ZOL851 Intro tutorial on GLM in R

Ian Dworkin

September 18, 2012

In this tutorial we are going to examine how **R** fits some simple linear models, and how we can interpret some of the outputs. I generally like to call all of the libraries that I will use first, and double check that there are no conflicts (in terms of names of functions).

```
> require(arm)
> require(car)
> require(lattice)
```

Reading in & and checking the data

As usual, we set our working directory and read in the data. You will want to change the working directory to your home directory.

```
> dll.data = read.csv("/Users/ian/R/R scripts/Dll data/dll.txt", header=TRUE)
```

I always start by looking at the input of the data first using `summary(your.data.frame)` and `str(your.data.frame)`. I also use `head(your.data.frame)` and `tail(your.data.frame)` to confirm it was all imported as expected. For space here, I will not call them but you should run them yourself `summary(dll.data)`, `str(dll.data)`, etc...

If you have used these functions, you will see that some of the variable in `dll.data` are treated numerically, when in fact we want them to be treated as factors. In particular `temp` and `replicate`. So we have **R** change these to factors.

```
> dll.data$temp <- factor(dll.data$temp)
> dll.data$replicate <- factor(dll.data$replicate)
```

We also want to make sure that the default level of the factor `genotype=="wt"`. By default factors in **R** go alpha-numerically. That is the lowest letter of the alphabet, or smallest numbers are the default base level. However, we want to reset this to aid in intepretation. To do this we use the relevel function, which has the name of the variable that we are changing, and the new default level

```
> dll.data$genotype <- relevel(dll.data$genotype, "wt")
>  # Setting the wild-type (wt) as reference
```

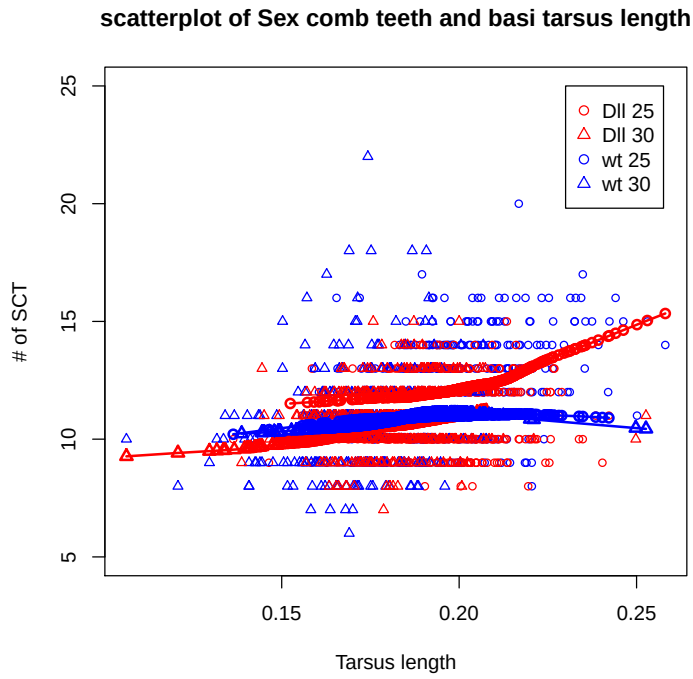
For the purposes of this tutorial we are also going to remove all rows that have any missing observations. This is not generally advisable, and the smarter approach would be to just remove specific rows that have missingness for the variables used in the model.

```
> dll.data <- na.omit(dll.data)
```

At this point it is wise to call `summary()` and `str()` for your data frame again to make sure the expected changes occurred.

Some quick data exploration

Before we fit the models, we will quickly do some exploratory plots. Please review the exploratory data analysis tutorial for more details.



In this case, the lowess lines are not actually very informative on the plot, but perhaps we can see the difference in the trend for the relationship between `SCT` and `tarsus` across the two genotypes.

We could do a fair bit of additional plotting, but I leave this as an exercise (see associated **R** code).

using the `lm()` function in R

To perform this regression in **R** we use the `lm()` function. lm stands for **L**inear **M**odel. As we will learn linear regression is just a special case of the *general linear model* in statistics.

