응용통계학: R 사용

인하대학교 통계학과

이 경 재

12 March, 2019

summary functions for Exploratory Data Analysis (EDA)

```
x = (1:10)*10; x
```

```
##  [1]  10  20  30  40  50  60  70  80  90 100
```

```
length(x)
```

```
## [1] 10
```

```
mean(x); sum(x)
```

```
## [1] 55
```

```
## [1] 550
```

summary functions for Exploratory Data Analysis (EDA)

```
var(x); sd(x)
```

```
## [1] 916.6667
```

```
## [1] 30.2765
```

```
min(x); max(x)
```

```
## [1] 10
```

```
## [1] 100
```

summary functions for Exploratory Data Analysis (EDA)

```
quantile(x, probs = c(0.25, 0.75))
```

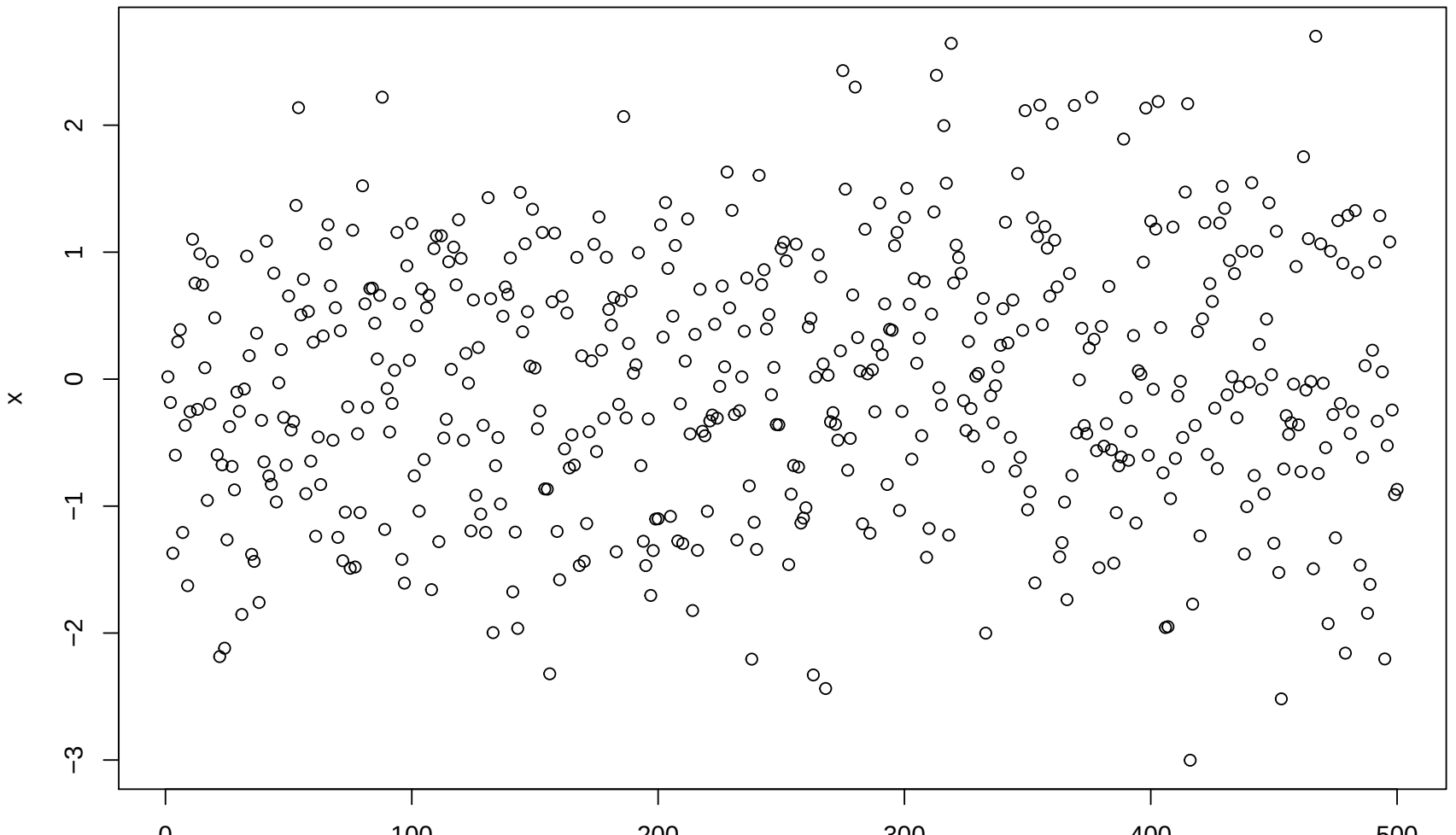
```
##  25%  75%
## 32.5 77.5
```

```
summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    10.0    32.5    55.0    55.0    77.5   100.0
```

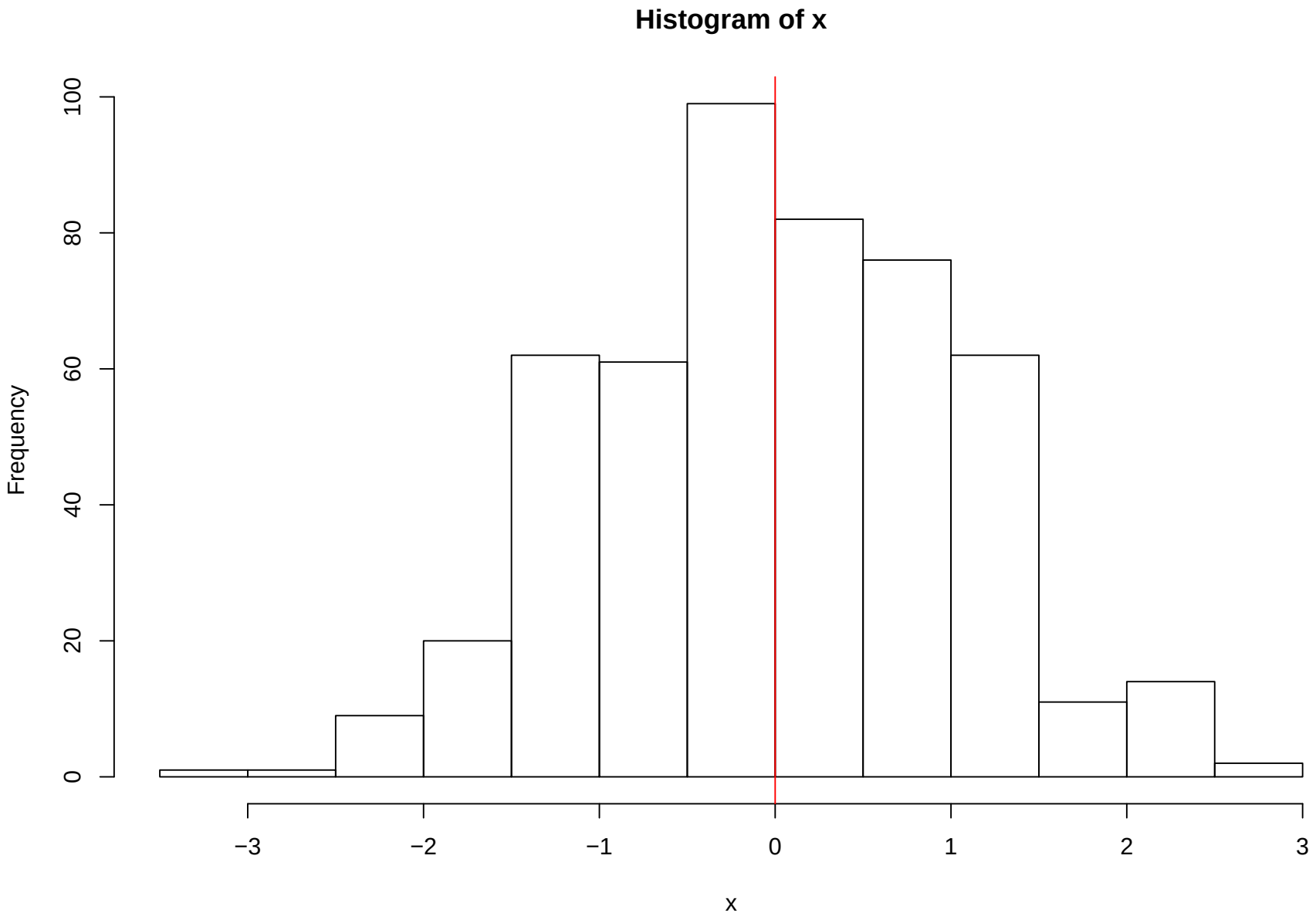
plot

```r
set.seed(10)
x = rnorm(n = 500, mean = 0, sd = 1)
# or x = rnorm(500)
plot(x)
```



