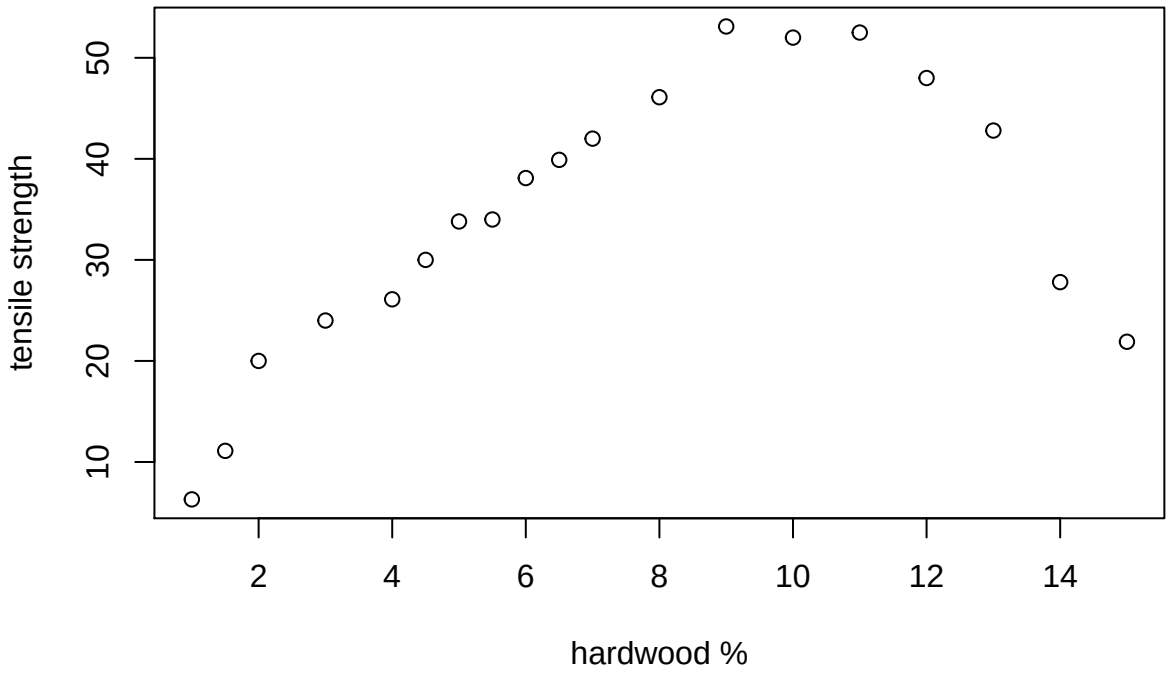
Introduction to Linear Regression

The linear regression model

The following data consists of of pairs $(x_i, y_i), i = 1, 2, \ldots, n$, where x is the proportion of hardwood in a piece of lumber, and y is a measure of tensile strength. A plot of x vs y indicates that the relationship is non-linear.

```
rm(list=ls())
x = c(1,1.5,2,3,4,4.5,5,5.5,6,6.5,7,8,9,10,11,12,13,14,15)
y = c(6.3,11.1,20,24,26.1,30,33.8,34.0,38.1,39.9,42,46.1,53.1,52,52.5,48,42.8,27.8,21.9)
n=length(y)
plot(x,y,xlab="hardwood %",ylab="tensile strength")
```



We would like to use x to predict y.

- Let's start with a **simple linear regression** model

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

- We estimate the intercept β_0 and slope β_1, using the method of **least squares**.
- The least squares estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ are used to get **predicted values** for the outcome variable y, as

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$$

- The difference between the observed and predicted y's are called **residuals**.
- The i'th residual is

$$e_i = y_i - \hat{y}_i$$

- the *sum of squared residuals*, also called the *error sum of squares* is

$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

In R the calculations are as follows:

```
lm.out=lm(y~x)
coef(lm.out) #least squares estimates
```

```
## (Intercept)           x
##   21.321262    1.770986
```

```
ypred=predict(lm.out,x=x) #predicted values
resids=y-ypred  #residuals
SSE=sum(resids^2) #error sum of squares
print(paste("Error sum of squares is ",SSE))
```

```
## [1] "Error sum of squares is  2373.45782945736"
```

```
lm(y~x,subset=sample(1:length(y),length(y),replace=T))
```

```
##
## Call:
## lm(formula = y ~ x, subset = sample(1:length(y), length(y), replace = T))
##
## Coefficients:
## (Intercept)            x
##       20.09         2.32
```

```
lm(y~x,subset=sample(1:length(y),length(y),replace=T))
```

```
##
## Call:
## lm(formula = y ~ x, subset = sample(1:length(y), length(y), replace = T))
##
## Coefficients:
## (Intercept)            x
##      10.220        4.162
```