In **R** we regress y onto x like:

`lm(y ~ x, data=YourData)`, where the tilde ('~') means 'is modeled as'. So you would read this as "y is modeled as a function of x". Using the `lm()`

additionally implies that we are assuming that the residual (unexplained) variation in the model is normally distributed, and that the response is (to a first approximation) a continuous variable. So we are really fitting the model.

$y = f(x) \sim N(\beta_0 + \beta_1 x, \sigma)$, where

$N()$ means normally distributed with mean (μ) equal to $\beta_0 + \beta_1 x$
β_0 is the intercept
β_1 is the slope of the regression line
σ is the distribution for the residual (unexplained) variation. The variation remaining in our response (y) after accounting for the model. Here I have it in the same units as y, but you will often see it expressed as a variance σ^2, and being labeled the unexplained variance of the model.

Using the `lm()` in **R** assumes that you want to fit an intercept term in your model (which we almost want to do). So we did not need to explicitly include it in the function call to `lm()`. However, we certainly can (and it is probably a good idea when you are first starting out). In any case it means that these two calls are equivalent:

`lm(y ~ 1 + x)` means the same as
`lm(y ~ x)`

Now we actually fit the model in **R**

```
> regression.1 <- lm(SCT ~ tarsus, data=dll.data) # The intercept is implicit
> regression.1.Intercept <- lm(SCT ~ 1 + tarsus, data=dll.data)
>   # With explicit intercept, but the same as above!
```

It is useful to remind ourselves of the design matrix. We can use the `model.matrix()`. Remember that it will have as many rows as observations! We will just look at the first few observations.

```
> head(model.matrix(regression.1))
```

```
  (Intercept) tarsus
1           1  0.219
2           1  0.214
3           1  0.211
5           1  0.207
6           1  0.207
7           1  0.204
```

This is just a matrix so we can use indexing like normal in **R**.

```
> model.matrix(regression.1)[500:508,]
```

```
    (Intercept) tarsus
520           1  0.200
521           1  0.199
522           1  0.199
523           1  0.198
```

```
524           1  0.196
525           1  0.196
526           1  0.195
527           1  0.194
528           1  0.190
```

INTERPRETING THE MODEL OUTPUT

Let us first look at the estimated coefficients from this regression. We use the `coef()` function.

```
> coef(regression.1)
```

```
(Intercept)      tarsus
       6.08       26.92
```

These represent the intercept and the slope of the model. However (and we will return to this), the intercept tells us the expected number of `SCT` when `tarsus=0`, which is clearly not a sensible value. The tarsus could not have a zero length, and how could sex comb teeth be present on a structure with a zero length?

We should also look at the confidence intervals for our estimated slope and intercept. There is a handy function `confint()` to get the 95% confidence intervals for these parameters.

```
> confint(regression.1)
```

```
             2.5 % 97.5 %
(Intercept)  5.35   6.82
tarsus      23.02  30.81
```

One thing that is clear from these values, the 95% CI do not overlap with zero, consistent with a so-called 'statistically significant' effect.

As we often do with **R**, we will use the `summary()` function. However before we do for the linear model, I want to make a slight detour for a discussion about *generic functions* in **R**. When you call `summary()`, the first thing it does is look at the class of the object, and then it uses a class specific method. What this means is that calling `summary()` actually calls different functions depending upon the class of the object.

If you look up the help for `summary()`, `?summary` under the description it will say "summary is a generic function used to produce result summaries of the results of various model fitting functions. The function invokes particular methods which depend on the class of the first argument." What this means is that if you call `summary()` on a data frame, i.e. `summary(your.data)`, it is actually calling the function `summary.data.frame(your.data)`. If you call `summary(your.model.object)` on a linear model object, `lm(your.model)`, it will actually call the function `summary.lm(your.model.object)`. To get a better sense of this type `methods(summary)` into the R console, and you will see a list of different class specific methods for the `summary()` generic function. This is why summary seems to do so many different things!!! It also means if you want to understand more about `summary.lm` you need to use `?summary.lm`.

