7.1 Jackknife estimate of the bias and standard error of the correlation statistic.

```
require(bootstrap)
require(boot)
data(law)
n <- dim(law)[1]
rho <- cor(law$LSAT, law$GPA)
rho.jack <- numeric(n)
for (i in 1:n) {
    rho.jack[i] <- cor(law$LSAT[-i], law$GPA[-i])
}
bias <- (n - 1) * (mean(rho.jack) - rho)
rho.mean <- mean(rho.jack)
se <- sqrt((n - 1) * mean((rho.jack - rho.mean)^2))
```

The bias is calculated as -0.0065 and the standard error is calculated as 0.1425.

7.4 The pdf of the exponential distribution is $f(x) = \lambda e^{-\lambda x} I_{(0,\text{inf})}(X)$. The maximum likelihood estimator $\hat{\lambda}$ is:

$$L(\lambda|x) = \prod_{i=1}^{n} \lambda e^{-\lambda x_i} I_{(0,\text{inf})}(x_i) = \lambda^n e^{-\lambda \sum x_i} I_{(0,\text{inf})}(x_i)$$

$$log(L(\lambda|x)) = l = nlog(\lambda) - \lambda \sum x_i$$

$$l' = \frac{n}{\lambda} - \sum x_i$$

$$\frac{n}{\hat{\lambda}} = \sum x_i$$

$$\hat{\lambda} = \frac{n}{\sum x_i} = \frac{1}{\bar{x}}$$

```
set.seed(36)
data(aircondit)
air <- aircondit  #shorten the name for easier coding
lambda <- 1/mean(air$hours)
r <- 10000
set.seed(36)
lambda.boot <- boot(data = air, statistic = function(x, i) {
    1/mean(x[i, ])
}, R = r)
lambda.boot

##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = air, statistic = function(x, i) {
```

```
##     1/mean(x[i, ])
## }, R = r)
##
##
## Bootstrap Statistics :
##     original   bias    std. error
## t1* 0.009252 0.001406    0.004397
```

7.5 Bootstrap confidence intervals via standard normal, basic, percentile, and BCa methods.
From the invariance property of the MLE, $\frac{1}{\hat{\lambda}} = \bar{x}$ is a function of a MLE estimator and is therefore also a MLE estimator.
The standard normal CI is calculated without bootstraping:

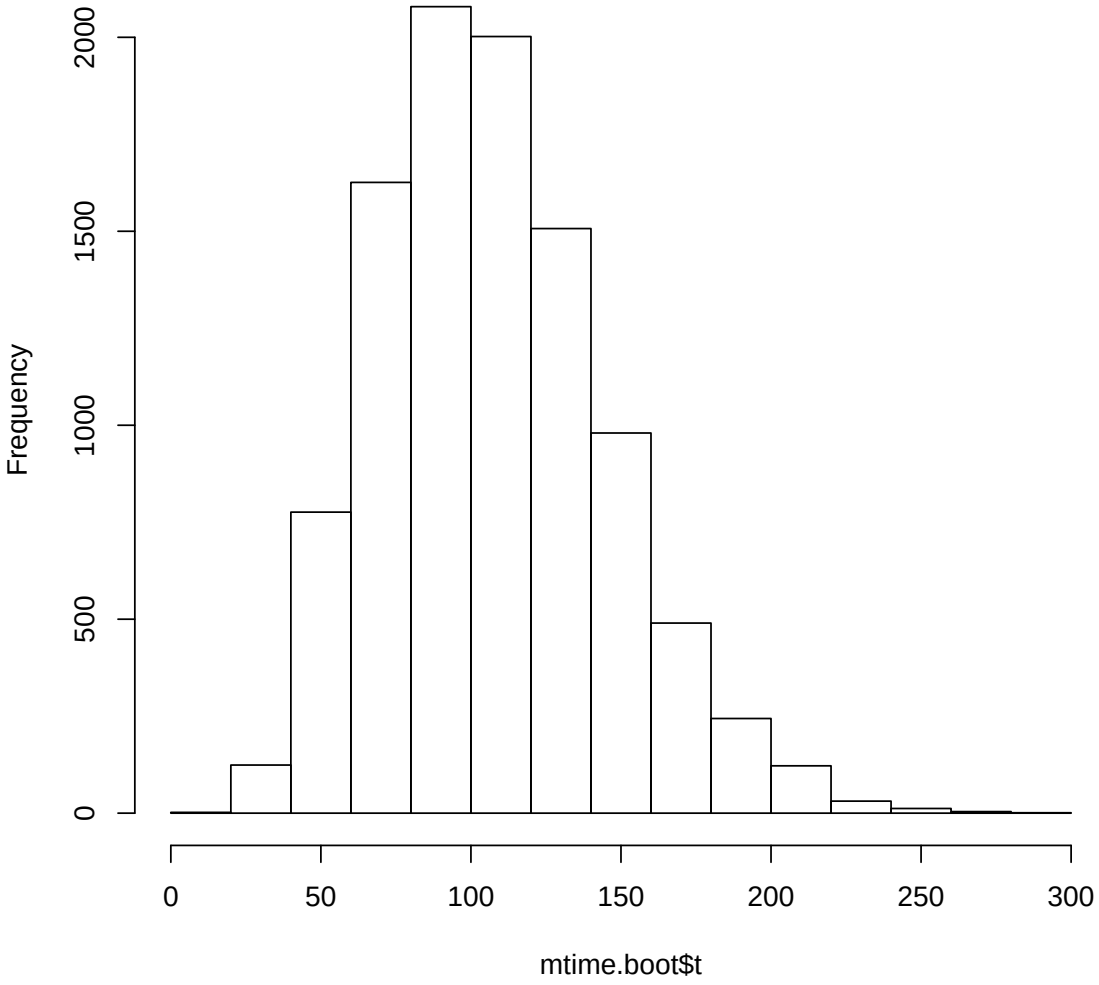
```
mtime <- mean(air$hours)
se.norm <- sd(air$hours)/sqrt(dim(air)[1])
mtime.CI <- c(mtime - (1.96 * se.norm), mtime, mtime + (1.96 * se.norm))
mtime.CI

## [1]  31.0 108.1 185.2
```

The bootstrap confidence intervals are:

```
set.seed(36)
mtime.boot <- boot(data = air, statistic = function(x, i) {
    mean(x[i, ])
}, R = r)
einf.jack <- empinf(mtime.boot, type = "jack")
boot.ci(mtime.boot, type = c("basic", "perc", "bca"), L = einf.jack)

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = mtime.boot, type = c("basic", "perc", "bca"),
##     L = einf.jack)
##
## Intervals :
## Level      Basic              Percentile            BCa
## 95%   ( 24.3, 169.8 )   ( 46.3, 191.8 )   ( 56.9, 226.2 )
## Calculations and Intervals on Original Scale
```

All of these CI's differ substantially. This is likely because the distribution of the mean interval time is not normal. To check this we can examine the distribution:

