- Date: August 2014

LOADING IN PACKAGES, DATA FOR ANALYSIS

First we need to load in "packages" from our "library." Packages are groups of functions and code that the R community at large writes. They allow us to do specific things. If R doesn't have a function to do something you want, then feasibly you could write it yourself, but then share it with the community as a package.

To download and install a package, you can go to "Packages & Data", then select "Package Installer." Make sure "CRAN (binaries)" is selected. Then in the search bar type "lme4", as this is a package that we need to download. Hit "Get List". In the results you should see the "lme4" package. Select "lme4", and then below this window you should see a check-box "Install dependencies". Check the box, as this will also install any ancillary functions that lme4 requires to run. Also make sure that install location "At System Level (in R framework) is selected. Then select"Install Selected" and installation should proceed.

You should also download the packages "ggplot2" and "languageR" as you did for "lme4" above

Now, we are ready to "load in" the packages we will be using. To load in a package, we use the function library() and put as the argument the package we would like to load.

```
library(lme4)
```

```
Warning: package 'lme4' was built under R version 3.3.2
```

```
Loading required package: Matrix
```

```
library(ggplot2)
```

```
Warning: package 'ggplot2' was built under R version 3.3.2
```

```
library(languageR)
```

Not only can we load in functions that others have written using packages and the library() function, but we can write our own functions in an R session itself. Below is a useful function written by Julin. We will use it later. But for now, let's make this function. The new function we will create is called "modelcheck". You can see that a new function is created with the "function" command. Select and execute the below to make the function modelcheck().

```
modelcheck <- function(model,h=8,w=10.5) {## because plot(lmer.obj) doesn't work
    rs <- residuals(model)
    fv <- fitted(model)
    quartz(h=h,w=w)
    plot(rs~fv)
    quartz(h=h,w=w)
    plot(sqrt(abs(rs))~fv)
    quartz(h=h,w=w)
    qqnorm(rs)
    qqline(rs)
    }
```

OK, now we are ready to load in the data that we will analyze. You can read in multiple file types into R, but today we will be loading in a .txt file using the read.table() function. The resulting dataframe will be called the object "data".

```
stomdata <- read.table("Modeling_example.txt", header=TRUE)
```

Lets just check the names of the columns, and check that everything is working OK with the function summary.

```
names(stomdata)
```

```
[1] "plant"     "abs_stom"  "epi_count" "il"        "row"       "tray"
[7] "col"
```

```
summary(stomdata)
```

```
    plant        abs_stom       epi_count            il
 A1     :  1   Min.   :13.00   Min.   :22.00   cvm82   : 27
 A2     :  1   1st Qu.:24.50   1st Qu.:34.50   IL_1.1.2: 10
 A20    :  1   Median :27.50   Median :37.50   IL_1.1.3: 10
 A22    :  1   Mean   :27.59   Mean   :37.99   IL_1.3  : 10
 A23    :  1   3rd Qu.:30.50   3rd Qu.:41.00   IL_1.4  : 10
 A24    :  1   Max.   :52.50   Max.   :71.00   IL_10.1 : 10
 (Other):721                                   (Other) :650
     row           tray     col
 B      : 75   M      : 50   A:149
 D      : 74   E      : 48   B:131
 E      : 74   G      : 48   C:147
 G      : 74   J      : 48   D:152
 I      : 73   K      : 48   E:148
 C      : 72   B      : 47
 (Other):285   (Other):438
```

These are the exact types of traits that the tomato interns will be measuring this summer! Let's go over all the different factors in this dataframe, as displayed in the summary() results:

plant: an ID, a name for each individual. This is a "factor" in R, meaning that it is not treated as numerical data, which should make sense, because how would you mathematically treat "A1"? In summary() output is shown how many of each level of the factor is represented. Note that each individual is only represented once.

abs_stom: this is one of the traits we will analyze. It's short for "absolute somtata density on the adaxial side of the cotyledon". It is a count of stomata in a field of view of the microscope. For each individual, this number actually results as an average of two measurements for each individual. This is called "pseudoreplication", and in this case we just averaged the multiple measurements we took for an individual (although there are more sophisticated ways to deal with pseudoreplication). Note that abs_stom is treated numerically rather than a factor, and that in the output of summary we see the min and max values for this term, its median, and quartile values.

epi_count: this trait is very similar to "abs_stom", except that it is a count of the number of pavement cells in a field of view.

il: This is a factor that simply says which IL each individual was. Note that "cvm82", which stands for cultivar M82, is alphanumerically before all other ILs. This is important for modeling later, as all other ILs will be compared relative to M82.

row: This was the row in the tray that the individual plant was in in the lathouse from which epidermal impressions were made. It is a factor, with values "A"-"J"

tray: Which tray the individual comes from. There were trays "A" through "P"

col: This is a factor stating which column in the tray the individuals come from, "A" - "E"

