Practicing regression: the lm() function

author: Maria Paniw date: February 8, 2016; UCA

The `lm` function in R deals with multiple linear regression. Here, I want to show you the mathematics behind it.

Let's create some data

We want to fit the following model:

$$Y_i = \beta_0 + \beta_1 X1_i + \beta_2 X2_i + \epsilon_i$$

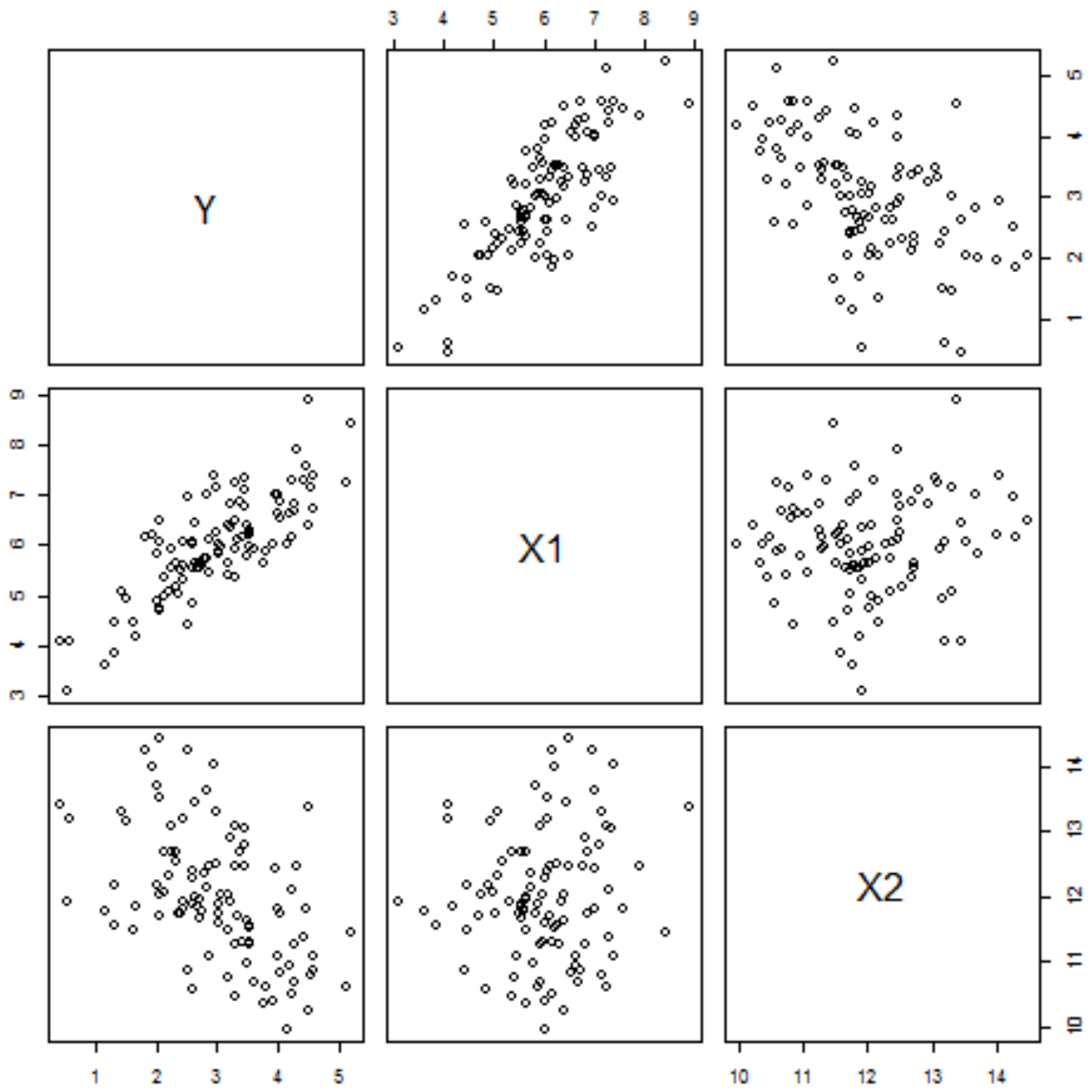
```
# To simulate data, the first thing we do is a create a matrix
# that describes the correlation between X1, X2, and Y.

# Y is negatively correlated  with X2
# Y is stronly positively correlated with X1
# X1 and X2 are not correlated.

varCov=matrix(cbind(1,.80,-.5,  .80,1,.1,  -.5,.1,1),nrow=3)
```

```
colnames(varCov) = rownames(varCov) = c("Y","X1","X2")
varCov
```

```
      Y  X1   X2
Y   1.0 0.8 -0.5
X1  0.8 1.0  0.1
X2 -0.5 0.1  1.0
```

```
library(MASS)
# simulate the data using the mvrnorm function
set.seed(1) # fix the random number generator
data=mvrnorm(n = 100, mu=c(3,6,12), Sigma=varCov, empirical = T)
```

```
pairs(data)
```