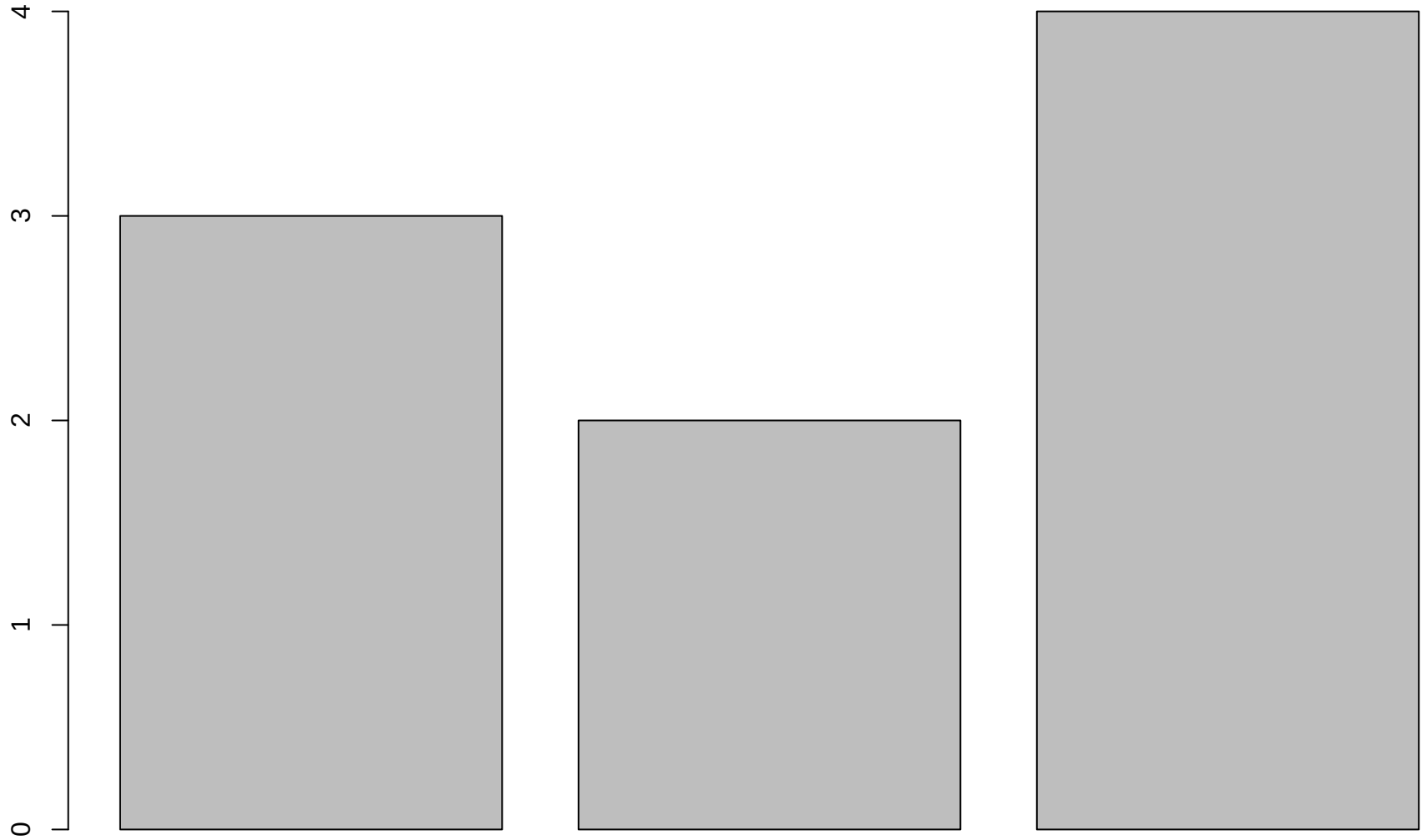
histogram

```
hist(x); abline(v = 0, col="red")
```



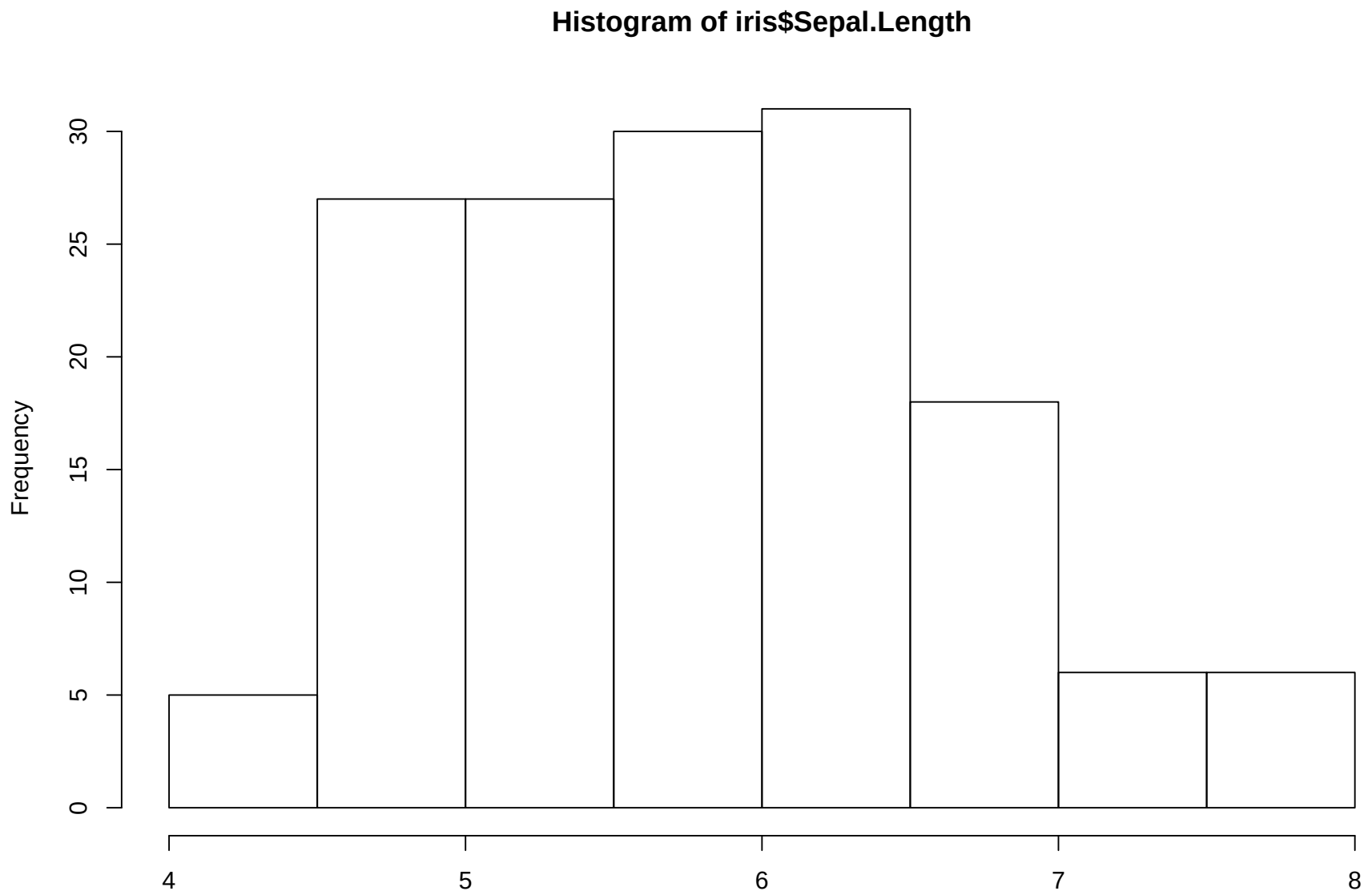
histogram

```r
# histogram of factors
a=factor (c("a", "b", "c", "b", "c", "a", "c", "c","a"))
plot(a)
```



