Tema 4. Estimación de errores estándar mediante remuestreo

Limitaciones del TCL

Se toma el ejemplo de una distribución binomial con $n = 25$ en los casos de $p = 0.25$ y $p = 0.9$.

```r
n = 25

p09 = rbinom(20000, n, 0.9)/n
p025 = rbinom(20000, n, 0.25)/n
```

```r
head(p09)
```

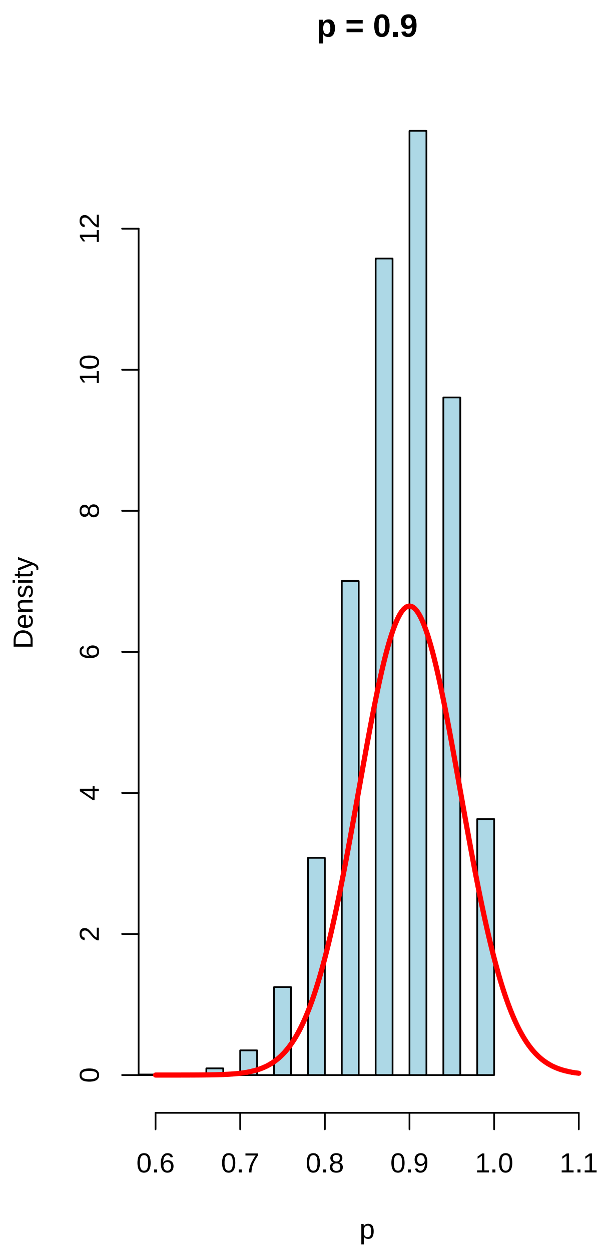
```
[1] 0.88 0.84 0.88 0.88 1.00 0.88
```

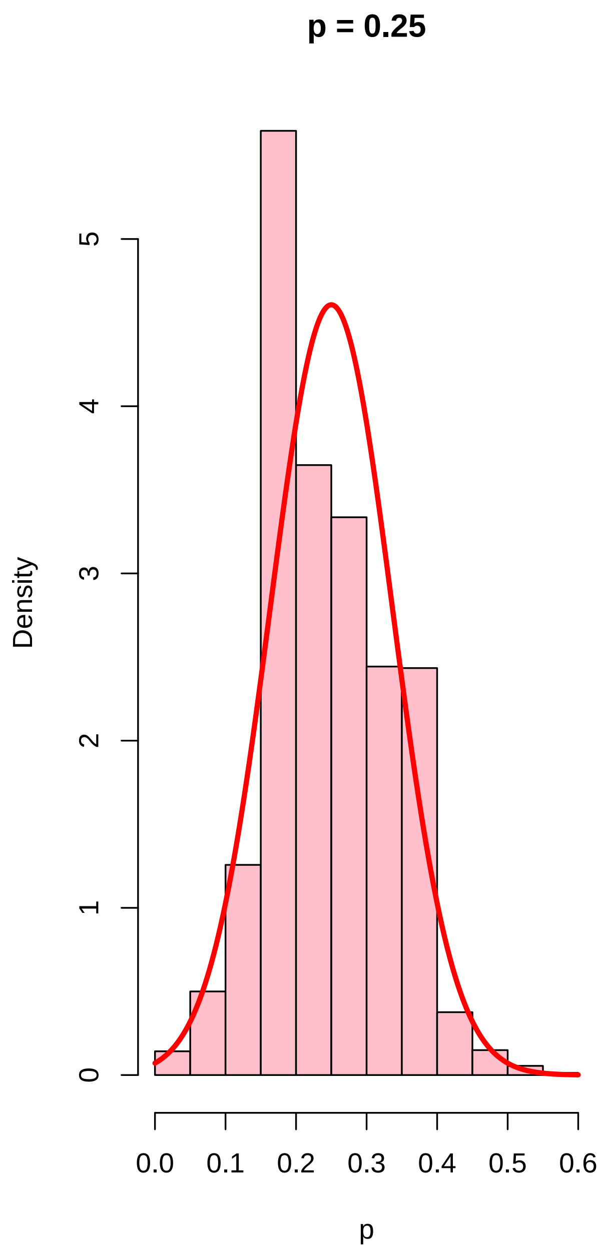
```r
head(p025)
```

```
[1] 0.32 0.12 0.28 0.16 0.12 0.16
```

Para el caso de $p = 0.9$ la aproximación a la normal por el TCL **no** es muy buena.

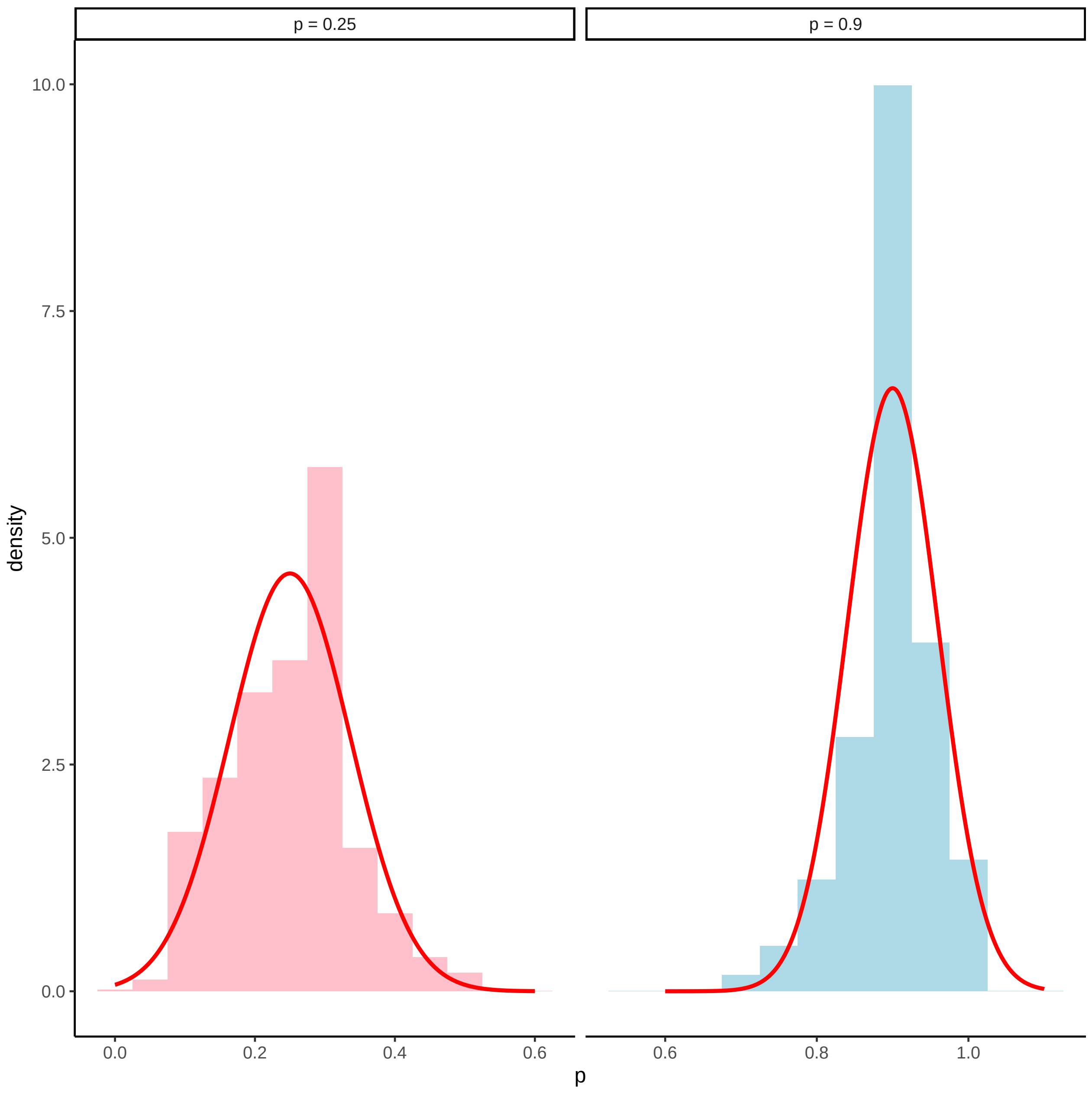
```r
par(mfrow=c(1,2))

hist(p09, prob=T, xlim=c(0.6,1.1), col="lightblue",
     xlab="p", main="p = 0.9")
xs1 = seq(0.6,1.1,0.0001)
ys1 = dnorm(xs1, 0.9, sqrt((0.9*0.1)/n))
lines(xs1,ys1,lwd=3,col="red")

hist(p025, prob=T, xlim=c(0,0.6), col="pink",
     xlab="p", main="p = 0.25")
xs2 = seq(0,0.6,0.0001)
ys2 = dnorm(xs2, 0.25, sqrt((0.25*0.75)/n))
lines(xs2, ys2, lwd=3, col="red")
```





```
library(ggplot2)
library(tidyr)
library(dplyr)

# Se crean los dataframes
df = data.frame(
  p = c(p09, p025),
  group = c(rep('p = 0.9', length(p09)), rep('p = 0.25', length(p025)))
)

xs1 = seq(0.6, 1.1, 1e-04)
ys1 = dnorm(xs1, 0.9, sqrt((0.9 * 0.1) / n))
```

```r
xs2 = seq(0, 0.6, 1e-04)
ys2 = dnorm(xs2, 0.25, sqrt((0.25 * 0.75) / n))

normal_df = data.frame(
  p = c(xs1, xs2),
  density = c(ys1, ys2),
  group = c(rep('p = 0.9', length(xs1)), rep('p = 0.25', length(xs2)))
)

ggplot(df, aes(x = p)) +
    geom_histogram(data = df %>% filter(group == 'p = 0.9'),
    aes(y = ..density..), fill = "lightblue", binwidth = 0.05) +
    geom_histogram(data = df %>% filter(group == 'p = 0.25'),
    aes(y = ..density..), fill = "pink", binwidth = 0.05) +
    geom_line(data = normal_df, aes(y = density), color = "red", size = 1) +
    facet_wrap(~ group, scales = "free_x") +
    labs(x = "p") +
    theme_classic()
```