Find the betas:

Remember the model we wanted to fit!

$$Y_i = \beta_0 + \beta_1 X1_i + \beta_2 X2_i + \epsilon_i$$

Before taking the easy road (`lm`), let's actually see what the easy road does behind the scenes.

Numerical optimization

Remember, we want to find least-square estimates (LSE) of β_0, β_1, and β_2 that mimimize the RSS. Numerical optimization would loop through many values and find the LSE. This is what non-linear regression (`nls`) and machine-learning techniques of modeling do.

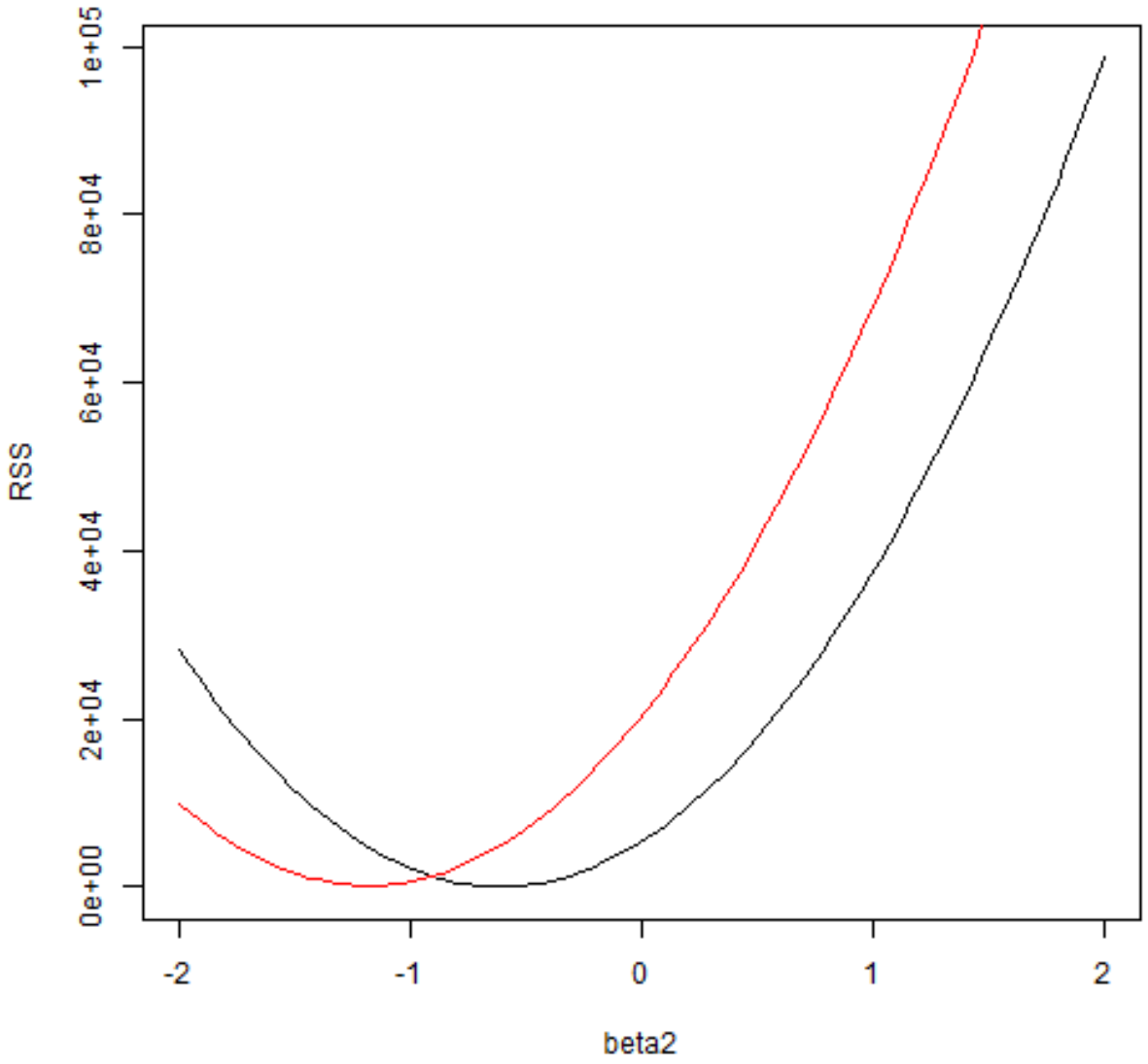
Let's take a look at a function that would do just that!

```
rss <- function(beta0,beta1,beta2){
r <- data[,1] - (beta0+beta1*data[,2]+beta2*data[,3])
return(sum(r^2))
}
```

