Statistics with Sparrows - 08

Julia Schroeder

July 2020

HO 08

Starting off with linear models

Learning aims

- Getting a feel for the code used for linear models
- Exploring the structure of models, and the structure of model objects, in R
- Understanding how to access the different statistical parameters of models
- Starting to explore correlations and covariances linked with simple linear models

Let's give this a go:

First, we will try to re-do what I've shown you in the lecture. We will re-create the vectors x and y, and then run the model. I want you to use the output (in your R) to see if you can find similar estimates to those we "guesstimated" during the lecture. It is really important now that you not only run the code in your own machine, but also, that you inspect the output, be it red (errors or warning messages) or not red. You need to look at all the numbers, put them into perspective of each other, and think about what each number means. I'll talk you through it but you need to start deducing this by yourself at this point:

```
rm(list=ls())
x<-c(1,2,3,4,8)
y<-c(4,3,5,7,9)
x
```

```
## [1] 1 2 3 4 8
```

```
mean(x)
```

```
## [1] 3.6
```

```
var(x)
```

```
## [1] 7.3
```

```
y
```

```
## [1] 4 3 5 7 9
```

```
mean(y)

## [1] 5.6

var(y)

## [1] 5.8
```

So this is how we made the data for our model. Now we run the linear model. As linear models are the most important function for any sort of statistic, it's a really important function to know it by heart. It's `lm()`. Accessing the help function with this gives:

```
?lm
```

It seems we need to pass a sort of formula to R, that's the model equation! Remember:

$$y = b_0 + b_1 * x$$

Where b_0 is the intercept, b_1 is the slope for the explanatory variable x, and y is the response variable. In R, we specify that in a very specific way. First, we give the response variable, y, then we use this sign "~" (called the tilde). Make sure you find it on your keyboard - you'll be using it **a lot** from here on in. If you can't find it, google where it is on your keyboard. Then, after the tilde comes our explanatory variable. We don't have to specify the intercept b_0 (at least not for most models), it is estimated automatically by default. For the slope b_1 we need to tell R what explanatory variable it is connected to. So we just say `y~x` in this case. If we have a data frame, we have to specify the data frame using the `data=` argument. There's an argument for `weights` that we ignore for now, as the one for `subset=`. There is one for the `na.action=`, this is self-explanatory and I want you to play around with that a bit later on. There will be little if any need to use any of the other arguments, but don't let that stop your curiosity. Yet, it won't be part of this course. It's all explained in the "Details" section of the help text. Then, under the "Details" section (you need to scroll down), there's a "Value" section, that explains what the function returns. It returns an object of class "lm", and then it goes on to say what sort of values you can access from that. There's `coefficients` - that sounds like the parameter estimates we're after. The next one, `residuals` should be clear to you. `fitted values` is interesting, and we'll look at that further below. Then we can glance of the rest. Scroll further down, until you reach "See Also". There's the `summary.lm` and `anova.lm` command, also further generic functions, and `predict.lm`. And lots more if you want to dive right in, but I'd say reserve that for later. The examples are interesting, and I urge you to run them. Very much like this hand-out, you can copy-paste the code and it should run, then inspect the output and see if you understand it. I recommend doing this after you've done this hand out.

Let's put this to work:

```
model1 <- (lm(y~x))
model1

##
## Call:
## lm(formula = y ~ x)
```

```
##
## Coefficients:
## (Intercept)            x
##      2.6164       0.8288
```

```
summary(model1)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##       1       2       3       4       5
##  0.5548 -1.2740 -0.1027  1.0685 -0.2466
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.6164     0.8214   3.185   0.0499 *
## x             0.8288     0.1894   4.375   0.0221 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.024 on 3 degrees of freedom
## Multiple R-squared:  0.8645, Adjusted R-squared:  0.8193
## F-statistic: 19.14 on 1 and 3 DF,  p-value: 0.0221
```

```
coefficients(model1)
```

```
## (Intercept)           x
##   2.6164384   0.8287671
```

```
resid(model1)
```

```
##          1          2          3          4          5
##  0.5547945 -1.2739726 -0.1027397  1.0684932 -0.2465753
```

```
mean(resid(model1))
```

```
## [1] 1.111578e-17
```

```
var(resid(model1))
```

```
## [1] 0.7859589
```

```
length(resid(model1))
```

```
## [1] 5
```

I think, give you've been through the lecture, you will now see how we can access our primary statistics, the intercept and the slope for x.