La correlación entre GPA y LSAT es

```
library(bootstrap)
data(law)
(lawCor = with(law,cor(GPA,LSAT)))
```

```
[1] 0.7763745
```

Siguiendo a Efron y Tibshirani (1993) tiene un error estándar igual a

```r
(se = (1-lawCor^2)/sqrt(dim(law)[1]-3))
```

```
[1] 0.1146741
```

```r
c((lawCor - 1.96*se), min(1, (lawCor + 1.96*se)))
```

```
[1] 0.5516133 1.0000000
```

Alternativamente se puede usar la librería `psychometric`

```r
psychometric::CIr(lawCor, dim(law)[1])
```

```
[1] 0.4385108 0.9219648
```

Usando bootstrap se puede *evitar* asumir que F se distribuye como una normal bivariante.

```r
samplesize = dim(law)[1]
ind = 1:samplesize

law.boot =
replicate(1000, {indB = sample(ind,replace=TRUE);
with(law[indB,], cor(GPA,LSAT))})

sd(law.boot)
```

```
[1] 0.1329626
```

Desde el punto de vista clásico, el error estándar del estimador de la correlación (asumiendo aproximadamente normalidad) es igual a

$$SE_r = \sqrt{\frac{1-r^2}{n-2}}$$

se puede estimar así:

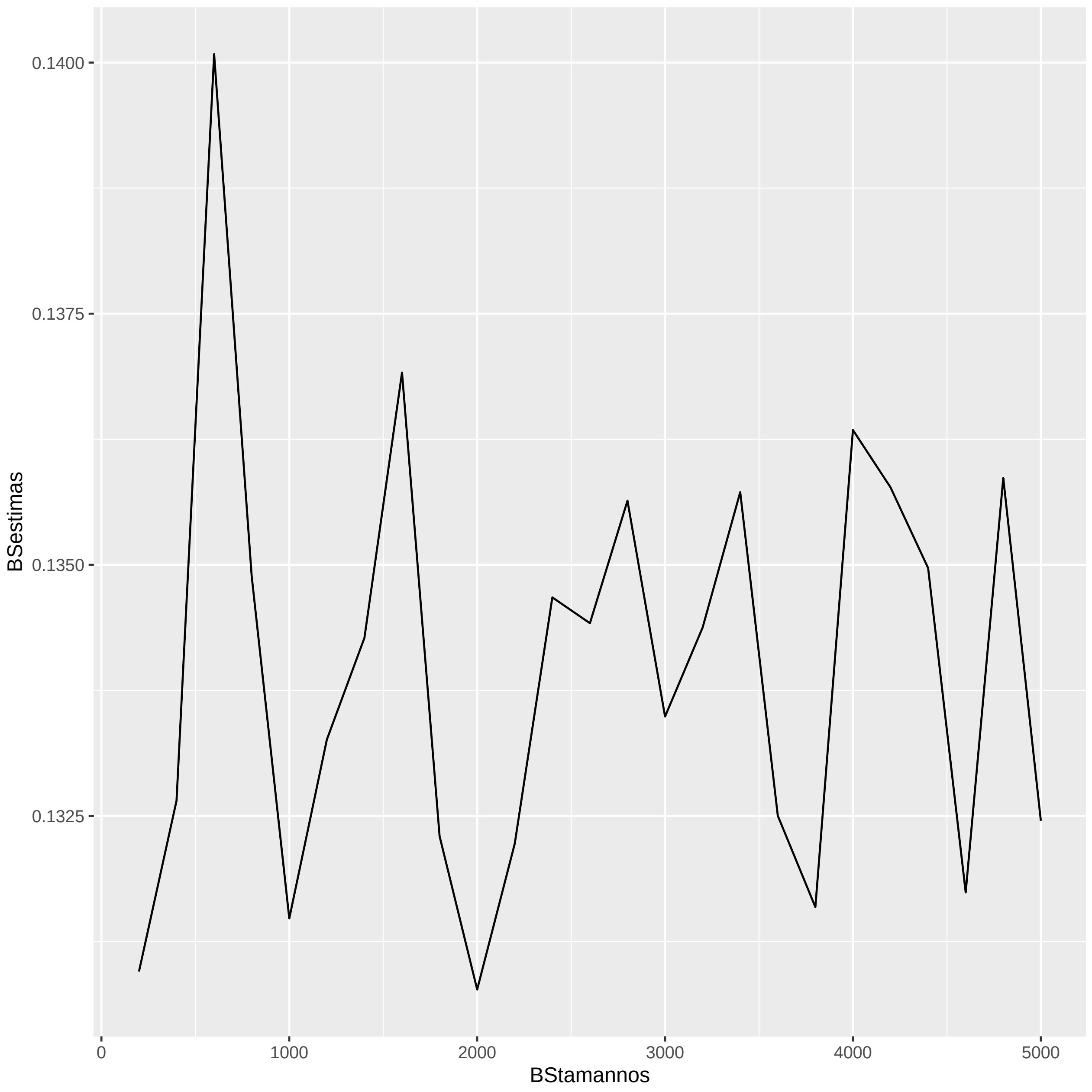
```r
# Asumes SE_r = sqrt((1-r^2)/(n-2))

cor.test.plus = function(x) {
  list(x,
   Standard.Error =
   unname(sqrt((1 - x$estimate^2)/x$parameter)))
}
```

```r
library(bootstrap)
cor.test.plus(cor.test(law$GPA, law$LSAT))
```

```
[[1]]

    Pearson's product-moment correlation

data:  law$GPA and law$LSAT
t = 4.4413, df = 13, p-value = 0.0006651
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.4385108 0.9219648
sample estimates:
      cor
0.7763745


$Standard.Error
[1] 0.174806
```

¿Cómo converge de rápido el estimador bootstrap?

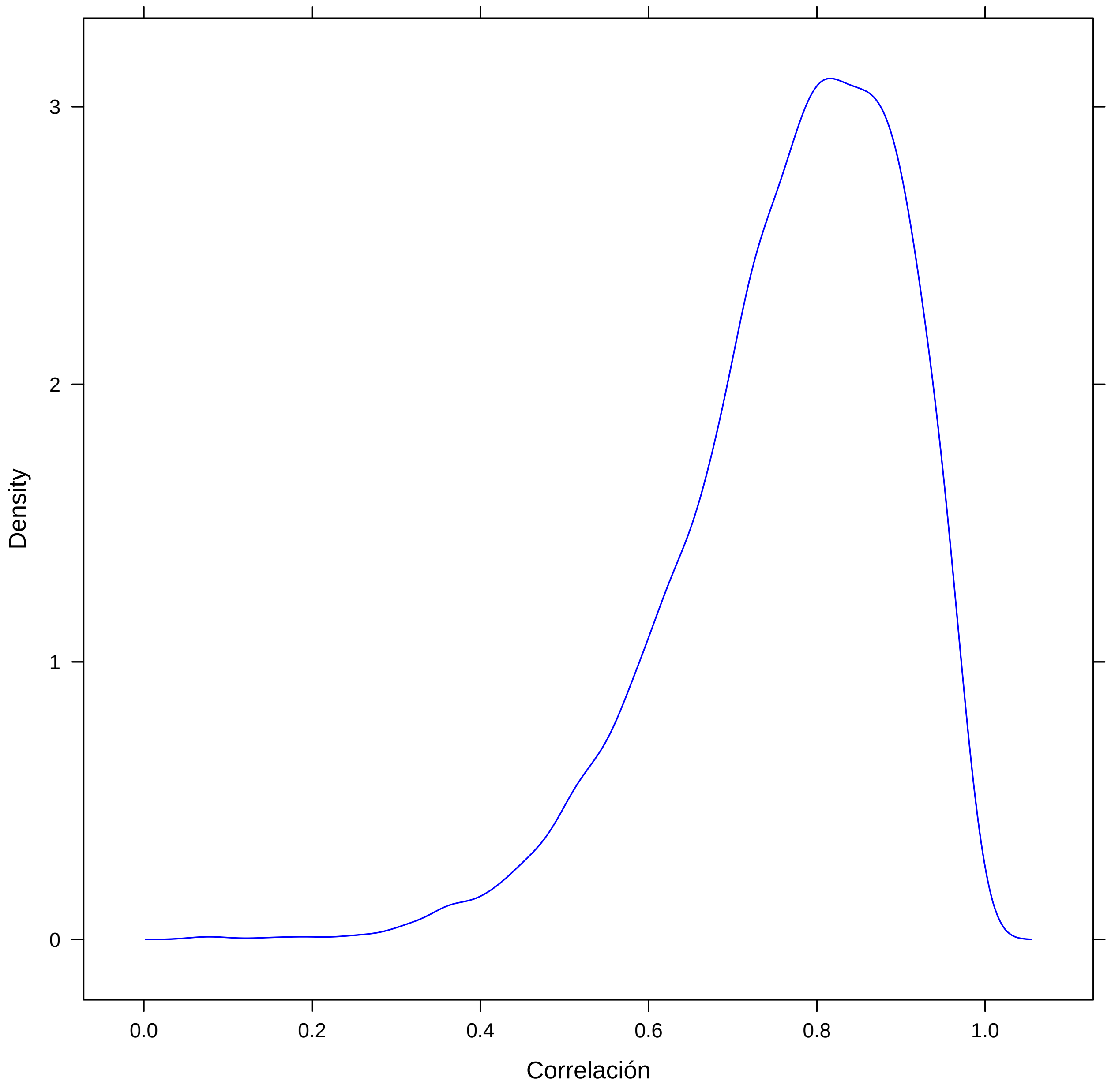
```r
samplesize = dim(law)[1]
ind = 1:samplesize
lawBS = function(B) sd(replicate(B,
{indB = sample(ind,replace=TRUE);
with(law[indB,],cor(GPA,LSAT))}))

BStamannos = seq(200,5000,200)
BSestimas = sapply(BStamannos,lawBS)

library(ggplot2)
qplot(BStamannos, BSestimas, geom="path")
```



Se obtiene la distribución empírica muestral de la población $\hat{F}(\hat{\theta}^*)$

```r
ind1 = 1:dim(law)[1]
law.boot = replicate(5000, {indB = sample(ind1,
size=length(ind1), replace=TRUE);
with(law[indB,], cor(GPA,LSAT))})
```

```r
library(latticeExtra)

densityplot(~law.boot, plot.points=FALSE, xlab="Correlación", col="blue")
```