Now back to the linear model.

```
> summary(regression.1)

Call:
lm(formula = SCT ~ tarsus, data = dll.data)

Residuals:
   Min     1Q Median     3Q    Max
-4.633 -1.012 -0.123  0.845 11.226

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    6.083      0.374    16.2   <2e-16
tarsus        26.915      1.987    13.6   <2e-16

Residual standard error: 1.55 on 1916 degrees of freedom
Multiple R-squared: 0.0874,        Adjusted R-squared: 0.0869
F-statistic:  184 on 1 and 1916 DF,  p-value: <2e-16
```

There are a couple of useful aspects to this output. In addition to the estimated coefficients for the slope and intercept, we have their standard errors, t-values (which is just the parameter estimate divided by the standard error) and p-values. We can double the standard errors to get approximate confidence intervals but the `confint()` allows us to get these anyways. We also get the quantiles for the residuals. We expect that these should be symmetric in value, with the median close to zero. That is clearly not the case here, so we may want to look further at the residuals (which we will do in a later tutorial).

R^2, also known as the coefficient of determination provides a quantitative assessment of *How much of the observed variation in our response (SCT) can be accounted for by our model?* In this case it is approximately 8%. While this is small, for a single variable such as a measure of length, that is not negligible by any means.

It is also worth interpreting the Residual Standard Error (RSE). The RSE of the model is ~ 1.55, which tells us we can predict the number of SCT on an individual with an accuracy of ~ 1.55 SCT *given this model*. See page 41 of Gelman and Hill for more discussion.

I also think it is generally wise to take a look at the co-variances for the estimated parameters. I am not speaking of co-variation or correlation of the observed values, but of the estimates we just produced for the slope and intercept. These helps you recognize whether the estimated values depend on each other (and suggests some lack of independence between predictors, or other issues in your model that need to be fixed).

```
> vcov(regression.1)

            (Intercept) tarsus
(Intercept)        0.14  -0.74
tarsus            -0.74   3.95
```

Question: How would we get the Standard errors for the estimates from this matrix? I will ask this in class.

To address the degree of co-variation between the estimated slope and intercept, we look at the off-diagonals. Sometimes these are difficult to interpret as co-variances, so we convert this to a correlation matrix using the `cov2cor()` function.

```
> cov2cor(vcov(regression.1))
```

```
            (Intercept) tarsus
(Intercept)       1.000 -0.995
tarsus           -0.995  1.000
```

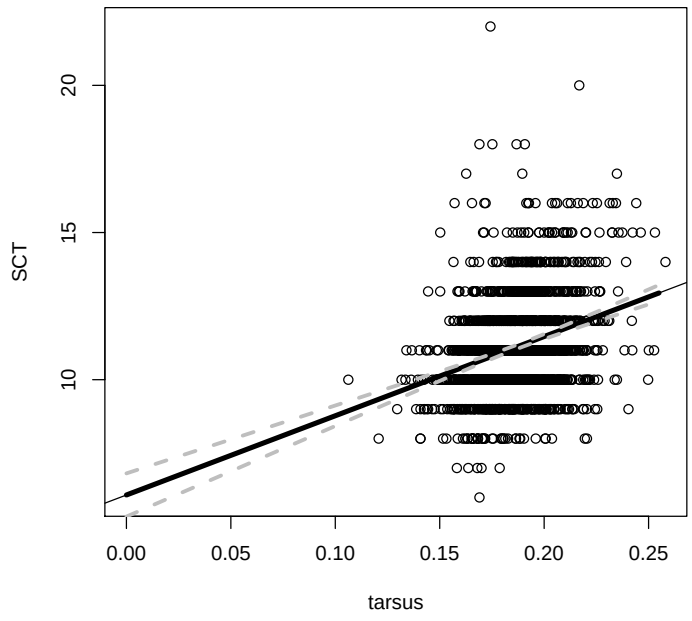
The diagonals are correlations of a parameter estimate on itself, so these have to be one, and are not interesting. We are again focused on the off-diagonal elements. In this case the value is ~ -0.99 which means that our parameter estimates are almost perfectly negatively correlated. This is clearly not good news. This problem is due almost entirely to the fact that we are estimating an intercept way outside the range of our data.