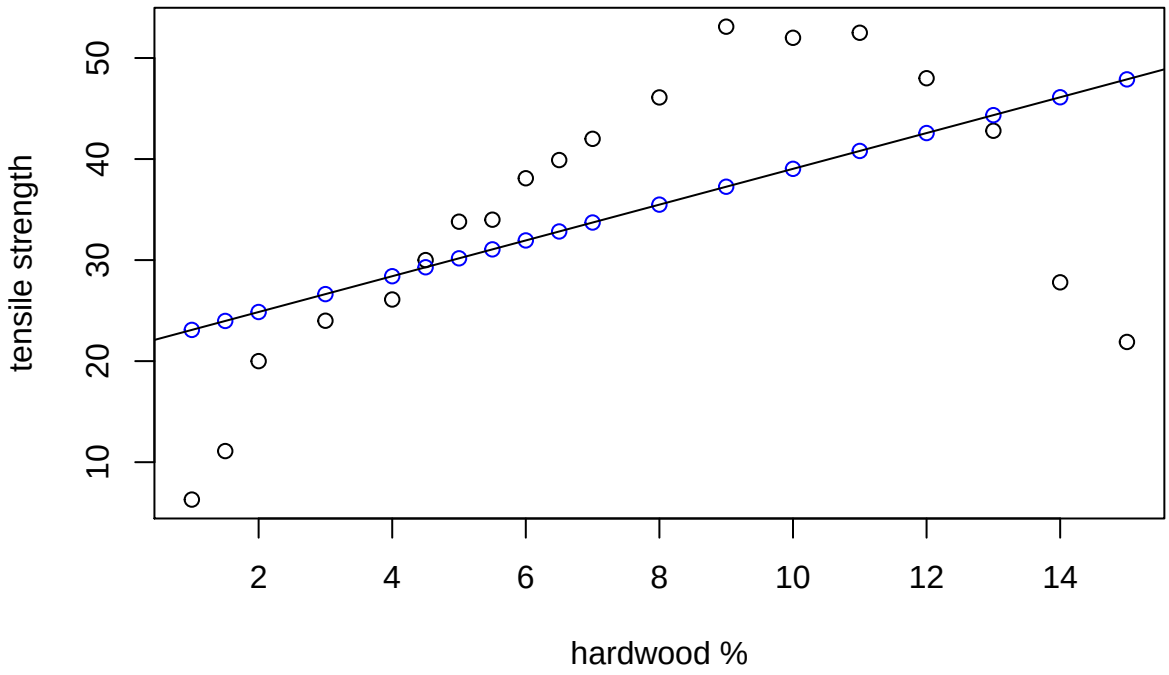
You can extract individual least squares estimates using **coef**. For example, to get back the least squares estimate of the slope β_1, use, say,

```
beta1hat=coef(lm.out)[2]
beta1hat
```

```
##        x
## 1.770986
```

Let's plot the data again, and add the least squares line.

```
plot(x,y,xlab="hardwood %",ylab="tensile strength")
points(x,ypred,col="Blue")
abline(lm.out)
```



The linear regression doesn't provide very good predictions, because the pattern in the data is clearly non-linear. We can improve the prediction by fitting a higher order polynomial function of x.

Now consider a quadratic model:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \epsilon_i$$

Let's plot the fits of both the linear and the quadratic model, together with the data points.