Obviously, having three parameters to estimate and no good starting points, we may get in many issues. Numerical optimization typically creates very efficient algorithms for such multidimensional tasks.

In our case, let us assume we know `beta0` and `beta1` and are interested in `beta2`. For `beta2`, we have some reasonable range of values that may fit.

```
#Simulate and plot results
beta2s= seq(-2,2,len=100)
plot(beta2s,sapply(beta2s,rss,beta0=4.9,beta1=0.9),
ylab="RSS",xlab="beta2",type="l")
##Let's add another curve fixing another pair:

lines(beta2s,sapply(beta2s,rss,beta0=5.2,beta1=2),col=2)
```



The matrix algebra approach

Numerical optimization, properly done (and not trial-and-error as we did above) is a great tool for modern data-driven statistical analyses. For linear regression, maybe THE most established statistical analysis on this planet, we use calculus to find the β - and this is `lm` behind the scenes.

Solving a system of equations

$$\begin{aligned} a+b+c &= 6 \\ 3a-2b+c &= 2 \\ 2a+b-c &= 1 \end{aligned}$$

This system can be rewritten and solved using matrix algebra:

$$\begin{pmatrix} 1 & 1 & 1 \\ 3 & -2 & 1 \\ 2 & 1 & -1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 6 \\ 2 \\ 1 \end{pmatrix} \implies \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 3 & -2 & 1 \\ 2 & 1 & -1 \end{pmatrix}^{-1} \begin{pmatrix} 6 \\ 2 \\ 1 \end{pmatrix}$$

The matrix representation of a linear regression is:

$$Y_i = \beta_0 + \beta_1 x_i + \varepsilon, i = 1, \ldots, N$$

$$\begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_N \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \\ 1 & x_N \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_N \end{pmatrix}$$

or simply:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

Doing some calculus, we can get the estimates of of β, $\hat{\beta}$ as

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}$$

For more detail, see this wonderful book: Data Analysis for the Life Sciences

Let's look concretely at our data example:

```
X1=data[,"X1"]
X2=data[,"X2"]

X=model.matrix(~X1+X2) # this functions will be important for the rest of the course

head(X)
```

```
  (Intercept)       X1       X2
1           1 6.157015 11.30207
2           1 7.573721 11.80915
3           1 7.157177 10.79826
4           1 5.737468 12.34323
5           1 3.639242 11.77003
6           1 8.892114 13.37466
```

```
Y=data[,"Y"]

beta=solve(crossprod(X))%*%crossprod(X,Y)

beta
```

```
                 [,1]
(Intercept)  4.8787879
X1           0.8585859
X2          -0.5858586
```

Now the lm function

Finally let's take the easy way out:

```
mod=lm(Y~X1+X2,data=data.frame(data))
summary(mod) #Get the summary output
```

```
Call:
lm(formula = Y ~ X1 + X2, data = data.frame(data))

Residuals:
     Min       1Q   Median       3Q      Max
-0.36087 -0.09233 -0.00368  0.08543  0.35775

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   4.8788     0.1872   26.06   <2e-16 ***
X1            0.8586     0.0145   59.20   <2e-16 ***
X2           -0.5859     0.0145  -40.39   <2e-16 ***

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1436 on 97 degrees of freedom
Multiple R-squared:  0.9798,    Adjusted R-squared:  0.9794
F-statistic:  2352 on 2 and 97 DF,  p-value: < 2.2e-16
```

```
#compare to the beta vector...
```

==========

```
#Get the standard deviation of the function:

s = sqrt( sum(residuals(mod)^2)/(nrow(X)-ncol(X)))
```