Let's take a look at the estimated parameters again.

```
> coef(regression.1)
```

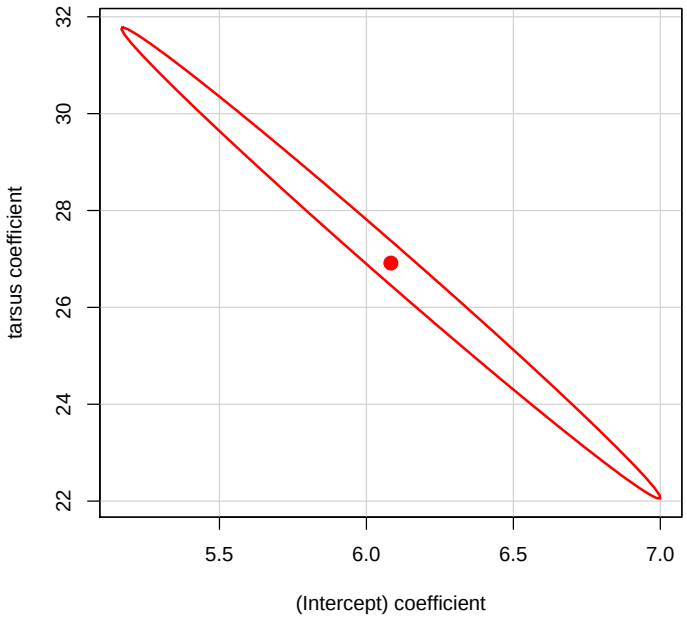
```
(Intercept)      tarsus
       6.08       26.92
```

The intercept 6.08 is interpreted as the number of `SCT` when `tarsus=0`. This is clearly a very silly interpretation, inflates the SE for the intercept, and the co-variation between parameters. This plot demonstrates how crazy this would be. The **R** code for this (including confidence intervals) is in the .R file of the same name with some comments to explain what we have done.



Clearly we are estimating well outside the range of the observed data, including the *confidence bands* around the best fit line. We can also look at a plot in parameter space (the previous plot was in the observation space). We do so by looking at the *confidence ellipse* around the estimated parameters (slope and intercept).

```
> confidenceEllipse(regression.1)
>     # The estimates are highly correlated with one another.
```



This makes it very clear how much the parameter estimates for slope and intercept depend upon each other. If we decrease (by slight changes in the data or estimation procedure) the estimate of the slope, we get a pretty substantial increase in the intercept. So what do we do?

Interpreting coefficients: Sometimes we need to center the covariates

Proper interpretation of the parameters of a model is the most important part of the inferential process (at least once you have a model to fit). This is what allows us to turn a hum drum statistical analysis into Biological inference!!!! However, often the "default" way we code our data and our analyses does not allow for easy interpretation. In this next part we are going to consider some simple things to make it easier to interpret our data.

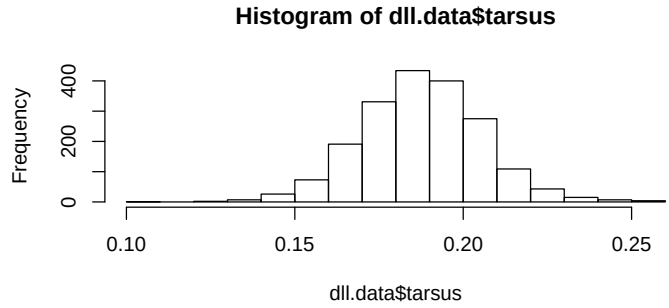
Not only is the intercept uninterpretable, but the CI's are really big at the intercept when `tarsus=0`. The fact that we are trying to estimate a parameter