El estimador bootstrap del error estándar a partir de la muestra $se_{\hat{F}}\left(\hat{\rho}\right)$ es

```
sd(law.boot)
```

```
[1] 0.1332714
```

Ejemplo de los institutos de máster en leyes

```r
library(bootstrap)

paraBoot = function(datos) {
   ndatos = dim(datos)[1]
   sigma = cov(datos)
   mu = sapply(datos, mean)

   c = sqrt(prod(diag(sigma))/sigma[1,2]^2-1)
  z1 = rnorm(ndatos)
  z2 = rnorm(ndatos)
  x = mu[1] + sqrt(sigma[1,1])*z1
  y = mu[2] + sqrt(sigma[2,2])*(z1+c*z2)/sqrt(1+c^2)
cbind(x,y)
}
```

Alternativamente, se puede programar con una librería específica que trata la normal multivariante: `mvrnorm`

```r
paraBoot = function(datos) {
   ndatos = dim(datos)[1]
   sigma = cov(datos)
   mu = sapply(datos, mean)

   x = MASS::mvrnorm(ndatos, mu=mu, Sigma=sigma)
return(x)
}
```

Aproximación con bootstrap paramétrico

```r
pBoot = replicate(5000, cor(paraBoot(law))[2,1])

sd(pBoot)
```

```
[1] 0.1217163
```

Con la librería `boot`:

```r
library(boot)

simulaBoot = function(datos, ...) {
  ndatos = dim(datos)[1]
  x = MASS::mvrnorm(ndatos, mu=mu0, Sigma=sigma0)
  return(x)
}
```

```r
sigma0 = Rfast::mvnorm.mle(as.matrix(law))$sigma
mu0 = Rfast::mvnorm.mle(as.matrix(law))$mu

estadistico = function(x,i){ cor(x)[2,1] }


saleB = boot(data=law, sim="parametric", ran.gen=simulaBoot, mle=list(mu0, sigma0),
statistic = estadistico, R=1000)

saleB
```

```
PARAMETRIC BOOTSTRAP


Call:
boot(data = law, statistic = estadistico, R = 1000, sim = "parametric",
    ran.gen = simulaBoot, mle = list(mu0, sigma0))


Bootstrap Statistics :
     original       bias    std. error
t1* 0.7763745 -0.002194144   0.1079905
```

```r
boot.ci(saleB,type = "perc")
```

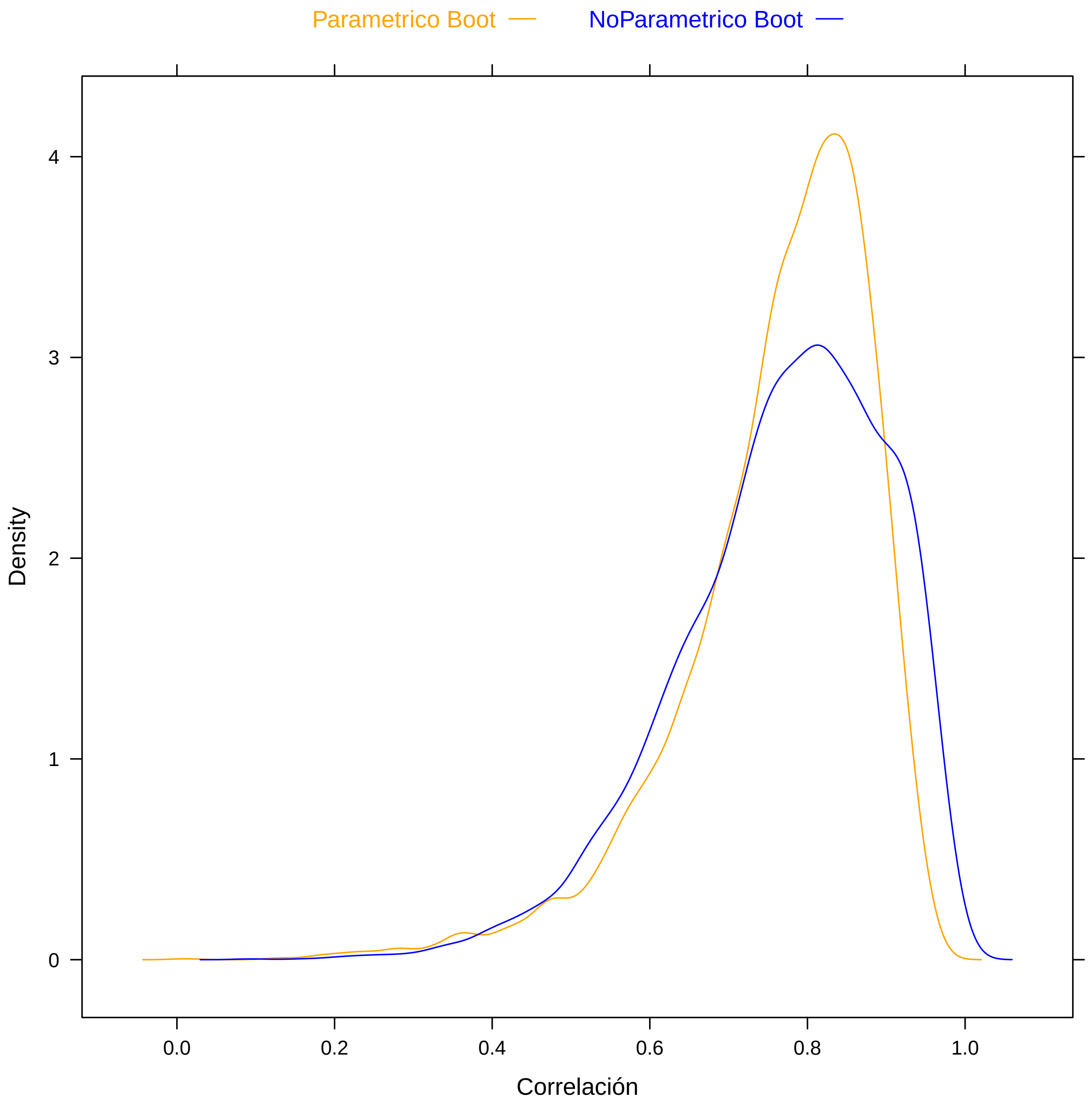
```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL : 
boot.ci(boot.out = saleB, type = "perc")

Intervals : 
Level     Percentile     
95%   ( 0.5123,  0.9256 )  
Calculations and Intervals on Original Scale
```

Aproximación asintótica

```r
samplesize = dim(law)[1]

(1-cor(law)[2,1]^2)/sqrt(samplesize-3)
```

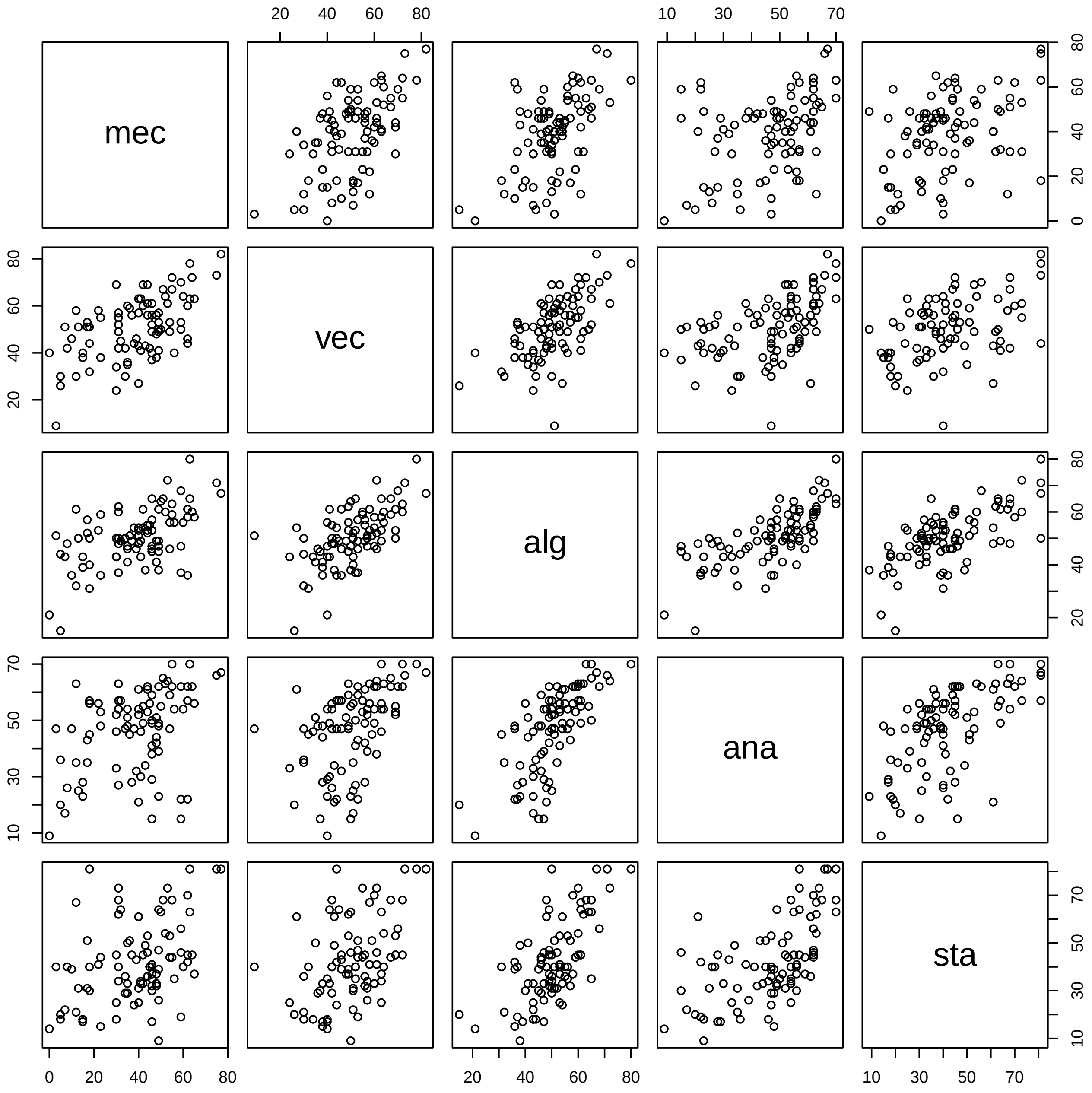
```
[1] 0.1146741
```

```
library(latticeExtra)

ind1 = 1:dim(law)[1]
law.boot = replicate(5000, {indB = sample(ind1,
size=dim(law)[1], replace=TRUE);
with(law[indB,], cor(GPA,LSAT))})

densityplot(~pBoot + law.boot, plot.points=FALSE,
auto.key=list(columns=2, size=2, between=1, col=c("orange","blue"),
text=c("Parametrico Boot", "NoParametrico Boot")), xlab="Correlación",
par.settings=list(superpose.line=list(col=c("orange","blue"))))
```