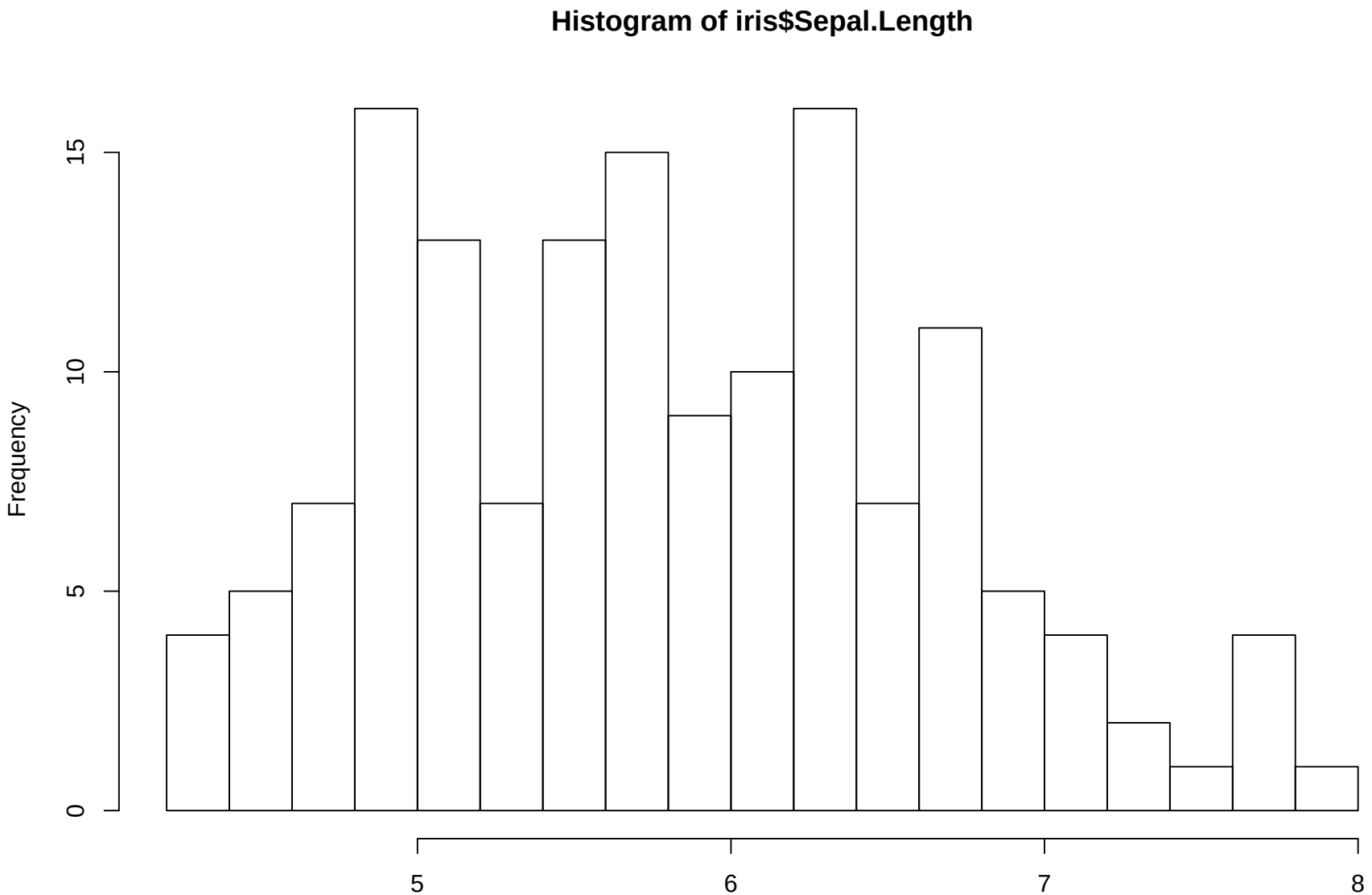
histogram

```
# histogram of the lenght in iris data set
hist(iris$Sepal.Length,xlab="cm")
```



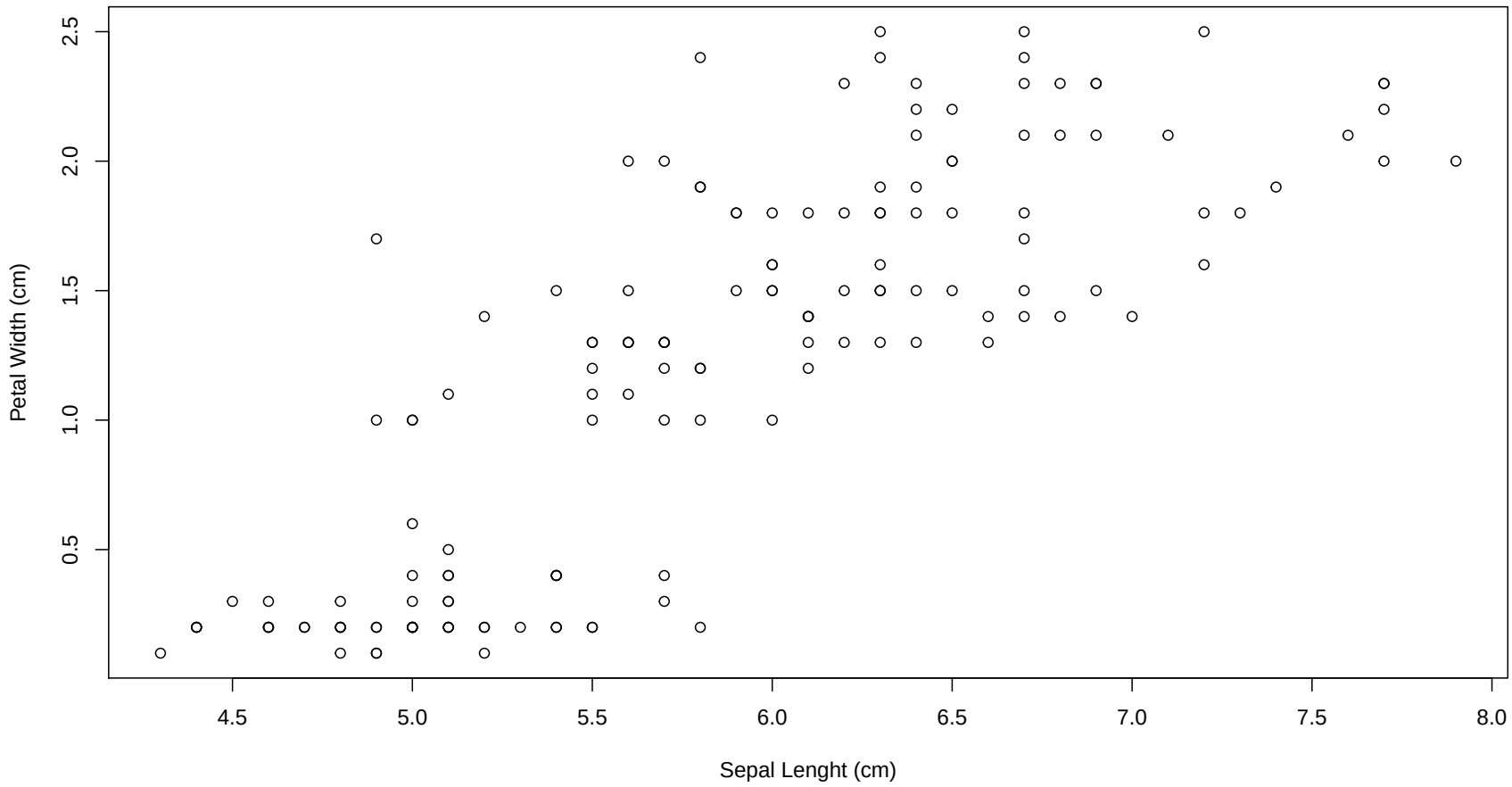
histogram

```
# we can increase the bins
hist(iris$Sepal.Length,breaks=20,xlab="cm")
```



