About Regression Models

Tiago A. Marques

oday

Introduction

This document is an attempt to provide a coherent framework for some of the examples about regression models that were presented in the "Modelação Ecológica" course on the 1st, 8th and 9th of October 2019. It was created for those students, but if you are reading it and your are not one of those students that's fine too.

We started looking into the simplest of models, a standard linear regression with a Gaussian error structure

$$y_i = a + b_x + e_i$$

where the assumption is that the e_i are Gaussian independent random deviates with a constant variance $sigma^2$.

Simulating regression data

This was part of what we did in class 6, on the 01 10 2019.

Simulate model

It was suggested that you would simulate data that considered that the TRUE relation between the weight and length of a lizard was given by

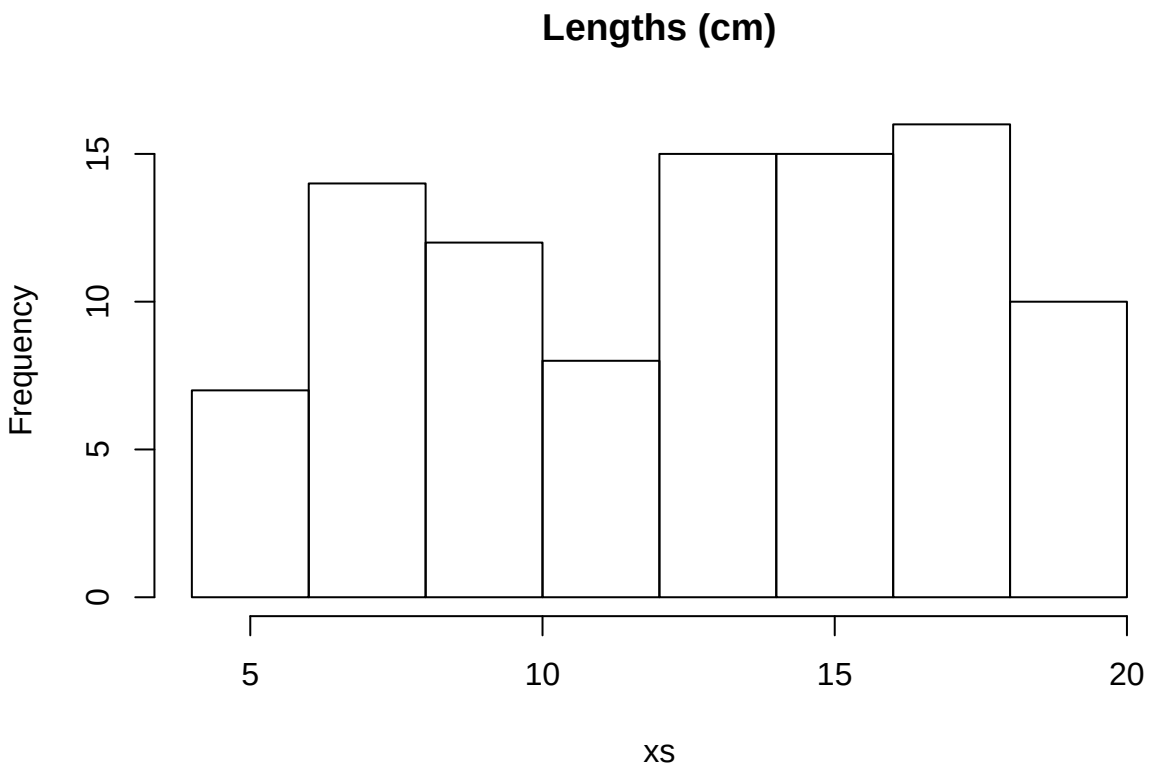
$$y = 12 + 1.2 * length$$

You were also told that the usual length of a lizard was between 5 and 20 cm.

We will have 97 lizards

Then you were told to create the lengths:

```
set.seed(121)
n=97
#lengths
xs=runif(n,5,20)
hist(xs,main="Lengths (cm)")
```

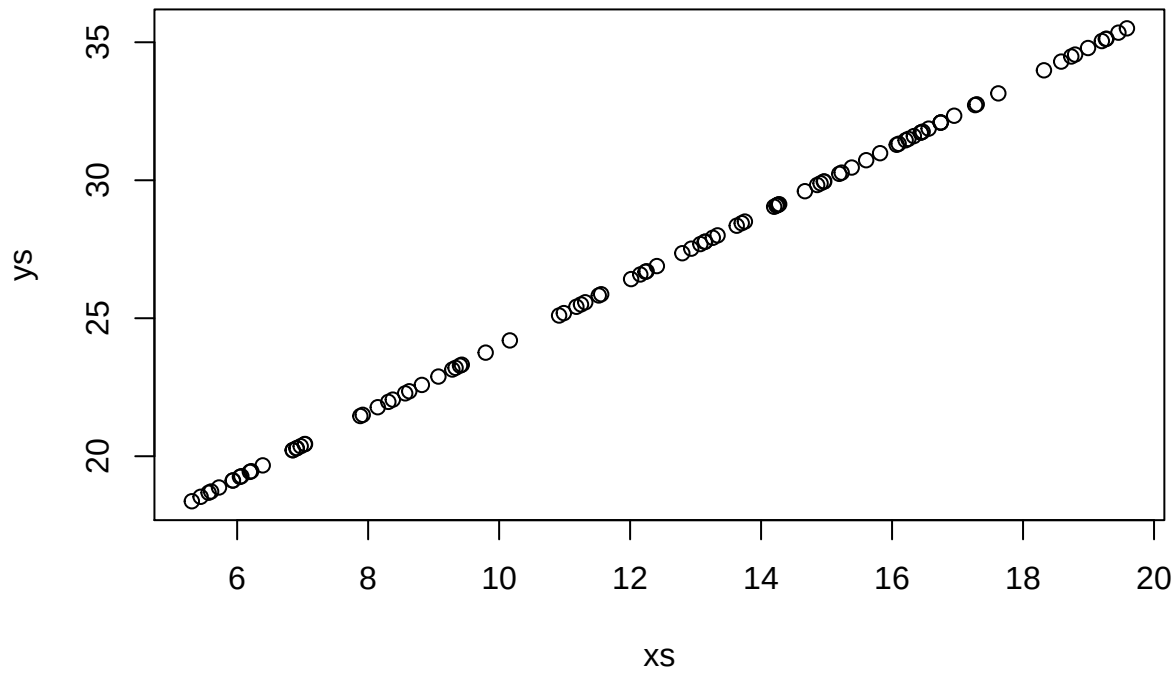


and then to create weights of lizards

```
a=12
b=1.2
ys=a+b*xs
```

If we plot the data, all points are in a single line. Why, because there is no randomness.

```
plot(xs,ys)
```



This means that if you try to run a model, it gives you a warning that the model might be unreliable

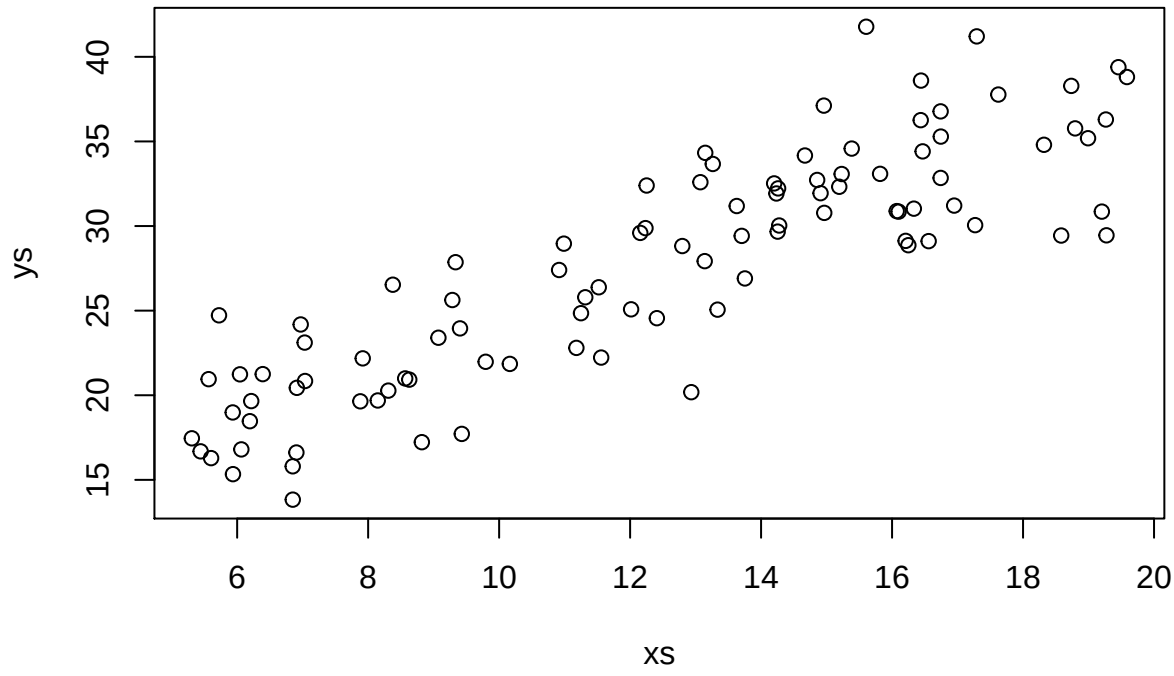
```
summary(lm(ys~xs))
```

```
## Warning in summary.lm(lm(ys ~ xs)): essentially perfect fit: summary may be
## unreliable
```

```
##
## Call:
## lm(formula = ys ~ xs)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -5.595e-15 -2.460e-15 -1.878e-15 -1.422e-15  1.873e-13
##
## Coefficients:
##              Estimate Std. Error   t value Pr(>|t|)
## (Intercept) 1.200e+01  6.050e-15 1.983e+15   <2e-16 ***
## xs          1.200e+00  4.611e-16 2.603e+15   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.934e-14 on 95 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 6.773e+30 on 1 and 95 DF,  p-value: < 2.2e-16
```

So... , we add some variance, and plot the data:

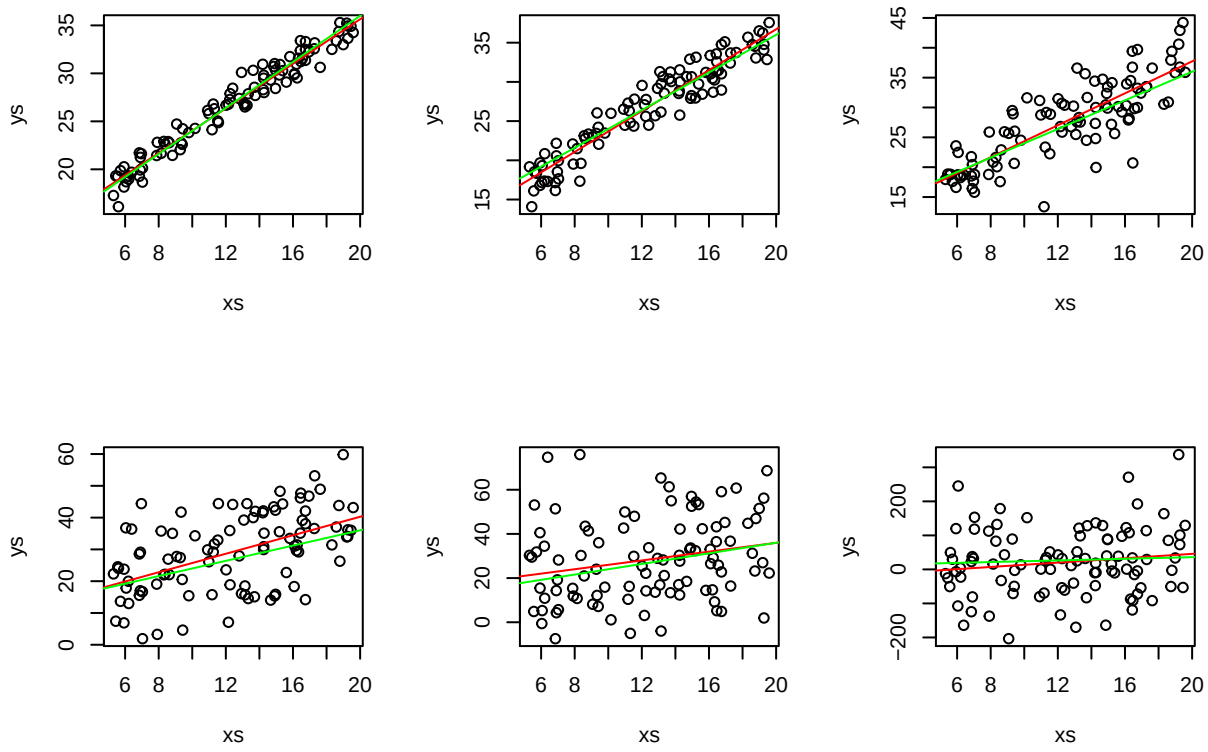
```
ys=a+b*xs+rnorm(n,0,4)
plot(xs,ys)
```



Now, let's consider there's more and less variance. We also add to each plot the real line (that with the true parameter values) and the one with the estimated parameter values.

```
par(mfrow=c(2,3))
ys=a+b*xs+rnorm(n,0,1)
plot(xs,ys)
mod1=lm(ys~xs)
abline(mod1,col="red")
abline(a,b,col="green")
ys=a+b*xs+rnorm(n,0,2)
plot(xs,ys)
mod2=lm(ys~xs)
abline(mod2,col="red")
abline(a,b,col="green")
ys=a+b*xs+rnorm(n,0,4)
plot(xs,ys)
mod4=lm(ys~xs)
abline(mod4,col="red")
abline(a,b,col="green")
ys=a+b*xs+rnorm(n,0,10)
plot(xs,ys)
mod10=lm(ys~xs)
abline(mod10,col="red")
```

```
abline(a,b,col="green")
ys=a+b*xs+rnorm(n,0,20)
plot(xs,ys)
mod20=lm(ys~xs)
abline(mod20,col="red")
abline(a,b,col="green")
ys=a+b*xs+rnorm(n,0,100)
plot(xs,ys)
mod100=lm(ys~xs)
abline(mod100,col="red")
abline(a,b,col="green")
```



Not surprisingly, as the variance increases, we get data that more and more looks like it is not coming from a real linear process.