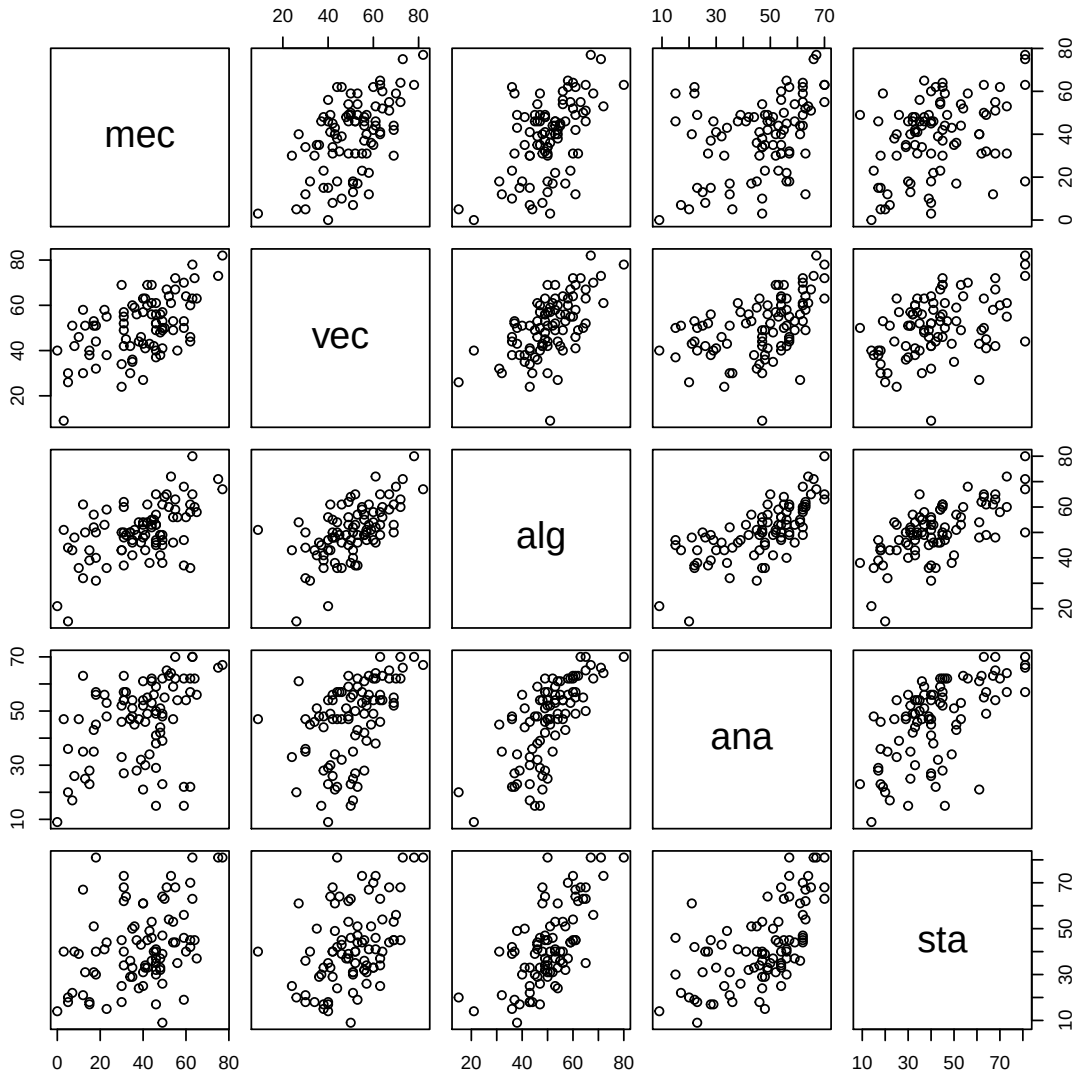
```
hist(mtime.boot$t, main = NULL)
```



Indeed the distribution has a clear right skew. The standard normal confidence interval assumes a symmetric normal distribution which is clearly violated in this case. The basic and percentile methods also assume a symmetric distribution and therefore the BCa method, which accounts for skew, is proabably the best in this case.

7.6 Test data correlations.

```
data(scor)
pairs(scor)
```



The sample correlation matrix contains the same information:

```
cor(scor)
```

```
##        mec    vec    alg    ana    sta
## mec 1.0000 0.5534 0.5468 0.4094 0.3891
## vec 0.5534 1.0000 0.6096 0.4851 0.4364
## alg 0.5468 0.6096 1.0000 0.7108 0.6647
## ana 0.4094 0.4851 0.7108 1.0000 0.6072
## sta 0.3891 0.4364 0.6647 0.6072 1.0000
```

All of the variables have reasonably strong correlations. The bootstrap estimates of the standard errors

```
set.seed(36)
corSE.boot <- function(var1, var2, rep = 10000) {
    #function to compute the se of a correlation
    dat <- cbind(var1, var2)
    r <- function(dat, i) {
        cor(dat[i, 1], dat[i, 2])
    }
    dat <- cbind(var1, var2)
    obj <- boot(data = dat, statistic = r, R = rep)
    y <- obj$t
    return(apply(y, 2, sd))
}
corSE.boot(scor$mec, scor$vec)

## [1] 0.07602

corSE.boot(scor$alg, scor$ana)

## [1] 0.04899

corSE.boot(scor$alg, scor$sta)

## [1] 0.06015

corSE.boot(scor$ana, scor$sta)

## [1] 0.06752
```

