R Code For Simple Linear Regression

The Heart Data

The heart data measure height (in) and weight (lb) of 12 children. These data also record the arterial distance to their heart. These data wree taken in order to improve the insertion of stents to treat a number of heart problems. For our purposes today, we will just look at the relationship between height and weight.

First, we must load in the data. If you have downloaded heart.txt from the blackboard site and put it into your working directory, you can proceed with.

```
heart = read.table('heart.txt',head=TRUE)
```

If you examine this, we can look at

```
heart
```

```
##    height weight dist
## 1    24.8   40.0 37.0
## 2    63.5   93.5 49.5
## 3    37.5   35.5 34.5
## 4    39.5   30.0 36.0
## 5    45.5   52.0 43.0
## 6    38.5   17.0 28.0
## 7    43.0   38.5 37.0
## 8    22.5    8.5 20.0
## 9    37.0   33.0 33.5
## 10   23.5    9.5 30.5
## 11   33.0   21.0 38.5
## 12   58.5   79.0 47.0
```

Where we see the column names and the data. You can access a column by

```
heart$weight
```

```
##  [1] 40.0 93.5 35.5 30.0 52.0 17.0 38.5  8.5 33.0  9.5 21.0 79.0
```

You can also access the second entry in the third row by

```
heart[3,2]
```

```
## [1] 35.5
```

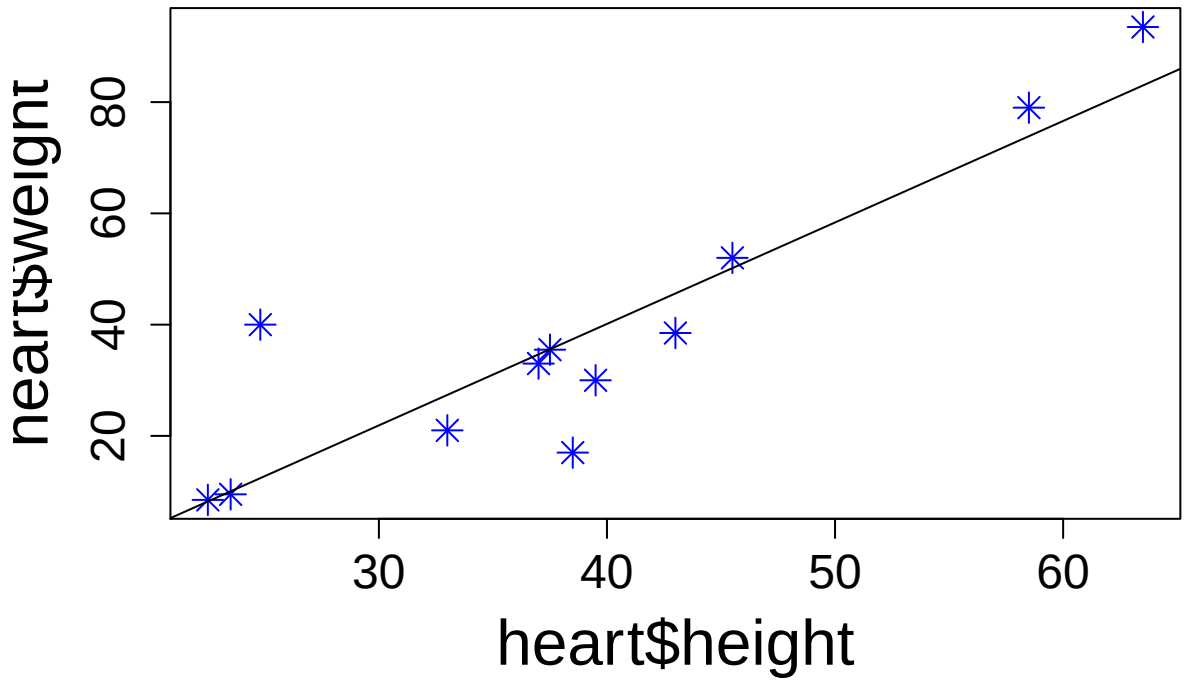
Or the whole of the third row by

```
heart[3,]
```

```
##   height weight dist
## 3   37.5   35.5 34.5
```

Now we will plot the data; making the axes and labels a bit larger and adding a line of best fit.

```
plot(heart$height,heart$weight,cex.axis=1.5,cex.lab=2,cex=1.5,col=4,pch=8)
abline(lm(heart$weight~heart$height)$coef)
```



From Silde 7 in Lecture 2, here are the calculations to get us to correlation

```
# Average value of weight
m.weight = mean(heart$weight)
m.weight
```

```
## [1] 38.125
```

```
# Average value of height
m.height = mean(heart$height)
m.height
```

```
## [1] 38.9
```

```
# Variance of weight
var.weight = var(heart$weight)
var.weight
```

```
## [1] 679.0057
```

```r
# Variance of height
var.height = var(heart$height)
var.height
```

```
## [1] 163.2291
```

```r
# Covariance between them
cov.heart = cov(heart$height,heart$weight)
cov.heart
```

```
## [1] 297.7045
```

```r
# Formula for correlation
cor.heart = cov.heart/sqrt( var.height * var.weight )
cor.heart
```

```
## [1] 0.8942315
```

```r
# Just using the correlation function gets you there directly
cor.heart2 = cor(heart$height,heart$weight)
cor.heart2
```

```
## [1] 0.8942315
```

Illustrating a Statistical Model with Made Up Data

In this section we will use made-up data (so we know what the truth is) to illustrate the statistical properties of the simple linear model and its estimation.

Setting up a linear model

We will start by setting up the linear model

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

We will start off by specifying β_0, β_1 and the error standard deviation σ:

```r
beta0 = 0;   beta1 = 1; sigma = 0.25
```

We will then take X to be spaced from 0 to 1 in intervals of 0.1

```r
X = seq(0,1,by=0.1)
X
```

```
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

and generate the ϵ_i from a normal distribution with standard deviation σ:

```
epsilon = rnorm(11,mean=0,sd=sigma)
epsilon
```

```
##  [1]  0.11639364  0.48311862 -0.04860514 -0.38991241  0.20278858
##  [6] -0.12634581  0.16216996  0.08243179 -0.10609095 -0.30087765
## [11] -0.12708452
```

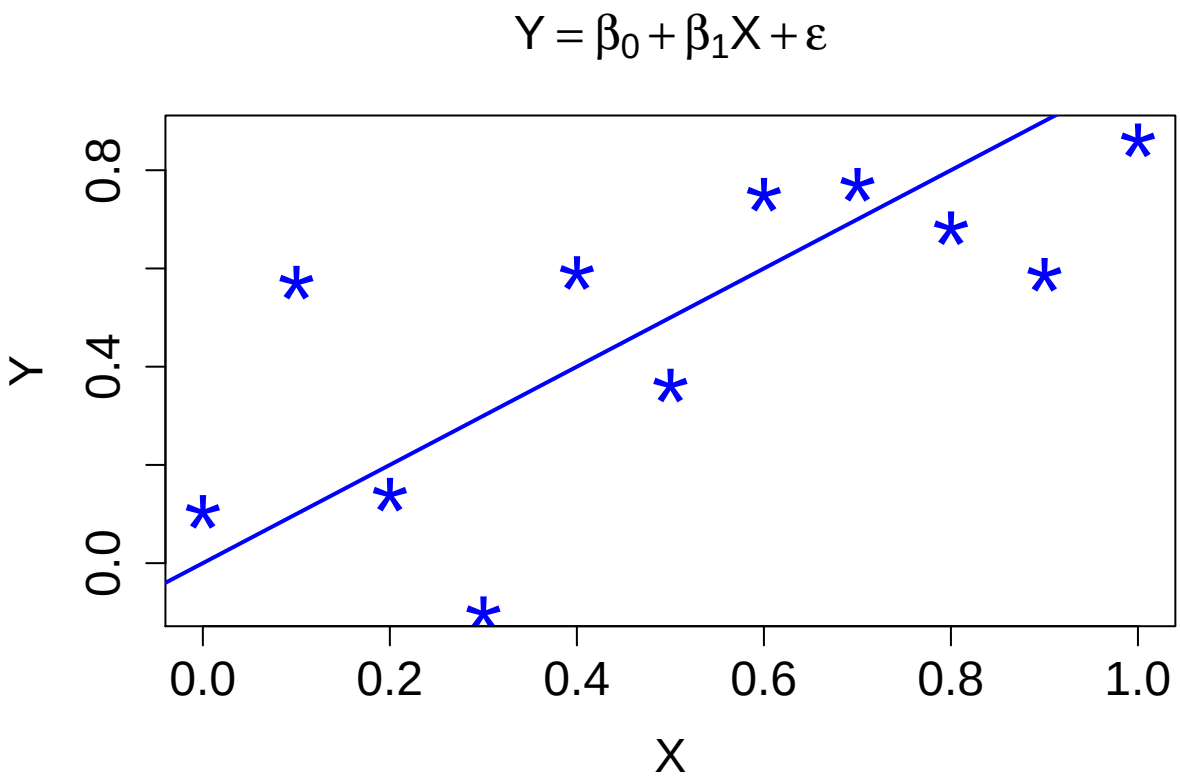
We can now generate response values

```
Y = beta0 + beta1*X + epsilon
Y
```

```
##  [1]  0.11639364  0.58311862  0.15139486 -0.08991241  0.60278858
##  [6]  0.37365419  0.76216996  0.78243179  0.69390905  0.59912235
## [11]  0.87291548
```

and plot these