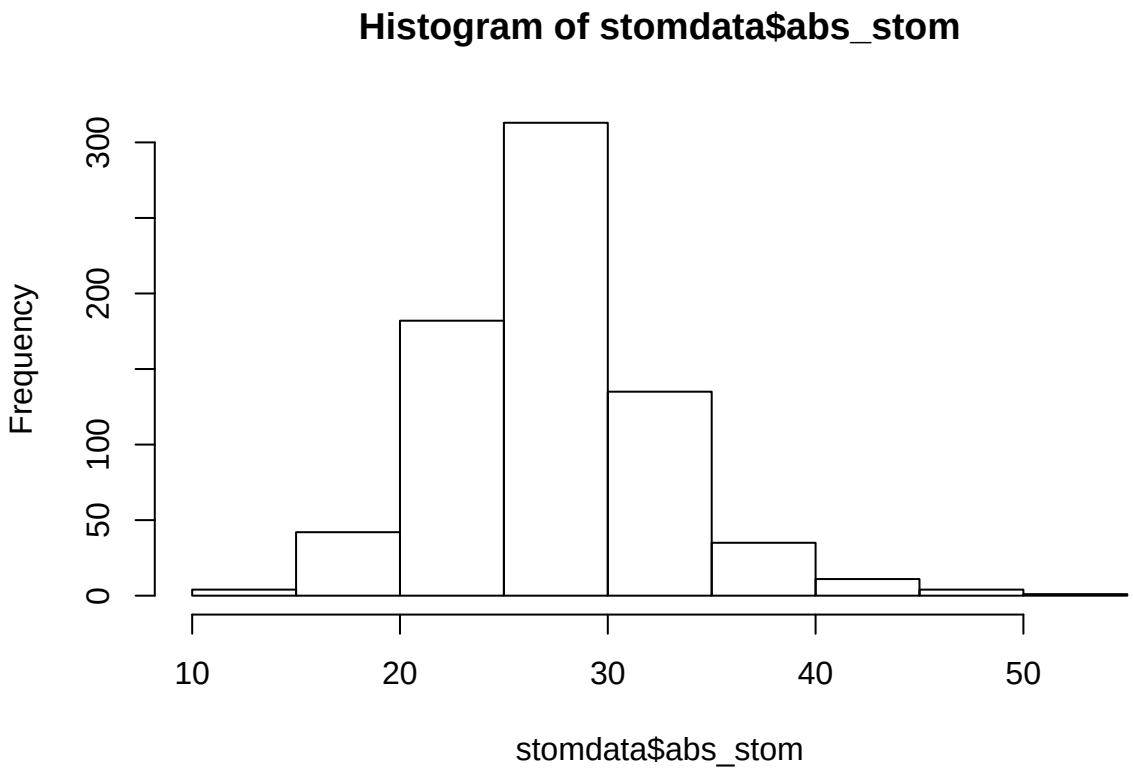
DESCRIPTIVE STATISTICS, TRANSFORMATIONS OF OUR TRAITS

The first step of modeling is to have a feel for your data. Ideally, we like our data to have parametric qualities, which means we make assumptions about the distribution of the trait. A distribution is simply the "spread" and "pattern" of values that you would obtain from a population after repeated measures. Very often, we like our data to be normal, which is a type of distribution that qualitatively looks like a bell curve. Let's see if our data fit this criteria. Additionally, it is ideal that the variance (a measure of the degree that data varies from a mean) in different levels of a factor is equivalent (this is called homoskedasticity, as opposed to heteroskedasticity)

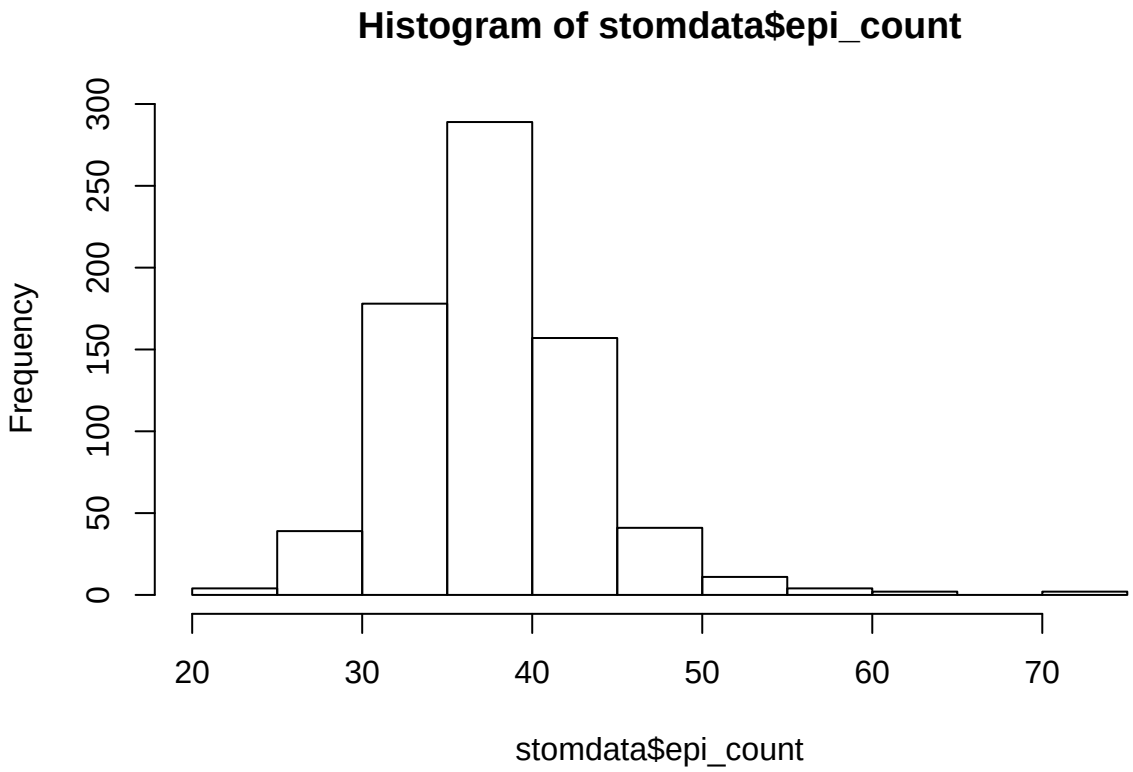
Before getting too far, though, as much as we like our data to be normally distributed before modeling, in the real world, our data will be something similar to but not quite normal. Additionally, its not so important that the data itself is normal, but that the residuals resulting from the fitted model are normally distributed (although having normally distributed data usually helps this goal). Further, the models that we use are robust to deviations from normally distributed data. But still, let's make sure our data is "normally distributed".

