00003724 -5.5393e-05
Age                   1.3719e-02  4.2015e-04 0.00709517  1.4030e-02
Duration             -3.2512e-02 -3.4442e-04 0.00813503 -3.2713e-02
Personal.Male.Single  4.6405e-01  1.6842e-03 0.15504132  4.6011e-01
```

```
Purpose.UsedCar       1.2100e+00  2.9664e-02 0.28741604  1.2318e+00
Property.RealEstate   4.8974e-01  7.5751e-03 0.17409900  4.9547e-01
```

```
confint(betahat.boot)
```

```
Bootstrap bca confidence intervals

                             2.5 %        97.5 %
(Intercept)           0.2201632306  1.363372e+00
Amount               -0.0001316000  1.750798e-05
Age                  -0.0007631093  2.727118e-02
Duration             -0.0480640307 -1.632889e-02
Personal.Male.Single  0.1786542409  7.838337e-01
Purpose.UsedCar       0.6323647972  1.757070e+00
Property.RealEstate   0.1579129667  8.436304e-01
```

```
hist(betahat.boot)
```

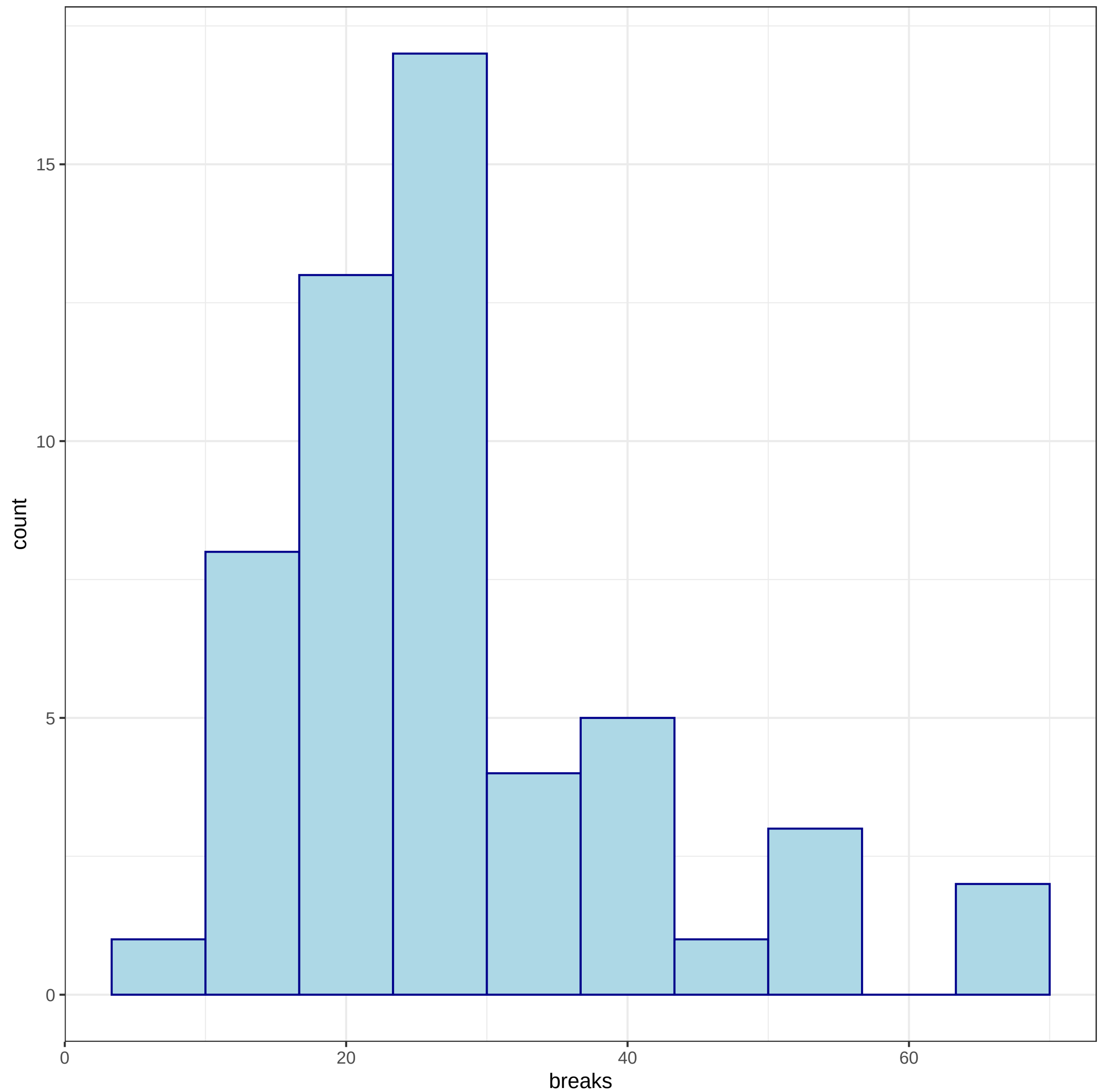


Regresión de Poisson

```
library(datasets)

# help(warpbreaks)

datos = warpbreaks
head(datos)
```

```
  breaks wool tension
1     26    A       L
2     30    A       L
3     54    A       L
4     25    A       L
5     70    A       L
6     52    A       L
```

```
library(ggplot2)
ggplot(datos, aes(x = breaks)) + geom_histogram(bins = 10, color = "darkblue", fill =
↪  "lightblue") +
    theme_bw()
```




```r
# qplot(breaks, data = datos, geom = 'histogram')
```

```r
poisson.modelo = glm(breaks ~ wool + tension, data = datos, family = poisson(link =
↪  "log"))
summary(poisson.modelo)
```