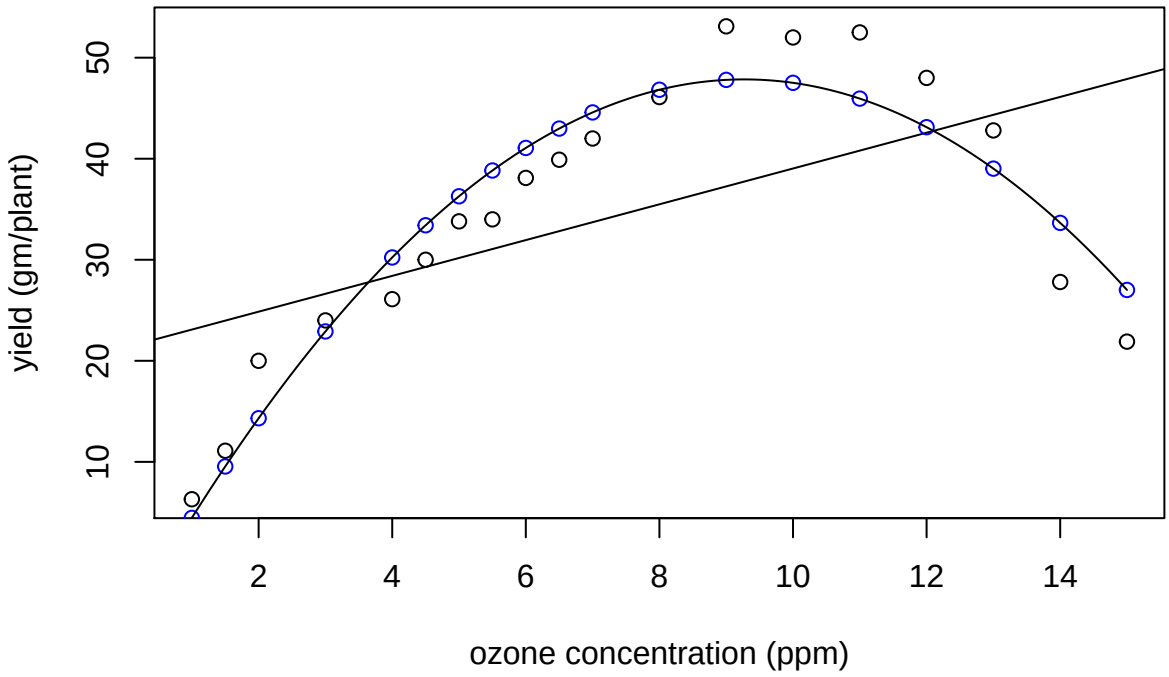
```
x2=x^2
lm2.out=lm(y~x+x2)
coef(lm2.out) #least squares estimates
```

```
## (Intercept)           x          x2
##  -6.6741916  11.7640057  -0.6345492
```

```
xplot=seq(min(x),max(x),length.out=300)
fits2=predict(lm2.out,data.frame(x=xplot,x2=xplot^2))
ypred2=predict(lm2.out) #predicted values
resids2=y-ypred2  #residuals
SSE2=sum(resids2^2) #error some of squares
print(paste("Error sum of squares of quadratic model is ",SSE2))
```

```
## [1] "Error sum of squares of quadratic model is  312.63828841154"
```

```
plot(x,y,xlab="ozone concentration (ppm)",ylab="yield (gm/plant)", main=" linear and quadratic least squ
abline(lm.out)
points(x,ypred2,col="Blue")
lines(list(x=xplot,y=fits2))
```



Predictions with the quadratic model are much better than for the linear model and so the error sum of squares for the quadratic model is dramatically reduced as compared to the linear model. Let's see if a cubic model is better still. The cubic model is:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \epsilon_i$$