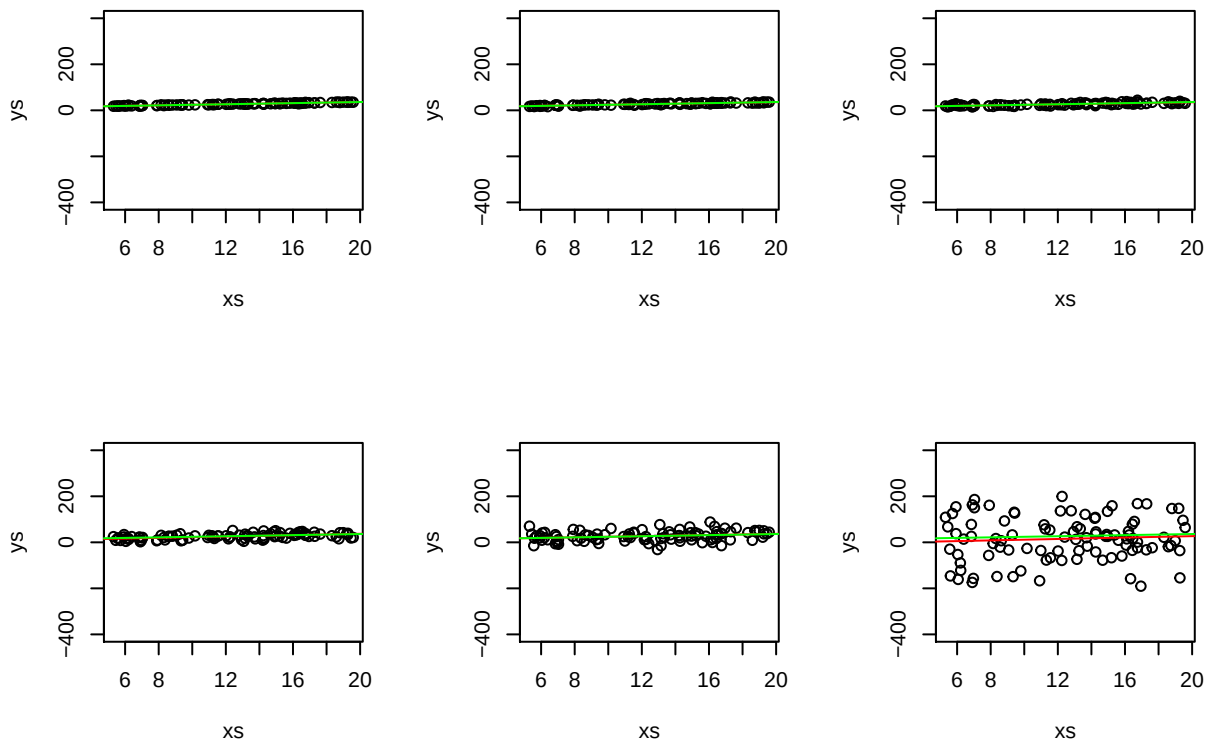
You can also look at the model summaries, and there you can see that, in fact, the models become essentially useless as the variance increases! You can see that both from the correlation, but also by the predictions generated from the model (comparing to the truth), and also the significance of the coefficients associated with the regression parameters.

Make no mistake, the reality is always the same, in terms of the fixed part of the model, it's just the variance that increases.

Also, don't get confused, the different green lines might look different, but they are always exactly the same line! You can check that by forcing the y axis to span the same limits.

```
par(mfrow=c(2,3))
ys=a+b*xs+rnorm(n,0,1)
```

```r
plot(xs,ys,ylim=c(-400,400))
mod1=lm(ys~xs)
abline(mod1,col="red")
abline(a,b,col="green")
ys=a+b*xs+rnorm(n,0,2)
plot(xs,ys,ylim=c(-400,400))
mod2=lm(ys~xs)
abline(mod2,col="red")
abline(a,b,col="green")
ys=a+b*xs+rnorm(n,0,4)
plot(xs,ys,ylim=c(-400,400))
mod4=lm(ys~xs)
abline(mod4,col="red")
abline(a,b,col="green")
ys=a+b*xs+rnorm(n,0,10)
plot(xs,ys,ylim=c(-400,400))
mod10=lm(ys~xs)
abline(mod10,col="red")
abline(a,b,col="green")
ys=a+b*xs+rnorm(n,0,20)
plot(xs,ys,ylim=c(-400,400))
mod20=lm(ys~xs)
abline(mod20,col="red")
abline(a,b,col="green")
ys=a+b*xs+rnorm(n,0,100)
plot(xs,ys,ylim=c(-400,400))
mod100=lm(ys~xs)
abline(mod100,col="red")
abline(a,b,col="green")
```



but since then you loose all the ability to look at the actual data in some of the plots, that is not really that useful!

Below I look at the summary of each model. Look at correlations, at the estimated values for the parameters, their corresponding variances and the R^2.

```
summary(mod1)
```

```
##
## Call:
## lm(formula = ys ~ xs)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.2297 -0.5755 -0.0093  0.5758  3.1122
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 12.13296    0.31728   38.24   <2e-16 ***
## xs           1.18951    0.02418   49.19   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.014 on 95 degrees of freedom
## Multiple R-squared:  0.9622, Adjusted R-squared:  0.9618
## F-statistic:  2420 on 1 and 95 DF,  p-value: < 2.2e-16
```

```r
summary(mod2)
```

```
##
## Call:
## lm(formula = ys ~ xs)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.0496 -1.0818  0.0395  1.3671  4.7968
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 12.78179    0.57907   22.07   <2e-16 ***
## xs           1.14051    0.04413   25.84   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.851 on 95 degrees of freedom
## Multiple R-squared:  0.8755, Adjusted R-squared:  0.8742
## F-statistic: 667.9 on 1 and 95 DF,  p-value: < 2.2e-16
```

```r
summary(mod4)
```

```
##
## Call:
## lm(formula = ys ~ xs)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.3229 -2.7291 -0.4323  2.3578 12.5807
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 12.47511    1.14862   10.86   <2e-16 ***
## xs           1.12565    0.08754   12.86   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.671 on 95 degrees of freedom
## Multiple R-squared:  0.6351, Adjusted R-squared:  0.6313
## F-statistic: 165.4 on 1 and 95 DF,  p-value: < 2.2e-16
```

```r
summary(mod10)
```

```
##
## Call:
## lm(formula = ys ~ xs)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -22.332  -7.700   1.157   6.395  25.753
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   7.4371     3.1514   2.360   0.0203 *
```

```
## xs            1.4793     0.2402   6.159 1.75e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.07 on 95 degrees of freedom
## Multiple R-squared:  0.2854, Adjusted R-squared:  0.2779
## F-statistic: 37.94 on 1 and 95 DF,  p-value: 1.746e-08
```

```
summary(mod20)
```

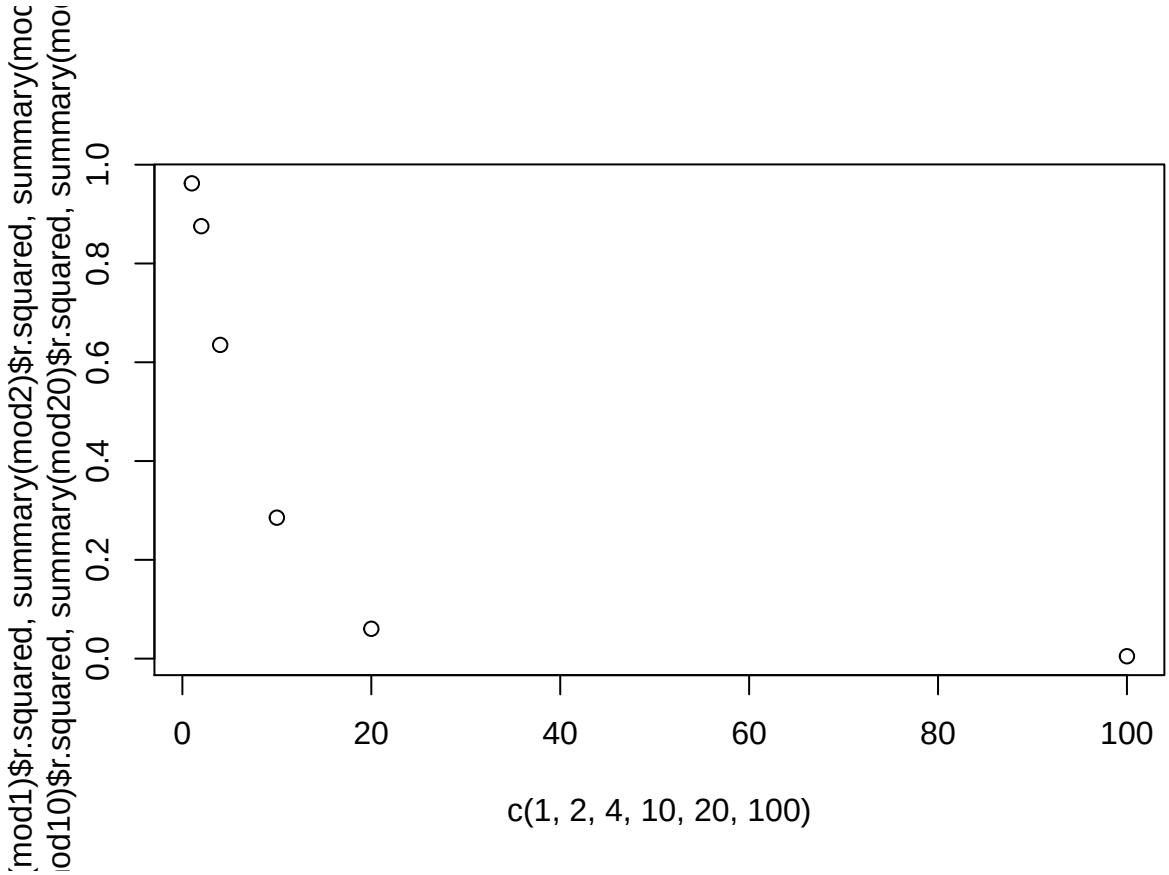
```
##
## Call:
## lm(formula = ys ~ xs)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -59.36 -14.88   0.09  12.18  56.18
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  11.6794     6.6835   1.747   0.0838 .
## xs            1.2592     0.5094   2.472   0.0152 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 21.36 on 95 degrees of freedom
## Multiple R-squared:  0.06044,    Adjusted R-squared:  0.05055
## F-statistic: 6.111 on 1 and 95 DF,  p-value: 0.01521
```

```
summary(mod100)
```

```
##
## Call:
## lm(formula = ys ~ xs)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -212.693  -56.234    2.761   69.244  184.494
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -4.037     29.856  -0.135    0.893
## xs             1.545      2.275   0.679    0.499
##
## Residual standard error: 95.42 on 95 degrees of freedom
## Multiple R-squared:  0.004829,   Adjusted R-squared:  -0.005646
## F-statistic: 0.461 on 1 and 95 DF,  p-value: 0.4988
```

As an example, we can plot the R^2 as a function of the variance

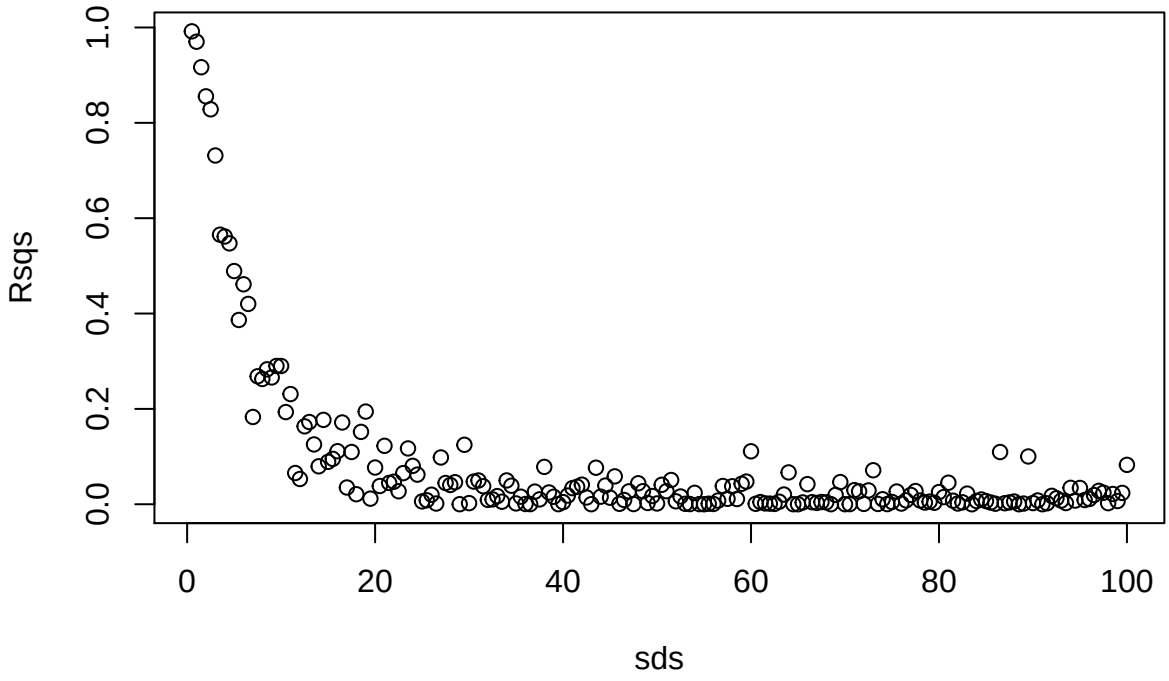
```
plot(c(1,2,4,10,20,100),c(summary(mod1)$r.squared,summary(mod2)$r.squared,summary(mod4)$r.squared,summa
```



That is quite interesting actually... There seems to be a nonlinear relationship, but we only have a sample size of six (different standard deviations, i.e., variances, as variance is standard deviation squared), so hard to tell...