Let's make a histrogram of of the abs_stom and epi_count traits using the function hist()

```
hist(stomdata$abs_stom)
```



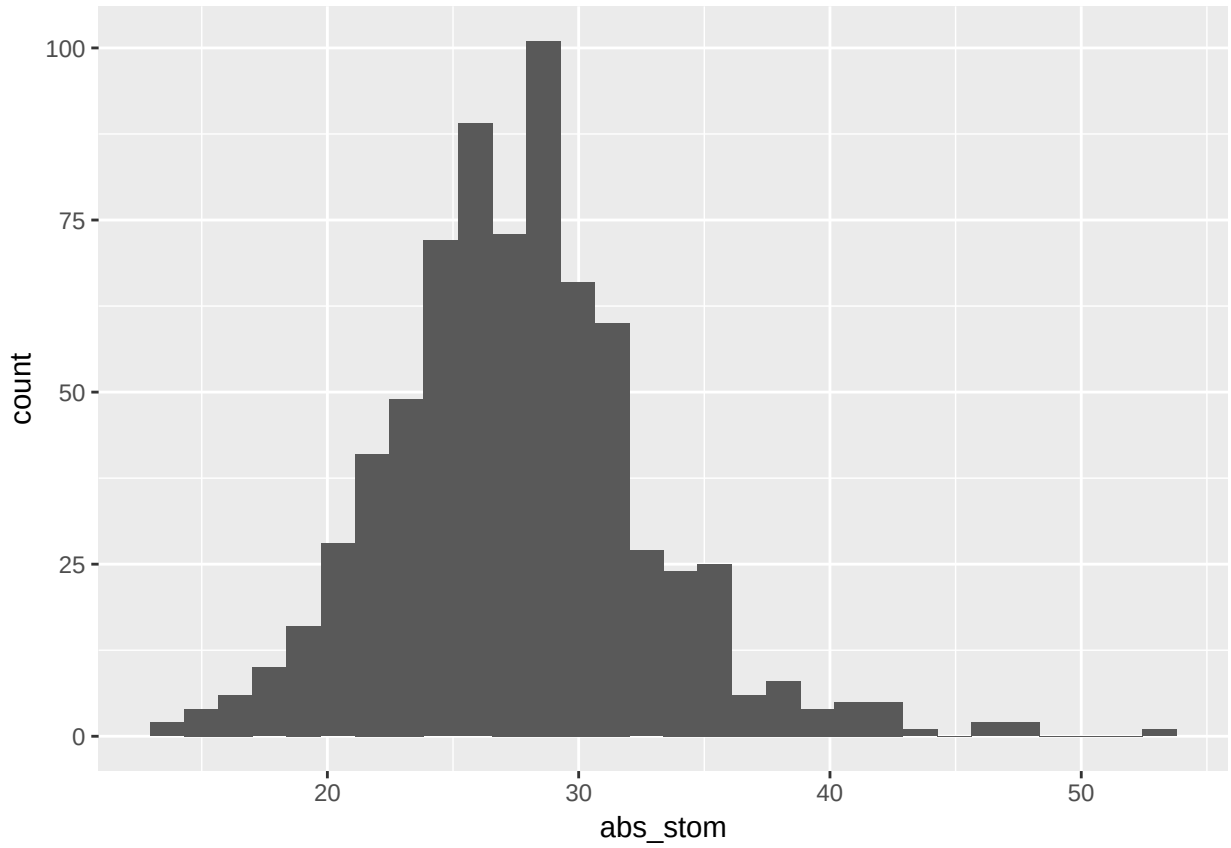

```
hist(stomdata$epi_count)
```



Things look reasonably OK, although there seems to be a little of a "tail", or "skew" of the data towards the right. Personally, I plot everything in ggplot2, so let's make a more aethetic histogram to take a look at these distributions better using functions from the package ggplot2