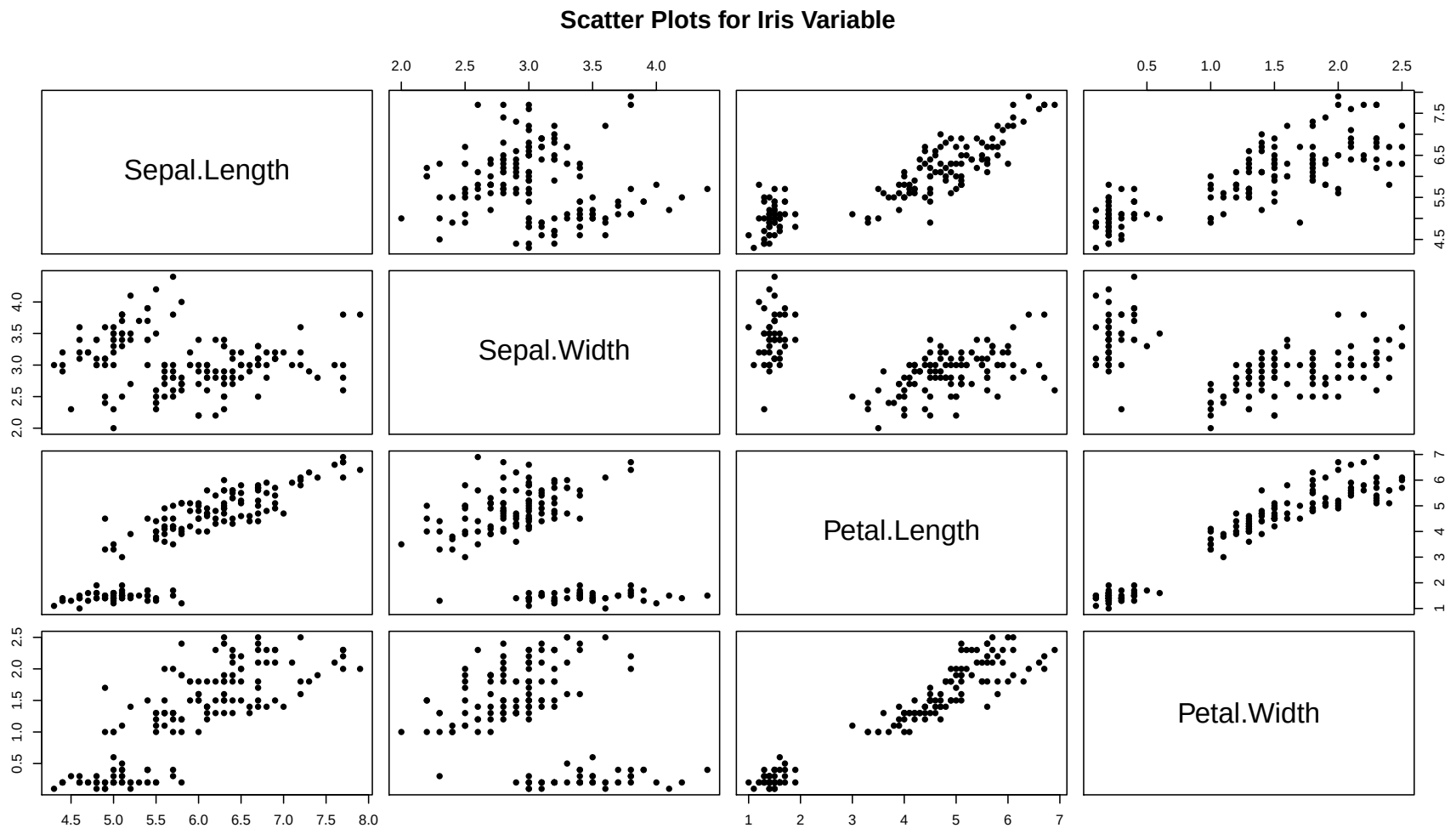
scatterplot

```r
# multiple variables: scatterplot
plot(iris$Sepal.L, iris$Petal.Width,
xlab=c("Sepal Lenght (cm)"),ylab=c("Petal Width (cm)"))
```



scatterplot

```
# multiple variables: scatterplot
plot(iris[,1:4], main="Scatter Plots for Iris Variable",
     pch=16)
```



User-defined function

```
myfun <- function(n, mu, Sigma=diag(length(mu))){
    p = length(mu) # dimensionality
    R = chol(Sigma) # Cholesky decomposition of Sigma
    # Sigma = t(R) %*% R

    z = matrix(rnorm(n*p), n, p)
    res = z%*%R + matrix(mu, n, p, byrow = T)
    return(res)
}
myfun(3, mu=1:2) # or myfun(3, mu=1:2, Sigma=diag(2))
```

```
##           [,1]     [,2]
## [1,] 1.8694750 1.840562
## [2,] 0.3199904 2.793499
## [3,] 1.1732145 3.694350
```

For loop

```
# here we print 1:5
for (i in 1:5){
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

For loop

```
# here we sum the first 10 numbers: 1+2+3+...+10
x = 0
for (j in 1:10){
  x = x + j
}
x
```

```
## [1] 55
```

For loop

```
vec = rep(NaN, 10); vec
```

```
##  [1] NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
```

```
s = 0
for (i in 1:10){
  s = s+i
  vec[i] = s;
}
# in fact
c(1+2,vec[2])
```

```
## [1] 3 3
```

```
c(1+2+3+4+5+6,vec[6])
```

```
## [1] 21 21
```

Simulation studies

```r
# this command is to allow replication:
# every time you start R you have the same random numbers
set.seed(1)

# removing all variables
rm(list=ls())

# in this study, we generate a sample from N(mu, sigma^2)
# N is the sample size
# mu is the population mean
# sigma the standard deviation, so sigma^2 is the variance

n=10
mu=4
sigma=10
```

```r
# this command generated a sample of size N
y=rnorm(n, mean=mu, sd=sigma)
y
```

```
##  [1] -2.2645381  5.8364332 -4.3562861 19.9528080  7.2950777
##  [7]  8.8742905 11.3832471  9.7578135  0.9461161
```