How do we calculate the standard errors of the betas?

```r
se_beta=sqrt((s^2)*diag(solve(t(X)%*%X)))
se_beta
```

```
(Intercept)          X1          X2
 0.18719985  0.01450421  0.01450421
```

This is exactly what the summary function gives us!

```r
summary(mod)
```

```
Call:
lm(formula = Y ~ X1 + X2, data = data.frame(data))

Residuals:
     Min       1Q   Median       3Q      Max
-0.36087 -0.09233 -0.00368  0.08543  0.35775

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   4.8788     0.1872   26.06   <2e-16 ***
X1            0.8586     0.0145   59.20   <2e-16 ***
X2           -0.5859     0.0145  -40.39   <2e-16 ***

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1436 on 97 degrees of freedom
Multiple R-squared:  0.9798,    Adjusted R-squared:  0.9794
F-statistic:  2352 on 2 and 97 DF,  p-value: < 2.2e-16
```

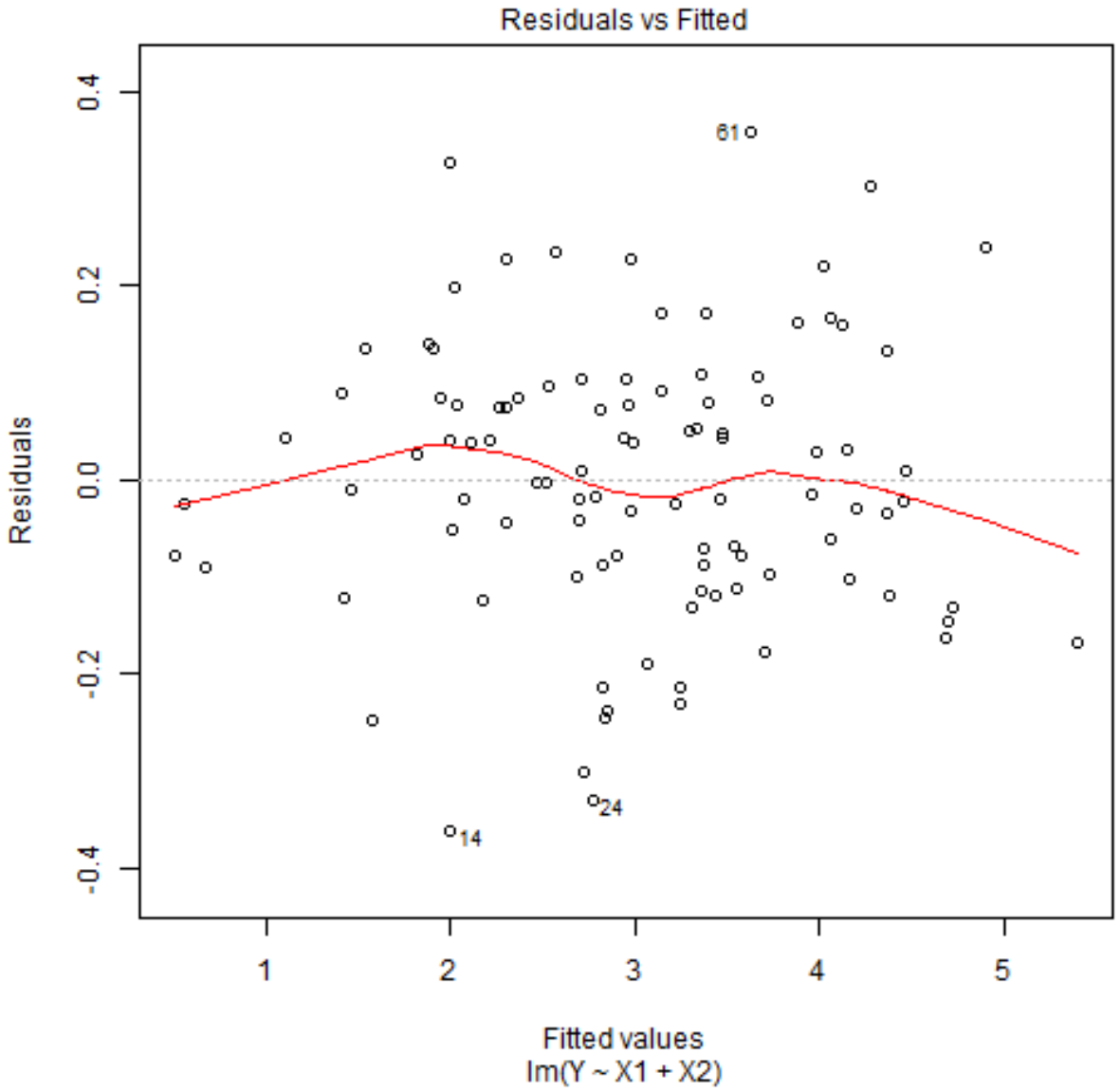
What about all the assumptions we make?