Let's show off in R...

```
sds=seq(0.5,100,by=0.5)
nsds=length(sds)
#an object to hold the correlations
Rsqs=numeric(nsds)
for (i in 1:nsds){
  #create data
  ys=a+b*xs+rnorm(n,0,sds[i])
  #estimate model
  modi=lm(ys~xs)
  #get R-squared
  Rsqs[i]=summary(modi)$r.squared
}
#and at the end... plot results
plot(sds,Rsqs)
```



How cool is that!! Actually, this means we can model the R^2 as a function of the original variance! But we would not be able to model it using a linear model...

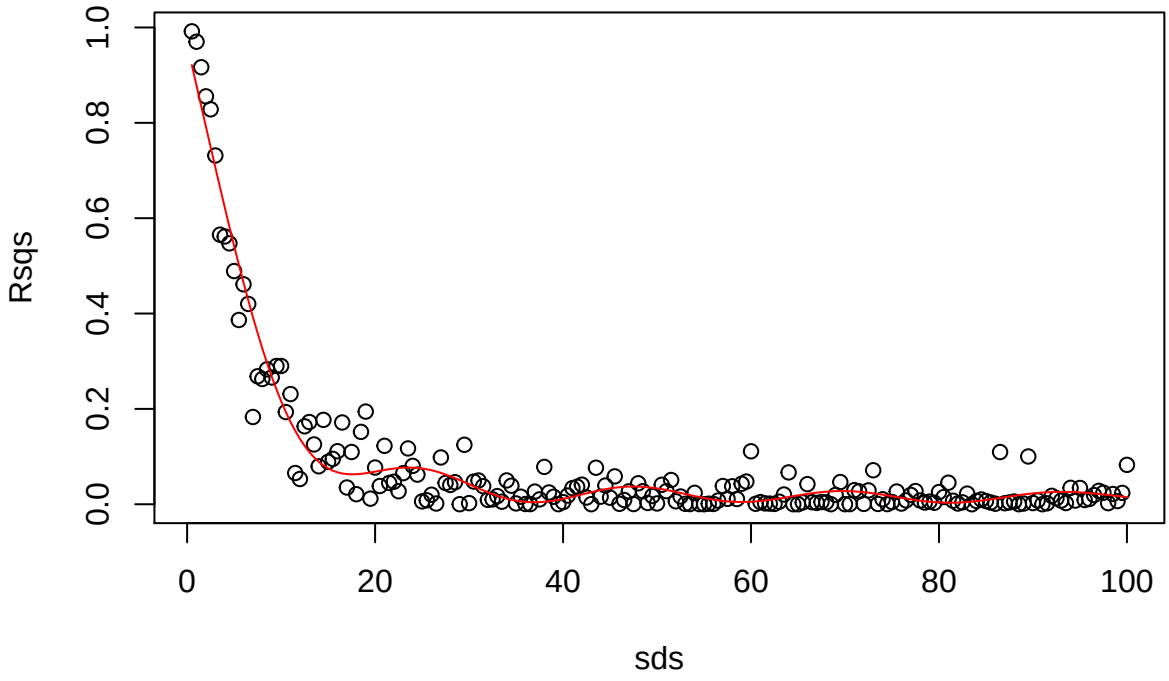
You are not supposed to know about this yet, but I'll continue to show off. Let's use a GAM

```
library(mgcv)
```

```
## Loading required package: nlme
```

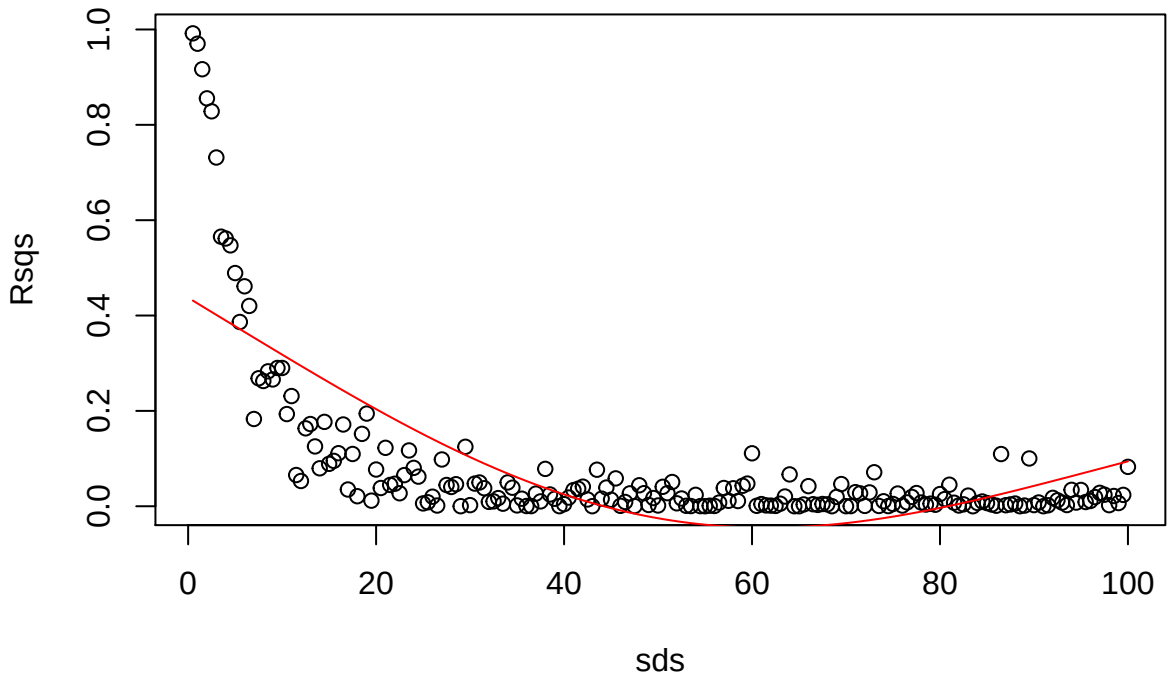
```
## This is mgcv 1.8-28. For overview type 'help("mgcv-package")'.
```

```
gam1=gam(Rsqs~s(sds),link=log)
#make predictions to plot the estimated GAM model
predRsqs=predict.gam(gam1,newdata = list(sds=sds),type="response")
plot(sds,Rsqs)
lines(sds,predRsqs,col="red")
```



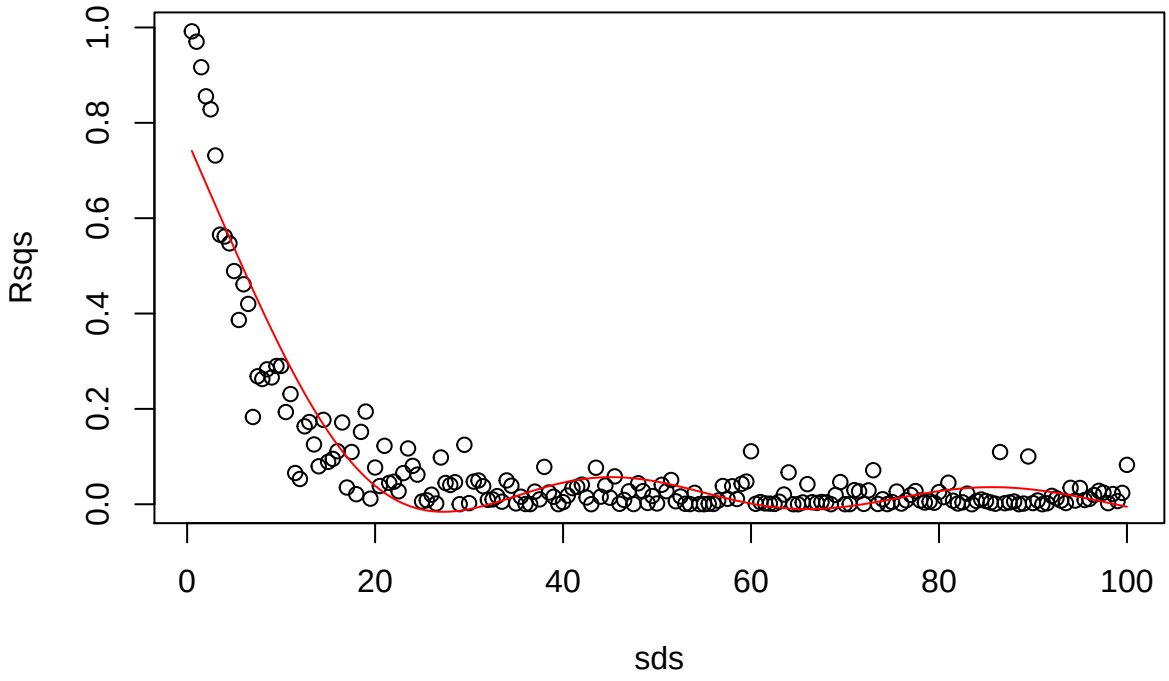
Aha... remember what we talked in class today? It seems like we have over-fitted. Then, I constrain the GAM.

```
library(mgcv)
gam1=gam(Rsqs~s(sds,k=3),link=log)
#make predictions to plot the estimated GAM model
predRsqs=predict.gam(gam1,newdata = list(sds=sds),type="response")
plot(sds,Rsqs)
lines(sds,predRsqs,col="red")
```



That was too much...

```
library(mgcv)
gam1=gam(Rsqs~s(sds,k=6),link=log)
#make predictions to plot the estimated GAM model
predRsqs=predict.gam(gam1,newdata = list(sds=sds),type="response")
plot(sds,Rsqs)
lines(sds,predRsqs,col="red")
```



That is already over-fitting... conclusion, the GAM is not the right tool here :)

What is...? Well, stay tuned and one day you'll learn!

Drawing data

This was part of what we did in class 7, on the 08 October 2019. Here the idea was to explore different regression models.

To illustrate the models the students were invited to go to drawdata.xyz and to create a dataset with regression data such that there was an increasing trend, a decreasing trend, no trend, and data that was not really a line. Then the data set should be downloaded, imported into R and regression models fit to it.

Here we have such a dataset, named "data.csv", and we read it in a and look at it

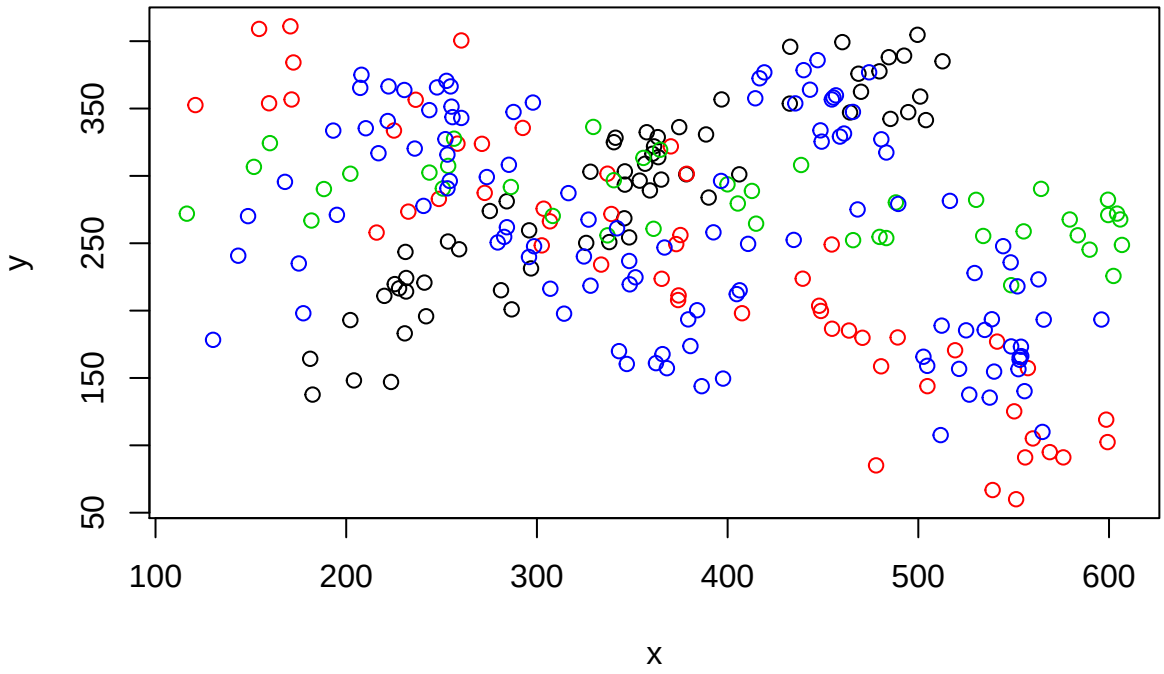
```
#TAM's path
#C:/Users/tam2/Dropbox/Trabalho/DBA/20192020/ME2019/aulas/OnRegressionModels/
#To work on any machine provided the file data.csv exists in said folder
data <- read.csv("data.csv")
summary(data)
```

```
##        x                y         z
##  Min.   :116.4   Min.   : 60.0   a: 61
##  1st Qu.:263.1   1st Qu.:211.5   b: 53
##  Median :367.5   Median :269.3   c: 41
##  Mean   :377.3   Mean   :264.1   d:111
##  3rd Qu.:482.6   3rd Qu.:325.3
##  Max.   :606.8   Max.   :411.0
```

We can see we have a dataset with 3 variables, an x, the independent variable, and y, the response, and the z, which represents the specific data. In my case, a was the increasing slope, b the decreasing slope, c a slope of approximately 0 and d was not a line.