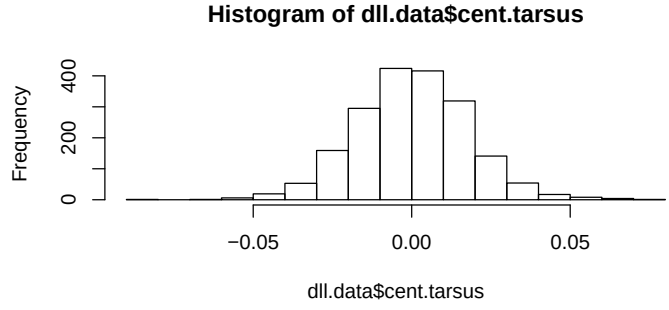
WAY OUTSIDE the range of our data is causing the highly inflated SE, and the covariance between parameter estimates. As we discussed in class, one simple way of dealing this is to center the covariates (in this case, just tarsus length), to help interpretability of the intercept. This will (as we will see) also get rid of the correlation between parameter estimates and the inflated standard error for the intercept. So let's center the data.

```
> dll.data$cent.tarsus <- dll.data$tarsus - mean(dll.data$tarsus)
```

You can also use the `scale(your.x, center=T, scale=F)`. If you keep `scale=T`, then it will also rescale the variable by its standard deviation (which we do not want at the moment).

```
> par(mfrow=c(2,1))
> hist(dll.data$tarsus, breaks=15)
> hist(dll.data$cent.tarsus, breaks=15) #just changes the mean
```





The mean changes as we expect

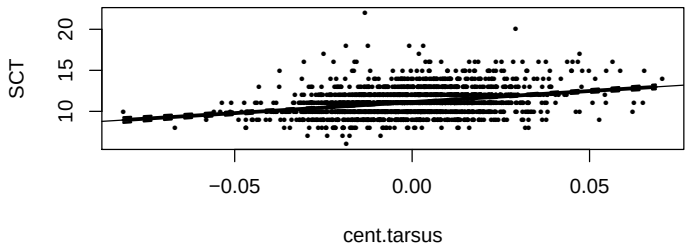
```
> mean(dll.data$tarsus)

[1] 0.188

> mean(dll.data$cent.tarsus)

[1] 2.06e-18
```

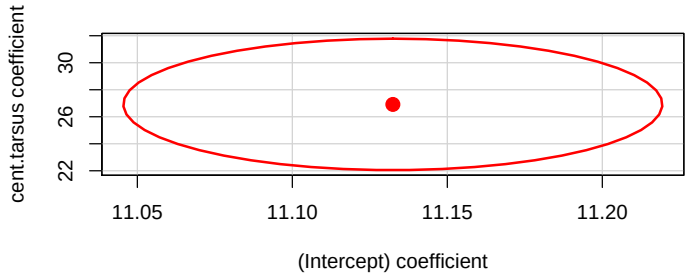
However the standard deviation does not change.

```
> sd(dll.data$tarsus)

[1] 0.0179

> sd(dll.data$cent.tarsus)

[1] 0.0179
```

We now re-run the model with the centered co-variate.

```
> regression.2 <-lm(SCT ~ cent.tarsus, data=dll.data)
```





We can see that we are estimating within the range of our data now. We can also see that the correlation between the parameter estimates is ~ 0.

```
> cov2cor(vcov(regression.2))

             (Intercept) cent.tarsus
(Intercept)      1.0e+00    -5.5e-16
cent.tarsus     -5.5e-16     1.0e+00
```

So how do we interpret the coefficients with the centered data? First off, you will notice that only the estimate of the intercept and its standard error have changed for this model.