- Linearity
- Normality
- Homoscedasticity

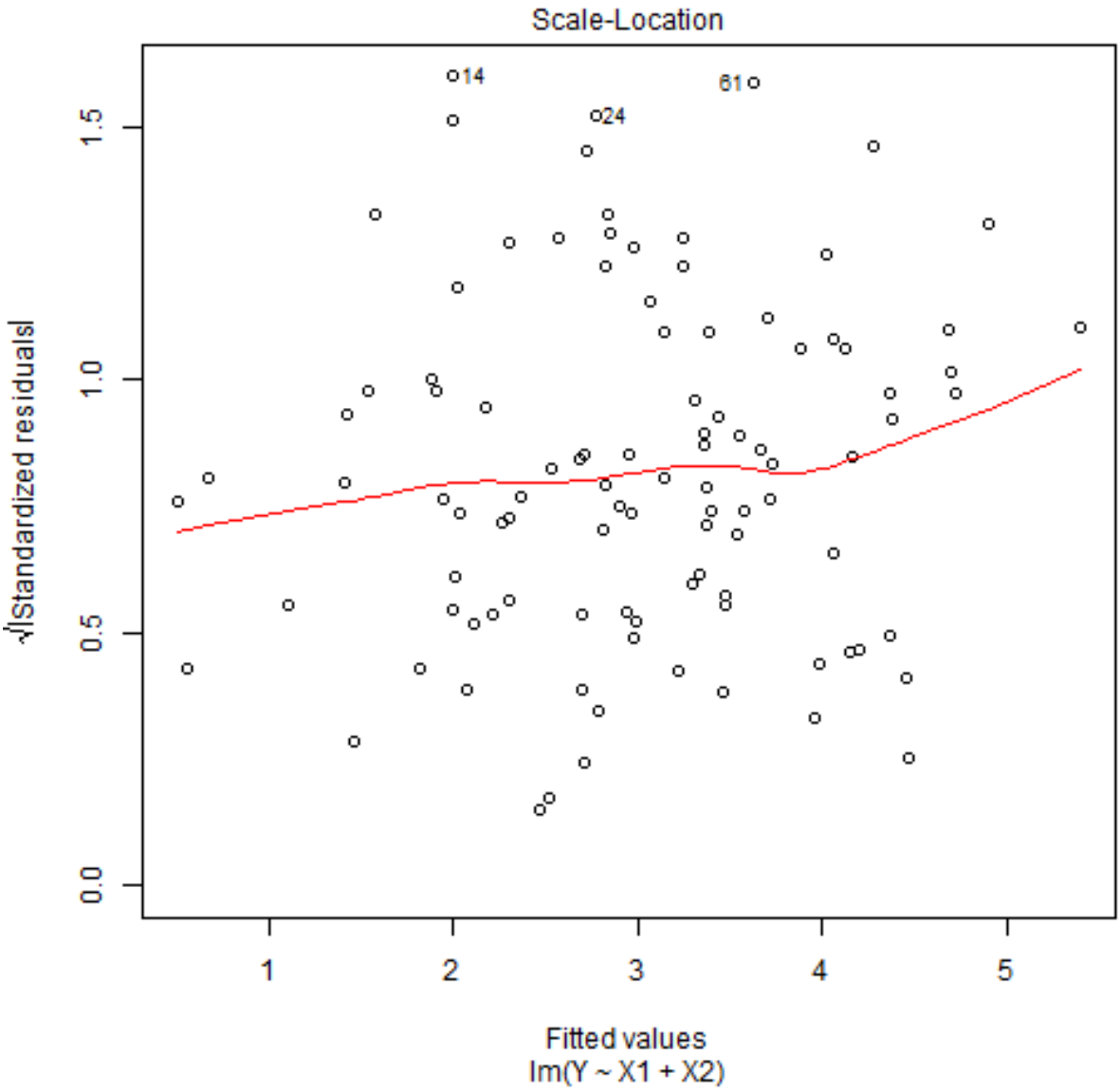
We can plot our `mod` object!

```r
plot(mod)
```