7.7 Bootstrap estimate of the bias and standard error of the estimate of the proportion of variance included in the first PCA component.

```
set.seed(36)
l <- eigen(cov(scor))$values
theta <- l[1]/sum(l)
thetafun <- function(x, i) {
    l <- eigen(cov(x[i, ]))$values
    theta <- l[1]/sum(l)
    return(theta)
}
theta.boot <- boot(data = scor, statistic = thetafun, R = r)
theta.boot

##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = scor, statistic = thetafun, R = r)
```

```
##
##
## Bootstrap Statistics :
##     original   bias    std. error
## t1*   0.6191 0.000862     0.04808
```

7.8 Jackknife estimate of the bias and standard error of the estimate of the proportion of variance included in the first PCA component.

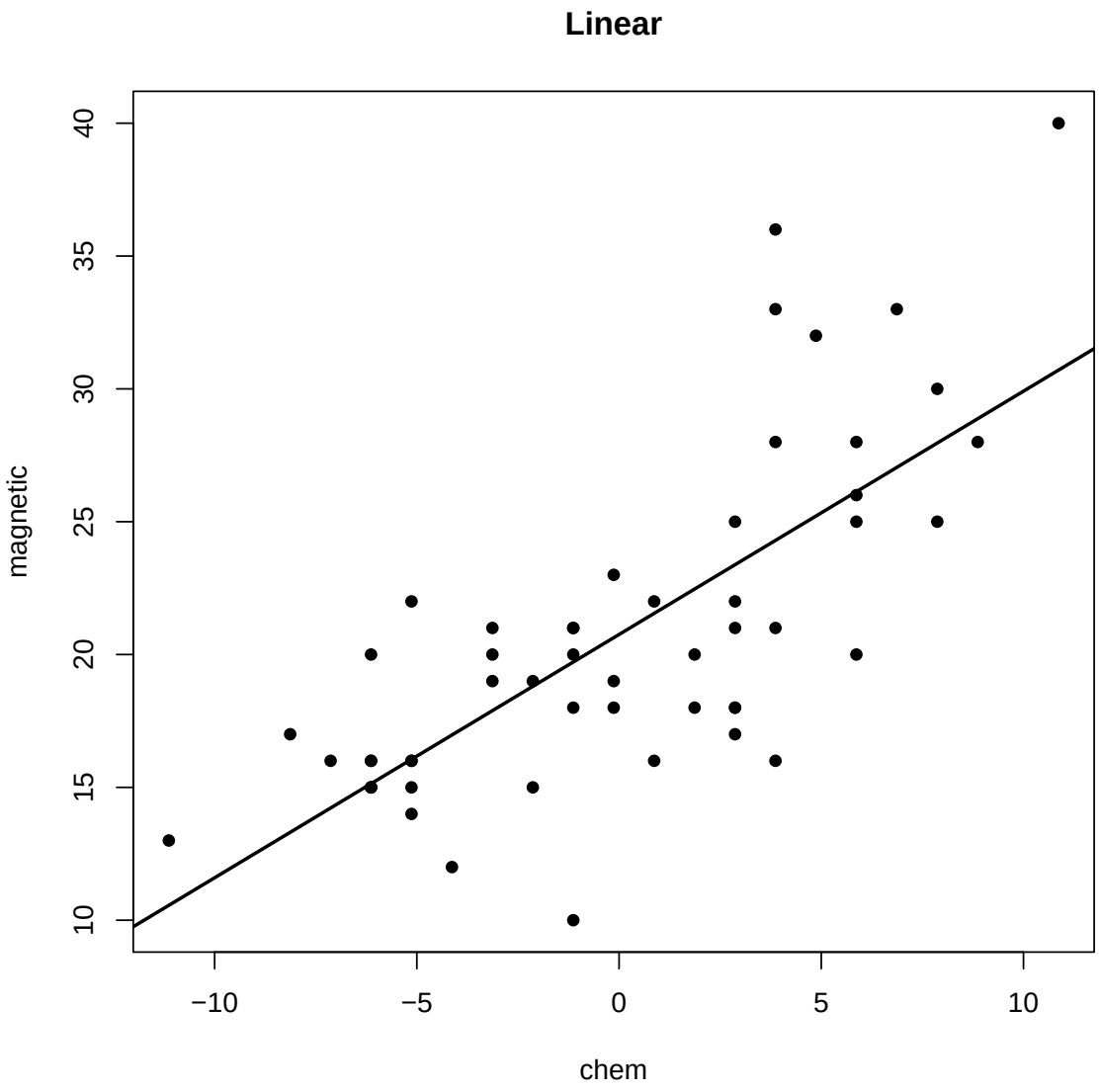
```
theta.jack <- numeric(dim(scor)[1])
for (i in 1:length(theta.jack)) {
    theta.jack[i] <- eigen(cov(scor[-i, ]))$values[1]/sum(eigen(cov(scor[-i,
        ]))$values)
}
bias <- (length(theta.jack) - 1) * (mean(theta.jack) - theta)
theta.mean <- mean(theta.jack)
se <- sqrt((length(theta.jack) - 1) * mean((theta.jack - theta.mean)^2))
bias

## [1] 0.001069

se

## [1] 0.04955
```