```
plot(X, Y,pch='*',lwd=2,col='blue',xlab='X',ylab='Y',
    main=expression(Y==beta[0]+beta[1] * X+epsilon),
    cex.lab=1.5,cex.axis=1.5,cex.main=1.5,cex=3)

abline(c(beta0,beta1),lwd=2,col='blue')
```



Estimating parameters from data

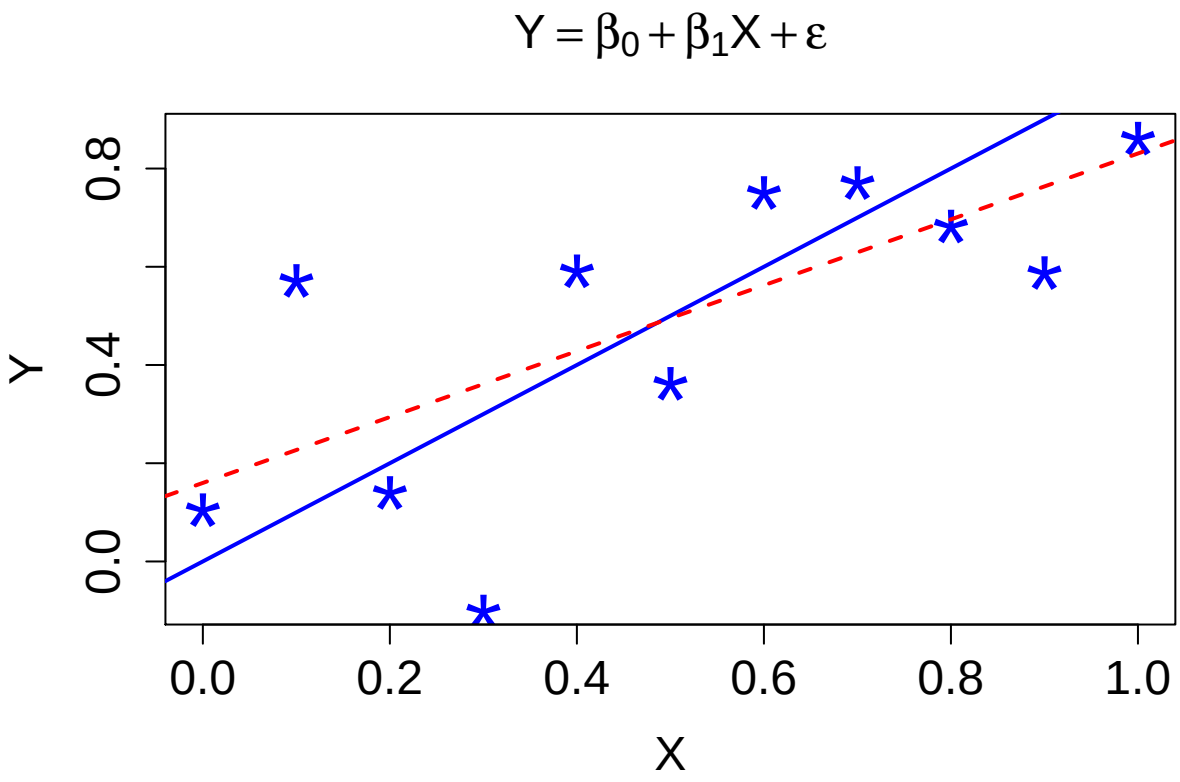
We can estimate parameters from data using the lm function

```r
mod = lm(Y~X)
mod$coefficients    # 'truth' is 0 and 1
```

```
## (Intercept)           X
##   0.1598971   0.6707488
```

and add this line to the plot

```r
plot(X, Y,pch='*',lwd=2,col='blue',xlab='X',ylab='Y',
    main=expression(Y==beta[0]+beta[1] * X+epsilon),
    cex.lab=1.5,cex.axis=1.5,cex.main=1.5,cex=3)

abline(c(beta0,beta1),lwd=2,col='blue')

abline(mod,lty=2,lwd=2,col='red')
```

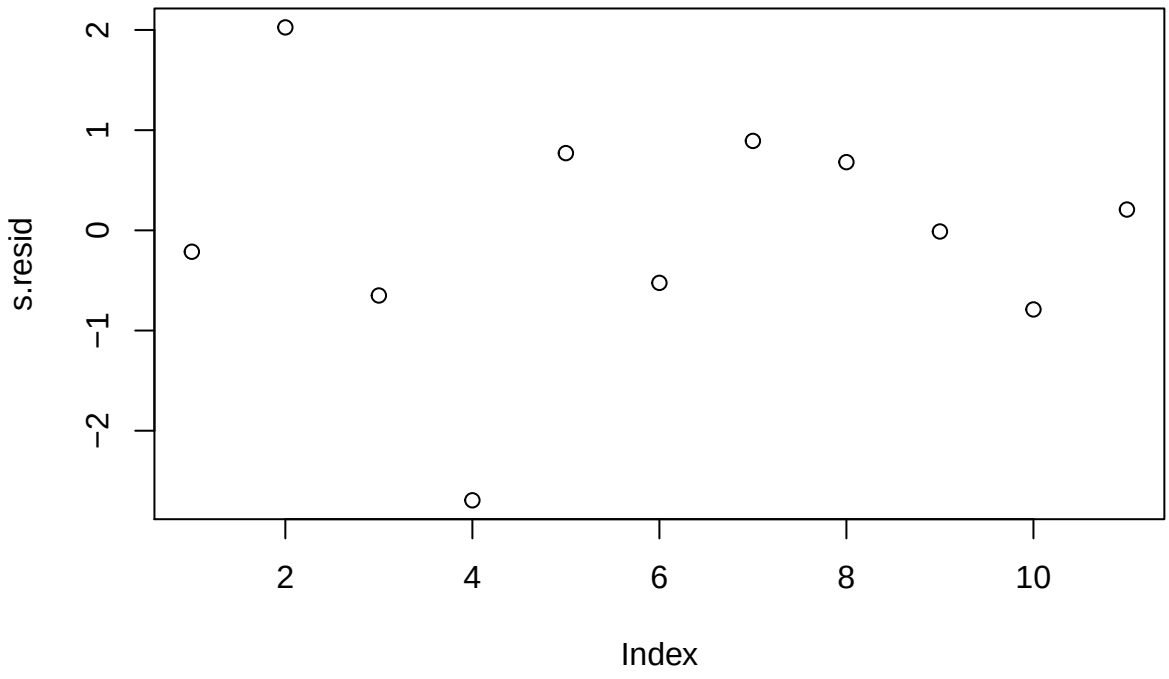


Looking at some diagnostics

We can obtain studentized residuals from the MASS packages

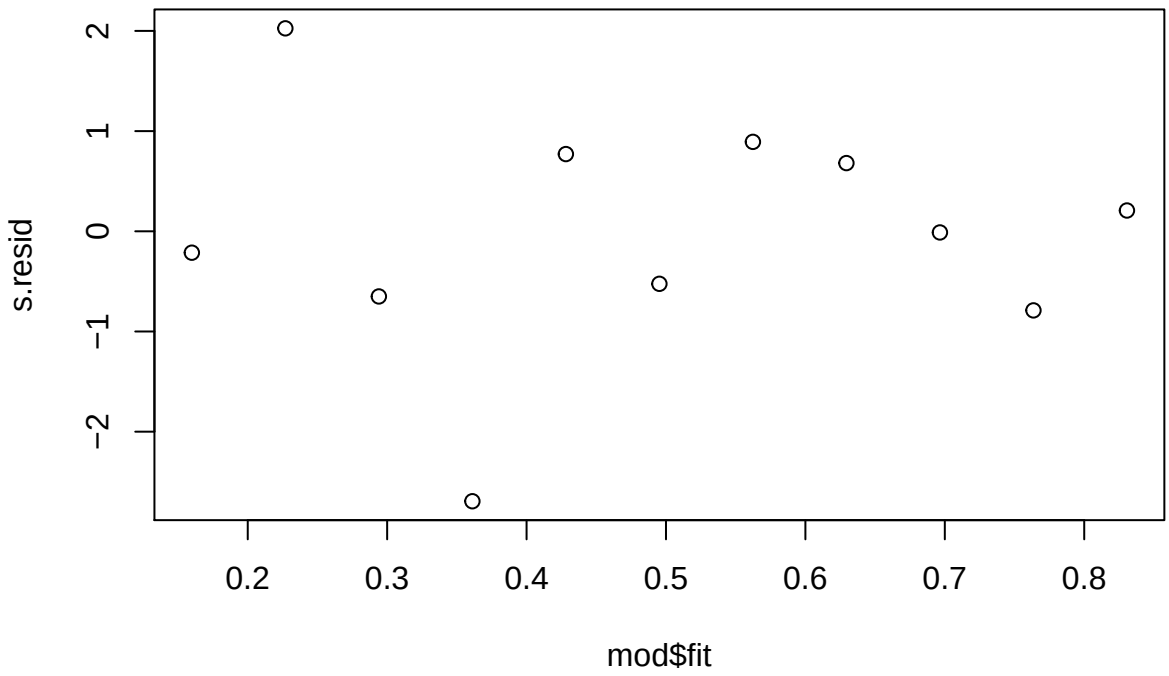
```r
library('MASS')