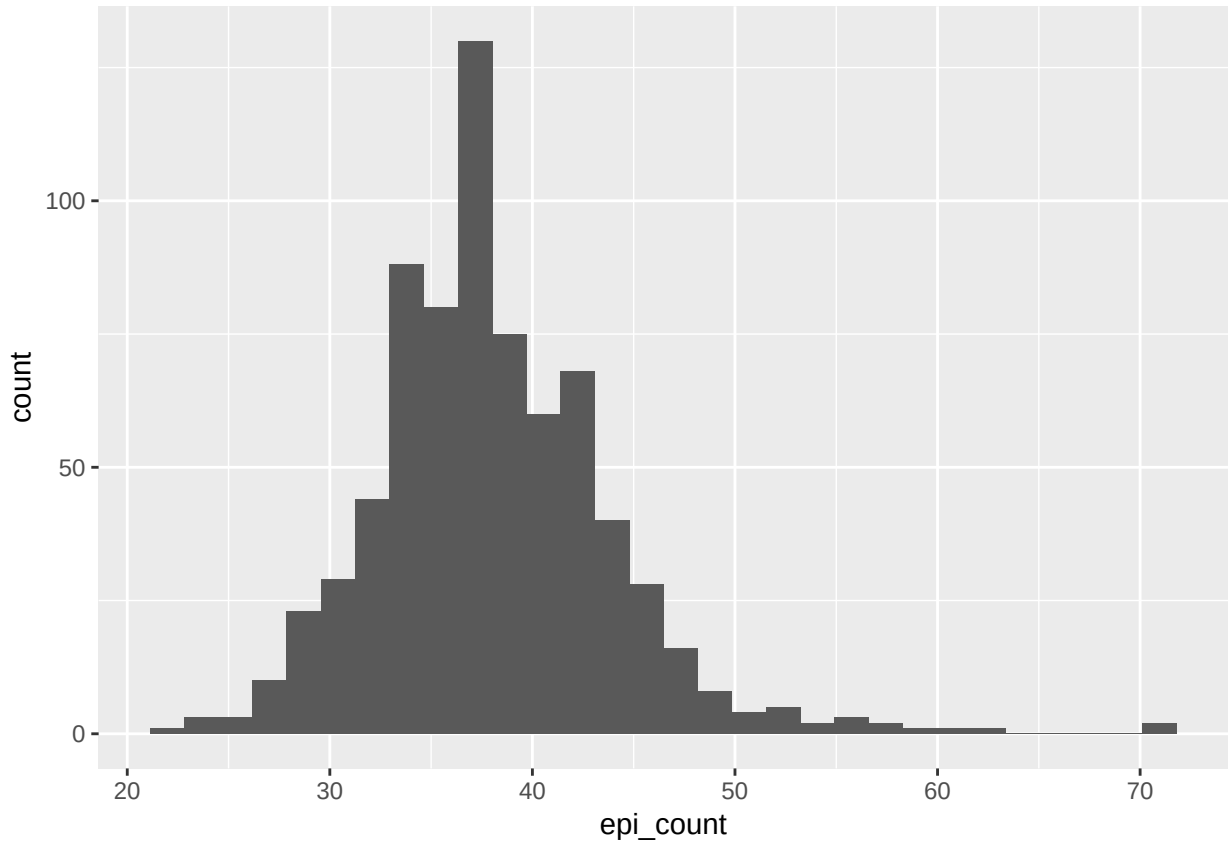
```
qplot(abs_stom, data=stomdata, geom="histogram")
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
qplot(epi_count, data=stomdata, geom="histogram")
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

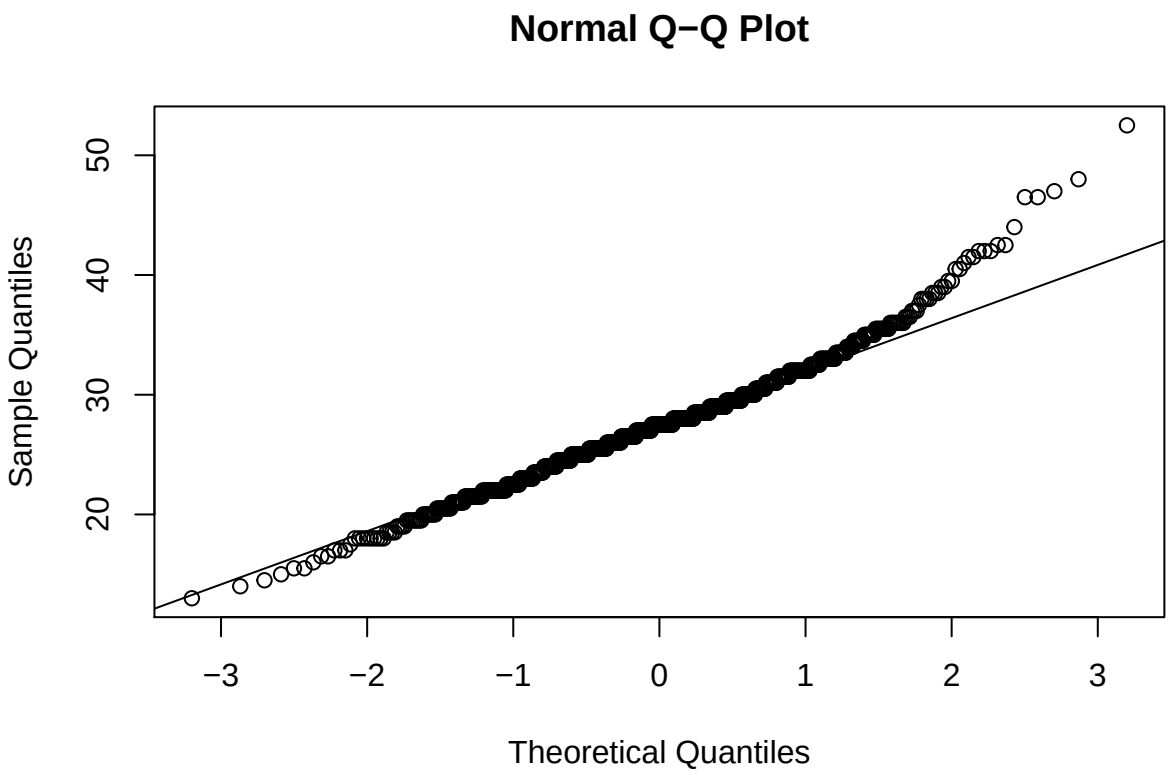


Certainly abs_stom looks more "normal" than epi_count, although it has a slight tail to the right. epi_count is definitely skewed to the right, and has a "shoulder" to the data on the right as well. Still, it has the characteristic bell-curved shape of a normal distribution.

Things are not always what they seem, though. Is there a way that we can better "see" if a distribution is normal? One of the best ways to visualize distributions is using a "QQ plot." A QQ plot compares two different distributions. In this case, we want to compare our distribution to what our data would look like if it were normally distributed. A QQ plot does this by plotting the actual quantiles of our data against the quantiles of our data if it had come from a normal distribution. We expect to see a staight, diagonal line (an x=y line) if our data is perfectly normal: deviations from a straight diagonal are proprotional to the degree that our data is not normal.

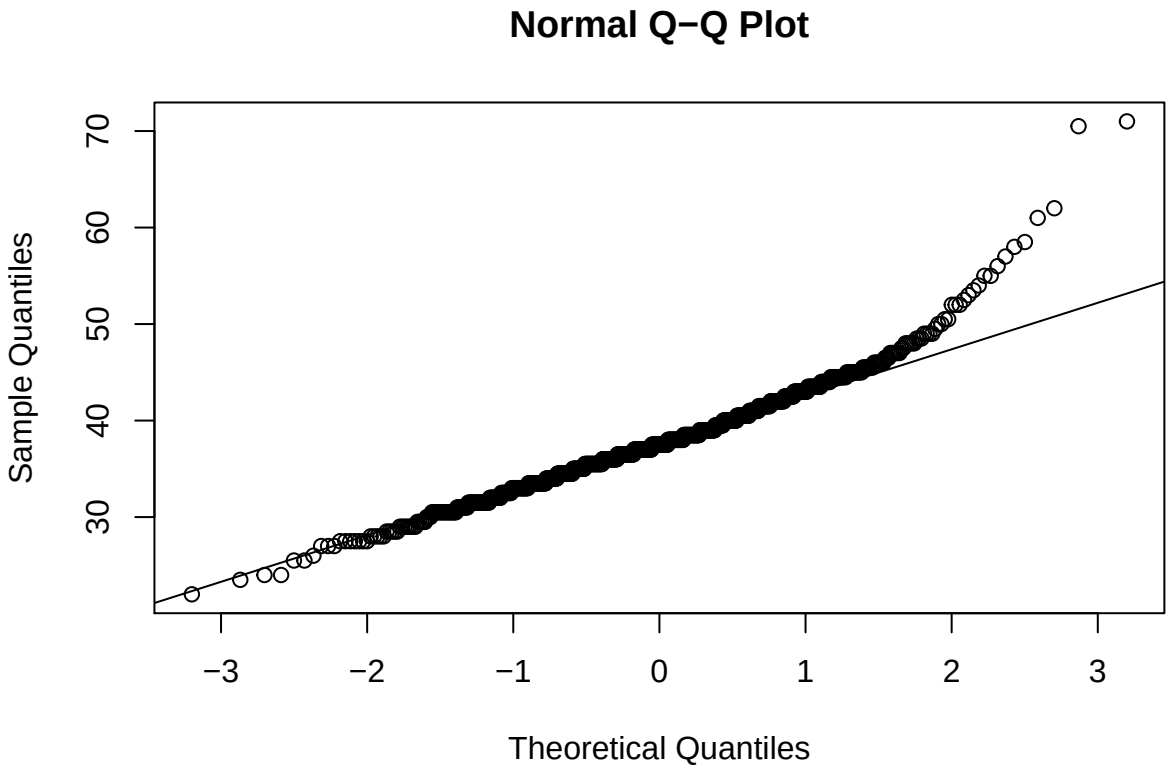
Execute together for a QQ plot of abs_stom