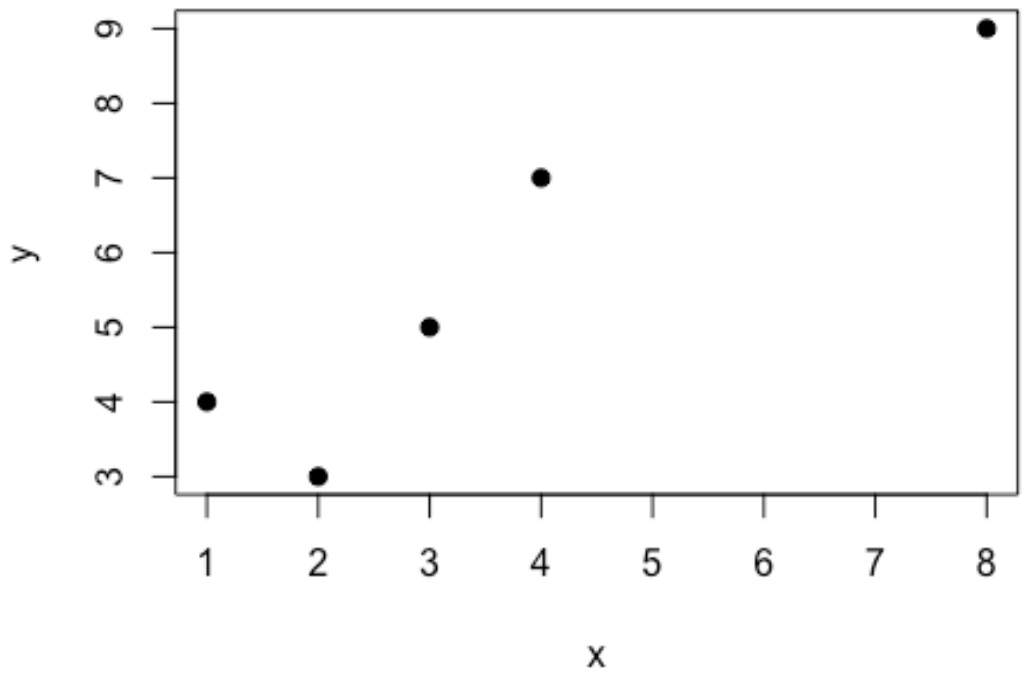
However, there is clearly more - the most comprehensive thing came through with the `summary()` function:

```r
summary(model1)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##       1       2       3       4       5
##  0.5548 -1.2740 -0.1027  1.0685 -0.2466
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.6164     0.8214   3.185   0.0499 *
## x             0.8288     0.1894   4.375   0.0221 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.024 on 3 degrees of freedom
## Multiple R-squared:  0.8645, Adjusted R-squared:  0.8193
## F-statistic: 19.14 on 1 and 3 DF,  p-value: 0.0221
```

It tells us the residuals. Then the coefficients, and their standard errors, and associated t-values and p-values. Then there's a row of significance codes, a row about the residual SE, the dfs, and some stuff on R^2 and F statistics that we haven't spoken about just yet. But we will. For now, we stick to the Coefficents. As they are estimated from data, we can calculate a standard error around them. And if their 95%CI (or, as we know, 2xSE) does not span 0, we can assume they are not statistically significant. For example, the slope is 0.829 and its SE of 0.19. Twice the SE plus minus the estimate (2x0.19+0.829 and 2x0,19-0.089) does not span zero thus the slope is statistically significant. That's confirmed by the t-value, and the p-value. Later more about this, but it's good to see. Also, there are 3 degrees of freedom. That makes sense because we have 5 data points, minus 2 for the two coefficients (intercept and slope) that we estimated. So far, I hope you have understood everything. Now, let's have a look at the line that this model predicts. We can plot it with abline into the plot with the datapoints. Note, that when I plot a model, I use the code y ~ x to denotate that y is the response variable, and x is the explanatory variable. That makes r plot y on the y axis (where the response belongs), and x on the x-axis (again, where it belongs).

```r
plot(y~x, pch=19)
```