s.resid = studres(mod)
```

```
plot(s.resid)
```



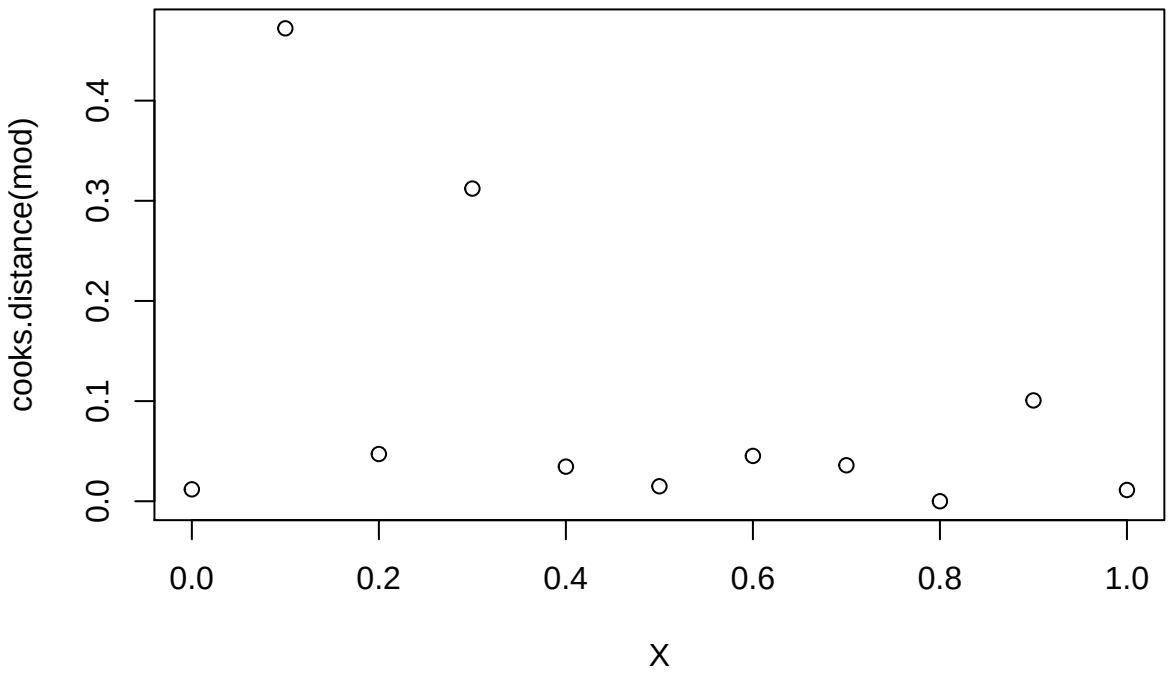
It also makes some sense to plot studentized residuals versus fitted values

```
plot(mod$fit,s.resid)
```



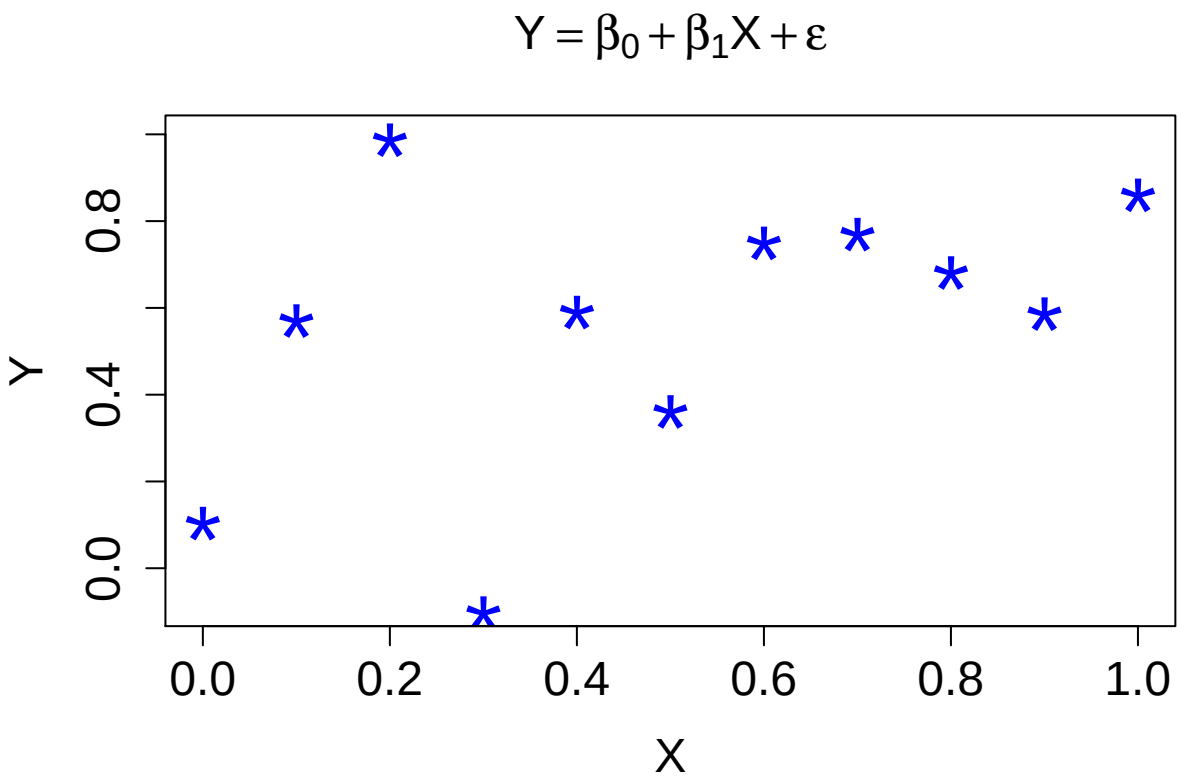
We can also plot Cook's distance

```
plot(X,cooks.distance(mod))
```



To see what happens for influential points, we'll create an outlier artificially

```
Y[3] = 1

plot(X, Y,pch='*',lwd=2,col='blue',xlab='X',ylab='Y',
    main=expression(Y==beta[0]+beta[1] * X+epsilon),
    cex.lab=1.5,cex.axis=1.5,cex.main=1.5,cex=3)
```



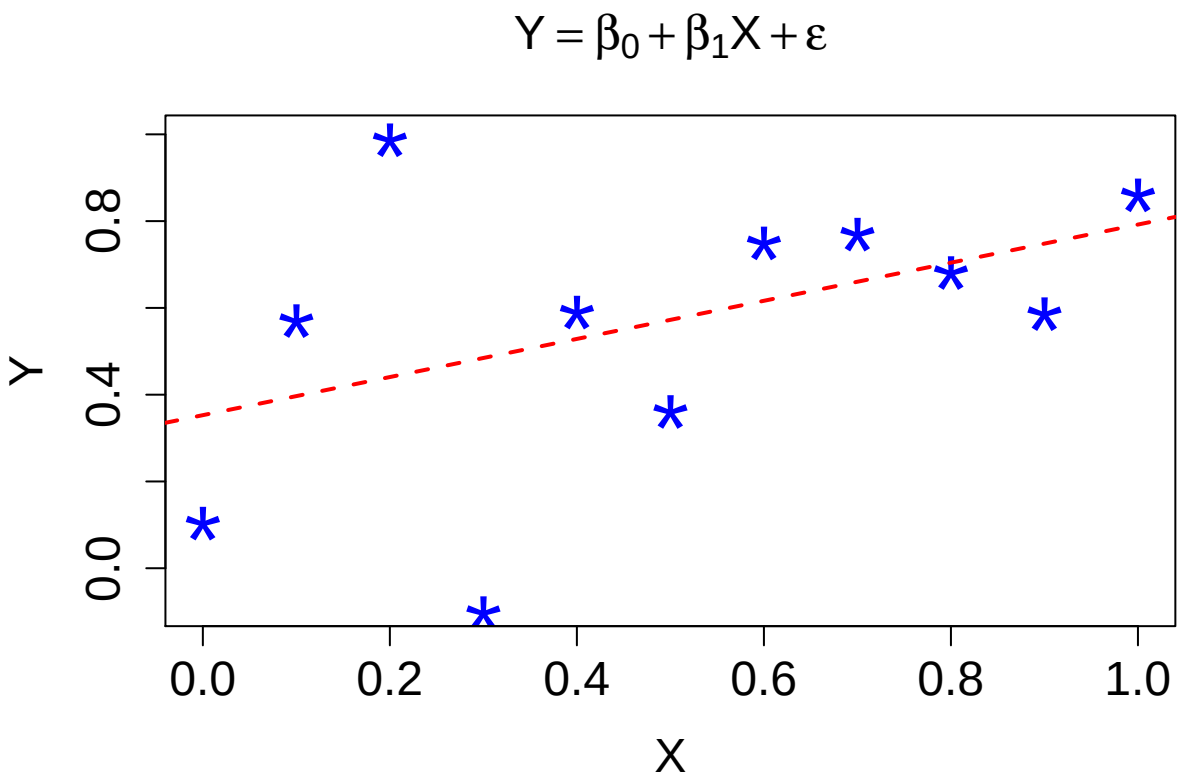
And estimate a linear model with these data, adding the best fit line to the plot