7.9 95% percentile and BCa confidence intervals for $\hat{\theta}$.

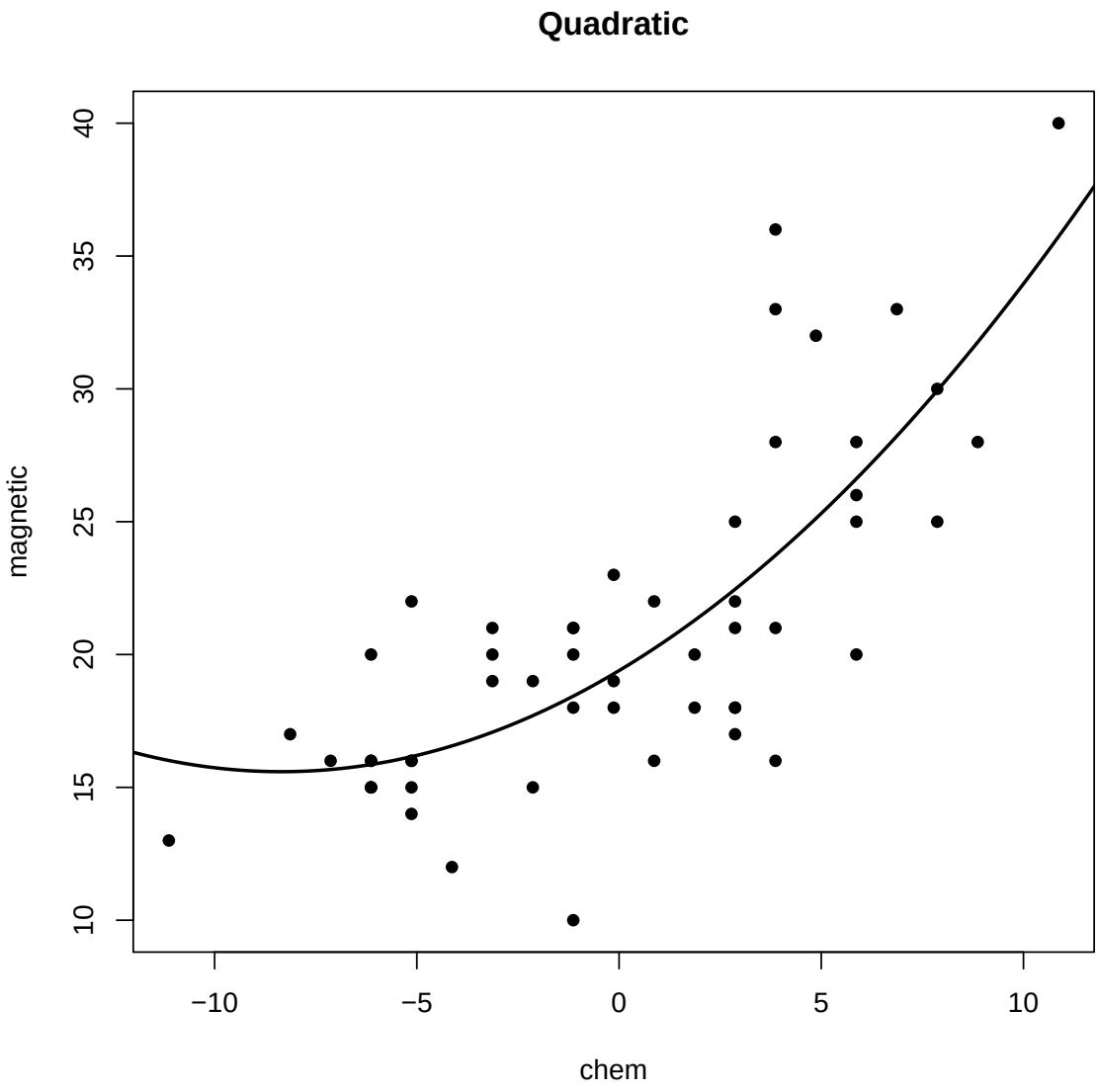
```
set.seed(36)
boot.ci(theta.boot, type = c("perc", "bca"), L = empinf(theta.boot, type = "jack"))

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = theta.boot, type = c("perc", "bca"), L = empinf(theta.boot,
##     type = "jack"))
##
## Intervals :
## Level     Percentile            BCa
## 95%   ( 0.5200,  0.7090 )   ( 0.5179,  0.7074 )
## Calculations and Intervals on Original Scale
```