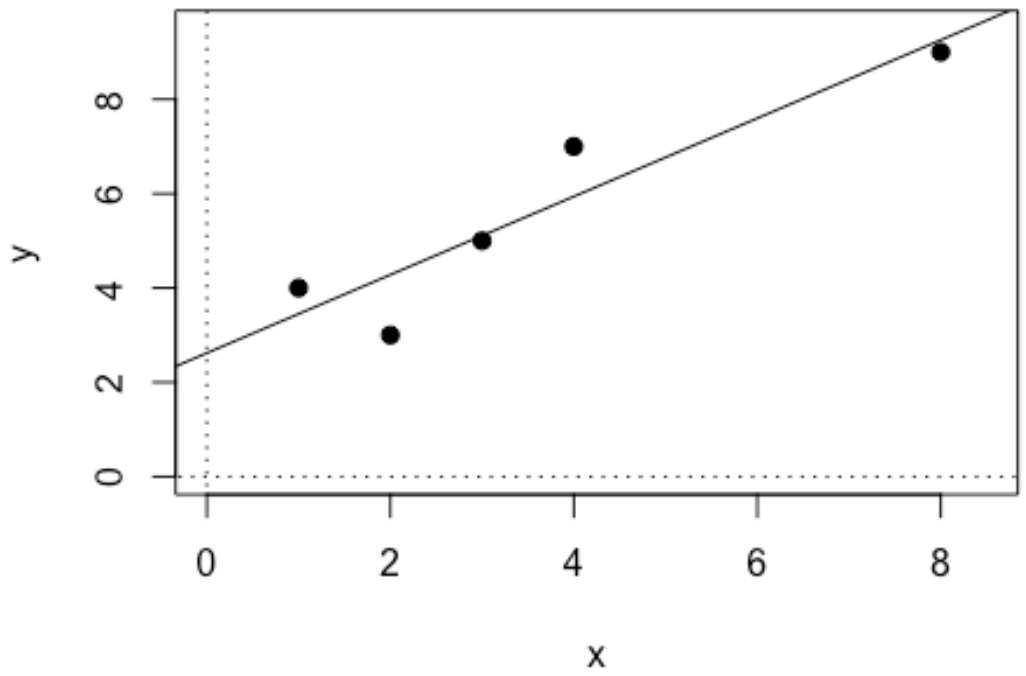


If I plot it with a comma (,), then it will be plotted such that the first variable will be interpreted as x, and the second one as y. It's a bit like with the t-test. Try it! To make sure you're not confused it is really good to stick to the tilde as the response vs explanatory variable stuff is always clear - response is always to the left of the tilde. I plot the y and x graph, and I change the x- and y-axes length (xlim and ylim) so that they include zero, because I want to see the intercept when I plot the line through my data. I do that with `abline()`. I know now the coefficients, so I can put that into the abline function. I bring the coefficients up again so I don't need to scroll I also add the cartesian coordinate axes explicitely so I'll be better able to assess the intercept:

```
plot(y~x, pch=19, xlim=c(0,8.5), ylim=c(0,9.5))
segments(0,-30,0,30, lty=3)
segments(-30,0,30,0,lty=3)
coefficients(model1)

## (Intercept)           x
##   2.6164384   0.8287671

abline(2.62, 0.83)
```



That looks like a neatly fitted line indeed. The intercept looks like somewhere between 2 and 3, so 6.6 seems like a good enough guess. And when you squint, you can see about what the slope would be (remember: one to the right, and then the slope is how much you have to go up until you hit the line).