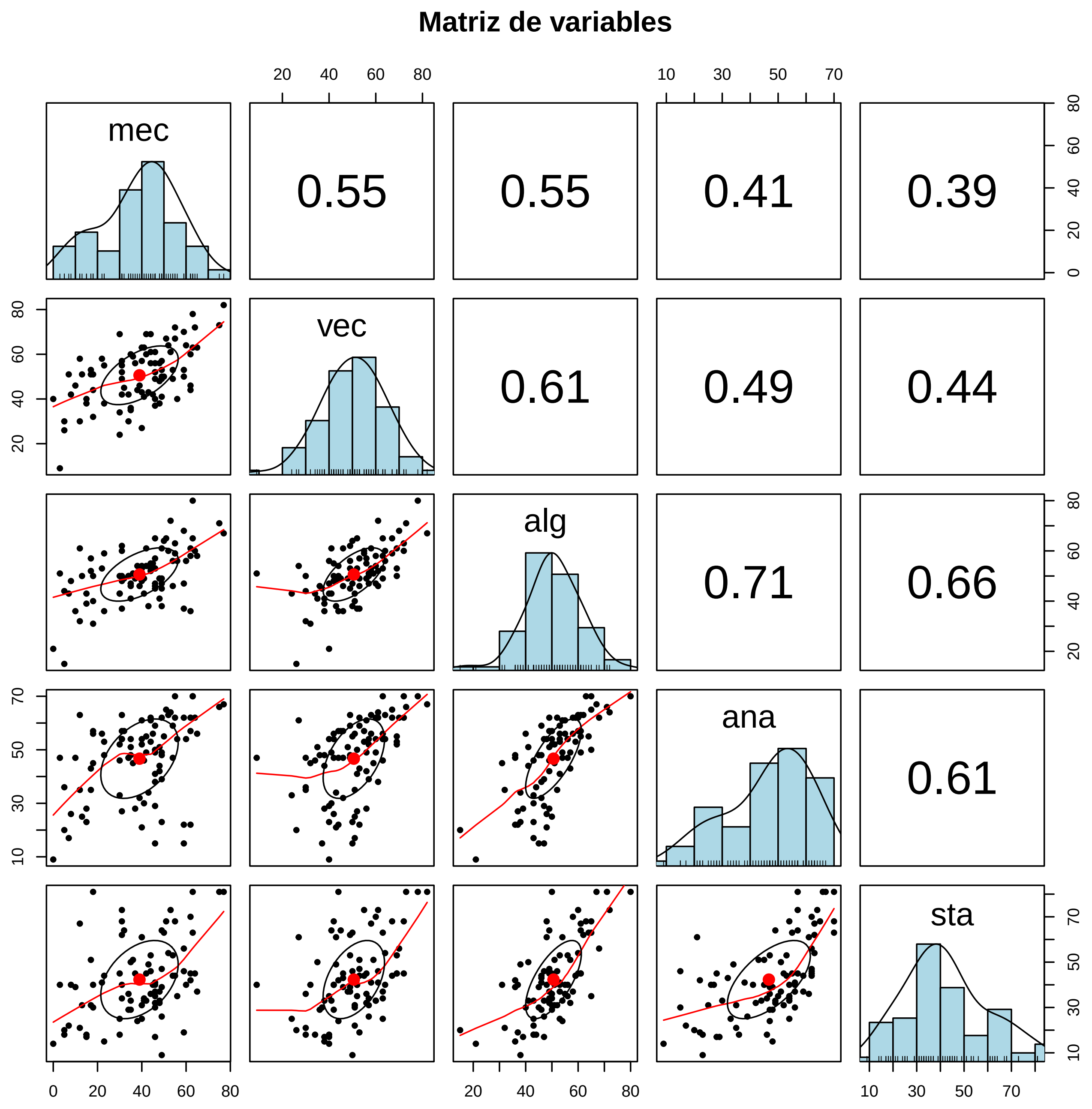
Ejemplo sobre calificaciones de alumnos

```r
library(bootstrap)
data(scor)
plot(scor)
```



Alternativamente

```
library(psych)

pairs.panels(scor, method = "pearson",
             hist.col = "lightblue",
             density = TRUE,
             ellipses = TRUE,
main="Matriz de variables")
```



El vector de medias y la correspondiente matriz de covarianzas son:

```r
colMeans(scor)
```

```
     mec      vec      alg      ana      sta 
38.95455 50.59091 50.60227 46.68182 42.30682 
```

```r
cov(scor)
```

```
         mec       vec       alg       ana       sta
mec 305.7680 127.22257 101.57941 106.27273 117.40491
vec 127.2226 172.84222  85.15726  94.67294  99.01202
alg 101.5794  85.15726 112.88597 112.11338 121.87056
ana 106.2727  94.67294 112.11338 220.38036 155.53553
sta 117.4049  99.01202 121.87056 155.53553 297.75536
```

Se calculan los autovalores y autovectores de la matriz de covarianzas.

```r
eigen(cov(scor))$values    # Autovalores
```

```
[1] 686.98981 202.11107 103.74731  84.63044  32.15329
```

```r
eigen(cov(scor))$vectors   # Autovectores
```

```
           [,1]        [,2]       [,3]         [,4]        [,5]
[1,] -0.5054457  0.74874751 -0.2997888  0.296184264 -0.07939388
[2,] -0.3683486  0.20740314  0.4155900 -0.782888173 -0.18887639
[3,] -0.3456612 -0.07590813  0.1453182 -0.003236339  0.92392015
[4,] -0.4511226 -0.30088849  0.5966265  0.518139724 -0.28552169
[5,] -0.5346501 -0.54778205 -0.6002758 -0.175732020 -0.15123239
```

En componente principales svd es numéricamente más estable que la descomposición por autovectores y autovalores, pero para aplicar bootstrap esta última es mas rápida

La función `prcomp()` es una de las múltiples funciones en R que realizan **PCA**.

Por defecto, `prcomp()` centra las variables para que tengan media cero, pero si se quiere además que su desviación estándar sea de uno, hay que indicar `scale = TRUE`.

```r
pca = prcomp(scor, scale = TRUE)
names(pca)
```

```
[1] "sdev"     "rotation" "center"   "scale"    "x"
```

```
pca$sdev^2
```

```
[1] 3.1809801 0.7395718 0.4449651 0.3878924 0.2465905
```

```
# Es equivalente a:
eigen(cor(scor))$values
```

```
[1] 3.1809801 0.7395718 0.4449651 0.3878924 0.2465905
```

```
prop_varianza = pca$sdev^2 / sum(pca$sdev^2)
prop_varianza
```

```
[1] 0.63619603 0.14791437 0.08899303 0.07757848 0.04931810
```

Analizar con detalle el vector de *loadings* que forma cada componente puede ayudar a interpretar que tipo de información recoge cada una de ellas

```
pca$rotation
```

```
           PC1        PC2         PC3        PC4        PC5
mec -0.3996045 -0.6454583  0.62078249 -0.1457865 -0.1306722
vec -0.4314191 -0.4415053 -0.70500628  0.2981351 -0.1817479
alg -0.5032816  0.1290675 -0.03704901 -0.1085987  0.8466894
ana -0.4569938  0.3879057 -0.13618182 -0.6662561 -0.4221885
sta -0.4382444  0.4704545  0.31253342  0.6589164 -0.2340223
```

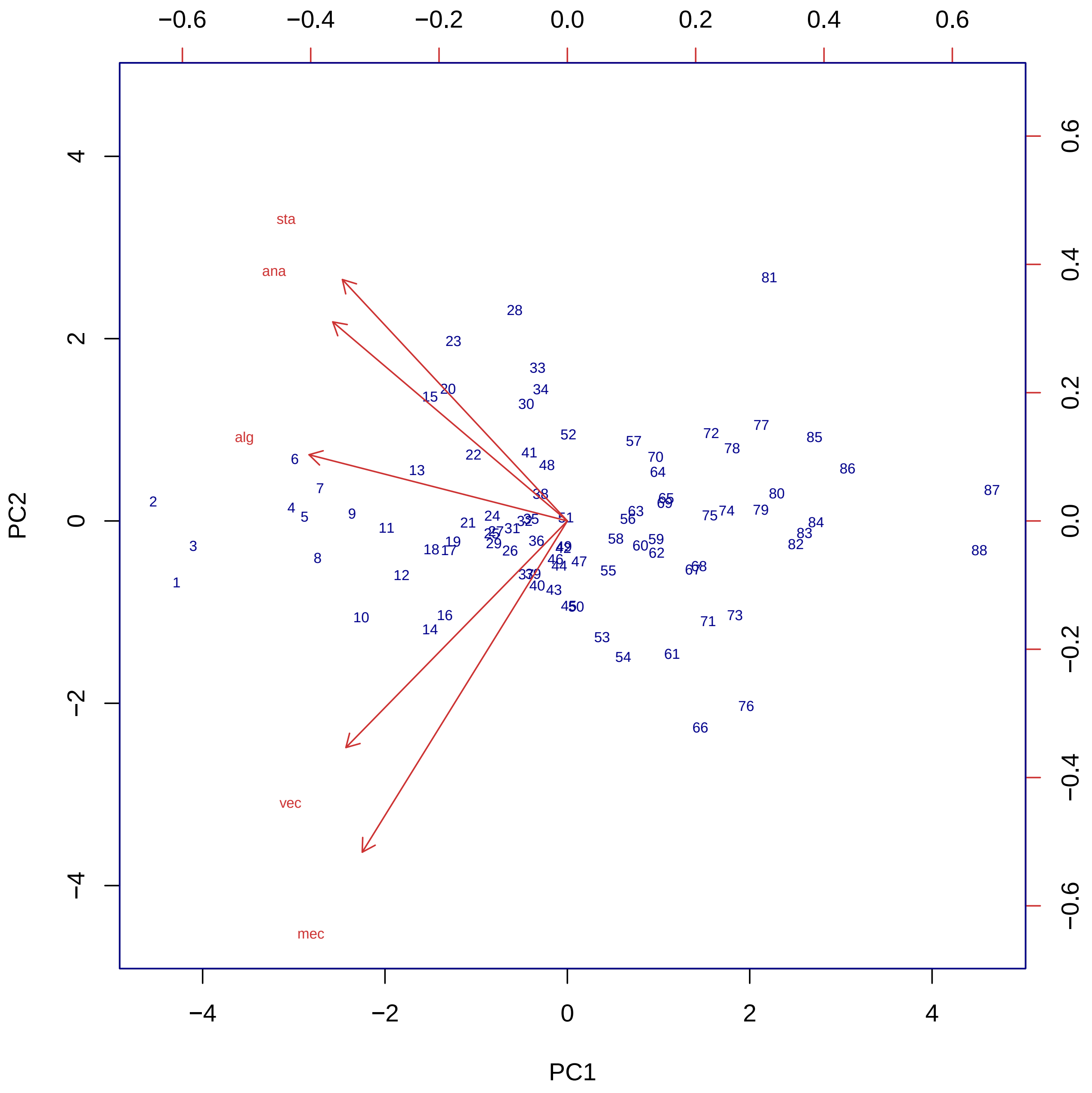
```
head(pca$x)
```

```
        PC1         PC2        PC3        PC4         PC5
1 -4.285041 -0.67410225  0.1235891  0.7931108 -0.51438033
2 -4.541989  0.21331176 -0.2317797  0.5516063  0.59618974
3 -4.102690 -0.27557530  0.5304377  0.6096862 -0.02781595
4 -3.026846  0.14916207 -0.3702154  0.1595890 -0.43950477
5 -2.882081  0.04408014  0.2988861 -0.3225740 -0.14767795
6 -2.988775  0.68126196  0.2628756  0.2950331  0.54754485
```

La función `prcomp()` calcula automáticamente el valor de las componentes principales para cada observación (principal component scores) multiplicando los datos por los vectores de *loadings*. El resultado se almacena en la matriz x.