7.10 Model selection via cross-validation.
Using code from the book's website

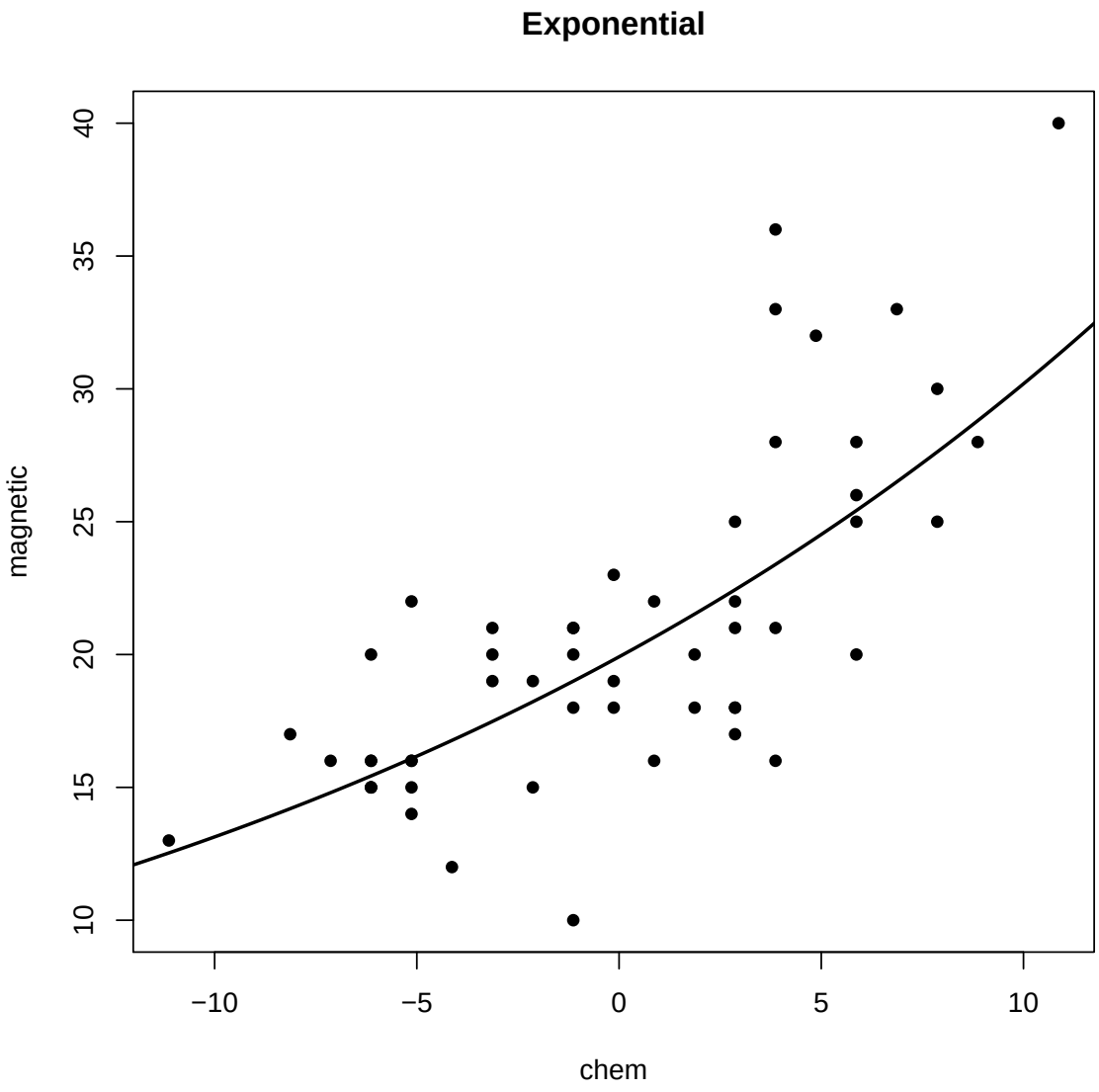
```r
library(DAAG)

## Loading required package:  lattice
##
## Attaching package:  'lattice'
##
## The following object(s) are masked from 'package:boot':
##
##    melanoma

attach(ironslag)
a <- seq(-12, 12, 0.1)  #sequence for plotting fits
mchem <- mean(chemical)
chem <- chemical - mchem
ironslag$chem <- chem  # center the chemical variable

L1 <- lm(magnetic ~ chem)
plot(chem, magnetic, main = "Linear", pch = 16)
yhat1 <- L1$coef[1] + L1$coef[2] * a
lines(a, yhat1, lwd = 2)
```