```r
y=rnorm(n, mean=mu, sd=sigma)
y
```

```
##  [1] 19.117812  7.898432 -2.212406 -18.146999 15.249309
##  [7]  3.838097 13.438362 12.212212  9.939013
```

```r
# the importance of set.seed(1),
# every time I can get the same results
set.seed(1)
y=rnorm(n, mean=mu, sd=sigma)
y
```

```
##  [1] -2.2645381  5.8364332 -4.3562861 19.9528080  7.2950777
##  [7]  8.8742905 11.3832471  9.7578135  0.9461161
```

```r
# we can now compute the mean and standard deviation
ybar=mean(y)
s2=var(y)

c(ybar,s2)
```

```
## [1]  5.322028 60.931444
```

```r
# compared to the true ones
c(mu,sigma)
```

```
## [1]  4 10
```

```r
# is this close enough?
# we can do this experiment as many times as we want
nrep=10000 # nrep: the number of replicates
s2=rep(NaN,nrep)
for (i in 1:nrep){
  y=rnorm(n, mean=mu, sd=sigma)
  ybar[i]=mean(y)
  s2[i]=var(y)
}
cbind(c(mean(ybar),mean(s2)),c(mu,sigma^2))
```

```
##              [,1] [,2]
## [1,]    3.977429    4
## [2,] 100.841920  100
```

```r
# we can compute the bias
cbind(c(mean(ybar)-mu, mean(s2)-sigma^2))
```

```
##                [,1]
## [1,] -0.02257143
## [2,]  0.84192029
```

```r
# the sample standard deviation is biased!
```

```r
# redo the same experiment with a much larger sample size
n=1000
for (i in 1:nrep){
  y=rnorm(n, mean=mu, sd=sigma)
  ybar[i]=mean(y)
  s2[i]=var(y)
}
cbind(c(mean(ybar)-mu,mean(s2)-sigma^2))
```

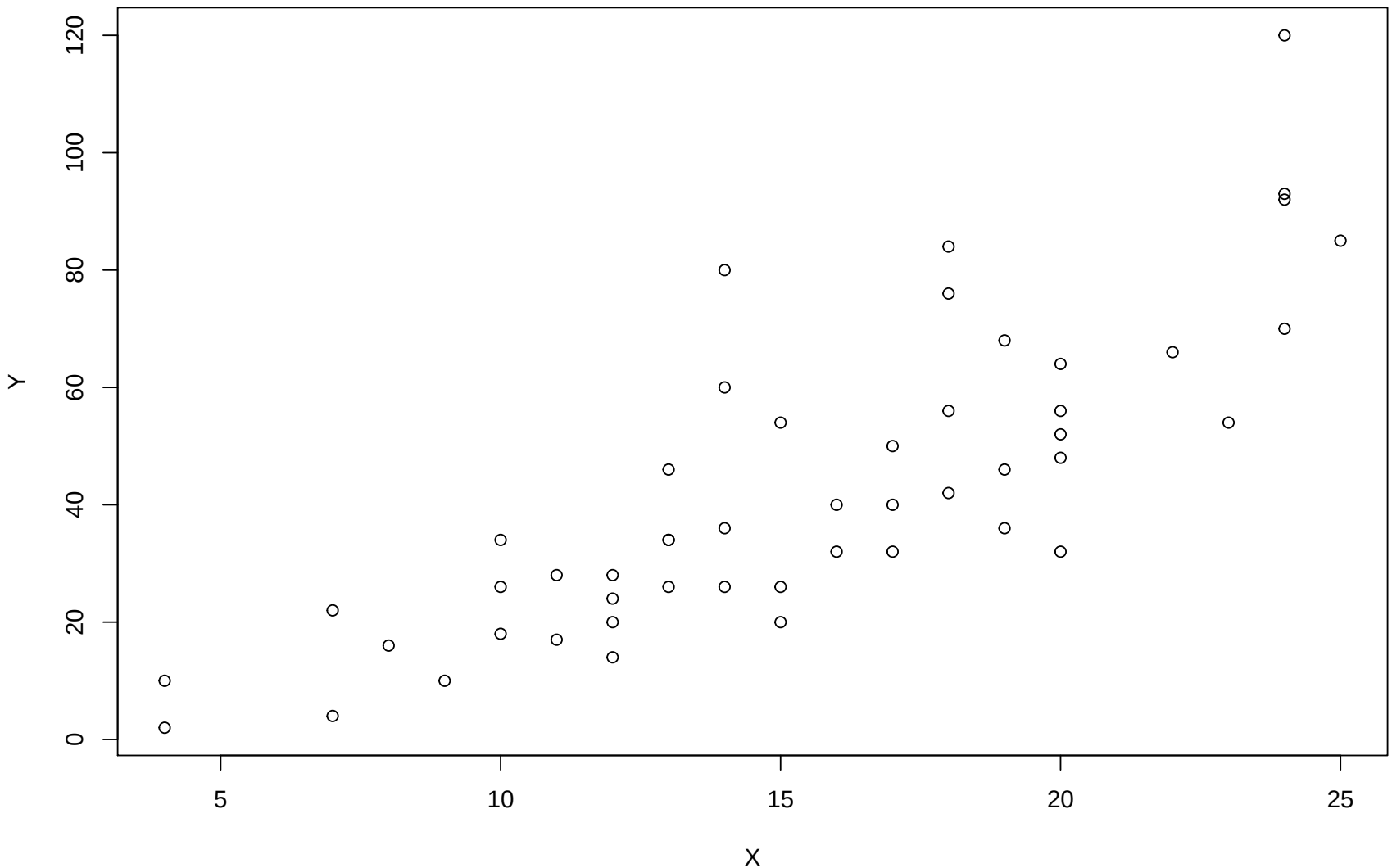
```
##                [,1]
## [1,] 0.004145326
## [2,] 0.045795265
```

Linear regression

```
data(cars)
head(cars)
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10
```

```
X = cars$speed
Y = cars$dist
plot(X,Y)
```



```r
n = length(Y)

mean_x = mean(X);
mean_y = mean(Y)
# this is divided by n-1
var_x = var(X);
var_y = var(Y)
cov_xy = cov(X,Y)

SS_xx <- (n-1)*var_x
SS_xy <- (n-1)*cov_xy
SS_yy <- (n-1)*var_y

b1 <- SS_xy/SS_xx
b0 <- mean_y - b1*mean_x
```

```r
yhat <- b0 + b1*X
e <- Y-yhat

SSE <- sum(e^2)
MSE <- SSE/(n-2)
s <- sqrt(MSE)