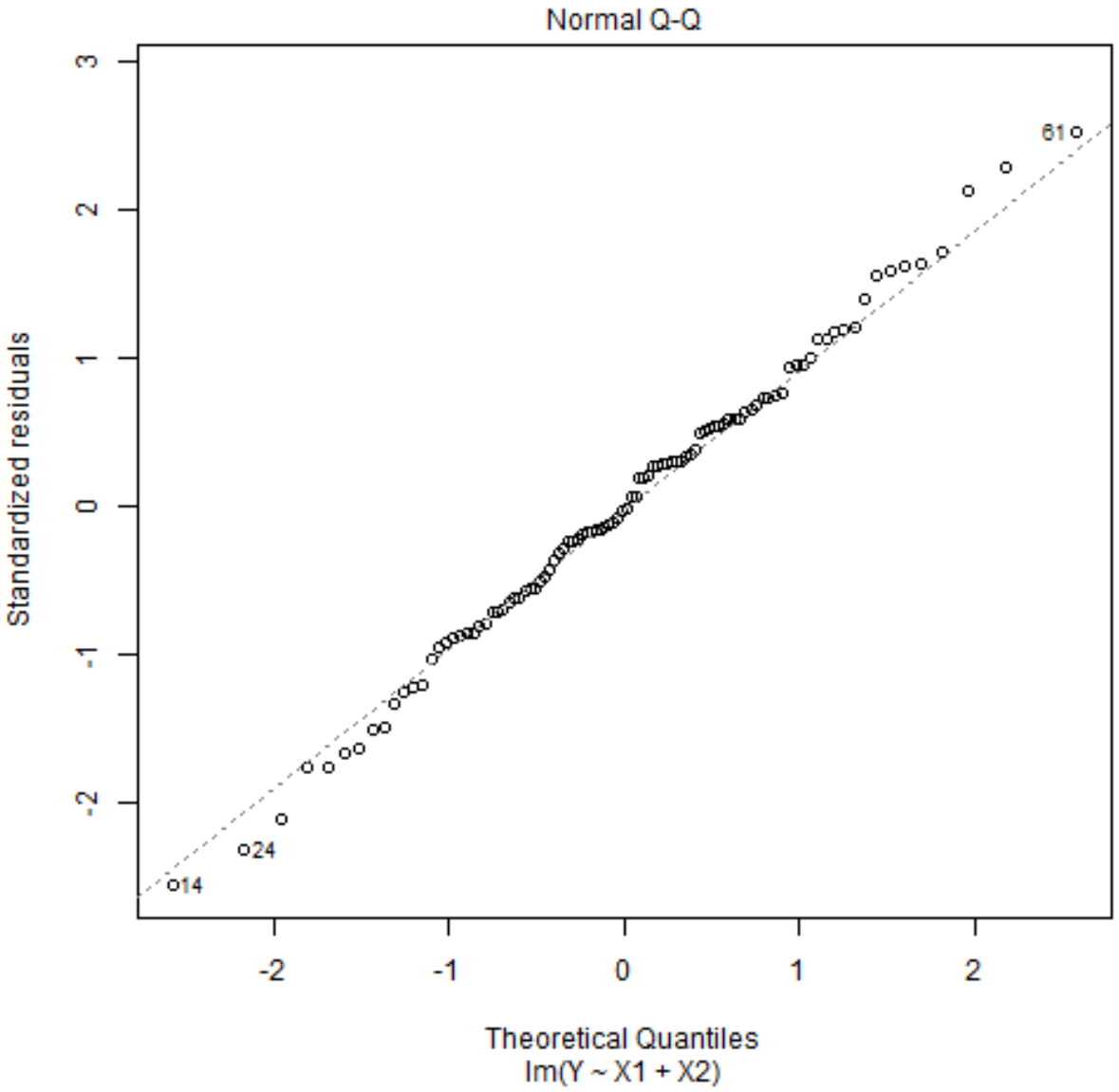
The distribution of residuals vs. predicted values should be random. It tells you something about the linearity and homoscedasticity assumptions, namely, if the distribution has a pattern, increase in variance for example, these assumptions are not met!