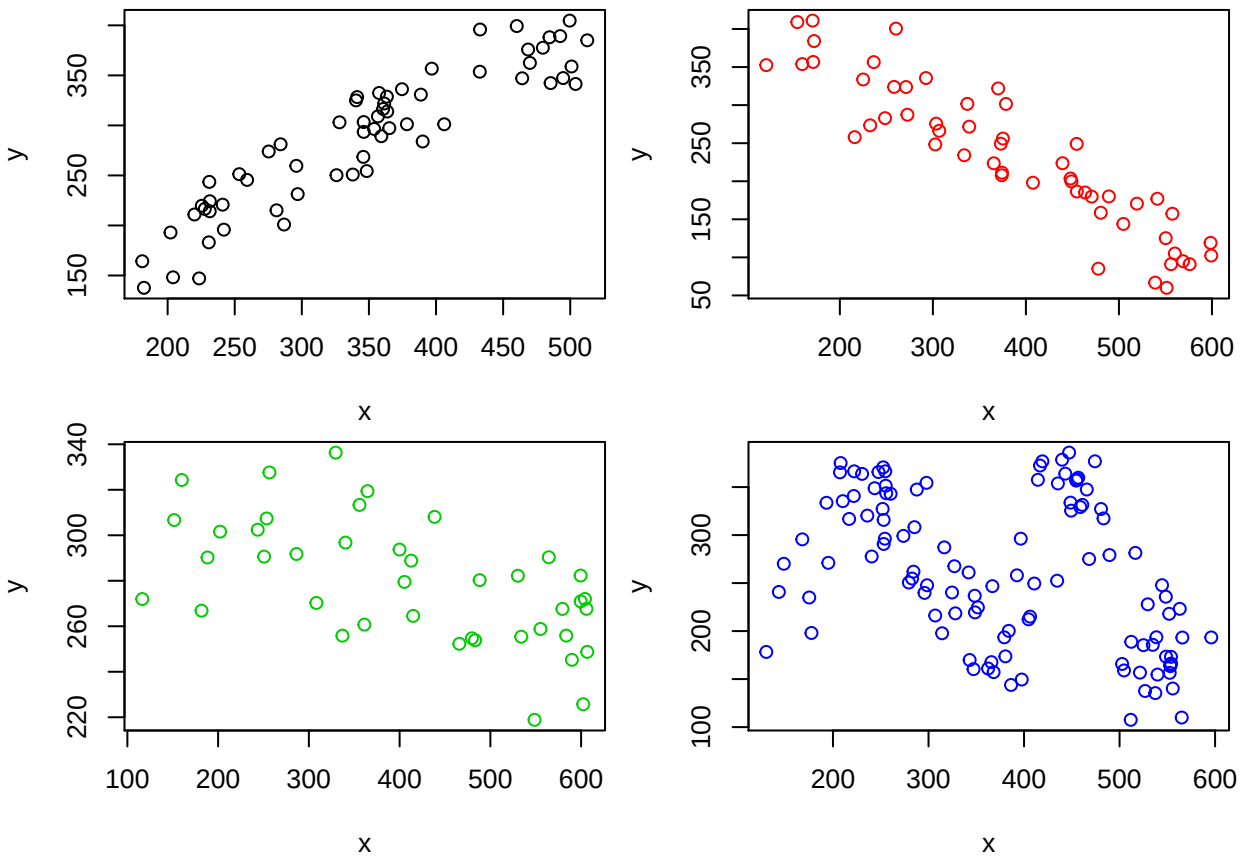
The data looks like this

```
with(data,plot(y~x,col=as.numeric(z)))
```



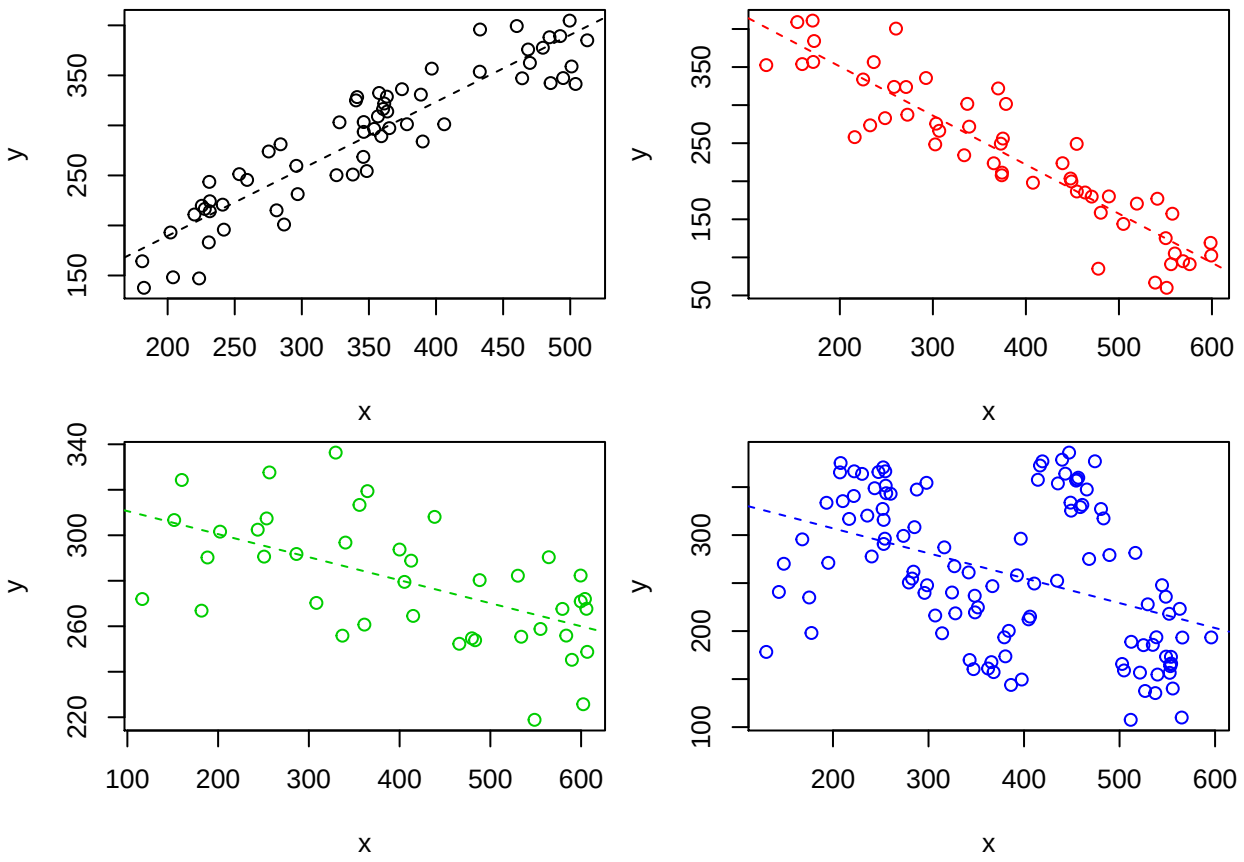
Not surprisingly, it looks quite messy, but if we separate the data by the 4 different subsets things become clearer.

```
par(mfrow=c(2,2),mar=c(4,4,0.5,0.5))
dataA=data[data$z=="a",]
with(dataA,plot(y~x))
dataB=data[data$z=="b",]
with(dataB,plot(y~x,col=2))
dataC=data[data$z=="c",]
with(dataC,plot(y~x,col=3))
dataD=data[data$z=="d",]
with(dataD,plot(y~x,col=4))
```



and of course now we can fit a line to each dataset

```
par(mfrow=c(2,2),mar=c(4,4,0.5,0.5))
with(dataA,plot(y~x))
lm1=with(dataA,lm(y~x))
abline(lm1,col=1,lty=2)
with(dataB,plot(y~x,col=2))
lm2=with(dataB,lm(y~x))
abline(lm2,col=2,lty=2)
with(dataC,plot(y~x,col=3))
lm3=with(dataC,lm(y~x))
abline(lm3,col=3,lty=2)
with(dataD,plot(y~x,col=4))
lm4=with(dataD,lm(y~x))
abline(lm4,col=4,lty=2)
```



and we can look at the output of the models. For the increasing slope

```
summary(lm1)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -58.216 -23.135   3.871  16.324  50.378
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 55.59455   13.20552    4.21 8.84e-05 ***
## x            0.66969    0.03676   18.21  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 27.36 on 59 degrees of freedom
## Multiple R-squared:  0.849,  Adjusted R-squared:  0.8465
## F-statistic: 331.8 on 1 and 59 DF,  p-value: < 2.2e-16
```

we have an intercept estimate of 55.59 (with corresponding standard error of 13.21 and a slope of 0.67 (with corresponding standard error of 0.04. The regression R^2 is 0.849.

Note these objects can be easily manipulated to obtain parameters and statistics of interest. That was how I wrote them dynamically in the text above

```
#intercept
round(summary(lm1)$coefficients[1,1],2)
```

```
## [1] 55.59
```

```
#intercept se
round(summary(lm1)$coefficients[1,2],2)
```

```
## [1] 13.21
```

```
#slope
round(summary(lm1)$coefficients[2,1],2)
```

```
## [1] 0.67
```

```
#slope se
round(summary(lm1)$coefficients[2,2],2)
```

```
## [1] 0.04
```

```
#R-squared
round(summary(lm1)$r.squared,3)
```

```
## [1] 0.849
```

but there's much more info in a regression object. Fee free to explore it

```
names(summary(lm1))
```

```
##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

For the decreasing slope we got

```
summary(lm2)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -86.518 -20.501   0.309  25.235  88.670
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 479.69909   15.72385   30.51   <2e-16 ***
## x            -0.64470    0.03878  -16.62   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 38.24 on 51 degrees of freedom
## Multiple R-squared:  0.8442, Adjusted R-squared:  0.8411
## F-statistic: 276.3 on 1 and 51 DF,  p-value: < 2.2e-16
```

for what was supposed to be a 0 slope (proving I am terrible at drawing!)

```
summary(lm3)
```

```
##
```

```
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -46.326 -15.776   1.289  12.427  48.951
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 320.7516     9.9933  32.097  < 2e-16 ***
## x            -0.1013     0.0229  -4.422 7.61e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 22.2 on 39 degrees of freedom
## Multiple R-squared:  0.3339, Adjusted R-squared:  0.3168
## F-statistic: 19.55 on 1 and 39 DF,  p-value: 7.614e-05
```

and what was not a line at all

```
summary(lm4)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -146.846  -51.075   -6.494   57.647  143.011
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 358.9257    21.7565  16.497  < 2e-16 ***
## x            -0.2595     0.0544  -4.771 5.73e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 71.24 on 109 degrees of freedom
## Multiple R-squared:  0.1727, Adjusted R-squared:  0.1651
## F-statistic: 22.76 on 1 and 109 DF,  p-value: 5.733e-06
```

notice in particular that while this was not a line, the computer is agnostic to it. You are telling it to fit a line to the data, it, will. And interestingly, the regression is highly significant. This should warn you about the dangers of fitting wrong models to data, you can end up with the wrong conclusions.

The students should by now understand every aspect of the outputs above. If you do not, come and talk to me.

ANOVA, ANCOVA and the likes

This was part of what we did in class 8, on the 9th October 2019. Here the idea was to explore different regression models and to see how they relate to statistical procedures one might not associate with a regression, when in fact, they are special cases of a regression.

The t-test

While we did not do the t-test in class, but this is useful because it allows you to see how a simple t-test is just a linear model too, and acts as a building block for the next examples. The t-test allows us to test the null hypothesis that two samples have the same mean.

Create some data

```
#Making up a t-test
#making sure everyone gets the same results
set.seed(980)
```

Then we define the sample size and the number of treatments

```
#define sample size
n=100
#define treatments
tr=c("a","b")
#how many treatments - 2 for a t test
ntr=length(tr)
#balanced design
n.by.tr=n/ntr
```

Now, we can simulate some data. First, the treatments

```
type=as.factor(rep(tr,each=n.by.tr))
cores=rep(1:ntr,each=n.by.tr)
```

Then we define the means by treatment - note that they are different, so the null hypothesis in the t-test, that the mean of a is equal to the mean of b, is known to be false in this case.

```
#define 4 means
ms=c(3,4)
```

Then, the key part, the response variable, with a different mean by treatment. Note the use of the `ifelse` function, which evaluates its first argument and then assigns the value of its second argument if the first is true or the value of the second if its first argument is false. An example

```
ifelse(3>4,55,77)
```

```
## [1] 77
```

```
ifelse(3<4,55,77)
```

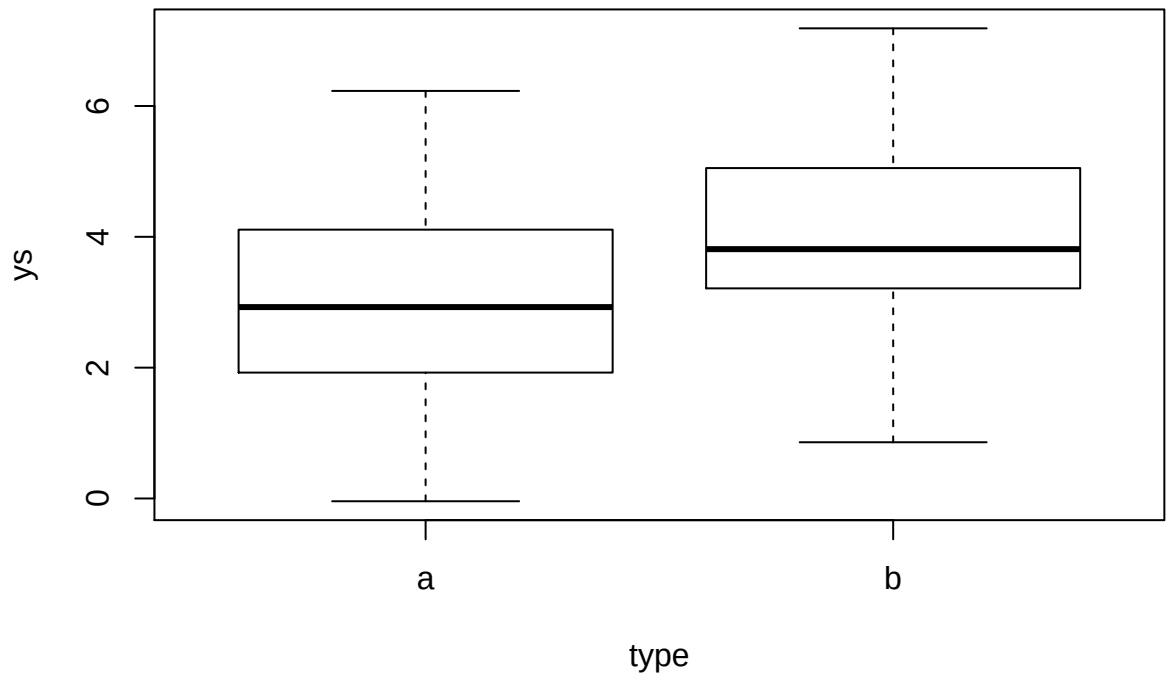
```
## [1] 55
```

So now, generate the response data