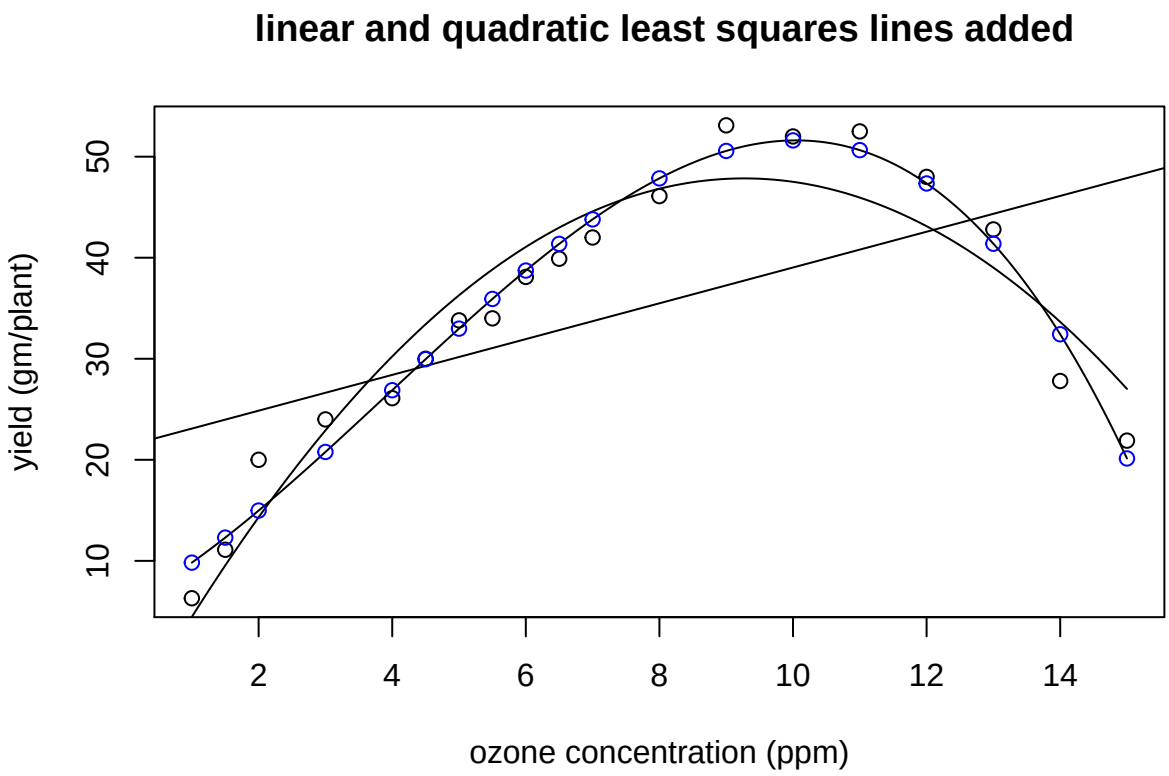
```
x3=x^3
lm3.out=lm(y~x+x2+x3)
coef(lm3.out) #least squares estimates
```

```
## (Intercept)           x          x2          x3
##   5.6483950   3.5784894   0.6536355  -0.0551876
```

```
xplot=seq(min(x),max(x),length.out=300)
fits3=predict(lm3.out,data.frame(x=xplot,x2=xplot^2,x3=xplot^3))
ypred3=predict(lm3.out) #predicted values
resids3=y-ypred3  #residuals
SSE3=sum(resids3^2) #error some of squares
print(paste("Error sum of squares of cubic model is ",SSE3))
```

```
## [1] "Error sum of squares of cubic model is  100.236905126369"
```

```
plot(x,y,xlab="ozone concentration (ppm)",ylab="yield (gm/plant)", main=" linear and quadratic least squ
abline(lm.out)
lines(list(x=xplot,y=fits2))
lines(list(x=xplot,y=fits3))
points(x,ypred3,col="Blue")
```



- The p'th order polynomial regression with a single predictor variable x is given by

$$y_i = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \ldots + \beta_p x^p + \epsilon_i$$

We could fit the p'th order model in the same manner.

Bootstrapping the standard error of the slope.

There is a builtin procedure "boot" which can be used to estimate the standard error of any statistic. In order to use it, you need to create a function which returns the statistic you're interesedt in. For the regression slope,

```
b1=function(data,index){
  x=data$x[index]
  y=data$y[index]
  lmout=lm(y~x)
  b1=coef(lmout)[2]
  return(b1)}
```

"index" is a subset of 1:n which will be passed in by the boot command.

```
library(boot)
data=data.frame(x=x,y=y)
boot(data,b1,R=1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = data, statistic = b1, R = 1000)
```

```
##
##
## Bootstrap Statistics :
##     original    bias    std. error
## t1* 1.770986 0.1327098   0.9612162
```

The original estimate ($\hat{\beta}_1$), the standard error (estimated variance of $\hat{\beta}_1$) and something called the bias of $\hat{\beta}_1$ are returned.