Now, we can move on and use a somewhat larger dataset. We can simulate that, and even make up the data that we want. Let's see how that works. We will first create an x variable that goes from -10 to +10. Then we decide on a random value for the slope, maybe -0.2. We also choose an intercept: 7.1. Then we simulate numbers using the normal number generator in R:

```
x<-seq(from=-10, to=10, by=0.2)
x

##   [1] -10.0  -9.8  -9.6  -9.4  -9.2  -9.0  -8.8  -8.6  -8.4  -8.2  -8.0  -
7.8
##  [13]  -7.6  -7.4  -7.2  -7.0  -6.8  -6.6  -6.4  -6.2  -6.0  -5.8  -5.6  -
5.4
##  [25]  -5.2  -5.0  -4.8  -4.6  -4.4  -4.2  -4.0  -3.8  -3.6  -3.4  -3.2  -
3.0
##  [37]  -2.8  -2.6  -2.4  -2.2  -2.0  -1.8  -1.6  -1.4  -1.2  -1.0  -0.8  -
0.6
##  [49]  -0.4  -0.2   0.0   0.2   0.4   0.6   0.8   1.0   1.2   1.4   1.6
```

```
1.8
##  [61]   2.0   2.2   2.4   2.6   2.8   3.0   3.2   3.4   3.6   3.8   4.0
4.2
##  [73]   4.4   4.6   4.8   5.0   5.2   5.4   5.6   5.8   6.0   6.2   6.4
6.6
##  [85]   6.8   7.0   7.2   7.4   7.6   7.8   8.0   8.2   8.4   8.6   8.8
9.0
##  [97]   9.2   9.4   9.6   9.8  10.0
```

```
y<- 7.1-0.2 * x
y
```

```
##   [1] 9.10 9.06 9.02 8.98 8.94 8.90 8.86 8.82 8.78 8.74 8.70 8.66 8.62 8.5
8 8.54
##  [16] 8.50 8.46 8.42 8.38 8.34 8.30 8.26 8.22 8.18 8.14 8.10 8.06 8.02 7.9
8 7.94
##  [31] 7.90 7.86 7.82 7.78 7.74 7.70 7.66 7.62 7.58 7.54 7.50 7.46 7.42 7.3
8 7.34
##  [46] 7.30 7.26 7.22 7.18 7.14 7.10 7.06 7.02 6.98 6.94 6.90 6.86 6.82 6.7
8 6.74
##  [61] 6.70 6.66 6.62 6.58 6.54 6.50 6.46 6.42 6.38 6.34 6.30 6.26 6.22 6.1
8 6.14
##  [76] 6.10 6.06 6.02 5.98 5.94 5.90 5.86 5.82 5.78 5.74 5.70 5.66 5.62 5.5
8 5.54
##  [91] 5.50 5.46 5.42 5.38 5.34 5.30 5.26 5.22 5.18 5.14 5.10
```

Now we can use that data to run a linear model. We expect the slope to be -0.2, and the intercept to be 7.1, of course:

```
summary(lm(y~x))
```