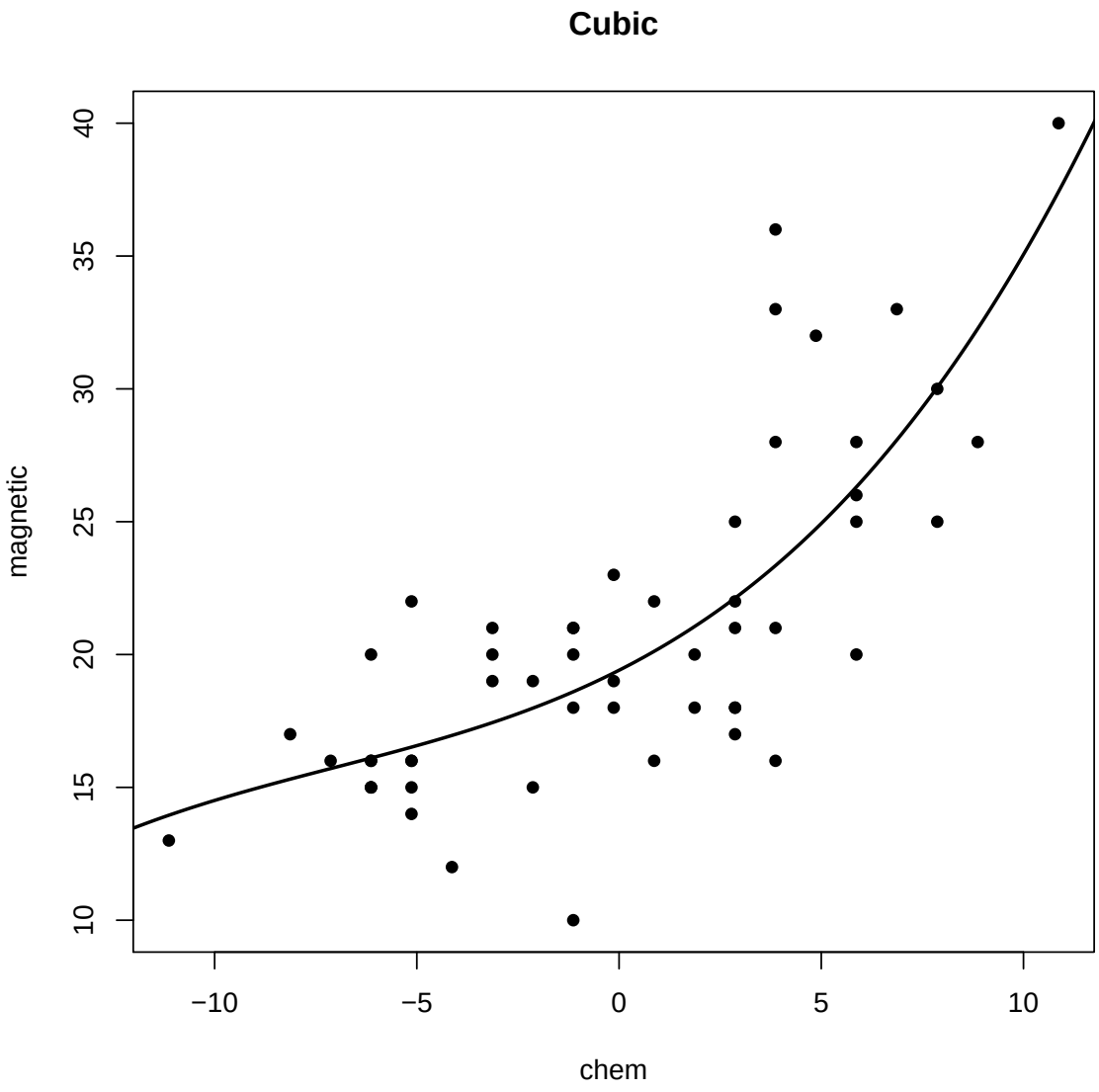

```
L2 <- lm(magnetic ~ chem + I(chem^2))
plot(chem, magnetic, main = "Quadratic", pch = 16)
yhat2 <- L2$coef[1] + L2$coef[2] * a + L2$coef[3] * a^2
lines(a, yhat2, lwd = 2)
```




```
L3 <- lm(log(magnetic) ~ chem)
plot(chem, magnetic, main = "Exponential", pch = 16)
logyhat3 <- L3$coef[1] + L3$coef[2] * a
yhat3 <- exp(logyhat3)
lines(a, yhat3, lwd = 2)
```




```
L4 <- lm(magnetic ~ chem + I(chem^2) + I(chem^3))
plot(chem, magnetic, main = "Cubic", pch = 16)
yhat4 <- L4$coef[1] + L4$coef[2] * a + L4$coef[3] * a^2 + L4$coef[4] * a^3
lines(a, yhat4, lwd = 2)
```




```
n <- length(magnetic)  #in DAAG ironslag
e1 <- e2 <- e3 <- e4 <- numeric(n)
# for n-fold cross validation fit models on leave-one-out samples
for (k in 1:n) {
    y <- magnetic[-k]
    x <- chem[-k]

    J1 <- lm(y ~ x)
    yhat1 <- J1$coef[1] + J1$coef[2] * chem[k]
    e1[k] <- magnetic[k] - yhat1