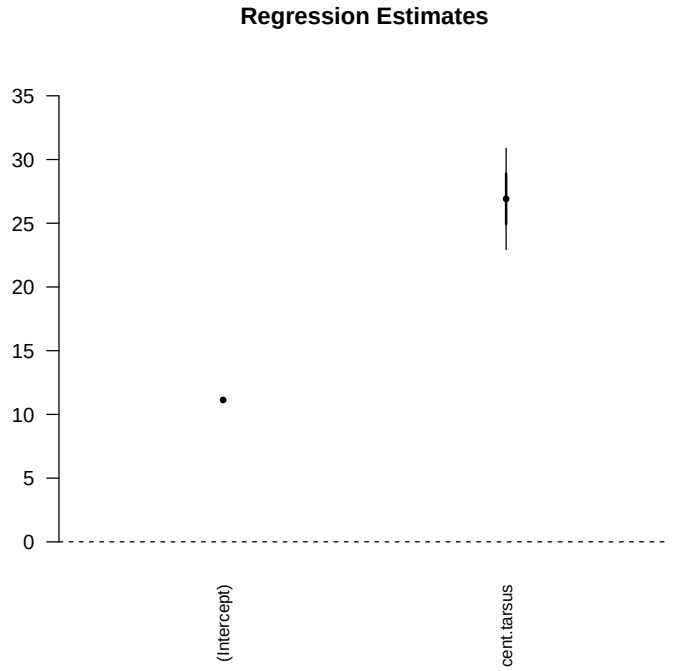
```
> summary(regression.2)
```

```
Call:
lm(formula = SCT ~ cent.tarsus, data = dll.data)

Residuals:
   Min     1Q Median     3Q    Max
-4.633 -1.012 -0.123  0.845 11.226

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  11.1324     0.0355   313.6   <2e-16
cent.tarsus  26.9151     1.9867    13.6   <2e-16

Residual standard error: 1.55 on 1916 degrees of freedom
Multiple R-squared: 0.0874,        Adjusted R-squared: 0.0869
F-statistic:  184 on 1 and 1916 DF,  p-value: <2e-16
```

The intercept now tells us what the expected number of `SCT` is when tarsus is at its mean, i.e. 11.1. You can also see that the standard error associated with the intercept is much smaller.

We can use the `coefplot()` in the arm library to take a quick look at our estimates. The thick error bars represent one standard error, and the thinner bars are $2X$ standard errors.

```
> coefplot(regression.2, intercept=T, vertical=F, ylim=c(0, 35))
```



Interpreting coefficients: Making sense of the slope

One thing to keep in mind is what the estimated slope 26.9 is telling us. It is saying that for every increase by "1" unit of `tarsus` (of whatever unit `tarsus` is measured in), the number of `SCT` is predicted to increase by 26.9. However, let us look at the range of tarsus values.

```
> quantile(dll.data$tarsus)
```

```
   0%   25%   50%   75%  100%
0.106 0.175 0.188 0.200 0.258
```

The range of tarsus lengths is between $0.10 - 0.258$, so a change of unit length (i.e. from 0.10 to 1.1) goes far outside the biologically plausible range for this variable.

What can we do to interpret the slope? One thing you can do is just interpret carefully. Instead of thinking about what happens if we increase tarsus length by 1, we can instead think about what we would predict if tarsus length increased by 0.1 (we would predict an increase of $26.9/10 = 2.69$ `SCT` for a 0.1 increase in `tarsus`). This is much more reasonable, and easier to explain.

In this instance you can think about how the range of values for tarsus length is

```
> range(dll.data$tarsus)
```

```
[1] 0.106 0.258
```

or

```
> max(dll.data$tarsus) - min(dll.data$tarsus)
```

```
[1] 0.152
```

Which is $\sim$ 0.15mm. So we can just calculate $0.15 * 26.9$, the slope times the range of values. So over the whole range of values for `tarsus` we increase `SCT` by 4.035. Is this a lot? Let's compare it to the standard deviation for the observed sample.

```
> sd(dll.data$SCT)
```

```
[1] 1.63
```

It is $\sim 2*$ greater than the sd for `SCT,` suggesting that `tarsus` length is probably a biologically (as well as statistically) significant covariate for accounting for variation for `SCT`. In combination with R^2, I find I am usually able to think about the biological importance of my variables in this manner. We will go through a more formal analysis for effect sizes in the coming weeks.

It is also important to remember that there is uncertainty in your point estimate, so it is worth doing this for the confidence intervals.

```
> confint(regression.2)
```

```
            2.5 % 97.5 %
(Intercept)  11.1   11.2
cent.tarsus  23.0   30.8
```

So the lower bounds for the estimate is $0.15 * 23.01 \sim 3.5$ `SCT`. The upper bound is $0.15*30.8 \sim 4.62$ `SCT`. Neither of these really changes our interpretation of the biological importance of tarsus length as a covariate.