print(cbind(mean_x,mean_y))
```

```
##       mean_x mean_y
## [1,]    15.4  42.98
```

```r
print(cbind(SS_xx,SS_xy,SS_yy))
```

```
##       SS_xx  SS_xy    SS_yy
## [1,]  1370 5387.4 32538.98
```

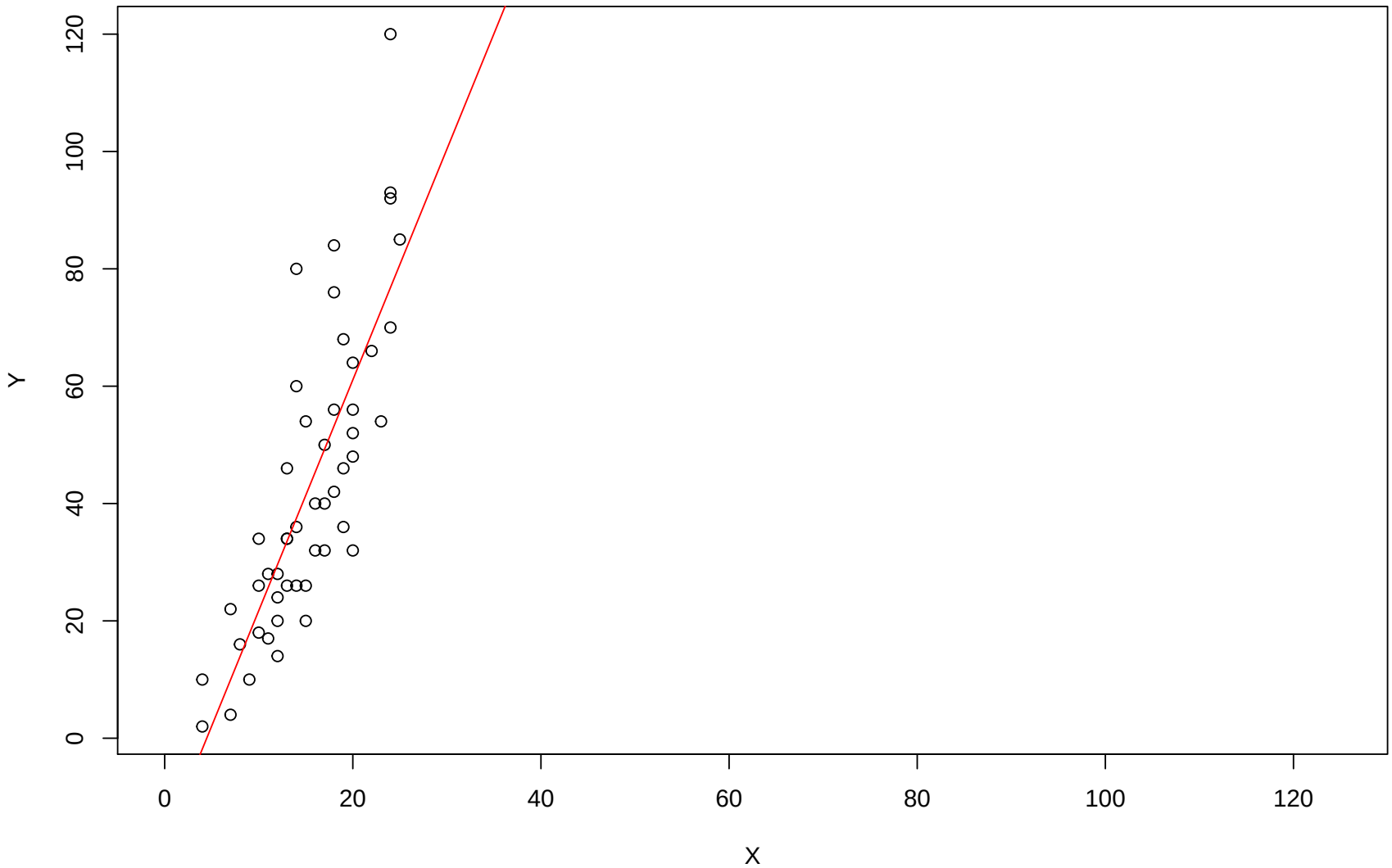
```r
print(cbind(b0,b1))
```

```
##              b0       b1
## [1,] -17.57909 3.932409
```

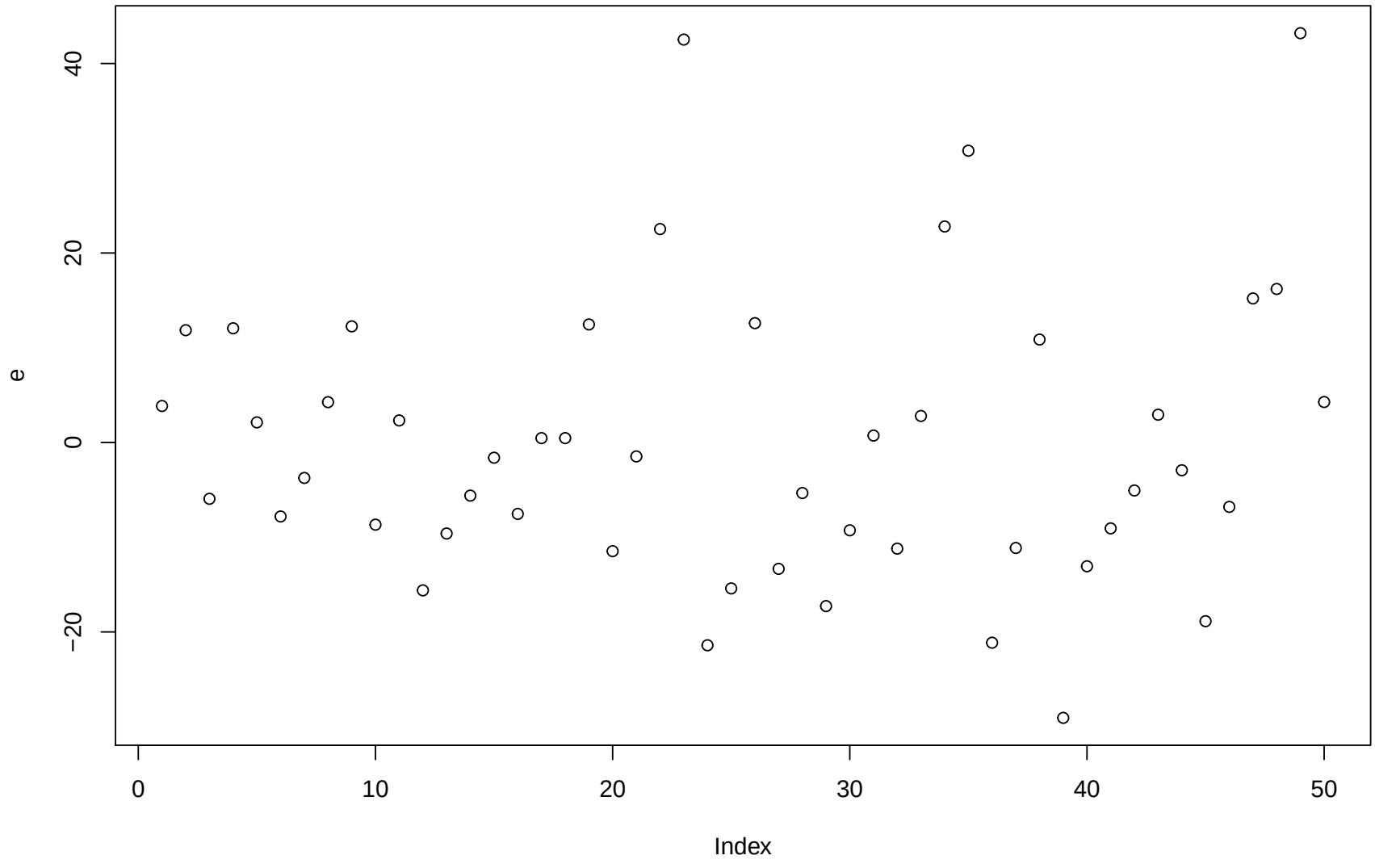
```r
print(cbind(Y,yhat,e))
```