```
qqnorm(stomdata$abs_stom)
qqline(stomdata$abs_stom)
```



Execute together for a QQ plot of epi_count

```
qqnorm(stomdata$epi_count)
qqline(stomdata$epi_count)
```



In the QQ plots for both traits, you can see a "lift" towards the end of the QQ plot, towards the upper righthand corner, compared to a straight, diagonal line. If you remember from the histograms, these points reflect the "skew" towards the right, the right "tail", that we observe in the distributions.

One way to try to "make" your data normal is through transformation. All transformation is is the application of a mathematical function to "transform" your data. Our data is count data (meaning that it is just a "count" of how many things). A tranformation that often helps this sort of data is the square root function. Let's transform our traits by using the square root:

Transform abs_stom:

```
stomdata$trans_abs_stom <- sqrt(stomdata$abs_stom)
```

Transform epi_count:

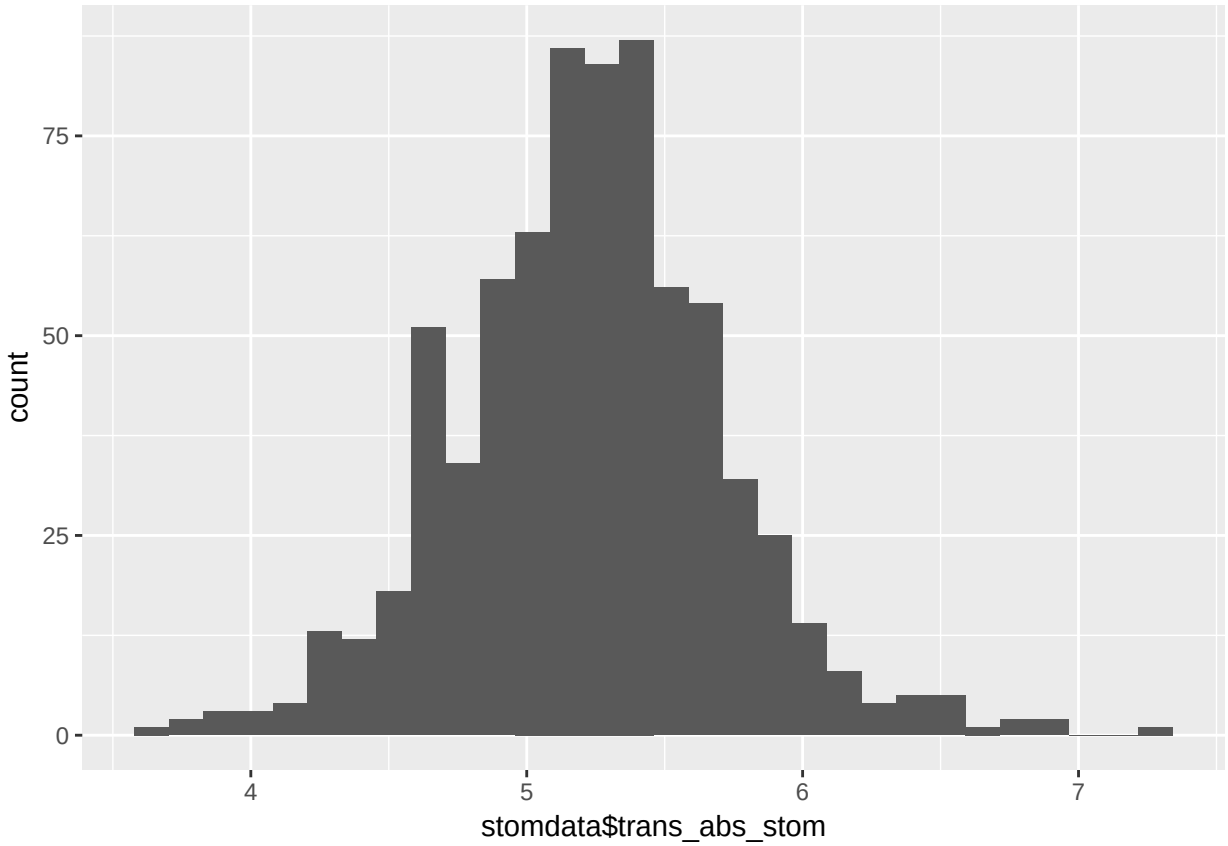
```
stomdata$trans_epi_count <- sqrt(stomdata$epi_count)
```

Did it help? Let's look at the new histograms and QQ plots of our transformed traits:

trans_abs_stom histogram

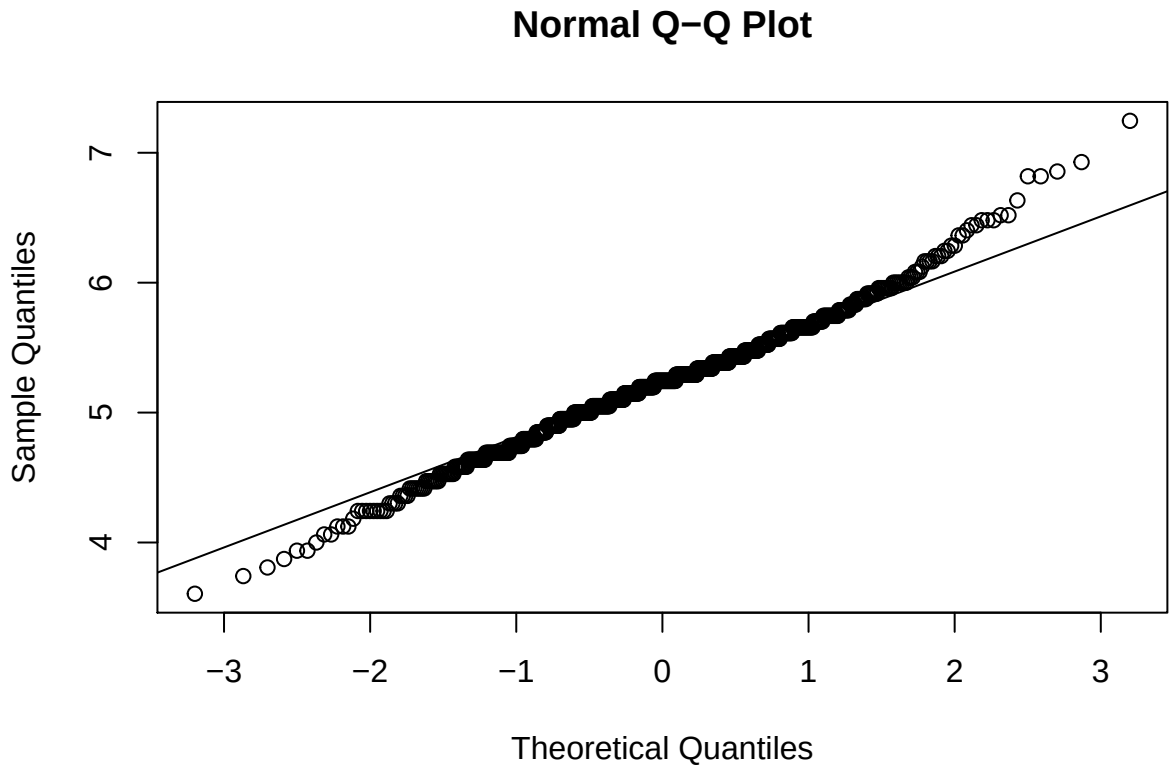
```
qplot(stomdata$trans_abs_stom, data=stomdata, geom="histogram")
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