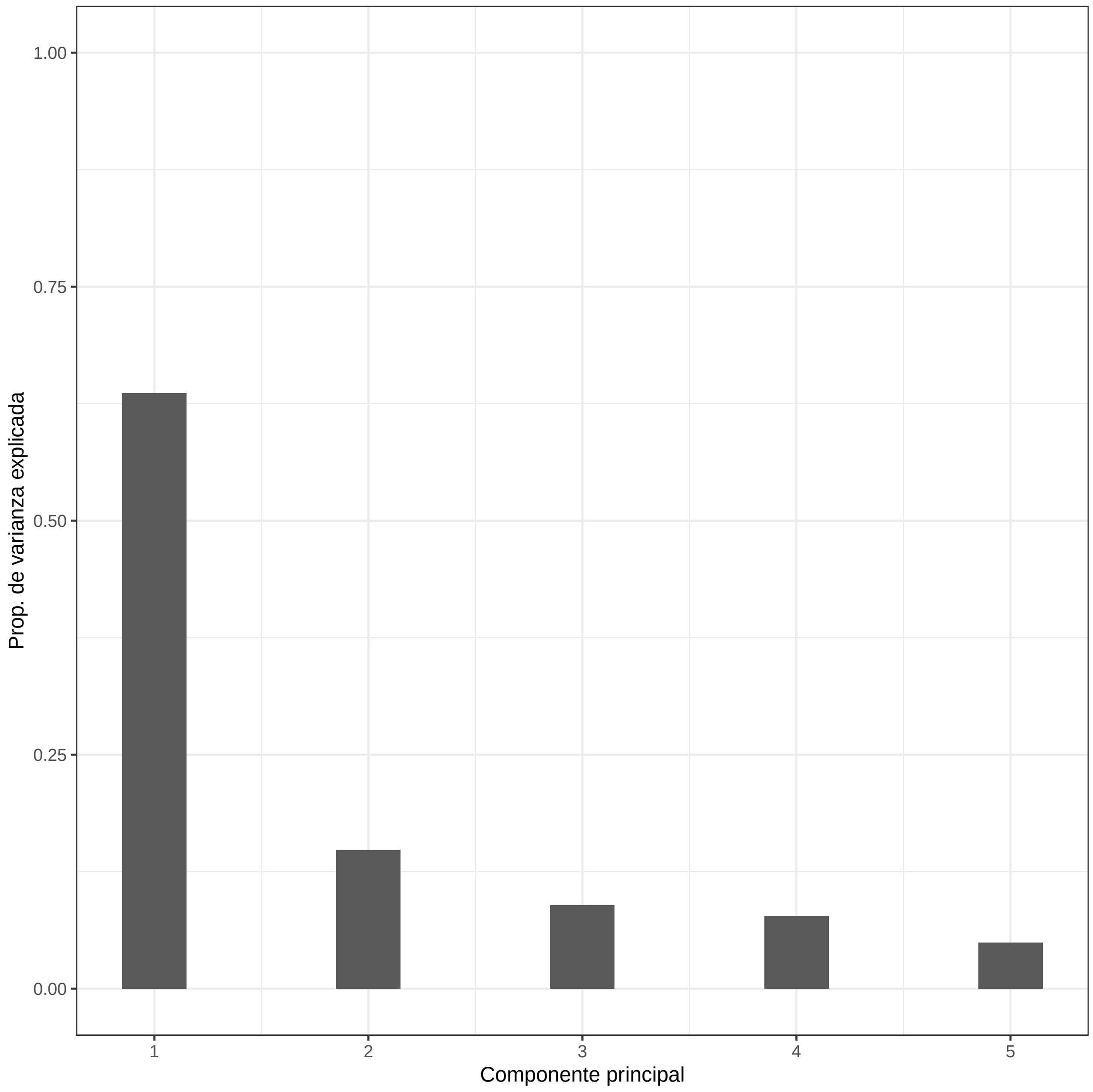
Mediante la función `biplot()` se puede obtener una representación bidimensional de las dos primeras componentes. Es recomendable indicar el argumento `scale = 0` para que las flechas estén en la misma escala que las componentes.

```r
biplot(x = pca, scale = 0, cex = 0.6, col = c("blue4", "brown3"))
```



Una vez calculadas las componentes principales, se puede conocer la varianza explicada por cada una de ellas, la proporción respecto al total y la proporción de varianza acumulada.

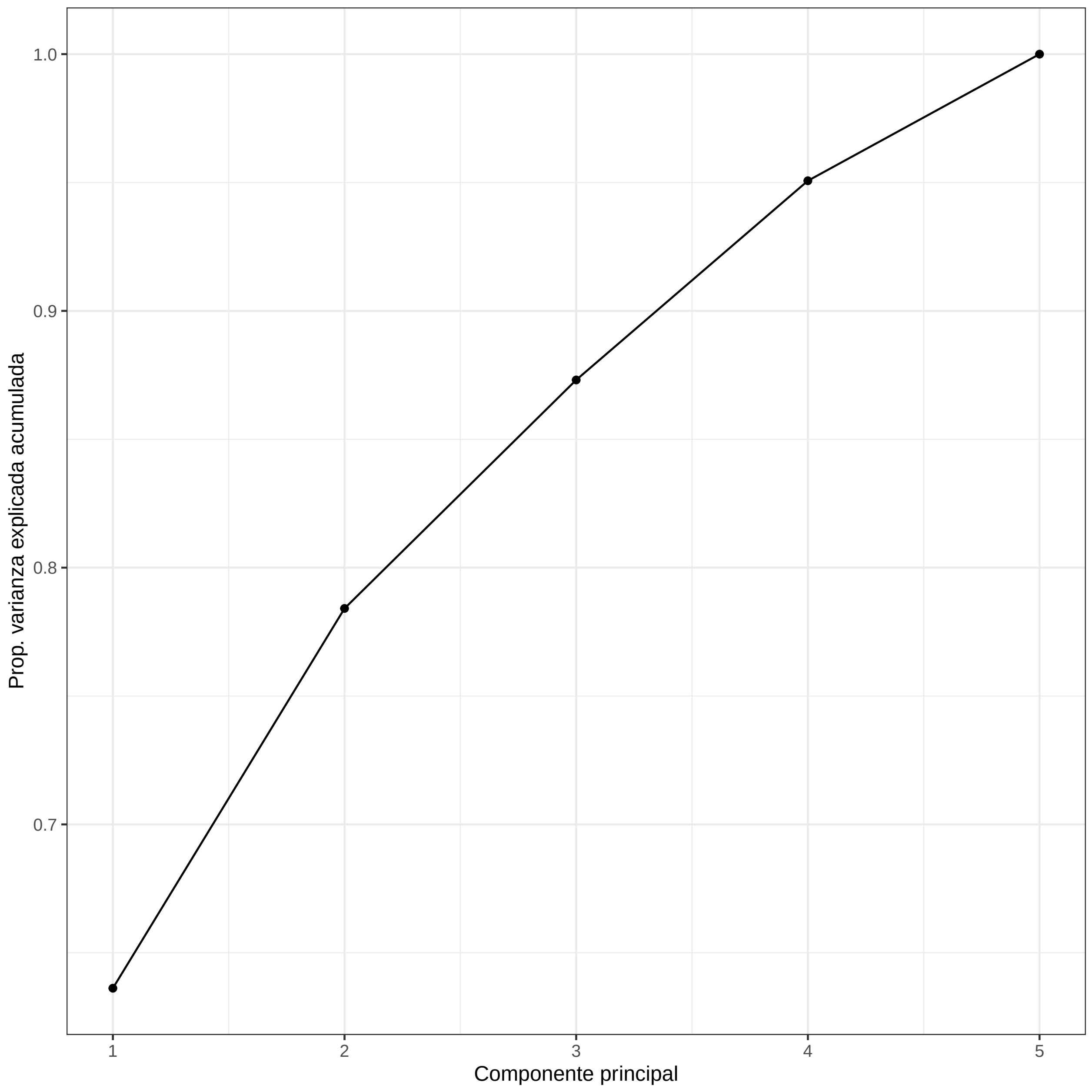
```r
library(ggplot2)

ggplot(data = data.frame(prop_varianza, pc = 1:5),
       aes(x = pc, y = prop_varianza)) +
  geom_col(width = 0.3) +
  scale_y_continuous(limits = c(0,1)) +
  theme_bw() +
  labs(x = "Componente principal",
       y = "Prop. de varianza explicada")
```



```r
prop_varianza_acum = cumsum(prop_varianza)
prop_varianza_acum
```

```
[1] 0.6361960 0.7841104 0.8731034 0.9506819 1.0000000
```

```r
ggplot(data = data.frame(prop_varianza_acum, pc = 1:5),
       aes(x = pc, y = prop_varianza_acum, group = 1)) +
  geom_point() +
  geom_line() +
  theme_bw() +
  labs(x = "Componente principal",
       y = "Prop. varianza explicada acumulada")
```



```r
library(boot)

autovals = eigen(var(scor), symmetric=TRUE,
only.values=TRUE)$values
(teta = autovals[1] / sum(autovals))
```

```
[1] 0.619115
```

```r
autoval = function(X, ind){
  vals = eigen(var(X[ind,]), symmetric=TRUE,
  only.values=TRUE)$values
  vals[1] / sum(vals)
}

scor.boot = boot(scor, statistic = autoval, R=500)
names(scor.boot)
```

```
 [1] "t0"        "t"         "R"         "data"      "seed"      "statistic"
 [7] "sim"       "call"      "stype"     "strata"    "weights"
```