```
##          Y      yhat           e
##  [1,]    2 -1.849460   3.849460
##  [2,]   10 -1.849460  11.849460
##  [3,]    4  9.947766  -5.947766
##  [4,]   22  9.947766  12.052234
##  [5,]   16 13.880175   2.119825
##  [6,]   10 17.812584  -7.812584
```

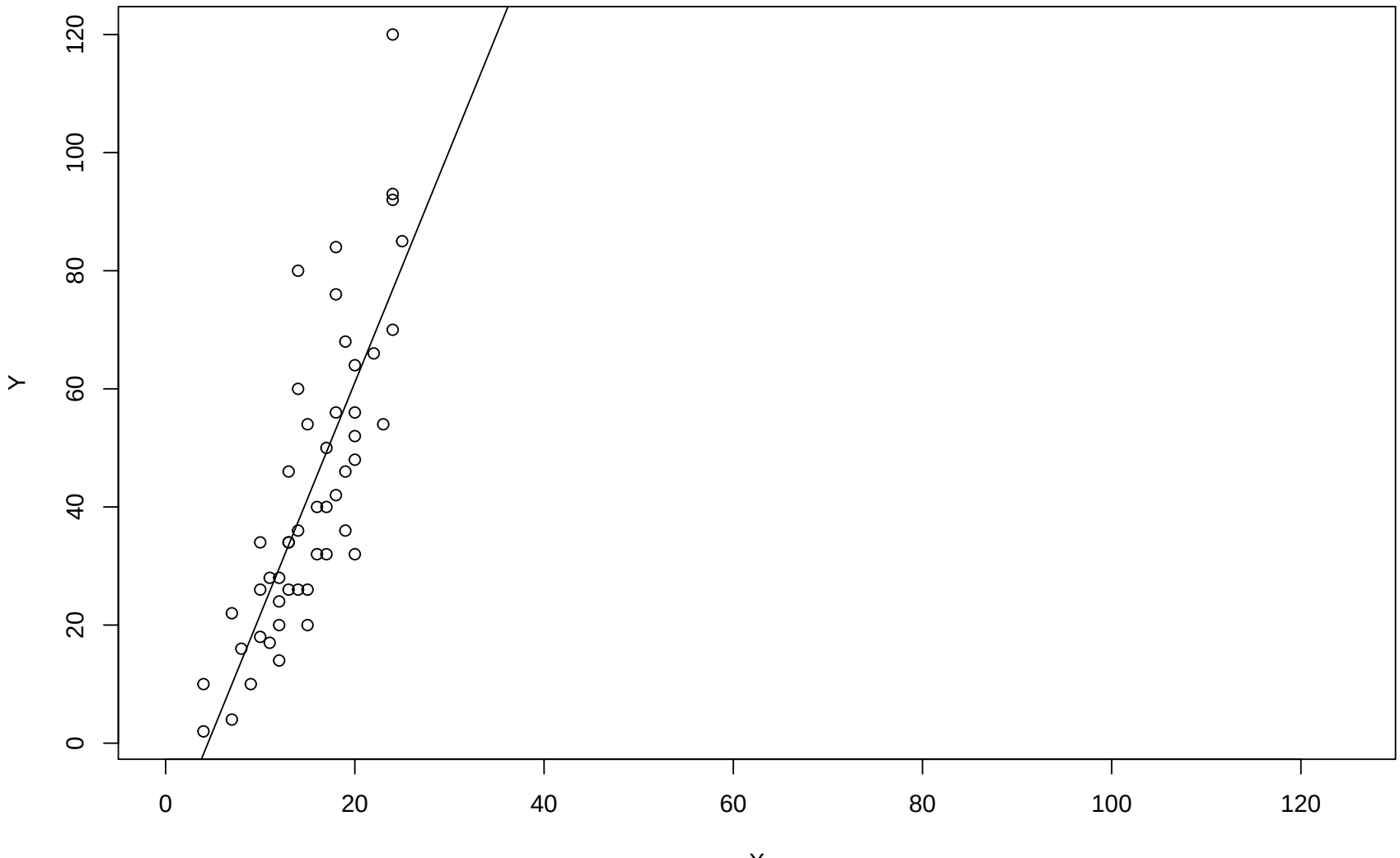
```r
# plot the estimated regression function
plot(X,Y,xlim=c(0,125))
abline(a=b0,b=b1, col='red')
```



```r
# how to the residuals look like?
plot(e) # e <- Y-yhat
```



```r
# what we have done can actually be done in one command!
mod=lm(Y~X)

plot(X,Y,xlim=c(0,125))
abline(mod)
```



```r
# b0 and b1 should be equal to the coefficients from lm
cbind(c(b0,b1),mod$coefficients)
```

```
##                    [,1]       [,2]
## (Intercept) -17.579095 -17.579095
## X             3.932409   3.932409
```

```r
# ... the estimated sigma ...
cbind(s,summary(mod)$sigma)
```

```
##                s
## [1,] 15.37959 15.37959
```

```r
# ... and the fitted values
cbind(yhat,mod$fitted)
```

```
##          yhat
## 1  -1.849460 -1.849460
## 2  -1.849460 -1.849460
## 3   9.947766  9.947766
## 4   9.947766  9.947766
## 5  13.880175 13.880175
## 6  17.812584 17.812584
## 7  21.744993 21.744993
## 8  21.744993 21.744993
## 9  21.744993 21.744993
## 10 25.677401 25.677401
```

Unbiasedness

```
# We have seen that the lm commands are correct
# I will show you now how to practically see than an
# estimator is unbiased
# one number (estimate) doesn't tell you anything about
# the estimator being unbiased
Nsim=100
N=101
beta0=1
beta1=3
X=seq(0,1,1/(N-1))
sigma=2

Y=matrix(rep(N*Nsim),nrow=N,ncol=Nsim)
# Y is a matrix with a simulation for each column
# we will run lm for every column and
# store the estimated values
```

```r
coeff_matrix=matrix(rep(2*Nsim),nrow=2,ncol=Nsim)
sigma_matrix=matrix(rep(Nsim),nrow=1,ncol=Nsim)
for (i in 1:Nsim){
  epsilon=rnorm(N,mean=0,sd=sigma)
  Y[,i]=beta0+beta1*X+epsilon
  mod=lm(Y[,i]~X)
  coeff_matrix[,i]=mod$coefficients
  sigma_matrix[i]=summary(mod)$sigma
}
```

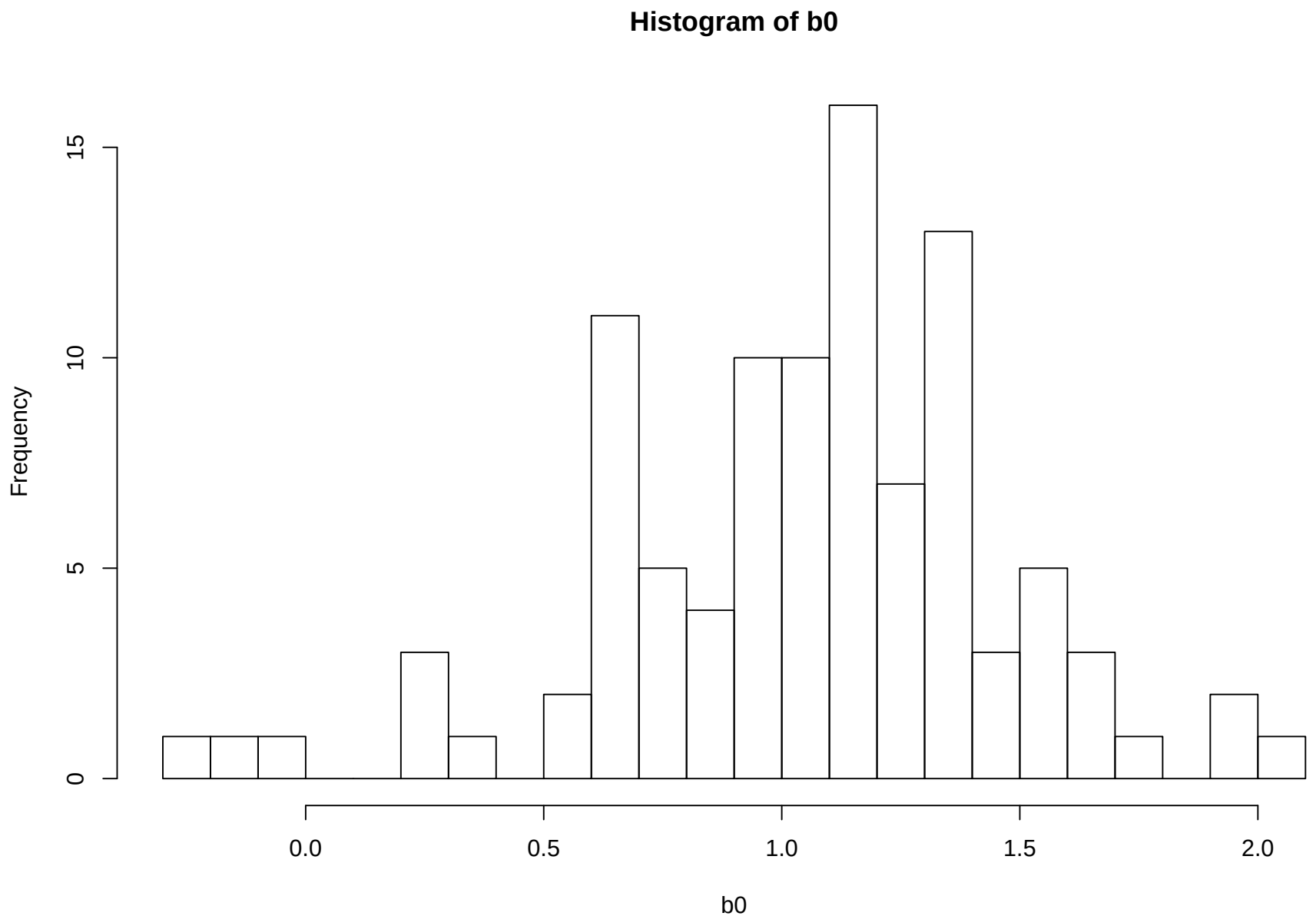
```r
# we have the estimated coefficients for each simulation
# on average...
apply(coeff_matrix,1,mean) # = rowMeans(coeff_matrix)
```