```
ys=ifelse(type=="a",ms[1],ms[2])+rnorm(n,0,1.5)
```

Look at the data

```
plot(ys~type)
```



Now, we can run the usual t-test

```
t.test(ys~type)
```

```
##
##  Welch Two Sample t-test
##
## data:  ys by type
## t = -2.8043, df = 97.475, p-value = 0.006087
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.4263293 -0.2441277
## sample estimates:
## mean in group a mean in group b
##        3.106656        3.941884
```

and now we can do it the linear regression way

```
lm0=lm(ys~type)
summary(lm0)
```

```
##
## Call:
## lm(formula = ys ~ type)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.1489 -0.9131 -0.1315  1.0295  3.2450
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.1067     0.2106  14.751  < 2e-16 ***
## typeb         0.8352     0.2978   2.804  0.00608 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.489 on 98 degrees of freedom
## Multiple R-squared:  0.07428,    Adjusted R-squared:  0.06484
## F-statistic: 7.864 on 1 and 98 DF,  p-value: 0.006081
```

and as you can see, we get the same result for the test statistic. It is the same thing! And we can naturally get the estimated means per group. The mean for a is just the intercept of the model. To get the mean of the group b we add the mean of group b to the intercept, as

```
#mean of ys under treatment a
summary(lm0)$coefficients[1]
```

```
## [1] 3.106656
```

```
#mean of ys under treatment b
summary(lm0)$coefficients[1]+lm0$coefficients[2]
```

```
##    typeb
## 3.941884
```

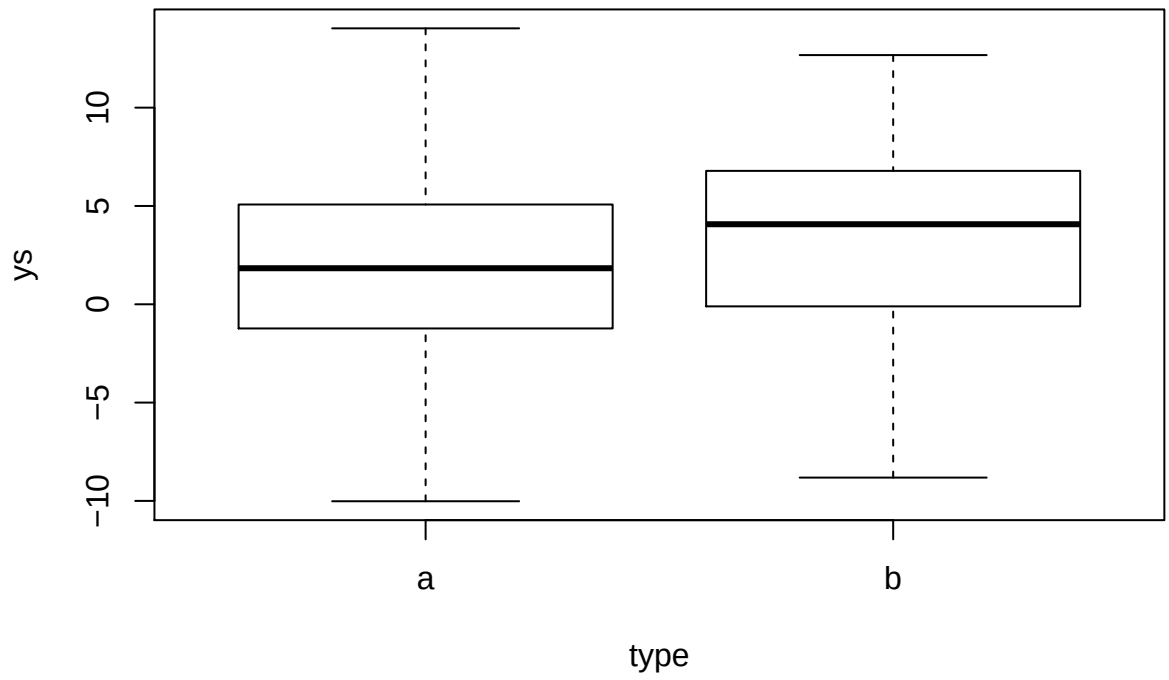
This is required because in a linear model, all the other parameters associated with levels of a factor will be compared to a reference value, that of the intercept, which happens to be the mean under treatment a. Below you will see more examples of this.

Note we were able to detect the null was false, but this was because we had a decent sample size compared to the variance of the measurements and the magnitude of the true effect (the difference of the means). If we keep the sample size constant but we increase the noise or decrease the magnitude of the difference, we might not get the same result, and make a type II error!

```
#define 2 means
ms=c(3,4)
#increase the variance of the process
ys=ifelse(type=="a",ms[1],ms[2])+rnorm(n,0,5)
```

Look at the data, we can see much more variation

```
plot(ys~type)
```



Now, we can run the usual t-test

```
t.test(ys~type)
```

```
##
##  Welch Two Sample t-test
##
## data:  ys by type
## t = -1.3609, df = 97.949, p-value = 0.1767
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.2822693  0.6118174
## sample estimates:
## mean in group a mean in group b
##        2.024963        3.360189
```

and now we can do it the linear regression way

```
lm0=lm(ys~type)
summary(lm0)
```

```
##
## Call:
## lm(formula = ys ~ type)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.1746  -3.2719   0.2527   3.0578  12.0085
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.0250     0.6938   2.919  0.00436 **
## typeb         1.3352     0.9811   1.361  0.17667
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.906 on 98 degrees of freedom
## Multiple R-squared:  0.01855,    Adjusted R-squared:  0.008533
## F-statistic: 1.852 on 1 and 98 DF,  p-value: 0.1767
```

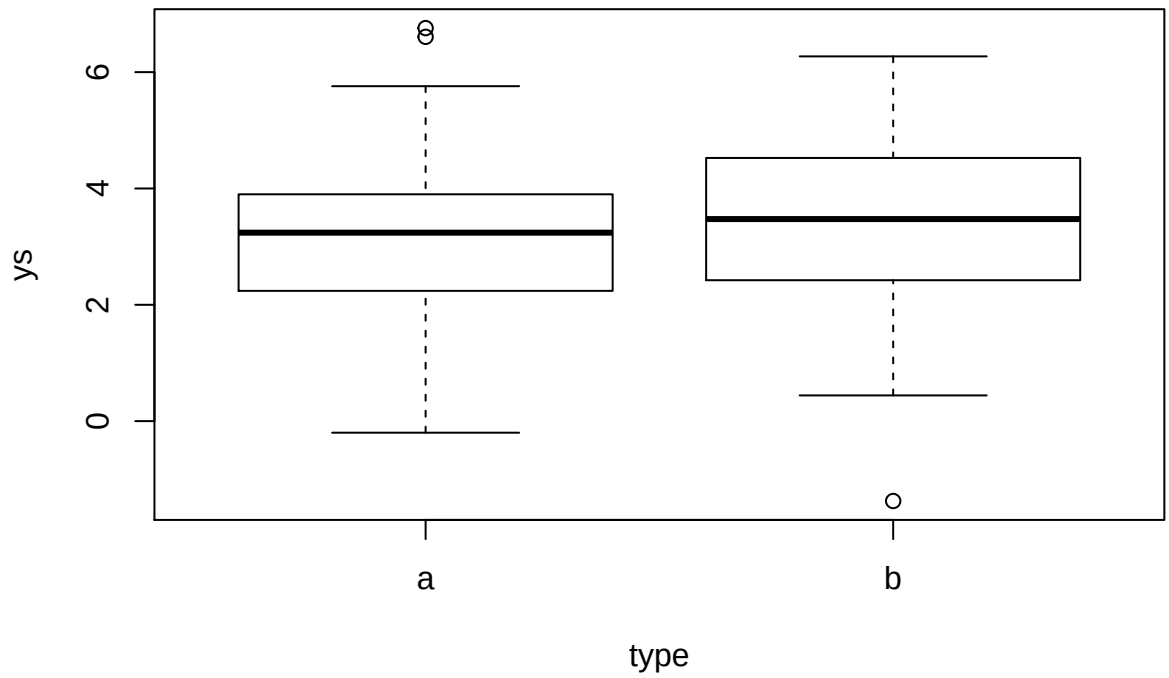
and as you can see, we get the same result for the test statistic, but now with a non significant test.

The same would have happened if we decreased the true difference, while keeping the original magnitude of the error

```
#define 2 means
ms=c(3,3.1)
#increase the variance of the process
ys=ifelse(type=="a",ms[1],ms[2])+rnorm(n,0,1.5)
```

Look at the data, we can see again lower variation, but the difference across treatments is very small (so, hard to detect!)

```
plot(ys~type)
```



Now, we can run the usual t-test

```
t.test(ys~type)
```

```
##
##  Welch Two Sample t-test
##
## data:  ys by type
## t = -0.7994, df = 97.455, p-value = 0.426
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.8149517  0.3469402
## sample estimates:
## mean in group a mean in group b
##        3.158868        3.392874
```

and now we can do it the linear regression way

```
lm0=lm(ys~type)
summary(lm0)
```

```
##
## Call:
## lm(formula = ys ~ type)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.7661 -0.9318  0.0812  0.9087  3.5981
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.1589     0.2070  15.261   <2e-16 ***
## typeb         0.2340     0.2927   0.799    0.426
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.464 on 98 degrees of freedom
## Multiple R-squared:  0.006479,   Adjusted R-squared:  -0.003659
## F-statistic: 0.639 on 1 and 98 DF,  p-value: 0.426
```

ANOVA

We move on with perhaps the most famous example of a statistical test/procedure, the ANOVA. An ANOVA is nothing but a linear model, where we have a continuous response variable, which we want to explain as a function of a factor (with several levels, or treatments).

We simulate a data set, beginning by making sure everyone gets the same results by using `set.seed`

```
#Making up an ANOVA
#An ANOVA
#making sure everyone gets the same results
set.seed(12345)
```

Then we define the sample size and the number of treatments

```
#define sample size
n=2000
#define treatments
tr=c("a","b","c","d")
```

```
#how many treatments
ntr=length(tr)
#balanced design
n.by.tr=n/ntr
```

now, we can simulate some data. First, the treatments, but we also generate a independent variable that is not really used for now (`xs`).

```
#generate data
xs=runif(n,10,20)
type=as.factor(rep(tr,each=n.by.tr))
#if I wanted to recode the levels such that c was the baseline
#type=factor(type,levels = c("c","a","b","d"))
#get colors for plotting
cores=rep(1:ntr,each=n.by.tr)
```

Then we define the means by treatment - note that they are different, so the null hypothesis in an ANOVA, that all the means are the same, is false.

```
#define 4 means
ms=c(3,5,6,2)
```

Then, the key part, the response variable, with a different mean by treatment. Note the use of the `ifelse` function, which evaluates its first argument and then assigns the value of its second argument if the first is true or the value of the second if its first argument is false. An example

```
ifelse(3>4,55,77)
```

```
## [1] 77
```

```
ifelse(3<4,55,77)
```

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

Note these can be used nested, leading to possible multiple outcomes, and I use that below to define 4 different means depending on the treatment of the observation

```
ifelse(3<4,55,ifelse(3>2,55,68))
```

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

```
ifelse(3>4,55,ifelse(3>2,666,68))
```

```
## [1] 666
```

```
ifelse(3>4,55,ifelse(3<2,666,68))
```

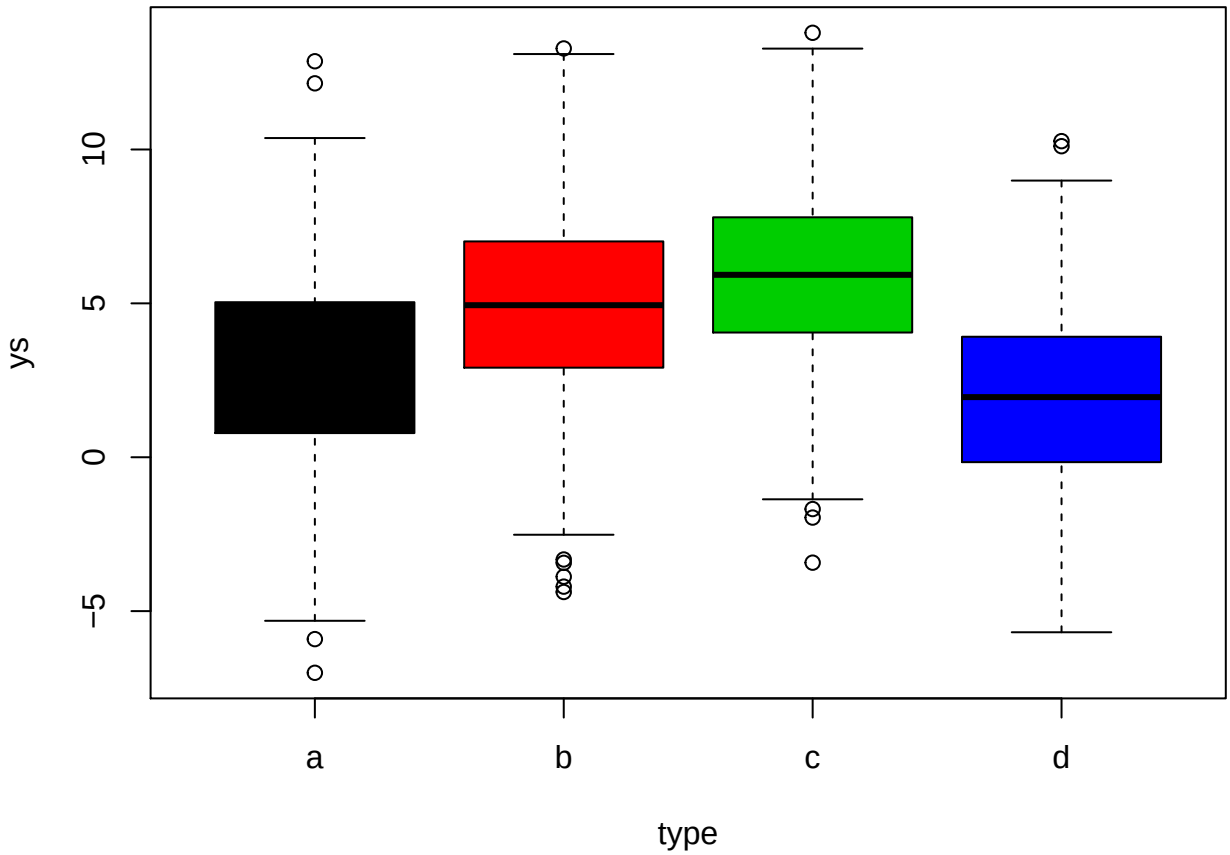
```
## [1] 68
```

So now, generate the data