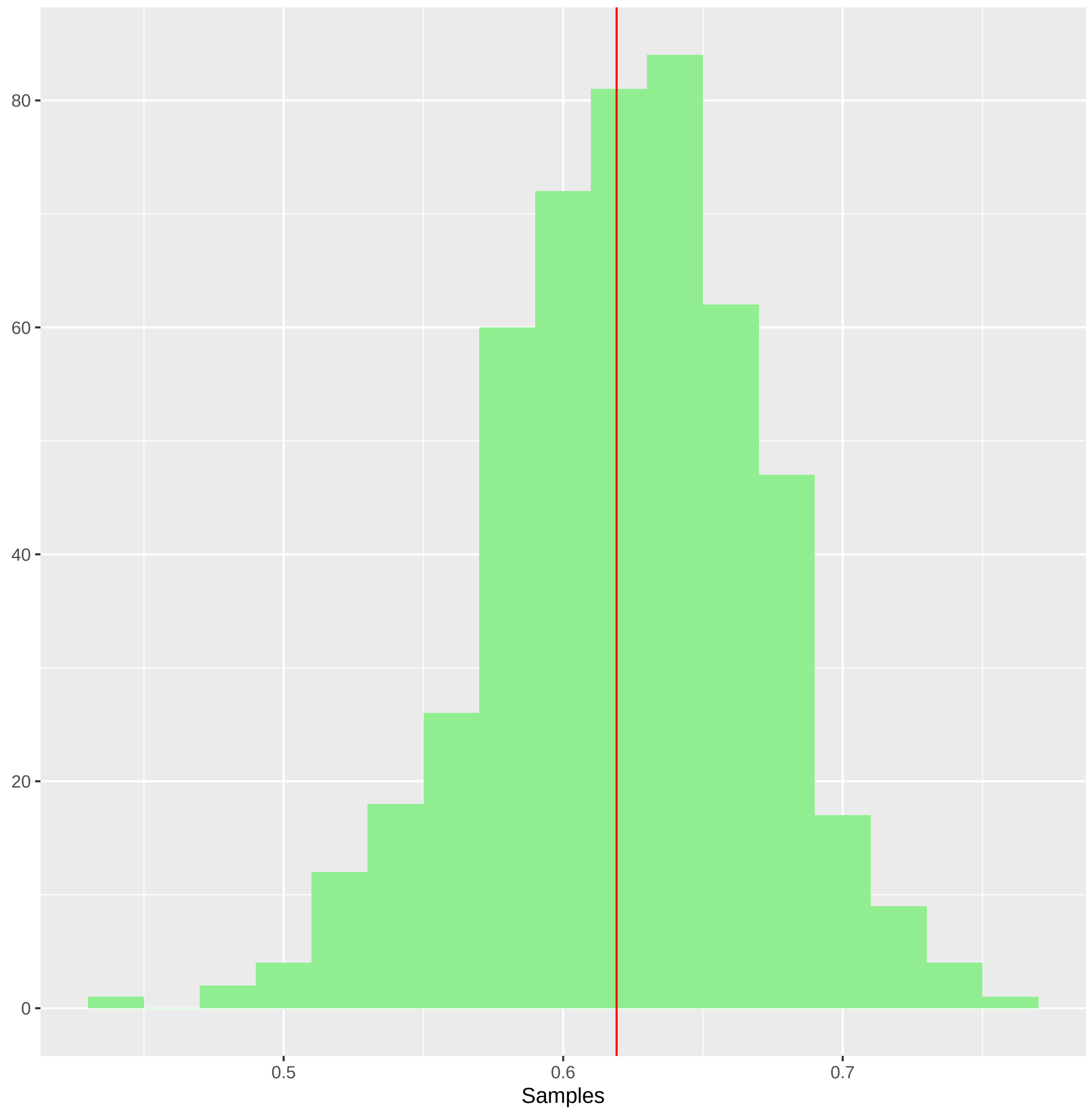
Error estándar del bootstrap

```r
sd(scor.boot$t)
```

```
[1] 0.04789772
```

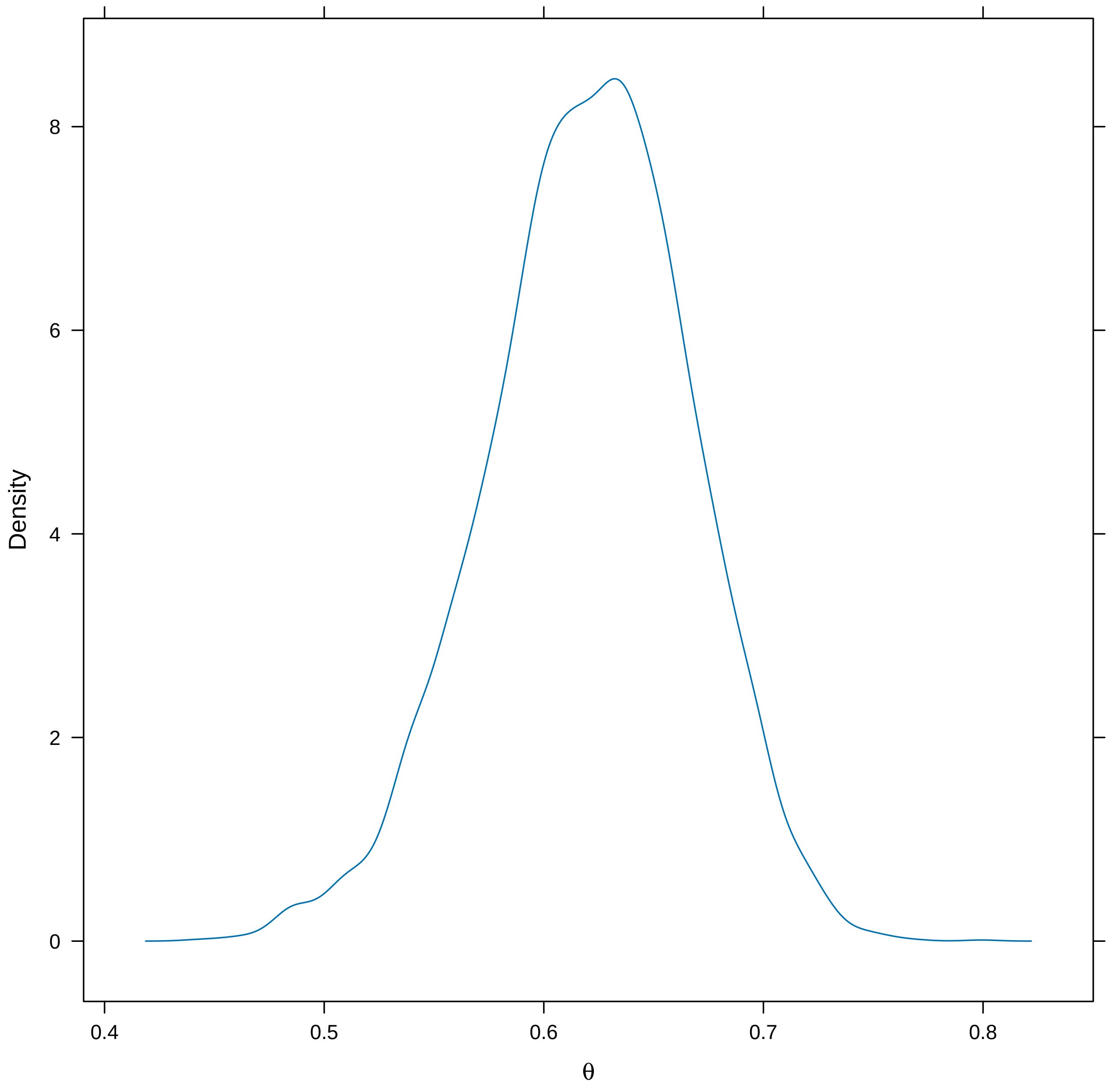
Distribución bootstrap

```r
library(ggplot2)

qplot(scor.boot$t, geom="histogram",binwidth=0.02,
fill=I("lightgreen"), xlab="Samples") +
geom_vline(xintercept=teta, col="red")
```



```r
library(bootstrap)
data(scor)
X = scor

eigenTeta = function(X) {
   ee = eigen(cov(X))[["values"]]
   ee[1]/sum(ee)
}

ind = 1:dim(X)[[1]]
```

```r
eigendist = replicate(5000,
eigenTeta(X[sample(ind, replace=TRUE),]))

library(latticeExtra)

densityplot(eigendist, plot.points=FALSE,
xlab=expression(theta))
```



```r
if (!require(data4PCCAR)) remotes::install_github("HerveAbdi/data4PCCAR")

library(data4PCCAR)
```

```
booteig = boot.eigen(scor, nIter = 1000)
booteig$fixed.eigs
```

```
[1] 3.1809801 0.7395718 0.4449651 0.3878924 0.2465905
```

```
# O de modo manual a partir de las correlaciones
eigen(cor(scor))$values
```

```
[1] 3.1809801 0.7395718 0.4449651 0.3878924 0.2465905
```

```
quantile(booteig$boot.eigs.sorted[,1], c(0.025,0.975))
```

```
    2.5%    97.5%
2.741854 3.551240
```

Tanto $\hat{\mathbf{v}}_1$ como $\hat{\mathbf{v}}_2$ son estadísticos del mismo modo que lo es $\hat{\theta}$, y de este modo se puede aplicar el bootstrap para calcular su variabilidad.

```
library(bootstrap)
data(scor)

X = scor

eigenVec = function(X) {
  ee = eigen(cov(X))[["vectors"]]
  return(cbind(ee[,1], ee[,2]))
}
```

```
ind = 1:dim(X)[[1]]
eigendist = replicate(500,
eigenVec(X[sample(ind, replace=TRUE),]))

apply(eigendist[1:5,1,], 1, sd)
```

```
[1] 0.2908766 0.2109680 0.1933209 0.2538687 0.3003810
```

Nota:

`eigendist` es un array de tres dimensiones

Por ejemplo:

```r
eigendist[1:5,,1:10]
```

```
, , 1

           [,1]        [,2]
[1,] -0.4348181 -0.64058557
[2,] -0.3611497 -0.30780680
[3,] -0.3913879 -0.05842695
[4,] -0.4416366  0.01949274
[5,] -0.5764344  0.70079343

, , 2

           [,1]       [,2]
[1,] -0.5653350  0.7003126
[2,] -0.3133882  0.1150995
[3,] -0.3569485 -0.0319872
[4,] -0.4770592 -0.2106664
[5,] -0.4766409 -0.6714990

, , 3

           [,1]        [,2]
[1,] -0.5882195 -0.58460446
[2,] -0.4068386 -0.31913620
[3,] -0.3192764  0.05138059
[4,] -0.4058974  0.39477102
[5,] -0.4709459  0.63079758

, , 4

           [,1]        [,2]
[1,] -0.5212370  0.75983894
[2,] -0.3918757 -0.10895787
[3,] -0.3649994  0.04925438
[4,] -0.4624598 -0.16641868
[5,] -0.4771288 -0.61696985

, , 5

           [,1]        [,2]
[1,] -0.5308490  0.72020281
[2,] -0.3858716  0.24192023
[3,] -0.2715896 -0.08178232
[4,] -0.4690164 -0.45443152
[5,] -0.5249431 -0.45780581

, , 6

           [,1]        [,2]
[1,] -0.5543524 -0.71381853
[2,] -0.3779566 -0.18051560
[3,] -0.3151340  0.06993942
```

```
[4,] -0.3973982  0.27705300
[5,] -0.5409320  0.61337373

, , 7

           [,1]       [,2]
[1,] -0.5175180  0.6949218
[2,] -0.4261540  0.2430267
[3,] -0.3296295 -0.2004155
[4,] -0.4109178 -0.5633560
[5,] -0.5225504 -0.3169943

, , 8

           [,1]        [,2]
[1,] -0.5152191  0.73919088
[2,] -0.3580193  0.22208641
[3,] -0.3207878 -0.06240543
[4,] -0.4271178 -0.27493189
[5,] -0.5666013 -0.56990568

, , 9

           [,1]        [,2]
[1,] -0.5082603  0.77650118
[2,] -0.3166010  0.17289814
[3,] -0.3375322 -0.03093558
[4,] -0.4551867 -0.35521778
[5,] -0.5659614 -0.48991373

, , 10

           [,1]        [,2]
[1,] -0.5136062 -0.74263232
[2,] -0.3816097 -0.18075070
[3,] -0.3503268  0.07043135
[4,] -0.4093375  0.24971227
[5,] -0.5479933  0.59034704
```

```
eigendist[1:5,1,1:10]
```

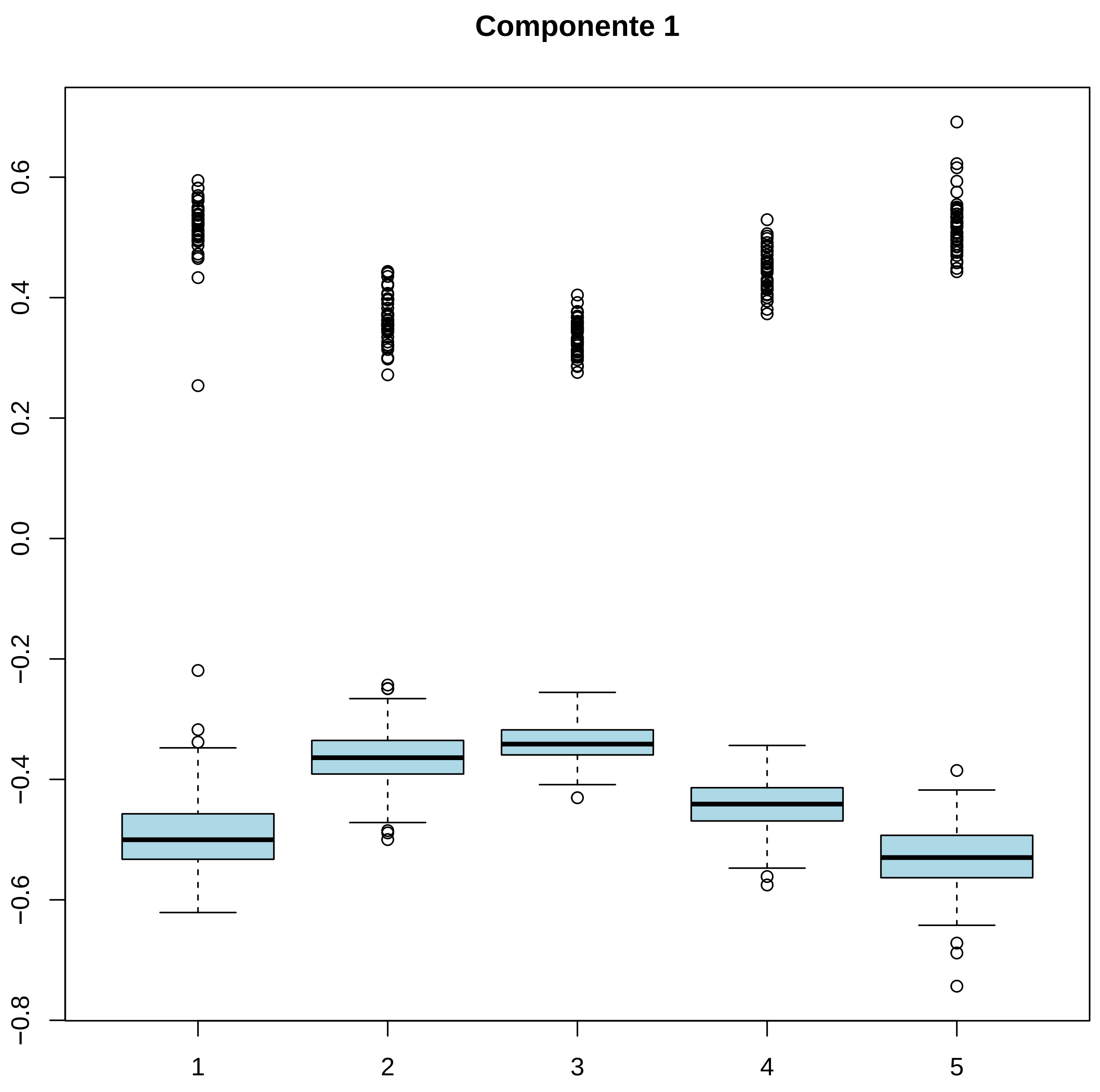
```
           [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
[1,] -0.4348181 -0.5653350 -0.5882195 -0.5212370 -0.5308490 -0.5543524
[2,] -0.3611497 -0.3133882 -0.4068386 -0.3918757 -0.3858716 -0.3779566
[3,] -0.3913879 -0.3569485 -0.3192764 -0.3649994 -0.2715896 -0.3151340
[4,] -0.4416366 -0.4770592 -0.4058974 -0.4624598 -0.4690164 -0.3973982
[5,] -0.5764344 -0.4766409 -0.4709459 -0.4771288 -0.5249431 -0.5409320
           [,7]       [,8]       [,9]      [,10]
[1,] -0.5175180 -0.5152191 -0.5082603 -0.5136062
[2,] -0.4261540 -0.3580193 -0.3166010 -0.3816097
[3,] -0.3296295 -0.3207878 -0.3375322 -0.3503268
[4,] -0.4109178 -0.4271178 -0.4551867 -0.4093375
[5,] -0.5225504 -0.5666013 -0.5659614 -0.5479933
```

Se calculan los errores estándar de cada componente:

```r
apply(eigendist[1:5,2,], 1, sd)
```

```
[1] 0.49635810 0.19976453 0.07478552 0.22834516 0.41802023
```

```r
boxplot(cbind(eigendist[1,1,], eigendist[2,1,],
eigendist[3,1,], eigendist[4,1,], eigendist[5,1,]),
main="Componente 1", col="lightblue")
```

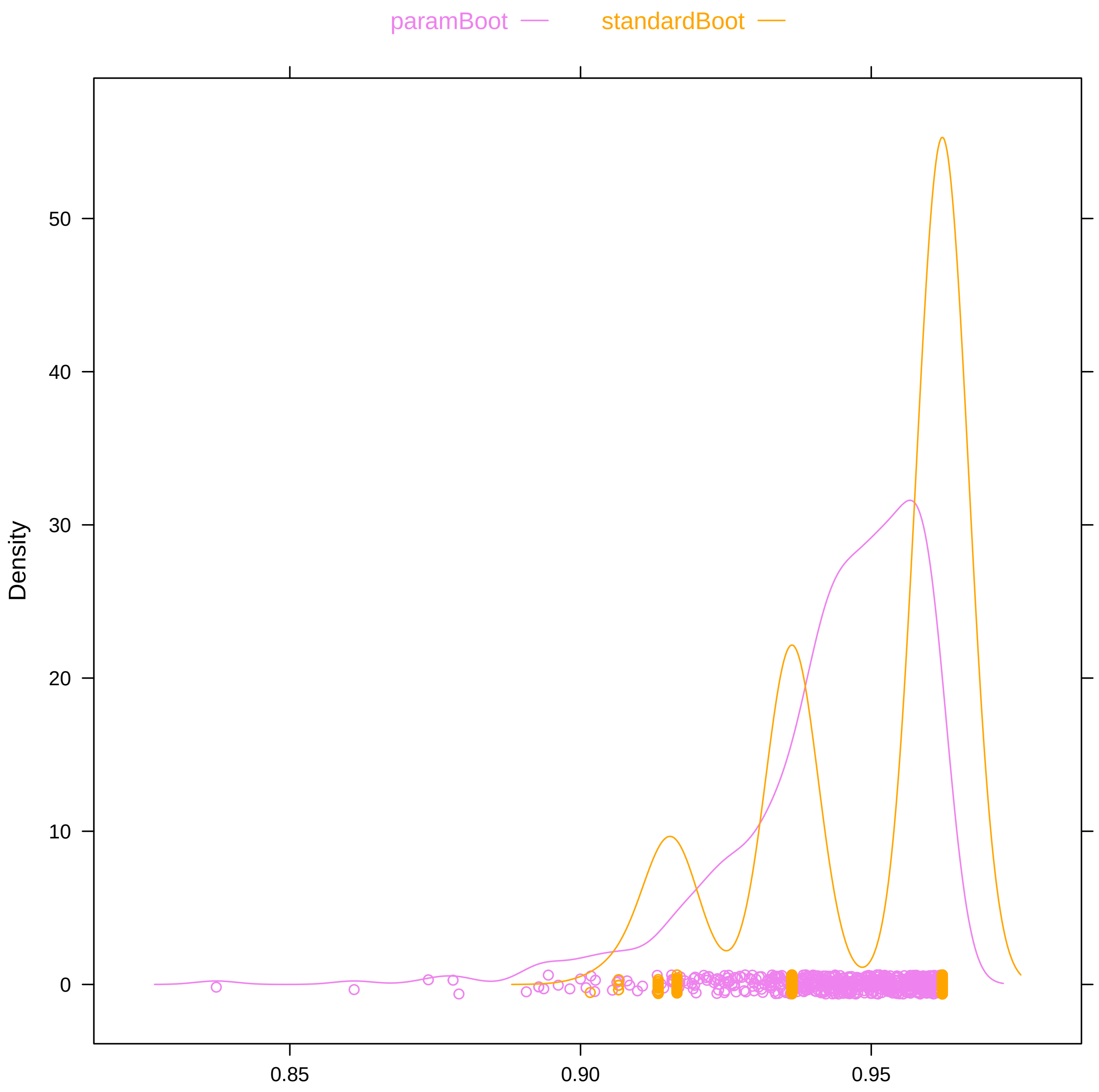


Cuando puede fallar el bootstrap

```r
N = 50
X = runif(N)
(thetaHat = max(X))
```

```
[1] 0.9622238
```

```r
standardBoot = replicate(500,
max(sample(X, N, replace=TRUE)))

paramBoot = replicate(500,
max(runif(N, min=0, max=thetaHat)))

library(latticeExtra)

densityplot(~paramBoot + standardBoot, xlab="",
auto.key=list(columns=2, size=2, between=1,
col=c("violet", "orange")),
par.settings=list(superpose.line=
list(col=c("violet", "orange"))),
col=c("violet", "orange"))
```