    J2 <- lm(y ~ x + I(x^2))
    yhat2 <- J2$coef[1] + J2$coef[2] * chem[k] + J2$coef[3] * chem[k]^2
```

```
    e2[k] <- magnetic[k] - yhat2

    J3 <- lm(log(y) ~ x)
    logyhat3 <- J3$coef[1] + J3$coef[2] * chem[k]
    yhat3 <- exp(logyhat3)
    e3[k] <- magnetic[k] - yhat3

    J4 <- lm(y ~ x + I(x^2) + I(x^3))
    yhat4 <- J4$coef[1] + J4$coef[2] * chem[k] + J4$coef[3] * chem[k]^2 + J4$coef[4] *
        chem[k]^3
    e4[k] <- magnetic[k] - yhat4
}

mean.error <- c(mean(e1^2), mean(e2^2), mean(e3^2), mean(e4^2))
mean.error

## [1] 19.56 17.85 18.44 18.18

which.min(mean.error)  #The smallest mean error

## [1] 2


R_squared <- c(summary(L1)$adj.r.squared, summary(L2)$adj.r.squared, summary(L3)$adj.r.squared,
    summary(L4)$adj.r.squared)
R_squared

## [1] 0.5282 0.5768 0.5281 0.5740

which.max(R_squared)

## [1] 2
```

The adjusted R^2 follows the same pattern as the cross validation error.