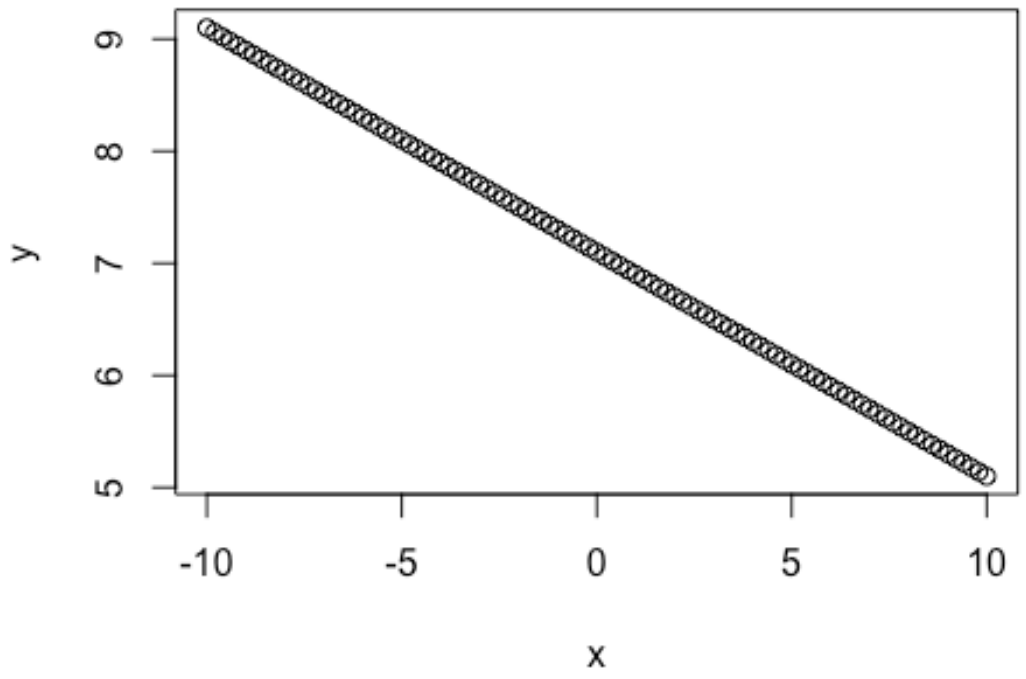
```
## Warning in summary.lm(lm(y ~ x)): essentially perfect fit: summary may be
## unreliable
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.385e-15 -4.873e-16 -1.301e-16  1.431e-16  1.882e-14
##
## Coefficients:
##               Estimate Std. Error    t value Pr(>|t|)
## (Intercept)  7.100e+00  1.955e-16  3.632e+16   <2e-16 ***
## x           -2.000e-01  3.352e-17 -5.966e+15   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.964e-15 on 99 degrees of freedom
```

```
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 3.56e+31 on 1 and 99 DF,  p-value: < 2.2e-16
```

Uuups. While the estimates are excellent, we get an odd error message. It says the fit is essentially perfect. What does that mean? Maybe it will clear up when we plot the data (plotting data when in trouble is always a good way to find the mistake):

```
plot(y~x)
```



Now it's clear - the data is too perfect. There is no uncertainty. So the standard error is super small, as are the residuals. Both are practically zero.

Well, we can try to simulate some uncertainty to our data. We'll use a random number generator to add the residual error when we create our y values. The function `runif()` does that - try `?runif` for more information. Basically, it creates random numbers between 0 and 1 by default, but you can set that interval to one of your choice.You also must specify the length of the vector you want to create. As you remember our residuals are the same number of values as x and y, so that's what we need to do.

Now, this time, we expect out coefficients to not be completely perfect, but still in the ballpark. Note: your own values should be different from those in the hand out because it's a random number generator!

```r
y<- 7.1-0.2 * x + runif(length(x))
summary(lm(y~x))

##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.56360 -0.21841 -0.03266  0.23890  0.52249
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.615939   0.029828  255.32   <2e-16 ***
## x           -0.211210   0.005116  -41.29   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2998 on 99 degrees of freedom
## Multiple R-squared:  0.9451, Adjusted R-squared:  0.9446
## F-statistic:  1705 on 1 and 99 DF,  p-value: < 2.2e-16
```

That looks much better.

Exercises

(no uploads)

1) Run the code you find under "examples" in the help file for `lm`
2) Look through the helpfile for `runif` and see if you understand it.
3) Increase and decrease the size of the random noise you add to the data, and see what it does to your linear model estimates! This will help you better understand the concepts behind it, and also help you to better interpret results. Further, I hope this will help you to use this tool (making up data where you know what the result should be like, and then analysing it) to better understand statistics.
4) Run the example code from the helpfile for `runif` and see if you can understand it.