trans_abs_stom QQ plot

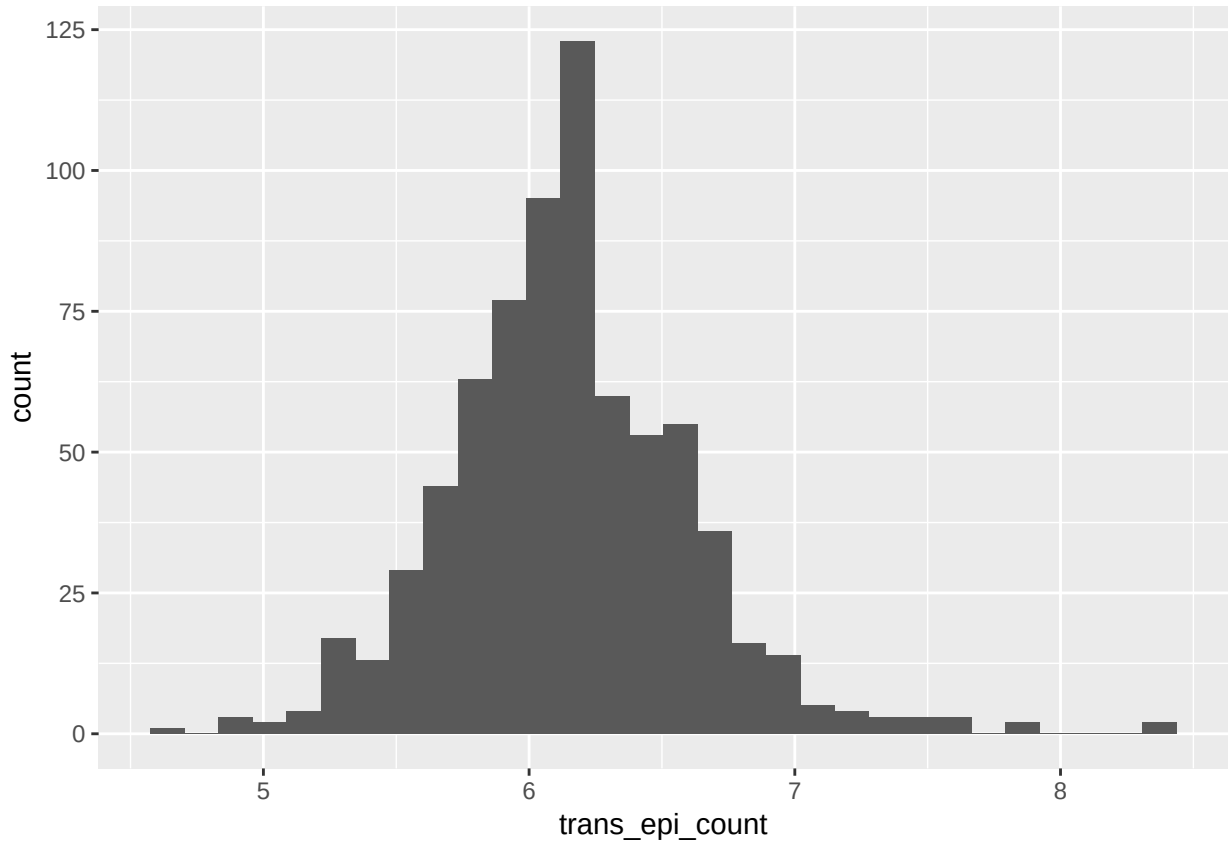
```
qqnorm(stomdata$trans_abs_stom)
qqline(stomdata$trans_abs_stom)
```