Interpreting the slope II: You could also recode tarsus length to micrometres instead of mm

We could also just rescale our `tarsus` measures from milli-metres to micro-metres.

```
> dll.data$cent.tarsus.microM <- 1000*dll.data$cent.tarsus
> regression.3 <- lm(SCT ~ cent.tarsus.microM, data=dll.data )
> coef(regression.3)

      (Intercept) cent.tarsus.microM
          11.1324             0.0269
```

This will not change anything other than the estimate for the slope and its SE (both 1000 times smaller). This may make it easier to interpret, as a unit increase in `tarsus` length (1 microM), increases expected number of `SCT` by ~ 0.0269.

Interpreting the slope III: Data normalizaition

You could also normalize the data by dividing all values of `tarsus` by `sd(tarsus)`. If you do this on the centered data, it is called unit standardized data.

You can also accomplish this using `scale(x, center=T, scale=T)`. Note that `scale=T` now.

If you normalize/standardize like this then you are no longer thinking in units of the measured variable, but in changes in units of standard deviation. This can be very useful for covariates that are difficult to compare (say nutrient concentration and water temperature). However it does add some complication to interpretation. I should also point out that several people (David Houle being a strong member of this community) suggest that more often scaling by the mean, not the standard deviation makes sense. I am happy to share references if you are interested. This form of re-scaling of data (either mean or sd) is very commonly done when estimating natural selection, so that estimates can be compared across studies.

Let's take a look how it influences our estimates.

```
> dll.data$tarsus.std <- as.numeric(scale(dll.data$tarsus))
> regression.4 <- lm(SCT ~ tarsus.std, data=dll.data )
> summary(regression.4)$coef[,1:2]

            Estimate Std. Error
(Intercept)   11.132     0.0355
tarsus.std     0.481     0.0355
```

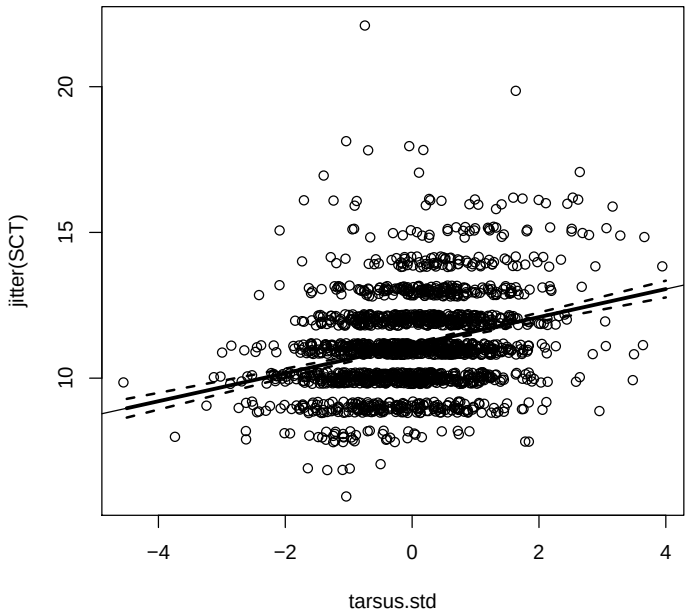
The estimate for the slope and its SE have changed... What do they mean? They are now in units of standard deviations of the `tarsus`. I find it helps to think about the range of new values.

```
> range(dll.data$tarsus.std)

[1] -4.55  3.94
```

This is now in terms of sd of the original variable (`tarsus` length) so the min value is -4.5 sd from the mean of `tarsus` length, and max $\sim$ 3.9 sd from the mean. But how much is a sd unit for `tarsus`?

```
> sd(dll.data$tarsus)

[1] 0.0179
```

So a sd "unit" is $\sim$ 0.018 mm of `tarsus` length. The slope is now interpreted as saying for every increase of 1 sd in `tarsus` length, `SCT` number increases by $\sim$ 0.48



As a note. We will start to build complex models. Especially once we build them in a MLE framework, it is useful to understand `formula` objects in **R**. `?formula` provides a useful example of how to do this with `paste()`, and I will use that.