```
#ys, not a function of the xs!!!
ys=ifelse(type=="a",ms[1],ifelse(type=="b",ms[2],ifelse(type=="c",ms[3],ms[4])))+rnorm(n,0,3)
```

We can actually look at the simulated data

```
par(mfrow=c(1,1),mar=c(4,4,0.5,0.5))
plot(ys~type,col=1:4)
```




```
#abline(h=ms,col=1:4)
```

finally, we can implement the linear model and look at its summary

```
lm.anova=lm(ys~type)
summary(lm.anova)
```

```
##
## Call:
## lm(formula = ys ~ type)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.8735 -2.0115  0.0301  2.0208  9.9976
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.8694     0.1319  21.753  < 2e-16 ***
## typeb         2.0788     0.1865  11.143  < 2e-16 ***
## typec         2.9806     0.1865  15.978  < 2e-16 ***
## typed        -0.8726     0.1865  -4.678 3.09e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.95 on 1996 degrees of freedom
## Multiple R-squared:  0.2163, Adjusted R-squared:  0.2151
## F-statistic: 183.6 on 3 and 1996 DF,  p-value: < 2.2e-16
```

note that, again, we can manipulate any sub-components of the created objects

```
#see the parameters
lm.anova$coefficients
```

```
## (Intercept)       typeb       typec       typed
##   2.8694412   2.0787628   2.9806367  -0.8726428
```

```
#see the third parameter
lm.anova$coefficients[3]
```

```
##    typec
## 2.980637
```

Not surprisingly, because the means were different and we had a large sample size, everything is highly significant. Note that the ANOVA test is actually presented in the regression output, and that is the corresponding F-test

```
summary(lm.anova)$fstatistic
```

```
##     value     numdf     dendf
##  183.6156    3.0000 1996.0000
```

and we can use the F distribution to calculate the corresponding P-value (note that is already in the output above)

```
ftest=summary(lm.anova)$fstatistic[1]
df1=summary(lm.anova)$fstatistic[2]
df2=summary(lm.anova)$fstatistic[3]
pt(ftest,df1,df2)
```

```
##         value
## 1.402786e-131
```

OK, this is actually the exact value, while above the value was reported as just a small value ($< 2.2 \times 10^{-16}$), but it is the same value, believe me!

Finally, to show (by example) this is just what the ANOVA does, we have the NAOVA itself

```
summary(aov(lm.anova))
```

```
##               Df Sum Sq Mean Sq F value Pr(>F)
## type           3   4792  1597.5   183.6 <2e-16 ***
## Residuals   1996  17365     8.7
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

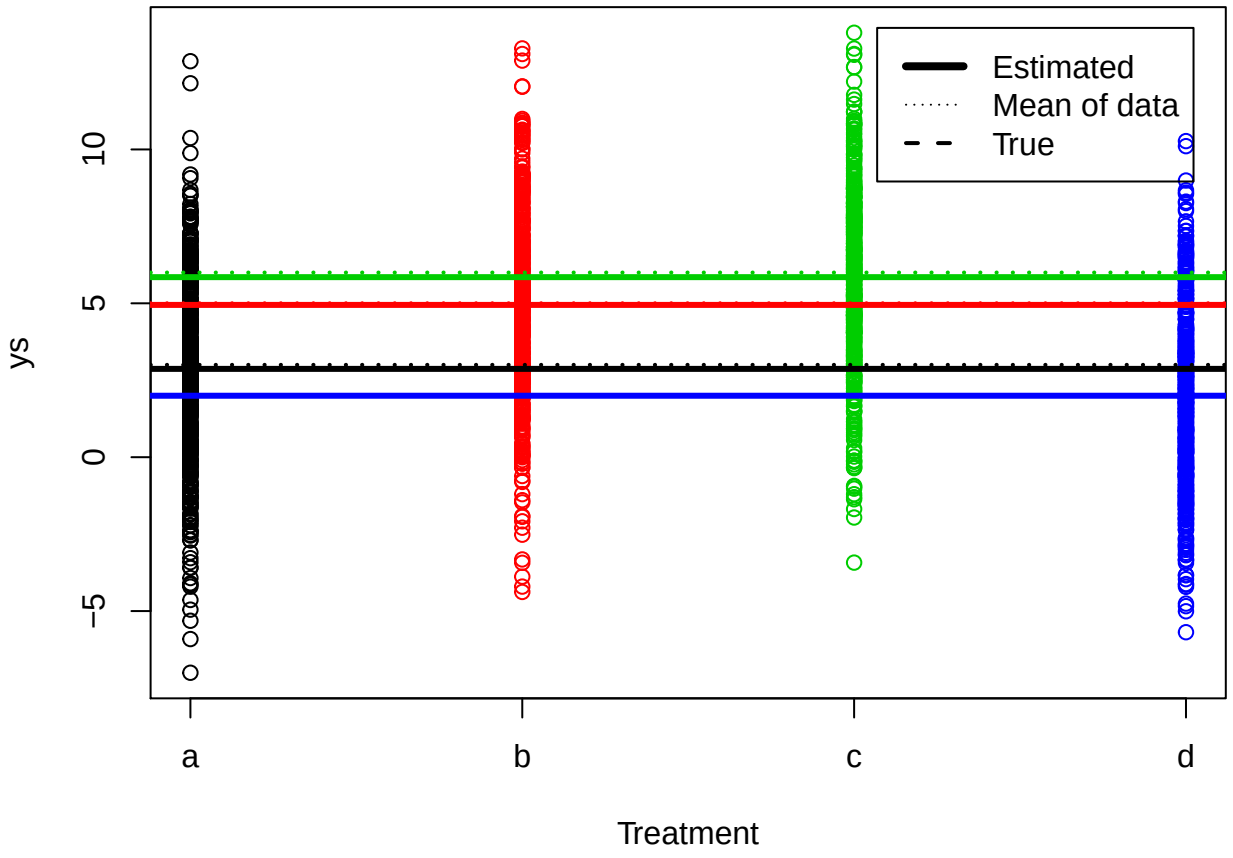
where everything is the same (test statistic, degrees of freedom and p-values).

Conclusion: an ANOVA is just a special case of a linear model, one where we have a continuous response variable and a factor explanatory covariate. In fact, a two way ANOVA is just the extension where we have a continuous response variable and 2 factor explanatory covariates, and, you guessed it, a three way ANOVA means we have a continuous response variable and a 3 factor explanatory covariates.

Just to finish up this example, we could now plot the true means per treatment, the estimated means per treatment

```
par(mfrow=c(1,1),mar=c(4,4,0.5,0.5))
plot(as.numeric(type),ys,col=as.numeric(type),xlab="Treatment",xaxt="n")
axis(1,at=1:4,letters[1:4])
#plot the estimated line for type a
abline(h=lm.anova$coefficients[1],lwd=3,col=1)
```

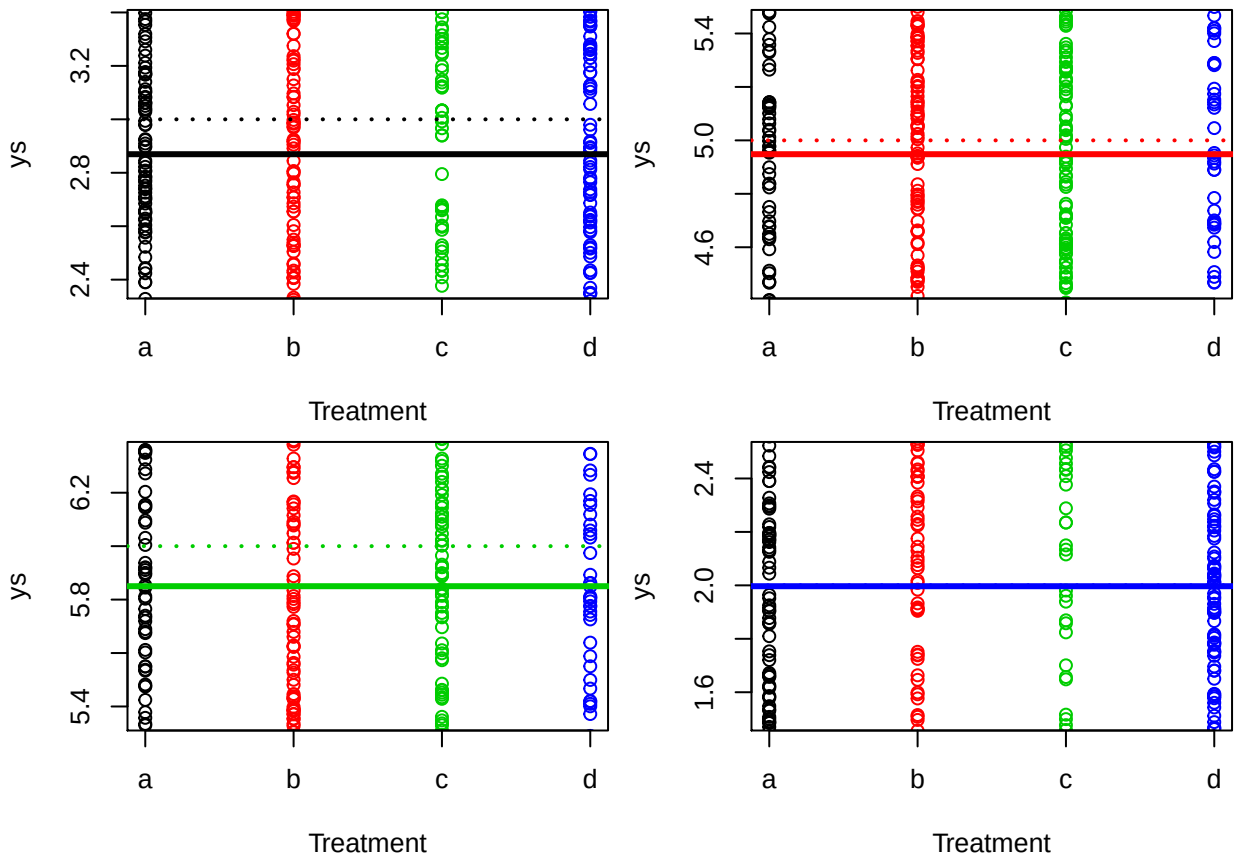
```
#plot the mean line for type a
abline(h=mean(ys[type=="a"]),lwd=1,col=1,lty=2)
#plot the real mean for type a
abline(h=ms[1],lwd=2,col=1,lty=3)
#and now for the other types
abline(h=lm.anova$coefficients[1]+lm.anova$coefficients[2],lwd=3,col=2)
abline(h=mean(ys[type=="b"]),lwd=1,col=2,lty=2)
#plot the real mean for type b
abline(h=ms[2],lwd=2,col=2,lty=3)
abline(h=lm.anova$coefficients[1]+lm.anova$coefficients[3],lwd=3,col=3)
abline(h=mean(ys[type=="c"]),lwd=1,col=3,lty=2)
#plot the real mean for type c
abline(h=ms[3],lwd=2,col=3,lty=3)
abline(h=lm.anova$coefficients[1]+lm.anova$coefficients[4],lwd=3,col=4)
abline(h=mean(ys[type=="d"]),lwd=1,col=4,lty=2)
#plot the real mean for type a
abline(h=ms[4],lwd=2,col=4,lty=3)
legend("topright",c("Estimated","Mean of data","True"),lwd=c(4,1,2),lty=c(1,3,2),inset=0.03)
```



It's not easy to see because these overlap (large sample size, high precision) but the estimated means are really close to the real means. It's a bit easier to see if we separate in 4 plots and zoom in on the mean of each treatment, but still the blue lines are all on top of each other, since the mean value was estimated real close to truth (truth=2, estimated = 1.9967984).

```
#see this in 4 plots, less blur
par(mfrow=c(2,2),mar=c(4,4,0.5,0.5))
plot(as.numeric(type),ys,col=as.numeric(type),xlab="Treatment",xaxt="n",ylim=mean(ys[type=="a"])+c(-0.5
```

```
axis(1,at=1:4,letters[1:4])
#plot the estimated line for type a
abline(h=lm.anova$coefficients[1],lwd=3,col=1)
#plot the mean line for type a
abline(h=mean(ys[type=="a"]),lwd=1,col=1,lty=2)
#plot the real mean for type a
abline(h=ms[1],lwd=2,col=1,lty=3)
#and now for the other types
plot(as.numeric(type),ys,col=as.numeric(type),xlab="Treatment",xaxt="n",ylim=mean(ys[type=="b"])+c(-0.5
axis(1,at=1:4,letters[1:4])
abline(h=lm.anova$coefficients[1]+lm.anova$coefficients[2],lwd=3,col=2)
abline(h=mean(ys[type=="b"]),lwd=1,col=2,lty=2)
#plot the real mean for type b
abline(h=ms[2],lwd=2,col=2,lty=3)
plot(as.numeric(type),ys,col=as.numeric(type),xlab="Treatment",xaxt="n",ylim=mean(ys[type=="c"])+c(-0.5
axis(1,at=1:4,letters[1:4])
abline(h=lm.anova$coefficients[1]+lm.anova$coefficients[3],lwd=3,col=3)
abline(h=mean(ys[type=="c"]),lwd=1,col=3,lty=2)
#plot the real mean for type c
abline(h=ms[3],lwd=2,col=3,lty=3)
plot(as.numeric(type),ys,col=as.numeric(type),xlab="Treatment",xaxt="n",ylim=mean(ys[type=="d"])+c(-0.5
axis(1,at=1:4,letters[1:4])
abline(h=lm.anova$coefficients[1]+lm.anova$coefficients[4],lwd=3,col=4)
abline(h=mean(ys[type=="d"]),lwd=1,col=4,lty=2)
#plot the real mean for type a
abline(h=ms[4],lwd=2,col=4,lty=3)
```




```
#legend("bottomright",c("Estimated","Mean of data","True"),lwd=c(4,1,2),lty=c(1,3,2),inset=0.05)
```

Now we can check how we can obtain the estimated means from the actual parameters of the regression model (yes, that is what the regression does, it calculates the expected mean of the response, conditional on the treatment).