```
mod = lm(Y~X)
mod$coefficients    # 'truth' is 0 and 1
```

```
## (Intercept)           X
##   0.3527619   0.4393110
```

and add this to the plot too

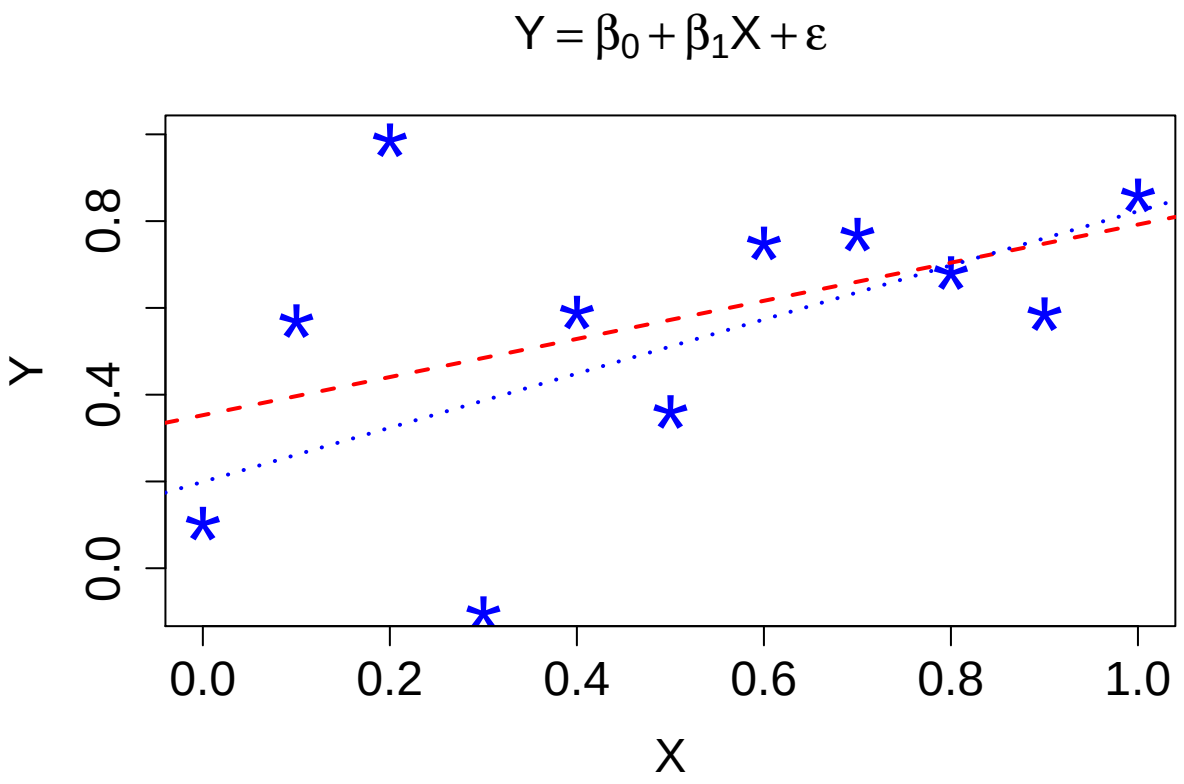
```
plot(X, Y,pch='*',lwd=2,col='blue',xlab='X',ylab='Y',
    main=expression(Y==beta[0]+beta[1] * X+epsilon),
    cex.lab=1.5,cex.axis=1.5,cex.main=1.5,cex=3)

abline(mod,lty=2,lwd=2,col='red')
```



We'll also look at the line that results when we don't include the outlier in the data set. To do this, we have

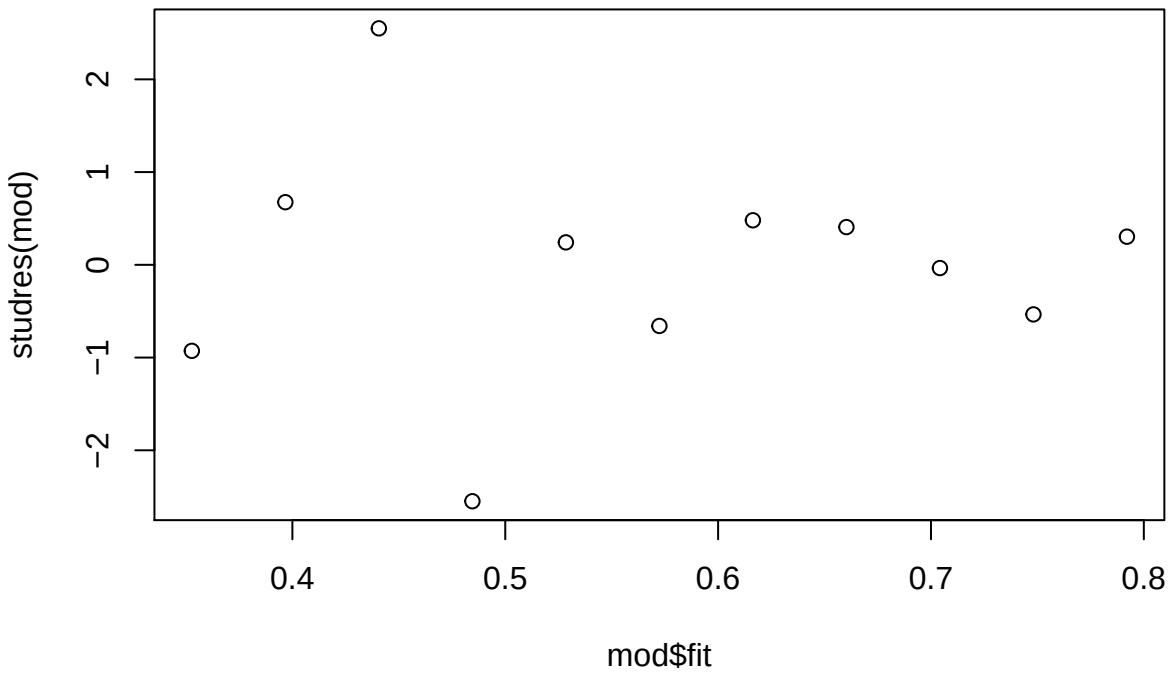
```
mod.red = lm(Y[-3]~X[-3])
summary(mod.red)
```

```
##
## Call:
## lm(formula = Y[-3] ~ X[-3])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.47612 -0.12364  0.02298  0.15234  0.32166
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.1991     0.1488   1.338   0.2177
## X[-3]         0.6237     0.2411   2.587   0.0322 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2412 on 8 degrees of freedom
## Multiple R-squared:  0.4556, Adjusted R-squared:  0.3875
## F-statistic: 6.694 on 1 and 8 DF,  p-value: 0.03225
```

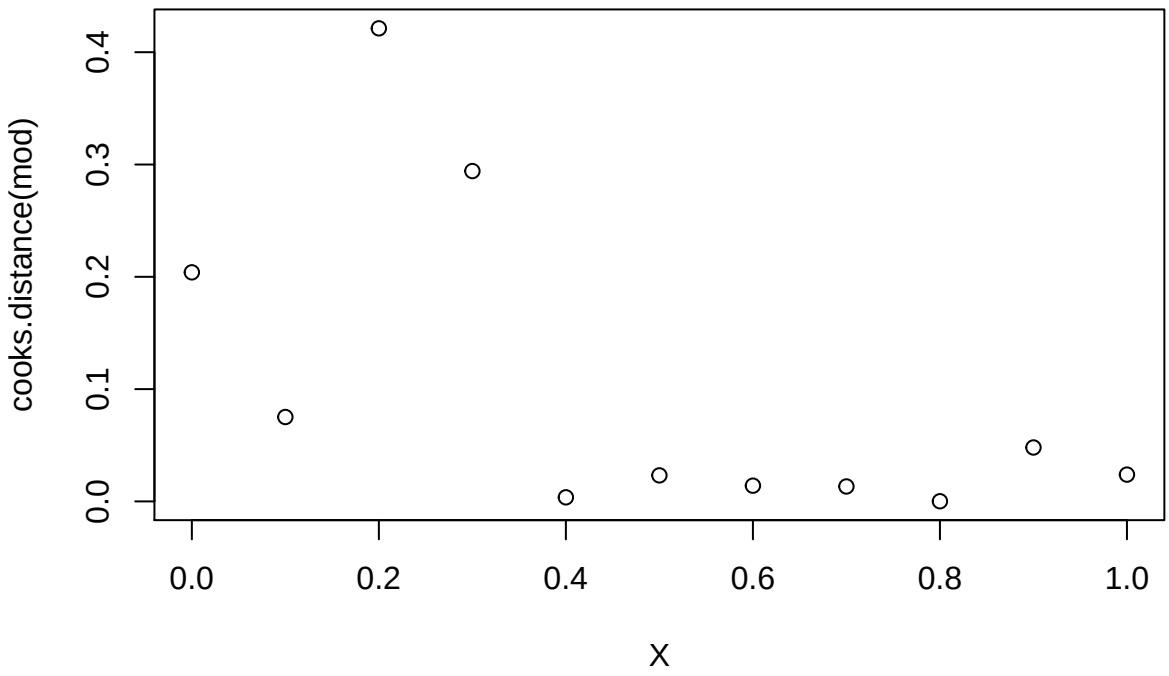


In this case, studentized residuals and Cooks distance ought to pick out the outlier

```
plot(mod$fit,studres(mod))
```




```
plot(X,cooks.distance(mod))
```



Computing Confidence Intervals

First we need toestimate the variance

```
sig.hat = sum( mod$resid^2 )/9
```

and obtain the sum of squares for X:

```
SXX = sum( (X-mean(X))^2 )
```

Then plug these into the variance expressions

```
sd.beta1 = sqrt(sig.hat/SXX)
sd.beta0 = sqrt(sd.beta1^2 * mean( X^2 ))
```

To obtain confidence intervals, we take the estimate plus and minus variance times the critical value of t-distribution (qt gives the quantiles of the t distribution)

```
c( mod$coef[1] - qt(0.975,9)*sd.beta0,mod$coef[1] + qt(0.975,9)*sd.beta0 )
```

```
## (Intercept) (Intercept)
##  -0.0379058   0.7434296
```

```
c( mod$coef[2] - qt(0.975,9)*sd.beta1,mod$coef[2] + qt(0.975,9)*sd.beta1 )
```