trans_epi_count histogram

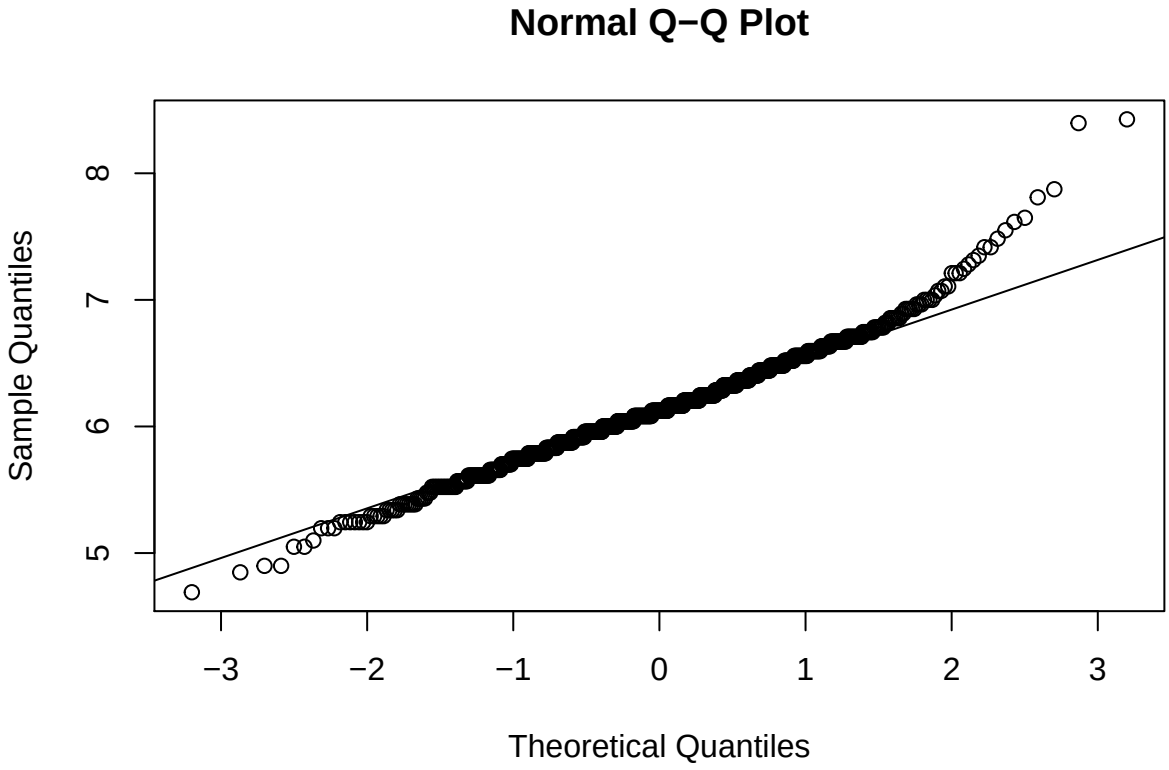
```
qplot(trans_epi_count, data=stomdata, geom="histogram")
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



trans_epi_count QQ plot

```
qqnorm(stomdata$trans_epi_count)
qqline(stomdata$trans_epi_count)
```



Things certainly "look" better for trans_abs_stom. Maybe they look better for epi_count. It'd be nice if we could quantitatively determine if the transformed traits are more "normal" than the untransformed traits. Fortunately, there are statistical tests for normality, like the Shapiro-Wilk normality test. Let's try it.

The p-value states the probability that we would be wrong if we declared that our distribution is NOT normal. Let's compare the p-values for departures from normality for our untranformed and transformed traits:

```
shapiro.test(stomdata$abs_stom)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  stomdata$abs_stom
## W = 0.977, p-value = 2.822e-09
```

```
shapiro.test(stomdata$trans_abs_stom)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  stomdata$trans_abs_stom
## W = 0.98986, p-value = 6.493e-05
```

```
shapiro.test(stomdata$epi_count)
```

```
    Shapiro-Wilk normality test

data:  stomdata$epi_count
W = 0.95471, p-value = 3.637e-14
```

```
shapiro.test(stomdata$trans_epi_count)
```

```
    Shapiro-Wilk normality test

data:  stomdata$trans_epi_count
W = 0.97669, p-value = 2.321e-09
```

By transforming our data with a square root function, the p-values change by a magnitude of 10^4! Certainly transforming our traits makes them more closely approximate a normal distribution! We should use transformed traits from here on out. Note that, still, the trans_epi_count trait is not as normal as we would like. We do our best to try to make things normal before proceeding. Maybe you can try different mathematical transforms to see if you can make the epi_count trait even more normal! Also, again bear in mind that we are most concerned with a normal distribution of residuals AFTER modeling, so maybe our epi_count transformation isn't so bad. Let's see in the next section about modeling.

MIXED-EFFECT LINEAR MODEL SELECTION

We will be using mixed effect linear models to model our traits. These are very much like ANOVA ("ANalysis Of VAriance"), but are called "mixed effect" models because effects can either be "fixed" or "random" effects. Trying to decide whether a term should be fixed or random can be sometimes ambiguous and complicated, but let me try my best to explain what each is below.

FIXED EFFECTS are often the things we are trying to measure. For example, here we are trying to measure differences in the introgression lines (ILs) relative to cvM82, so "il" might be a fixed effect. Another way to think of fixed effects is that they aren't randomly selected from another distribution. In our case, the ILs we are measuring are not drawn from a random population: these are very specific lines that were created for a very specific purpose in a non-random fashion. Finally, another hallmark of a fixed effect is that they are usually not extrapolatable or generalizble: for example, it is not as if our estimation of IL values can be extrapolated to a general population of IL. Again, this gets back to that they are not randomly selected from a population.

RANDOM EFFECTS are often just that . . . the result of "random", "incidental" effects. Random effects are often measured "at random" from a larger distribution. They are often the things we don't want to measure but still affect the value of our traits. In this case, random effects are the "tray" that the plant comes from, and its positional information such as "row" and "col"

We are using the "lme4" package to do our mixed-effect linear models. The function we use to do our mixed-effect linear models is "lmer". Writing out a model is fairly easy! It takes the form of trait ~ fixed_effect1 + fixed_effect2 + (1|rand_effect1) + (1|rand_effect2). You should read "~" as "as a function of." Notice that the fixed effects DON'T have brackets. Also notice that the random effects are both in parentheses and are proceeded by "1|". The "1|" means that the random effect is considered individually. The "|" character means "given" or "by" and the "1" can be replaced with another factor we would like to analyze the random effect in relation to. Interaction effects can also be specified by colons ":". Interactions and groupings in mixed effect models can get quite complicated, and this dataset doesn't deal with them. Consult a real statistician for more complicated modeling of interactions used mixed effect linear models!

OK, now we are ready to write a mixed-effect linear model. There are some rules about models, and one is that we should not include factors in our model that do not significantly explain the variance in our data. If we include such terms, we run the risk of "over-fitting" our data. Overfit data is less extapolatable to the phenomenon we are trying to describe, and extrapolatability is in many ways the entire reason we are trying to model in the first place: we are trying to estimate underlying parameters that are obscured by variability of repeated measures.

One way to select your model is to begin with the "max model" and remove terms that are not significant. The maximum model includes all possible factors that you might consider. You then remove factors, one by one, compare models with and without the factor of interest, and determine if the factor is significant or not and should be removed. This is called "backwards selection."

Let's write the max model for trans_abs_stom.