This is the estimated mean per treatment, using function `tapply` (very useful function to get any statistics over a variable, inside groups defined by a second variable, here the treatment)

```
tapply(X=ys,INDEX=type,FUN=mean)
```

```
##        a        b        c        d
## 2.869441 4.948204 5.850078 1.996798
```

and checking these are obtained from the regression coefficients. An important note. When you fit models with factors (like here), the intercept term will correspond to the mean of the reference level of the factor(s). Hence, to get the other means, you always have to sum the parameter of the corresponding level to the intercept. So we do it below

```
#check ANOVA is just computing the mean in each group
lm.anova$coefficients[1]
```

```
## (Intercept)
##    2.869441
```

```
lm.anova$coefficients[1]+lm.anova$coefficients[2]
```

```
## (Intercept)
##    4.948204
```

```
lm.anova$coefficients[1]+lm.anova$coefficients[3]
```

```
## (Intercept)
##    5.850078
```

```
lm.anova$coefficients[1]+lm.anova$coefficients[4]
```

```
## (Intercept)
##    1.996798
```

and we can see these are exactly the same values.

ANCOVA

We move on to the ANCOVA, which is like an ANOVA to which we add a continuous explanatory covariate. This is an extremely common situation in biology/ecology data. Consider, as an example, you are trying to explain how the weight of a fish depends on its length, but you want to see if that relationship changes per year or site.

Let's simulate some relevant data and fit the models

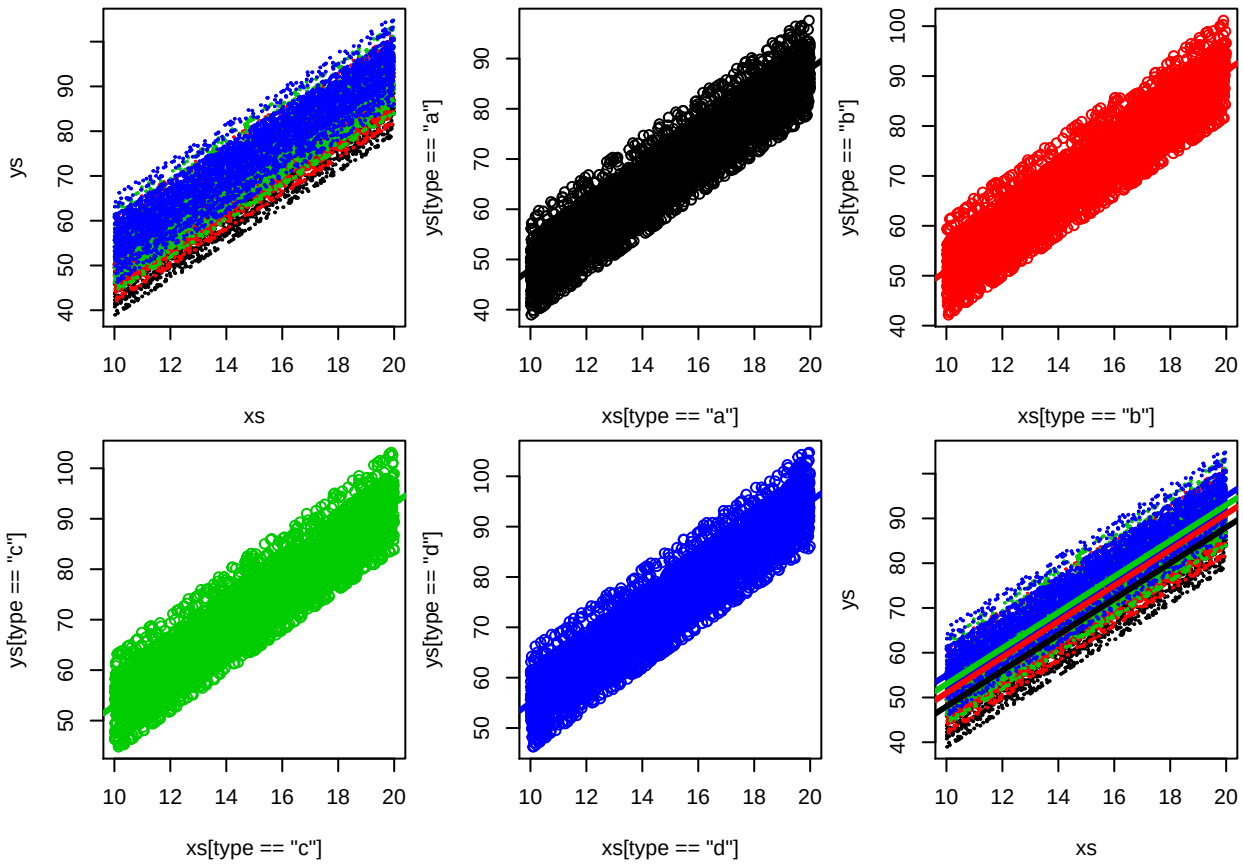
Common slope, different intercepts per treatment

We begin with a situation where there are different intercepts per group, but a common slope across all groups

```
#all slopes the same, diferent intercepts - no interactions
set.seed(1234)
xs=runif(20000,10,20)
type=rep(c("a","b","c","d"),each=5000)
cores=rep(1:4,each=5000)
ys=3+4*xs+
ifelse(type=="a",5,ifelse(type=="b",8,ifelse(type=="c",10,12)))+rnorm(200,0,4)
```

We plot the data, all together, per group, and at the end adding the generating line to the plot. It's not easy to make sense of it!

```
par(mfrow=c(2,3),mar=c(4,4,0.5,0.5))
#all the data - uma salganhada!
plot(xs,ys,col=cores,cex=0.2)
#plot the data
#par(mfrow=c(2,2),mar=c(4,4,0.5,0.5))
plot(xs[type=="a"],ys[type=="a"],col=cores[type=="a"])
abline(3+5,4,lwd=3,col=1)
plot(xs[type=="b"],ys[type=="b"],col=cores[type=="b"])
abline(3+8,4,lwd=3,col=2)
plot(xs[type=="c"],ys[type=="c"],col=cores[type=="c"])
abline(3+10,4,lwd=3,col=3)
plot(xs[type=="d"],ys[type=="d"],col=cores[type=="d"])
abline(3+12,4,lwd=3,col=4)
#the data with each line added to it
#par(mfrow=c(1,1),mar=c(4,4,0.5,0.5))
plot(xs,ys,col=cores,cex=0.2)
abline(3+5,4,lwd=3,col=1)
abline(3+8,4,lwd=3,col=2)
abline(3+10,4,lwd=3,col=3)
abline(3+12,4,lwd=3,col=4)
```



Now we run the linear model

```
#fit the model
lm.ancova1=summary(lm(ys~xs+type))
lm.ancova1
```

```
##
## Call:
## lm(formula = ys ~ xs + type)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.1325 -2.8197  0.0461  2.7266 10.6565
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.78940    0.16219   48.03   <2e-16 ***
## xs           4.00996    0.01009  397.38   <2e-16 ***
## typeb        3.00022    0.08186   36.65   <2e-16 ***
## typec        4.99925    0.08187   61.07   <2e-16 ***
## typed        7.00013    0.08186   85.51   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.093 on 19995 degrees of freedom
## Multiple R-squared:  0.8925, Adjusted R-squared:  0.8925
## F-statistic: 4.152e+04 on 4 and 19995 DF,  p-value: < 2.2e-16
```

We can check the model intercept coefficients

```
#estimated values of each intercept
lm.ancova1$coefficients[1]
```

```
## [1] 7.789395
```

```
lm.ancova1$coefficients[1]+lm.ancova1$coefficients[3]
```

```
## [1] 10.78962
```

```
lm.ancova1$coefficients[1]+lm.ancova1$coefficients[4]
```

```
## [1] 12.78864
```

```
lm.ancova1$coefficients[1]+lm.ancova1$coefficients[5]
```

```
## [1] 14.78952
```

and the common slope

```
lm.ancova1$coefficients[2]
```

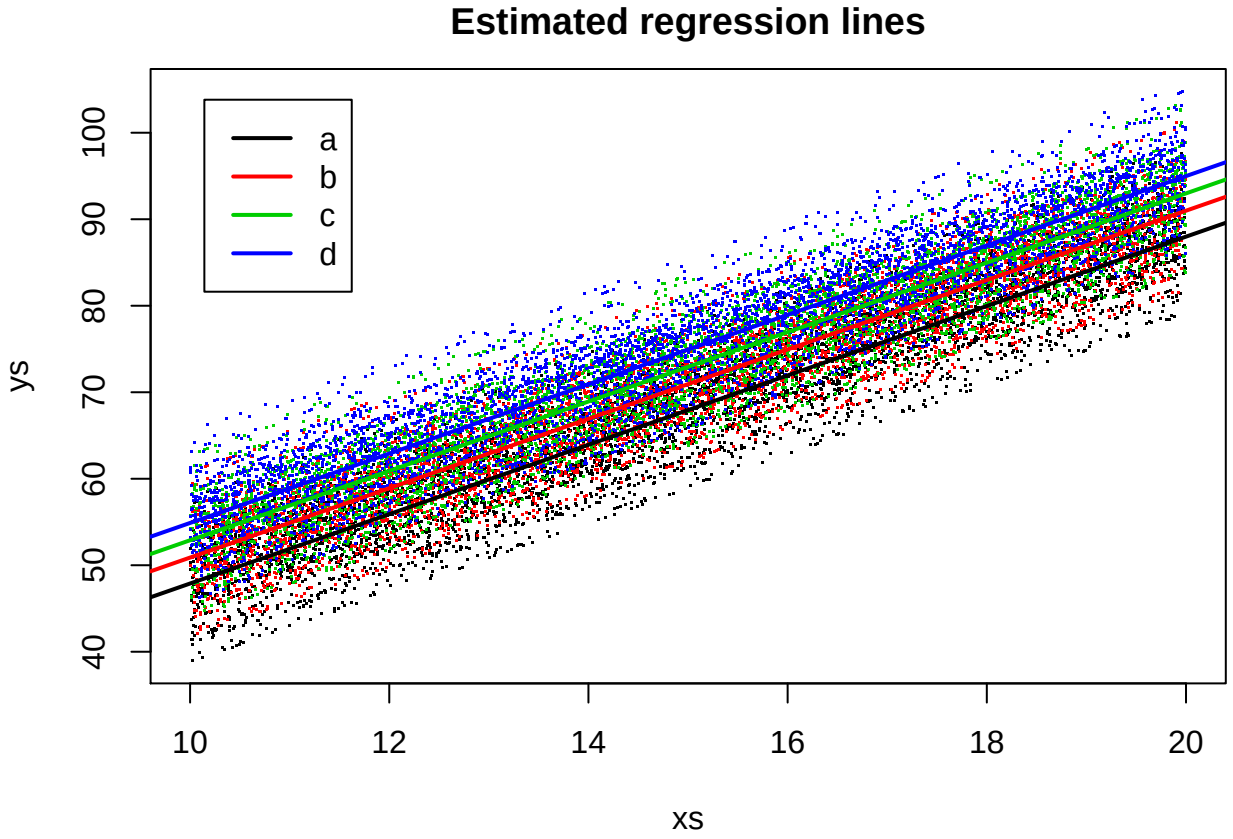
```
## [1] 4.009957
```

Check how these values are similar (they are estimates) to those we simulated above, slope was 4, and the intercepts were respectively 3+5,3+8,3+10 and 3+12.

We can plot the estimated regression lines

```
par(mfrow=c(1,1),mar=c(4,4,2.5,0.5))
plot(xs,ys,col=cores,pch=".",main="Estimated regression lines")
abline(lm.ancova1$coefficients[1],lm.ancova1$coefficients[2],col=1,lwd=2)
abline(lm.ancova1$coefficients[1]+lm.ancova1$coefficients[3],lm.ancova1$coefficients[2],col=2,lwd=2)
abline(lm.ancova1$coefficients[1]+lm.ancova1$coefficients[4],lm.ancova1$coefficients[2],col=3,lwd=2)

abline(lm.ancova1$coefficients[1]+lm.ancova1$coefficients[5],lm.ancova1$coefficients[2],col=4,lwd=2)
legend("topleft",legend = tr,lwd=2,col=1:4,inset=0.05)
```



Different intercepts and slopes per treatment

Now we extend the previous case to where the slope of the relationship is also different per treatment.