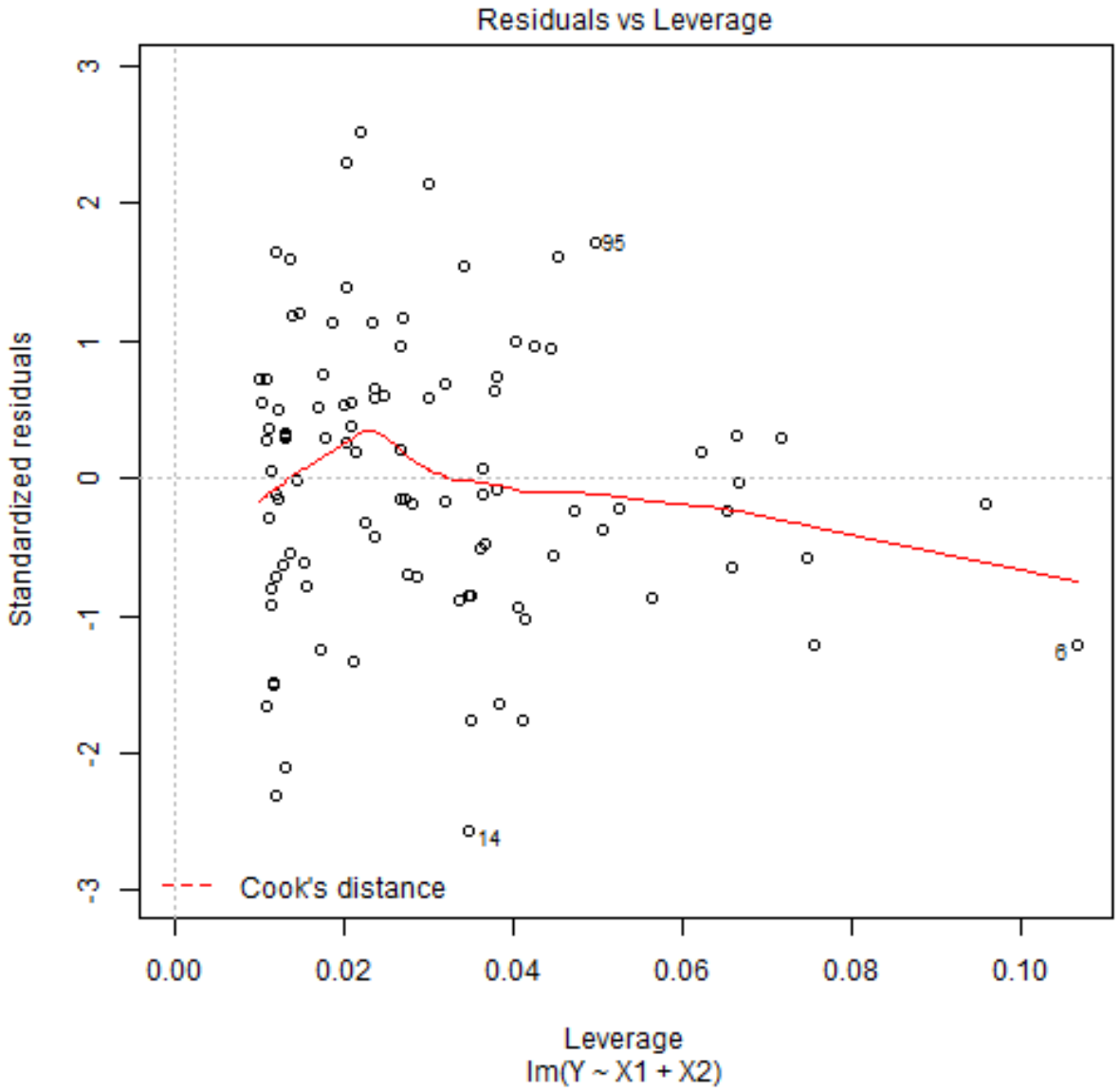
Same thing as the residual plot but with standardized values.



The Q-Q plot compares two vectors to see whether they have the same statistical distribution. Here, we are comparing the model residuals vs. a perfect normal distribution assuming the mean and sd of the residuals. If the points fall on the line, the residuals are normally distributed.



This plot shows you what point in your data have an exceptional weight (or leverage) on model fit. Such points may be potential outliers.


Residuals vs Leverage
Standardized residuals
Leverage
lm(Y ~ X1 + X2)
Cook's distance
95
14
6
3
2
1
0
-1
-2
-3
0.00
0.02
0.04
0.06
0.08
0.10