```
##          X          X
## -0.2210379  1.0996599
```

We can do this much more easily using the confint function:

```
confint(mod)
```

```
##                   2.5 %    97.5 %
## (Intercept) -0.0379058 0.7434296
## X           -0.2210379 1.0996599
```

A Simulation

All of statistical inference is really asking the question "What would happen if we ran the experiment again?". When we are generating the data, we can!

To run a simulation first we will define some arrays to hold the simulation values.

coefmat is a 1000-by-2 array to hold the two coefficients, β_0 and β_1 for each of the 1000 simulations

```
coefmat = matrix(0,1000,2)
```

We will also produce a 1000-by-11 array to hold the fitted values for each simulation

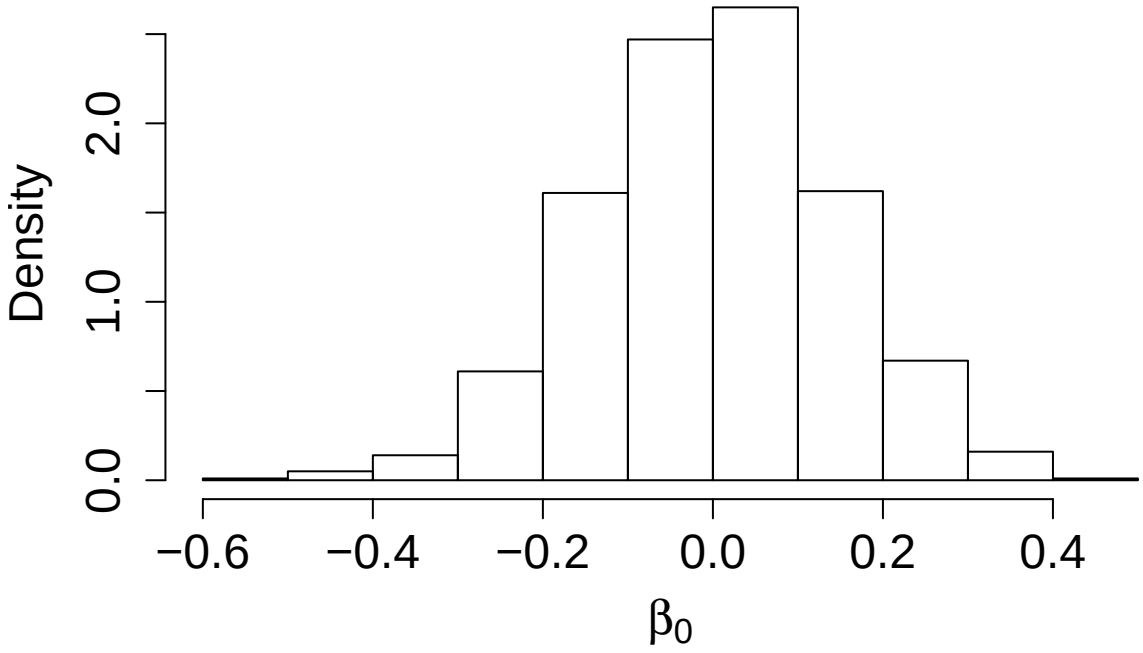
```
predmat = matrix(0,1000,11)
```

Now we will repeat the above 1000 times:

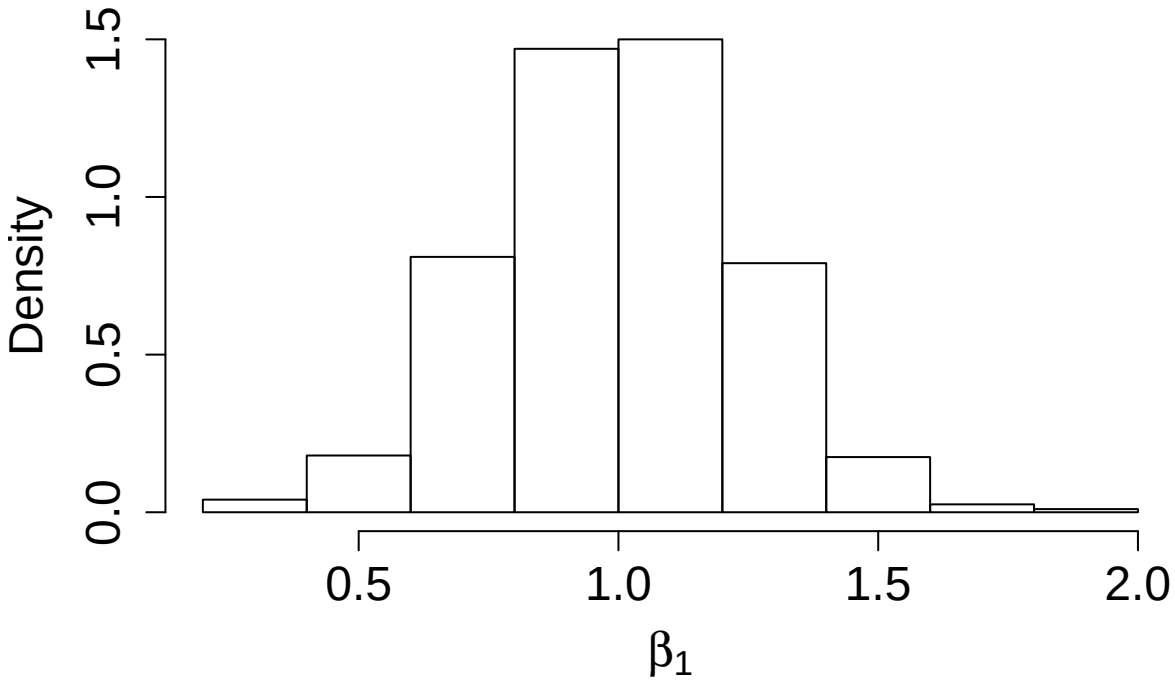
```
for(sim in 1:1000){
  epsilon = rnorm(11,mean=0,sd=sigma) # New observation errors
  Y =  beta0 + beta1*X + epsilon      # New response values

  mod = lm(Y~X)                       # Refit the model
  coefmat[sim,] = mod$coefficients    # Store fitted coefs

  predmat[sim,] = mod$fit             # Store fitted values
}
```

Now we look at the results. First histograms of the coefficients

```
hist(coefmat[,1],prob=TRUE,xlab=expression(beta[0]),main='',cex.lab=1.5,cex.axis=1.5)
```




```
hist(coefmat[,2],prob=TRUE,xlab=expression(beta[1]),main='',cex.lab=1.5,cex.axis=1.5)
```



Which look reasonably normal. Lets have a look at the variance of each of these

```
var(coefmat[,1])
```

```
## [1] 0.02036476
```

```
var(coefmat[,2])
```

```
## [1] 0.05922336
```

Ideally, the formulae that we saw in class will be pretty close to this (remeber that the results of the simulation are still random).

```
var.beta1 = sigma^2/SXX
var.beta1
```

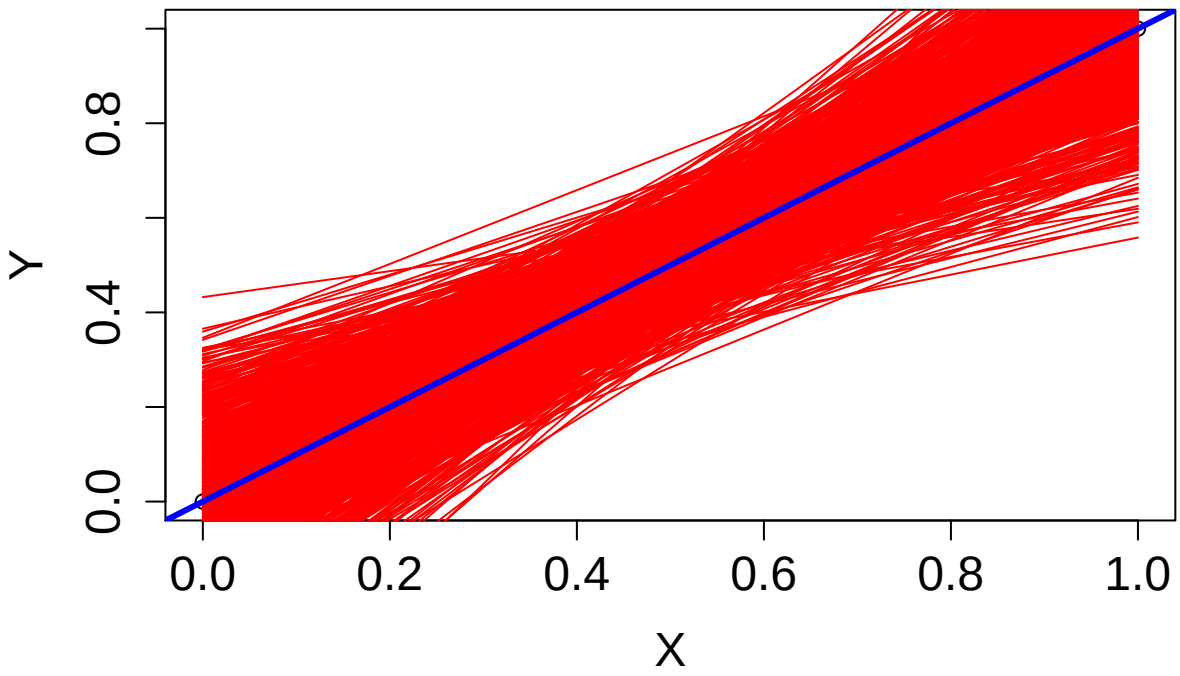
```
## [1] 0.05681818
```