Simulate treatments, same as before, but at least gives us the option to change later separately if we want.

```
#-----------------------------------------------------------------
#all slopes different
set.seed(1234)
xs=runif(200,10,20)
type=rep(c("a","b","c","d"),each=50)
cores=rep(1:4,each=50)
```

Now we simulate the response

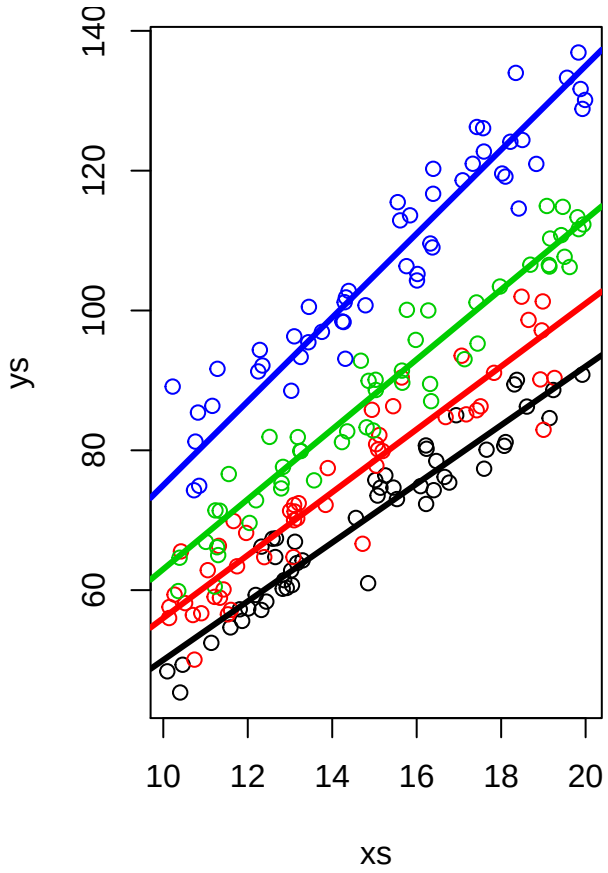
```
ys=3+
ifelse(type=="a",5,ifelse(type=="b",8,ifelse(type=="c",10,12)))+
4*xs+ifelse(type=="a",0.2,ifelse(type=="b",0.5,ifelse(type=="c",1,2)))*xs+
rnorm(200,0,4)
```

note this is the same as what we have below, but below it might be simpler to understand that these do correspond to different intercepts and slopes per treatment

```
#same as
intercept=3+ifelse(type=="a",5,ifelse(type=="b",8,ifelse(type=="c",10,12)))
slope=xs*(4+ifelse(type=="a",0.2,ifelse(type=="b",0.5,ifelse(type=="c",1,2))))
ys=slope+intercept+rnorm(200,0,4)
```

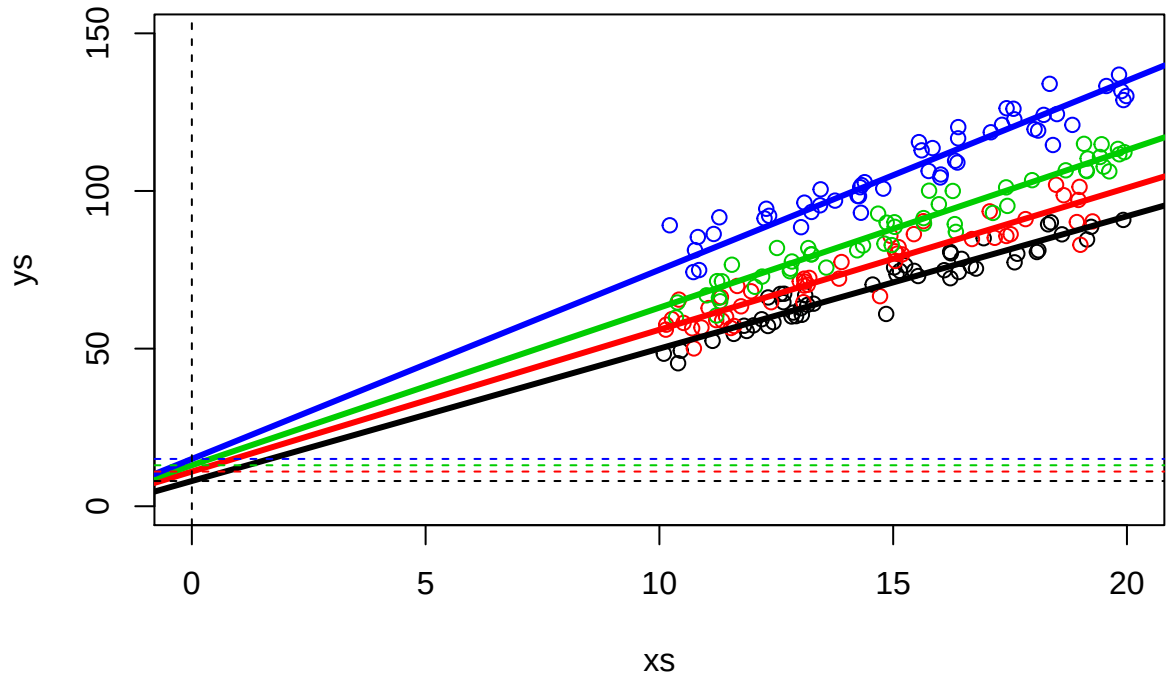
We can look at the data

```
par(mfrow=c(1,2),mar=c(4,4,0.5,0.5))
plot(xs,ys,col=cores)
abline(3+5,4+0.2,lwd=3,col=1)
abline(3+8,4+0.5,lwd=3,col=2)
abline(3+10,4+1,lwd=3,col=3)
abline(3+12,4+2,lwd=3,col=4)
```



it's actually not that easy to confirm the slopes and intercepts are different, as the intercept is not shown in the above plot. We can zoom out the plot and force that

```
plot(xs,ys,col=cores,xlim=c(0,20),ylim=c(0,150))
abline(3+5,4+0.2,lwd=3,col=1)
abline(3+8,4+0.5,lwd=3,col=2)
abline(3+10,4+1,lwd=3,col=3)
abline(3+12,4+2,lwd=3,col=4)
abline(h=c(3+5,3+8,3+10,3+12),v=0,col=c(1,2,3,4,1),lty=2)
```



Now, we implement the AANCOVA linear model

```
lm.ancova2=lm(ys~xs+type+xs*type)
sum.lm.ancova2=summary(lm.ancova2)
```

and we look at the output of the model

```
sum.lm.ancova2
```

```
##
## Call:
## lm(formula = ys ~ xs + type + xs * type)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.2514  -2.7831  -0.2006   3.0000  10.1051
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   7.0480     3.4994   2.014 0.045397 *
## xs            4.2667     0.2345  18.198  < 2e-16 ***
## typeb         4.5418     4.6361   0.980 0.328489
## typec         3.7997     4.7096   0.807 0.420787
## typed        15.9380     4.8844   3.263 0.001305 **
## xs:typeb      0.1848     0.3160   0.585 0.559468
## xs:typec      0.8442     0.3103   2.721 0.007106 **
## xs:typed      1.2325     0.3203   3.848 0.000162 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.331 on 192 degrees of freedom
## Multiple R-squared:  0.9578, Adjusted R-squared:  0.9562
## F-statistic: 622.3 on 7 and 192 DF,  p-value: < 2.2e-16
```

check how this is an output similar to the ANOVA (implemented via `aov`, the R function that produces ANOVA tables from expressions akin to linear models)

```
summary(aov(ys~xs+type+xs*type))
```

```
##              Df Sum Sq Mean Sq  F value   Pr(>F)
## xs            1  48776   48776 2600.680  < 2e-16 ***
## type          3  32550   10850  578.501  < 2e-16 ***
## xs:type       3    379     126    6.742 0.000239 ***
## Residuals   192   3601      19
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note that the overall F statistic from the regression model has an F-statistic of 622.3, with 7 and 192 degrees of freedom. That corresponds to the composite test with the null hypothesis "are all parameters equal to 0", which in the ANOVA table, is separated in 3 testes, one for each parameter, with 1, 3 and 3 degrees of freedom each. The residual degrees of freedom are naturally the same in all these tests.

The most interesting aspect it that, naturally, we can check the values of the estimated coefficients, and in particular how to estimate the corresponding regression lines per group

```
#type a
lm.ancova2$coefficients[1]
```

```
## (Intercept)
##    7.048022
```

```
lm.ancova2$coefficients[2]
```

```
##       xs
## 4.266671
```

```
#type b
lm.ancova2$coefficients[1]+lm.ancova2$coefficients[3]
```

```
## (Intercept)
##     11.5898
```

```
lm.ancova2$coefficients[2]+lm.ancova2$coefficients[6]
```

```
##       xs
## 4.451449
```

```
#type c
lm.ancova2$coefficients[1]+lm.ancova2$coefficients[4]
```

```
## (Intercept)
##    10.84769
```

```
lm.ancova2$coefficients[2]+lm.ancova2$coefficients[7]
```

```
##       xs
## 5.110884
```

```
#type b
lm.ancova2$coefficients[1]+lm.ancova2$coefficients[5]
```

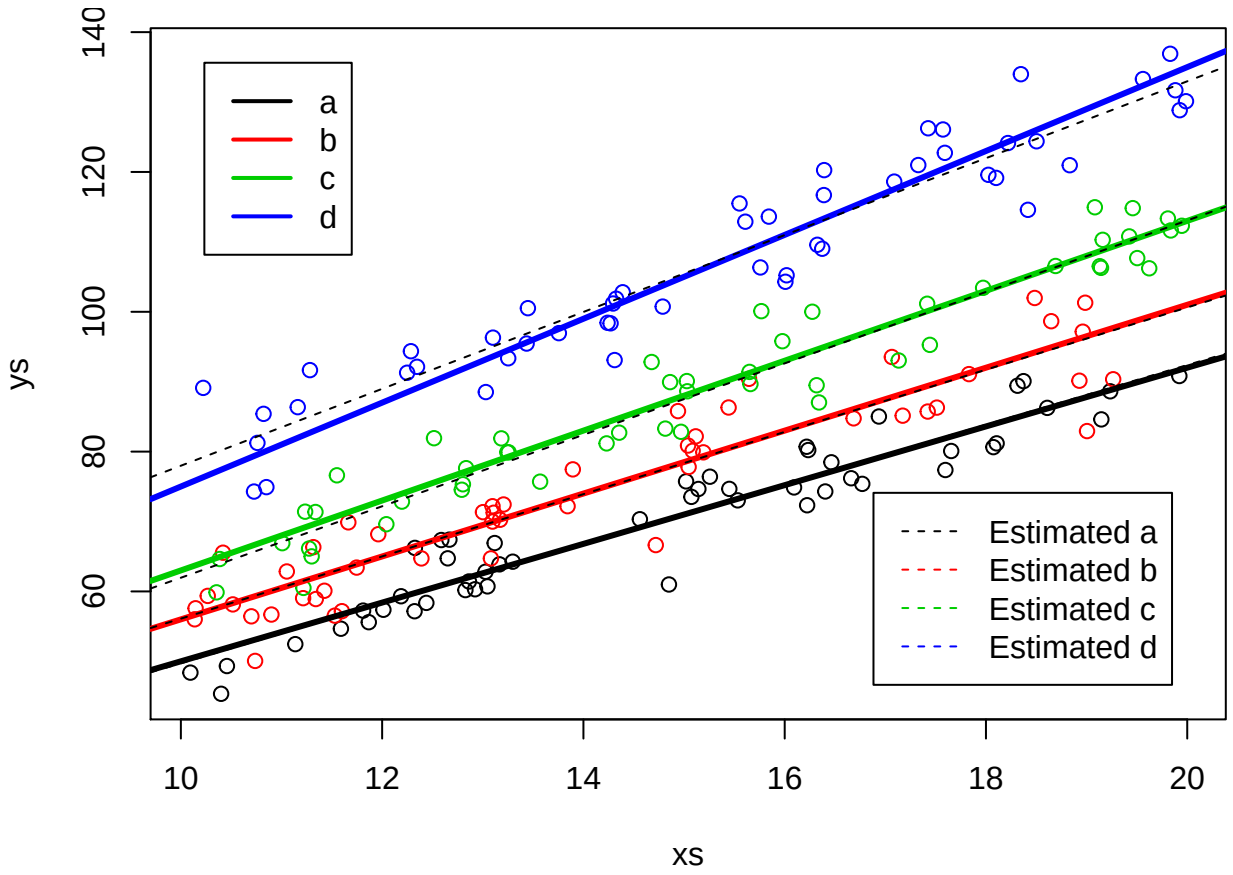
```
## (Intercept)
##    22.98597
```

```
lm.ancova2$coefficients[2]+lm.ancova2$coefficients[8]
```

```
##       xs
## 5.499171
```

we can now add these to the earlier plots, to see how well we have estimated the different lines per treatment

```
#real lines
par(mfrow=c(1,1),mar=c(4,4,0.5,0.5))
plot(xs,ys,col=cores)
abline(3+5,4+0.2,lwd=3,col=1)
abline(3+8,4+0.5,lwd=3,col=2)
abline(3+10,4+1,lwd=3,col=3)
abline(3+12,4+2,lwd=3,col=4)
#estimated lines
#type a
abline(lm.ancova2$coefficients[1],lm.ancova2$coefficients[2],lty=2,col=1)
#type b
abline(lm.ancova2$coefficients[1]+lm.ancova2$coefficients[3],
lm.ancova2$coefficients[2]+lm.ancova2$coefficients[6],lty=2,col=1)
#type c
abline(lm.ancova2$coefficients[1]+lm.ancova2$coefficients[4],
lm.ancova2$coefficients[2]+lm.ancova2$coefficients[7],lty=2,col=1)
#type b
abline(lm.ancova2$coefficients[1]+lm.ancova2$coefficients[5],
lm.ancova2$coefficients[2]+lm.ancova2$coefficients[8],lty=2,col=1)
legend("topleft",legend = tr,lwd=2,col=1:4,inset=0.05)
legend("bottomright",legend =paste("Estimated",tr),lwd=1,lty=2,col=1:4,inset=0.05)
```



Conclusion

The material above allows you to fully understand the outputs of simple regression models, to see how some statistical models that you know from other names are just a linear model.

It also helps you understand how the parameter values represent just features of the data and its generating process, and how we can recover estimates of the original relationships between the variables from said set of parameters.

I recommend you explore the code and output above, and that in particular you experiment with changing means (parameter values for the real models), variances (the precision of how you would measure variables) and sample sizes (which gives you an indication of how much information you have to estimate the underlying reality). Understanding the outputs under these new scenarios is fundamental for progressing towards more complex regression models, like GLMs or GAMs, of which the above cases are just particular cases.