```
Call:
glm(formula = breaks ~ wool + tension, family = poisson(link = "log"),
    data = datos)
```

```
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.6871  -1.6503  -0.4269   1.1902   4.2616

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  3.69196    0.04541  81.302  < 2e-16 ***
woolB       -0.20599    0.05157  -3.994 6.49e-05 ***
tensionM    -0.32132    0.06027  -5.332 9.73e-08 ***
tensionH    -0.51849    0.06396  -8.107 5.21e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 297.37  on 53  degrees of freedom
Residual deviance: 210.39  on 50  degrees of freedom
AIC: 493.06

Number of Fisher Scoring iterations: 4
```

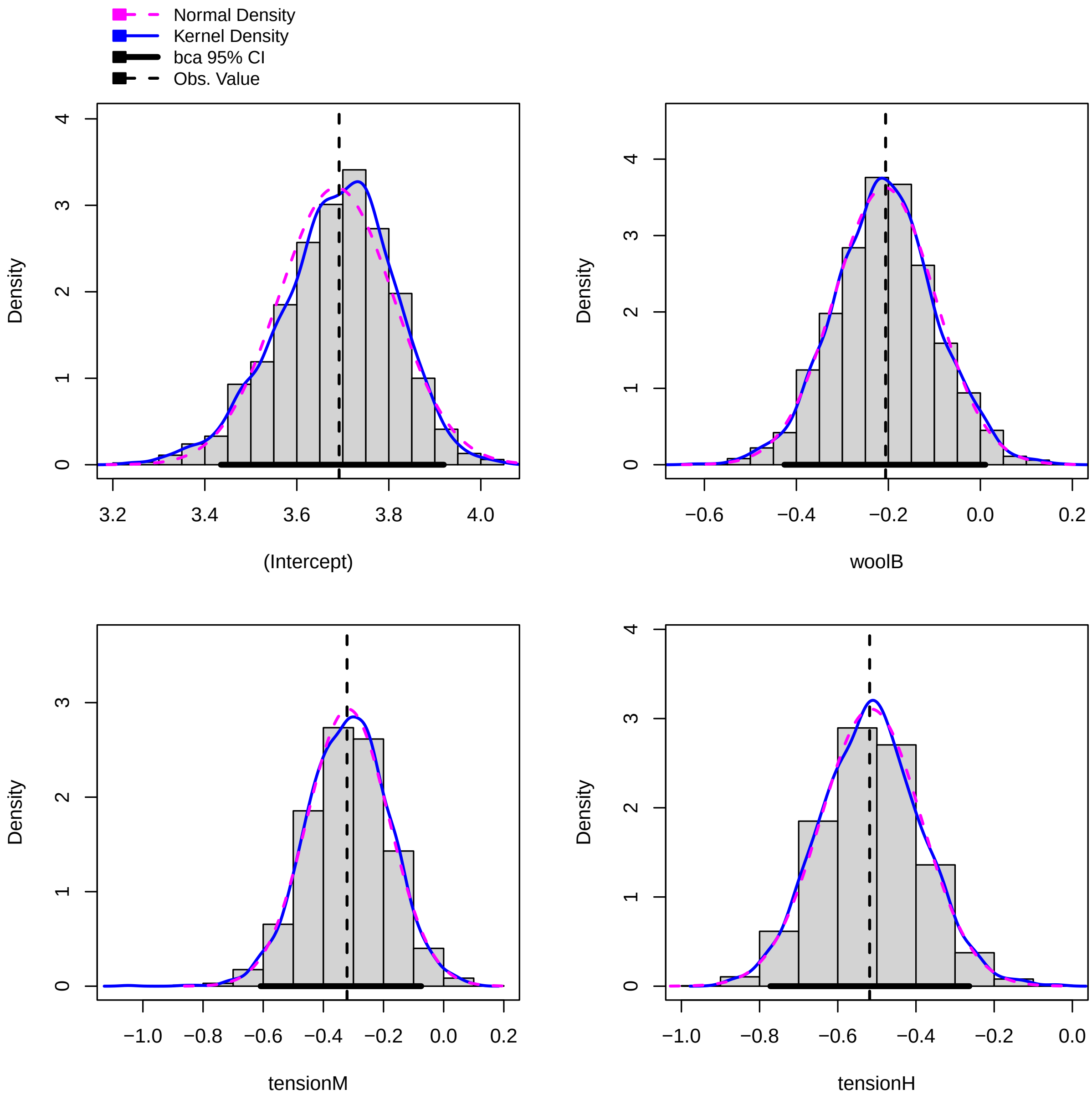
```r
betahat.boot2 = Boot(poisson.modelo, R = 2000)
summary(betahat.boot2)
```

```
Number of bootstrap replications R = 2000
            original   bootBias  bootSE  bootMed
(Intercept)  3.69196 -0.0065113 0.12448  3.69343
woolB       -0.20599 -0.0018534 0.10997 -0.20779
tensionM    -0.32132  0.0033683 0.13605 -0.31561
tensionH    -0.51849  0.0036645 0.12841 -0.51587
```

```r
confint(betahat.boot2)
```

```
Bootstrap bca confidence intervals

                 2.5 %      97.5 %
(Intercept)  3.4349687  3.91959926
woolB       -0.4260798  0.01085145
tensionM    -0.6086101 -0.07464100
tensionH    -0.7723661 -0.26348285
```

```r
hist(betahat.boot2)
```


Normal Density
Kernel Density
bca 95% CI
Obs. Value
Density
(Intercept)
woolB
tensionM
tensionH