```
var.beta0 = var.beta1 * mean( X^2 )
var.beta0
```

```
## [1] 0.01988636
```

To make this concrete, we will plot all 1000 estimated regression lines

```
plot(X,beta0+beta1*X,ylab='Y',cex.lab=1.5,cex.axis=1.5)
for(i in 1:1000){ lines(X,predmat[i,],col='red') }
abline(c(beta0,beta1),col='blue',lwd=3)
```

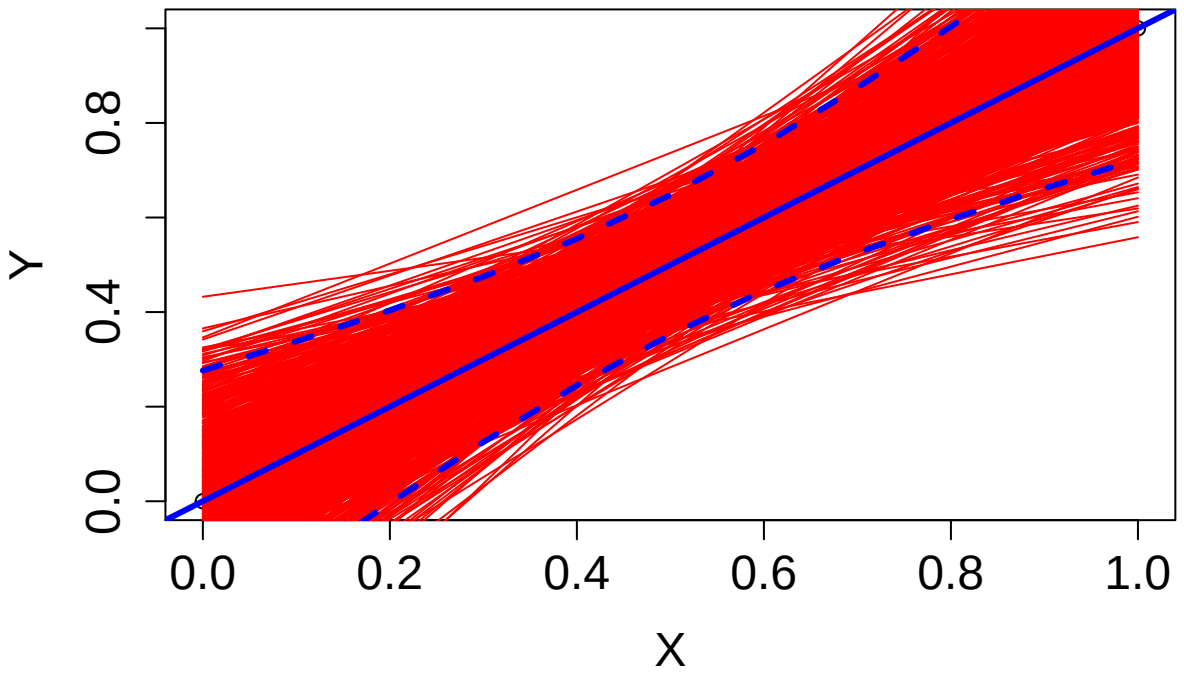


and add in the true line $\pm$ confidence and prediction intervals. The variance of the fitted values is

```
var.pred = sigma^2*(1/11 + (X - mean(X))^2/SXX)
```

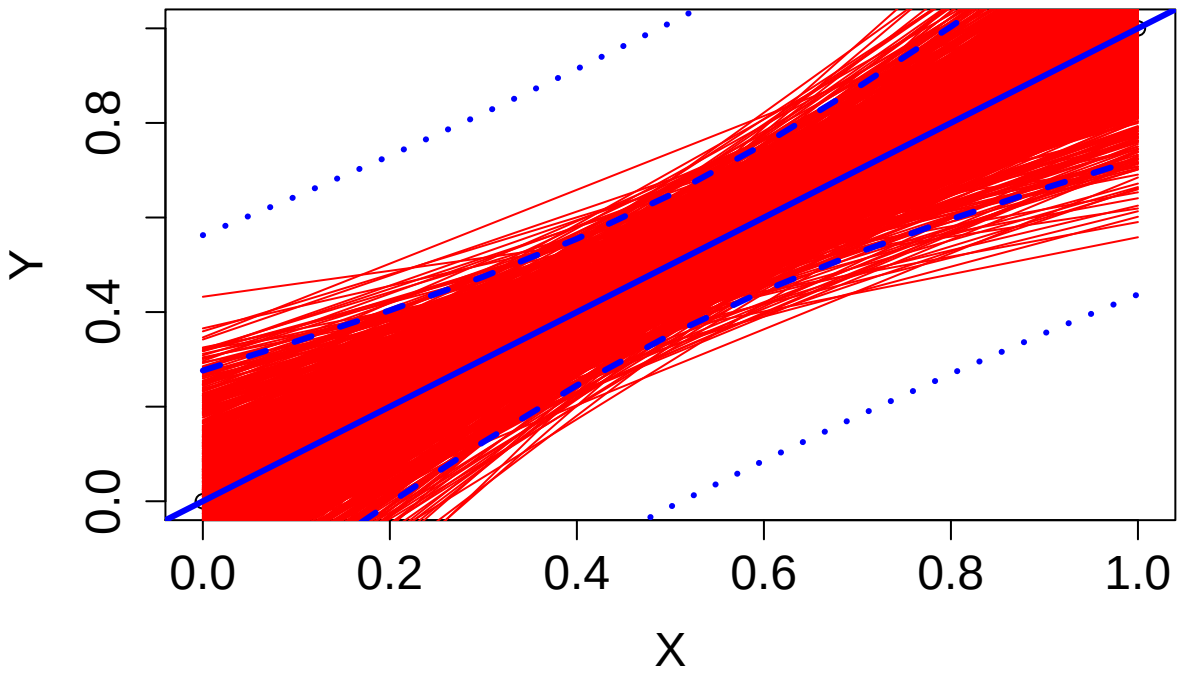
and we can 1.96 times the standard deviation of these fitted values to the plot

```
plot(X,beta0+beta1*X,ylab='Y',cex.lab=1.5,cex.axis=1.5)
for(i in 1:1000){ lines(X,predmat[i,],col='red') }
abline(c(beta0,beta1),col='blue',lwd=3)
lines(X,beta0+beta1*X + 1.96 * sqrt(var.pred), lty=2,lwd=3,col='blue')
lines(X,beta0+beta1*X - 1.96 * sqrt(var.pred), lty=2,lwd=3,col='blue')
```



The variance of a new point is the variance of the fitted value plus the standard deviation, so we can add prediction intervals this way

```
plot(X,beta0+beta1*X,ylab='Y',cex.lab=1.5,cex.axis=1.5)
for(i in 1:1000){ lines(X,predmat[i,],col='red') }
abline(c(beta0,beta1),col='blue',lwd=3)

lines(X,beta0+beta1*X + 1.96 * sqrt(var.pred), lty=2,lwd=3,col='blue')
lines(X,beta0+beta1*X - 1.96 * sqrt(var.pred), lty=2,lwd=3,col='blue')

lines(X,beta0+beta1*X + 1.96 * sqrt(sigma^2+var.pred), lty=3,lwd=3,col='blue')
lines(X,beta0+beta1*X - 1.96 * sqrt(sigma^2+var.pred), lty=3,lwd=3,col='blue')
```



Food pH data

The food data give the time that food has been allowed to sit and the corresponding pH values measured. We obtain these data by the following read-out

```
food = c(1,1,2,2,4,4,6,6,8,8,7.02,6.93,6.42,6.51,6.07,5.99,5.59,5.8,5.51,5.36)
food = data.frame(matrix(food,10,2))
names(food) = c("time","pH")
```

Looking at it:

```
print(food)
```

```
##    time   pH
## 1     1 7.02
## 2     1 6.93
## 3     2 6.42
## 4     2 6.51
## 5     4 6.07
## 6     4 5.99
## 7     6 5.59
## 8     6 5.80
## 9     8 5.51
## 10    8 5.36
```

We can work out regression coefficients manually (from Lecture 2 slides):

```
m.time = mean(food$time)   # Average time
m.pH = mean(food$pH)       # Average pH

s.time = sum( (food$time-m.time)^2 )   # S_X for time
s.timepH = sum( (food$time-m.time)*(food$pH-m.pH) )  # S_XY

beta1 =   s.timepH/s.time        # slope
beta0 =   m.pH - beta1*m.time    #intercept
```

Alternatively we simply call lm:

```
food.mod = lm(pH~time,data=food)
food.mod$coefficients
```

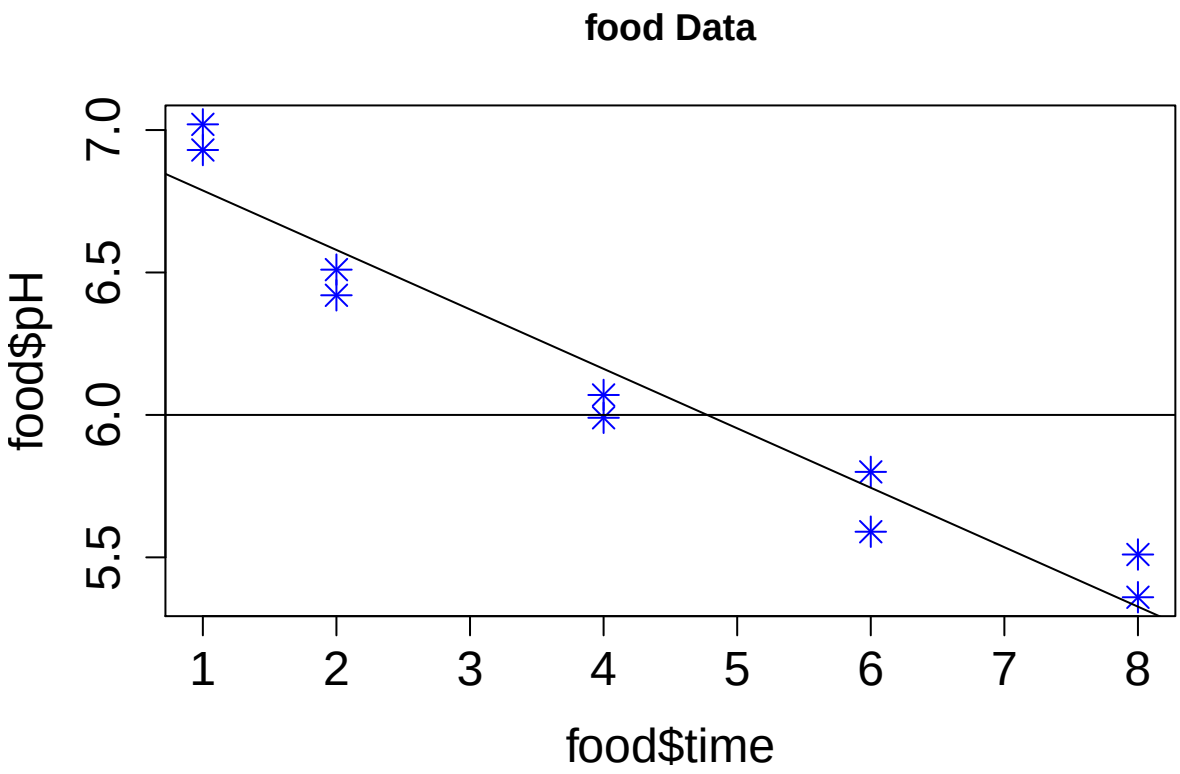
```
## (Intercept)        time
##    6.996494   -0.208689
```

We are interested in the value where the slope rosses pH = 6

```
t = (6 - beta0)/beta1
```

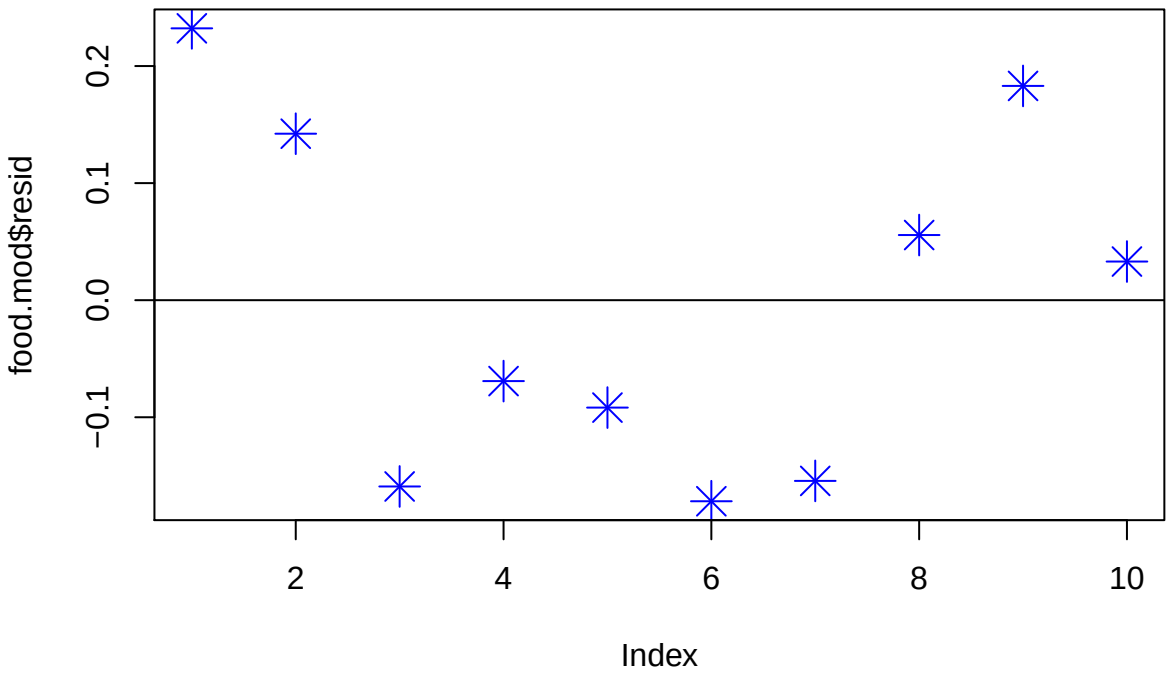
Now plot the data and the regression

```
plot(food$time,food$pH,cex.lab=1.5,cex.axis=1.5,col=4,main='food Data',cex=1.5,pch=8)
abline(food.mod)
abline(h=6)
```



We can also plot residuals

```
plot(food.mod$resid,pch=8,cex=2,col='blue')
abline(h = 0)
```



And we can look at the summary of the fitted model

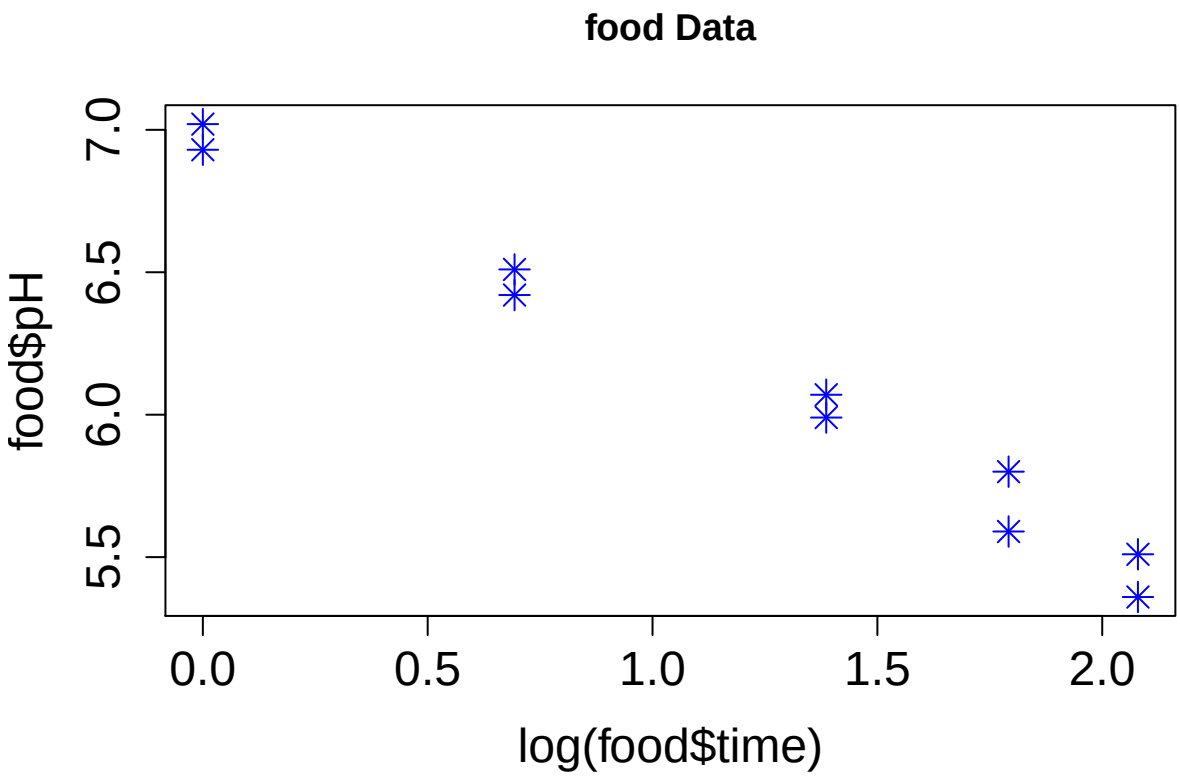
```
summary(food.mod)
```