```
## [1] 1.060704 2.924469
```

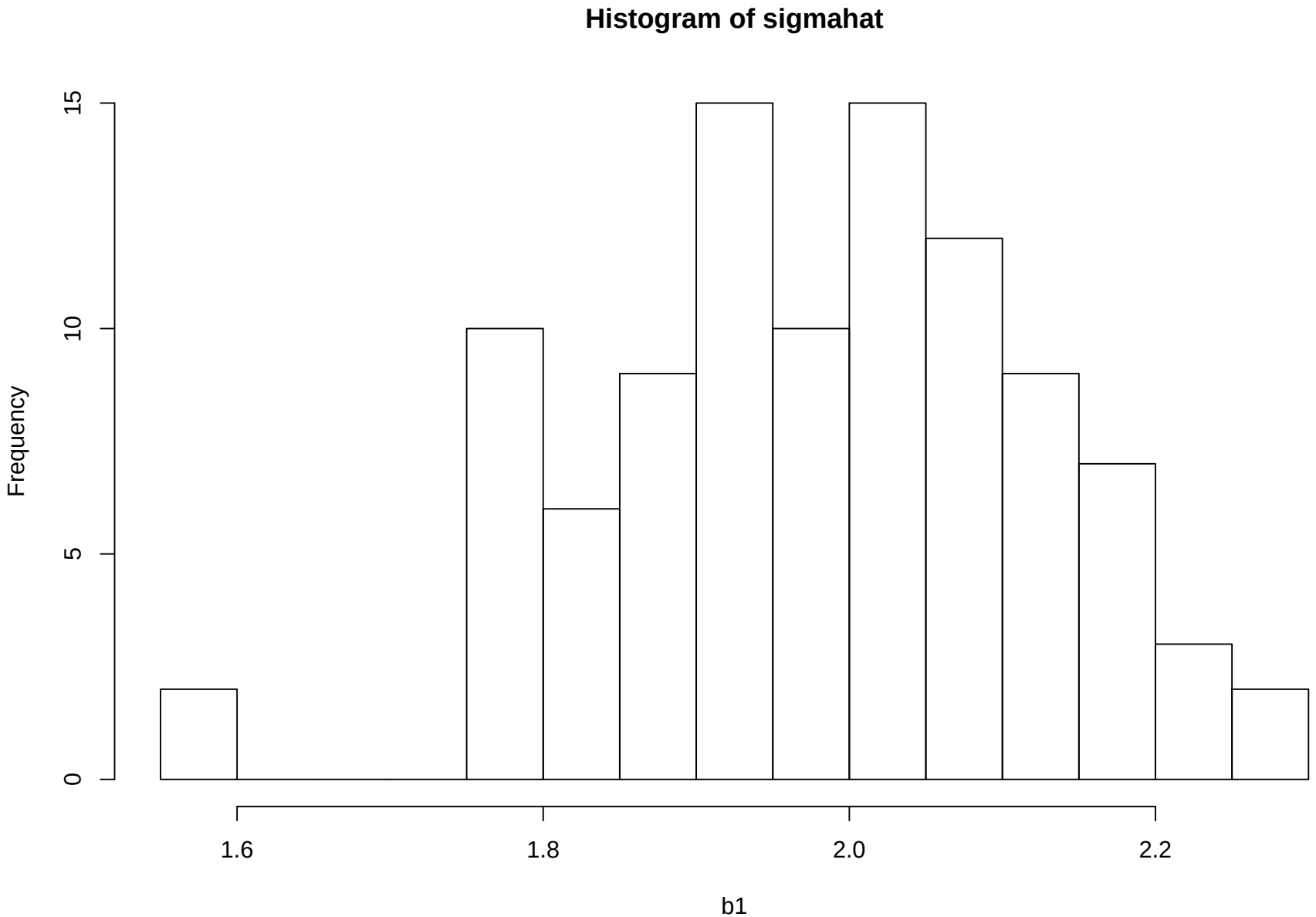
```r
# and the bias is...
bias_b=apply(coeff_matrix,1,mean)-c(beta0,beta1)
bias_s=mean(sigma_matrix)-sigma

c(bias_b,bias_s)
```

```
## [1]  0.06070448 -0.07553084 -0.01772397
```

```r
# histograms
hist(coeff_matrix[1,],main=c("Histogram of b0"),breaks=20,
     xlab="b0")
```



```
hist(sigma_matrix,main=c("Histogram of sigmahat"),breaks=20
     xlab="b1")
```



```r
# what happens as you have more simulations?
Nsim=10000

Y=matrix(rep(N*Nsim),nrow=N,ncol=Nsim)
coeff_matrix=matrix(rep(2*Nsim),nrow=2,ncol=Nsim)
sigma_matrix=matrix(rep(Nsim),nrow=1,ncol=Nsim)
```

```
for (i in 1:Nsim){
  epsilon=rnorm(N,mean=0,sd=sigma)
  Y[,i]=beta0+beta1*X+epsilon
  mod=lm(Y[,i]~X)
  coeff_matrix[,i]=mod$coefficients
  sigma_matrix[i]=summary(mod)$sigma
}
```

```r
# and the bias is...
bias_b_large=apply(coeff_matrix,1,mean)-c(beta0,beta1)
bias_s_large=mean(sigma_matrix)-sigma

c(bias_b_large,bias_s_large)
```

```
## [1] -0.0009302722  0.0021311711 -0.0060484544
```

```r
# comparison with the previous bias
cbind(bias_b,bias_b_large)
```

```
##             bias_b  bias_b_large
## [1,]  0.06070448 -0.0009302722
## [2,] -0.07553084  0.0021311711
```

```r
cbind(bias_s,bias_s_large)
```

```
##             bias_s bias_s_large
## [1,] -0.01772397 -0.006048454
```