Ejemplo de los ratones

Se toma el ejemplo de las diferencias de medias entre ratones según son tratamiento o control

```
Trata = c(94,197,16,38,99,141,23)
Cont = c(52,104,146,10,51,30,40,27,46)

mean(Trata) - mean(Cont)
```

```
[1] 30.63492
```

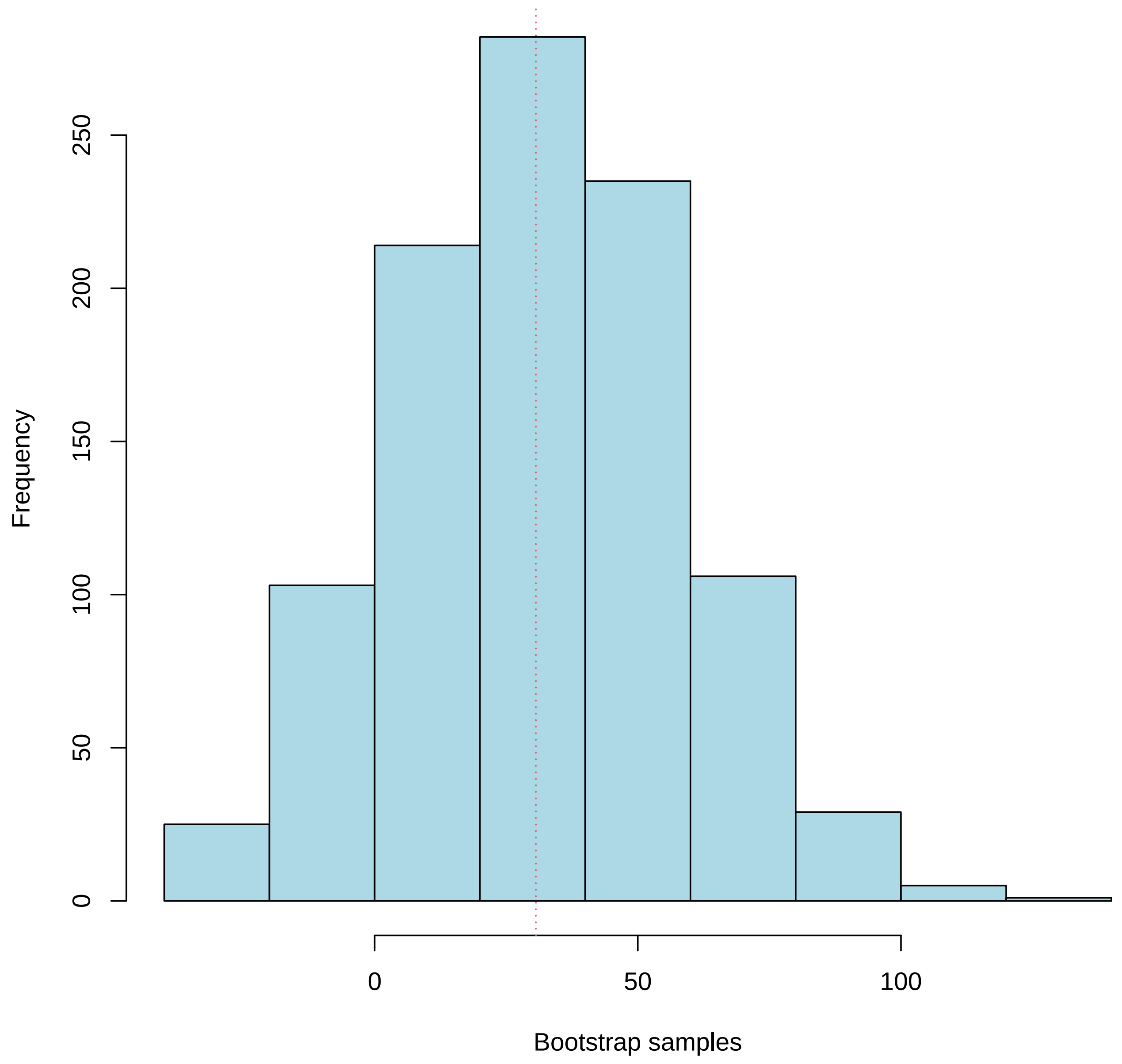
```r
B = 1000

sd(replicate(B, mean(sample(Trata,replace=TRUE)) -
mean(sample(Cont,replace=TRUE))))
```

```
[1] 27.0551
```

```r
library(simpleboot)

b = two.boot(Trata, Cont, mean, R=B)

sd(b$t)
```

```
[1] 26.53308
```

```r
hist(b, col="lightblue")
```



Programa original de la librería bootstrap

```
B = 1000
library(bootstrap)

mouse.boot.c = bootstrap(mouse.c, B, mean)
mouse.boot.t = bootstrap(mouse.t, B, mean)

mouse.boot.diff =
mouse.boot.t$thetastar - mouse.boot.c$thetastar
```

```r
Trata = c(94,197,16,38,99,141,23)
Cont = c(52,104,146,10,51,30,40,27,46)
B = 1000
n = length(Trata)
Losratones = c(Trata,Cont)

(t.obs = mean(Trata)-mean(Cont))
```

```
[1] 30.63492
```

```r
library(boot)
t.fun = function(data,i,n){
   bobo = data[i]
   mean(bobo[1:n])-mean(bobo[-c(1:n)])
}

(mouse.boot = boot(Losratones, t.fun, R=1000, n=n))
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = Losratones, statistic = t.fun, R = 1000, n = n)


Bootstrap Statistics :
    original     bias    std. error
t1* 30.63492 -31.52268    27.44754
```

Ejemplo sobre pastillas para dormir

Se toma el ejemplo de los datos correspondientes a 20 observaciones donde se mide el efecto de unas pastillas para dormir (el incremento de horas de sueño en relación a mediciones control).

Se usa primero el test de t-Student

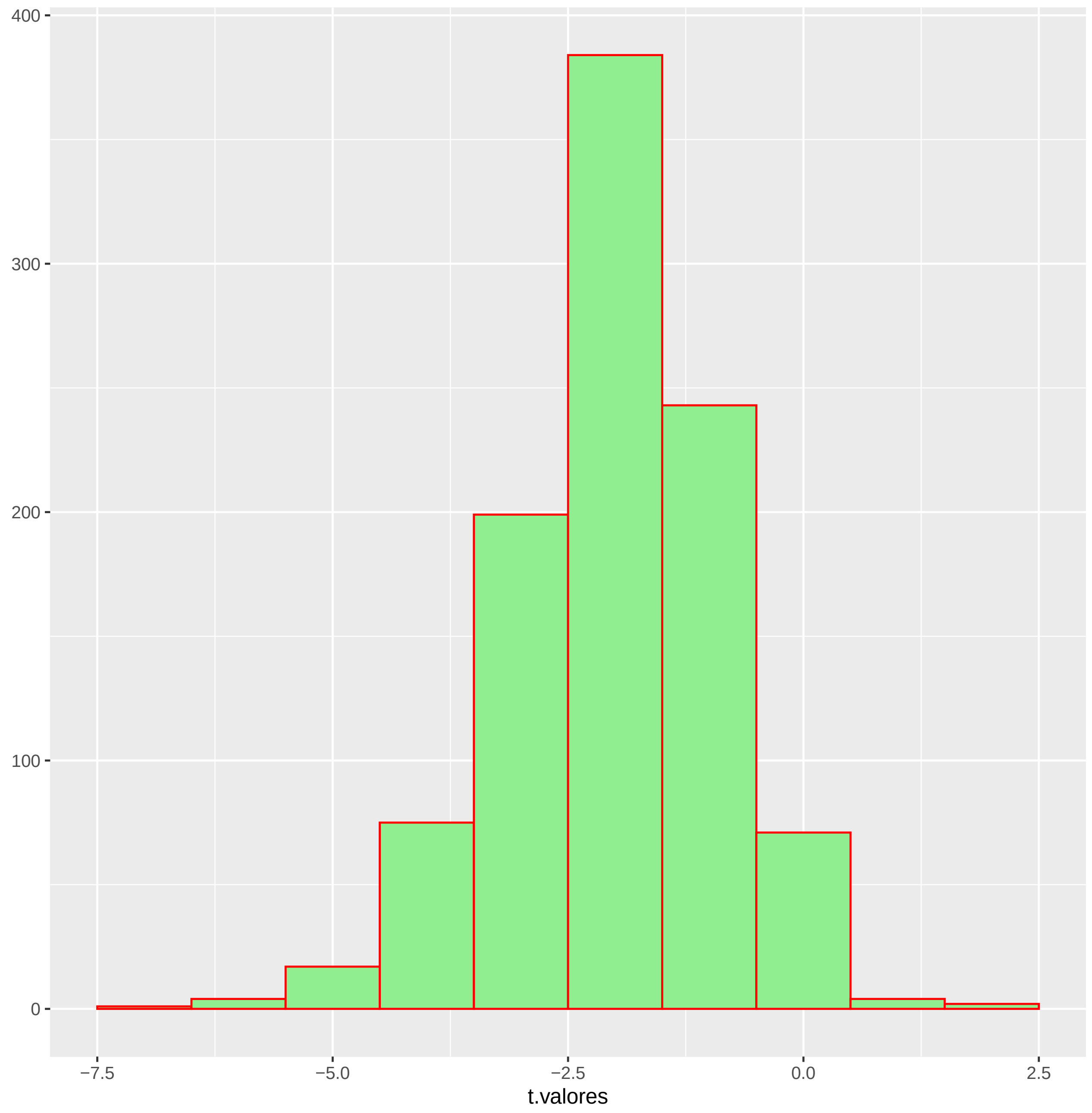
```r
data(sleep)

# test t Student
with(sleep, t.test(extra~group)$statistic)
```

```
        t
-1.860813
```

```r
scores = sleep$extra
R = 1000
t.valores = numeric(R)

scoresG1 = subset(scores, sleep$group==1)
scoresG2 = subset(scores, sleep$group==2)

for (i in 1:R) {
   grupo1 = sample(scoresG1, size=10, replace=T)
   grupo2 = sample(scoresG2, size=10, replace=T)
   t.valores[i] = t.test(grupo1, grupo2)$statistic
}

sd(t.valores)
```

```
[1] 1.114916
```

```r
ggplot2::qplot(t.valores, geom="histogram", binwidth=1,
fill=I("lightgreen"), col=I("red"))
```


400
300
200
100
0
−7.5
−5.0
−2.5
0.0
2.5
t.valores