```
##
## Call:
## lm(formula = pH ~ time, data = food)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.17174 -0.13870 -0.01805  0.12056  0.23220
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.99649    0.09691   72.20 1.51e-12 ***
## time        -0.20869    0.01970  -10.59 5.51e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1595 on 8 degrees of freedom
## Multiple R-squared:  0.9335, Adjusted R-squared:  0.9251
## F-statistic: 112.2 on 1 and 8 DF,  p-value: 5.509e-06
```

Fitting to $log(X)$ instead

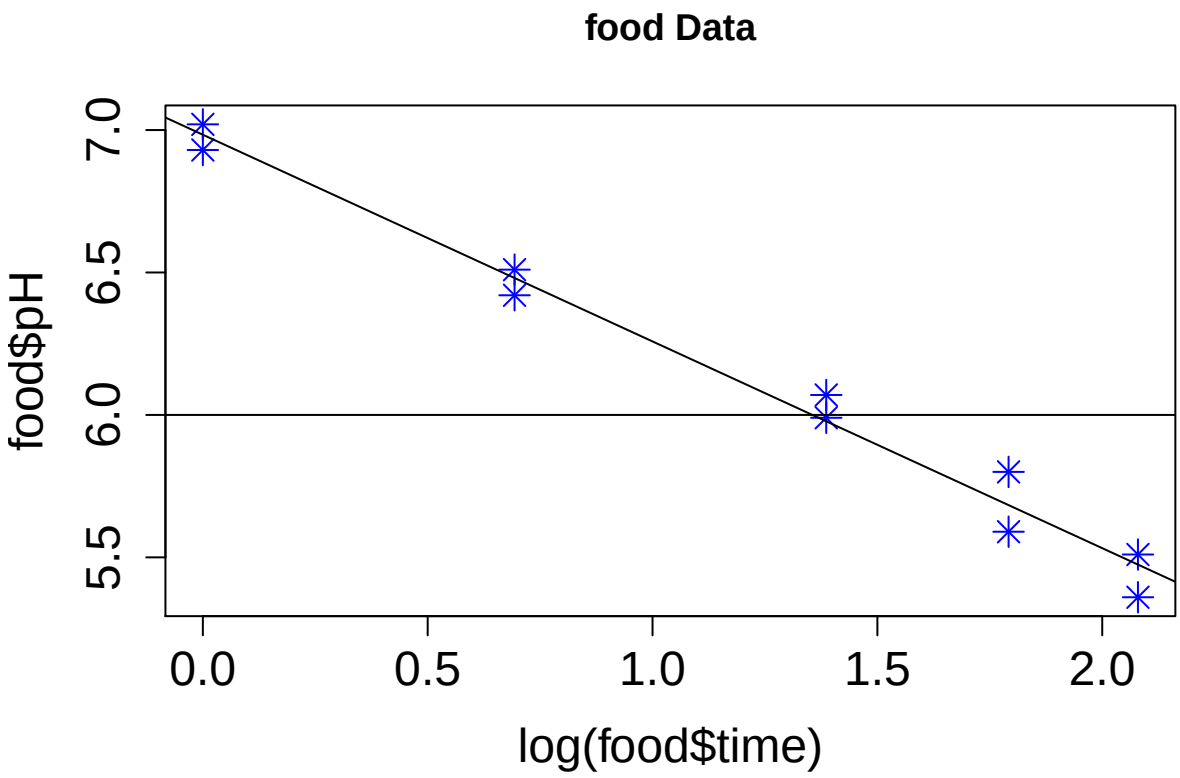
First a plot of Y against log(X)

```
plot(log(food$time),food$pH,cex.lab=1.5,cex.axis=1.5,col=4,main='food Data',cex=1.5,pch=8)
```



Estimate this model

```
food.mod2 = lm(pH~log(time),data=food)

plot(log(food$time),food$pH,cex.lab=1.5,cex.axis=1.5,col=4,main='food Data',cex=1.5,pch=8)
abline(food.mod2)
abline(h=6)
```

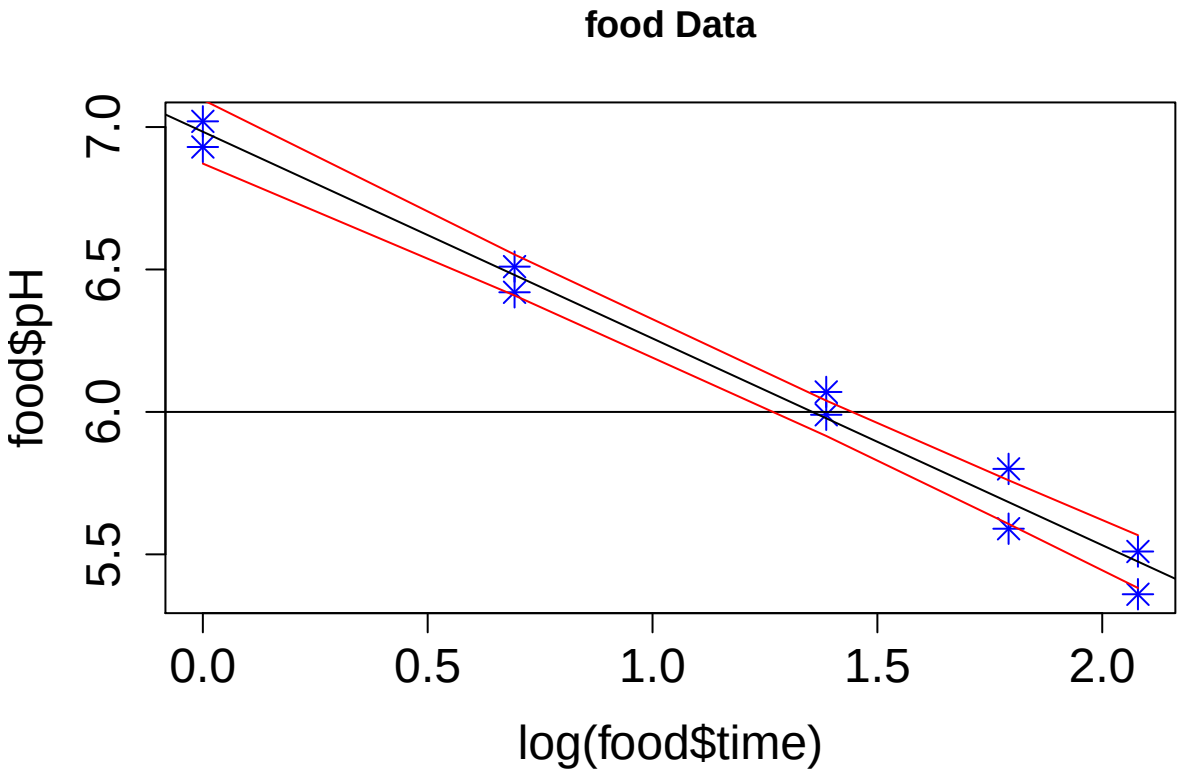


We can obtained confidence intervals at the time points in the data set from the predict function:

```
CIvals = predict(food.mod2,food,interval='confidence')
```

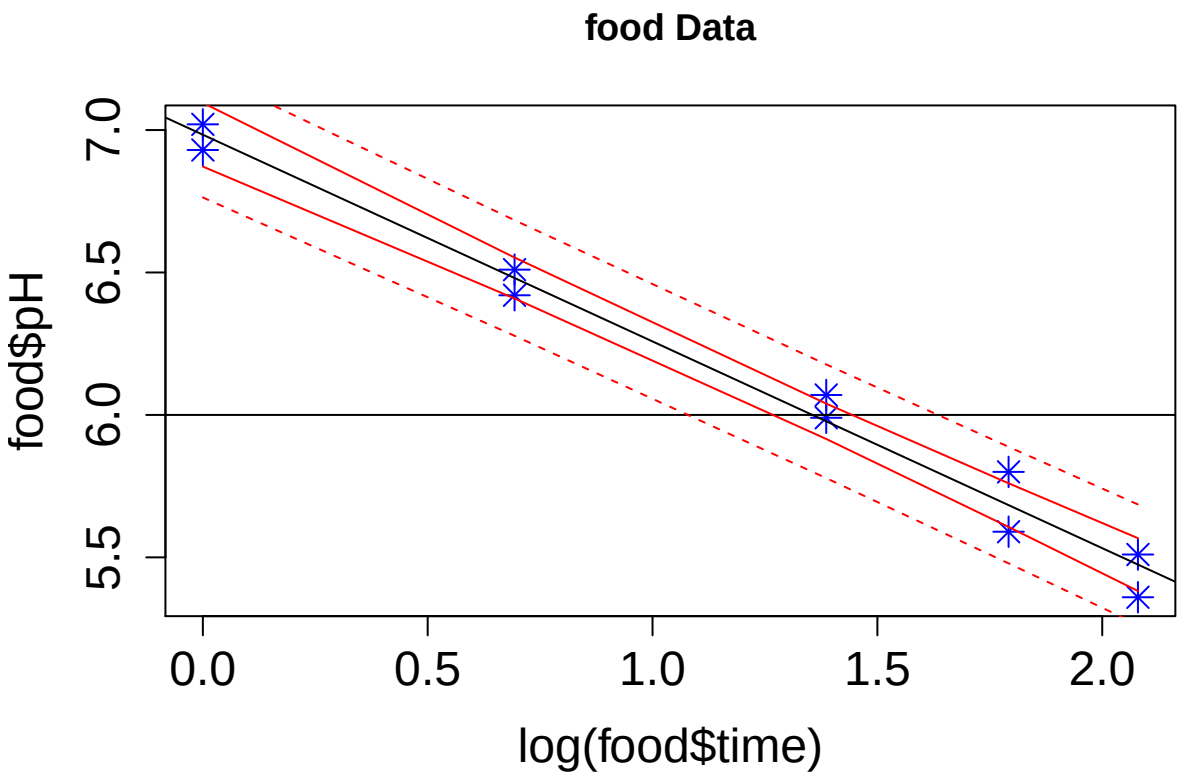
Note that CIvals has three colums – the fit, and the lower and upper values for the confidence intervals. Plot these

```
plot(log(food$time),food$pH,cex.lab=1.5,cex.axis=1.5,col=4,main='food Data',cex=1.5,pch=8)
abline(food.mod2)
abline(h=6)
lines(log(food$time),CIvals[,2],col='red')
lines(log(food$time),CIvals[,3],col='red')
```



We can also use predict to obtain prediction intervals and add these to the plot

```
plot(log(food$time),food$pH,cex.lab=1.5,cex.axis=1.5,col=4,main='food Data',cex=1.5,pch=8)
abline(food.mod2)
abline(h=6)
lines(log(food$time),CIvals[,2],col='red')
lines(log(food$time),CIvals[,3],col='red')
PIvals = predict(food.mod2,food,interval='prediction')
lines(log(food$time),PIvals[,2],lty=2,col='red')
lines(log(food$time),PIvals[,3],lty=2,col='red')
```


food Data
food$pH
7.0
6.5
6.0
5.5
0.0
0.5
1.0
1.5
2.0
log(food$time)