```
model1 <- lmer(stomdata$trans_abs_stom ~ stomdata$il + (1|stomdata$tray) + (1|stomdata$row) + (1|stomda
```

Now, let's write a model removing just the last term (1|stomdata$col). There is a nice function called update() that you can look into that facilitates this process of removing terms and comparing models, but it is not used here

```
model2 <- lmer(stomdata$trans_abs_stom ~ stomdata$il + (1|stomdata$tray) + (1|stomdata$row) )
```

Now that we have two models, only differing by the single term (1|stomdata$col), we can compare the ability of the models to explain the variance in our data using the anova() function. If removing a term significantly impacts the ability of a model to explain our data, then that term is significant and should remain in our model.

```
anova(model1, model2)
```

```
refitting model(s) with ML (instead of REML)

Data: NULL
Models:
model2: stomdata$trans_abs_stom ~ stomdata$il + (1 | stomdata$tray) +
model2:     (1 | stomdata$row)
model1: stomdata$trans_abs_stom ~ stomdata$il + (1 | stomdata$tray) +
model1:     (1 | stomdata$row) + (1 | stomdata$col)
       Df    AIC    BIC  logLik deviance  Chisq Chi Df Pr(>Chisq)
model2 78 914.68 1272.6 -379.34   758.68
model1 79 912.69 1275.2 -377.35   754.69 3.9875      1    0.04584 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Just barely, (1|stomdata$col) significantly explains our data at p = 0.046. For now, it should remain.

Let's remove other terms one by one from model1. Next is (1|stomdata$row):

```
model3 <- lmer(stomdata$trans_abs_stom ~ stomdata$il + (1|stomdata$tray) + (1|stomdata$col))
anova(model1, model3)
```

```
refitting model(s) with ML (instead of REML)

Data: NULL
Models:
model3: stomdata$trans_abs_stom ~ stomdata$il + (1 | stomdata$tray) +
model3:     (1 | stomdata$col)
model1: stomdata$trans_abs_stom ~ stomdata$il + (1 | stomdata$tray) +
model1:     (1 | stomdata$row) + (1 | stomdata$col)
       Df    AIC    BIC  logLik deviance  Chisq Chi Df Pr(>Chisq)
```

```
model3 78 925.02 1283.0 -384.51   769.02
model1 79 912.69 1275.2 -377.35   754.69 14.326      1  0.0001537 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

the p value of dropping the term (1|stomdata$row) is 0.000156

Let's remove (1|stomdata$tray) from model1:

```
model4 <- lmer(stomdata$trans_abs_stom ~ stomdata$il + (1|stomdata$row) + (1|stomdata$col))
```

```
anova(model1, model4)
```

```
refitting model(s) with ML (instead of REML)

Data: NULL
Models:
model4: stomdata$trans_abs_stom ~ stomdata$il + (1 | stomdata$row) +
model4:     (1 | stomdata$col)
model1: stomdata$trans_abs_stom ~ stomdata$il + (1 | stomdata$tray) +
model1:     (1 | stomdata$row) + (1 | stomdata$col)
       Df    AIC    BIC  logLik deviance  Chisq Chi Df Pr(>Chisq)
model4 78 973.72 1331.7 -408.86   817.72
model1 79 912.69 1275.2 -377.35   754.69 63.025      1  2.041e-15 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

the p value of dropping the term (1|stomdata$tray) is 2.08x10^-15

Finally, let's remove the il term:

```
model5 <- lmer(stomdata$trans_abs_stom ~ (1|stomdata$tray) + (1|stomdata$row) + (1|stomdata$col))
```

```
anova(model1, model5)
```

```
refitting model(s) with ML (instead of REML)

Data: NULL
Models:
model5: stomdata$trans_abs_stom ~ (1 | stomdata$tray) + (1 | stomdata$row) +
model5:     (1 | stomdata$col)
model1: stomdata$trans_abs_stom ~ stomdata$il + (1 | stomdata$tray) +
model1:     (1 | stomdata$row) + (1 | stomdata$col)
       Df    AIC    BIC  logLik deviance  Chisq Chi Df Pr(>Chisq)
model5  5 984.88 1007.8 -487.44   974.88
model1 79 912.69 1275.2 -377.35   754.69 220.19     74  < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

the p vlaue of dropping the term il is <2.2 x 10^-16. Note that il is the most significant term in our model. This is good! It means the effect we are trying to measure, il, explains the variance in our data to a much larger degree than the other things we are not interested in. However, "tray" has a VERY large effect as well.

Tray effects are usually caused by differences in watering and/or position, and demonstrate how susceptible the cellular traits we are measuring are to enviornemntal influences.

Let's recap the significance of each term in our model from the model comparisons that we just performed:

il: $p < 2.2 \times 10^{-16}$

(1|stomdata$tray): $p = 2.08 \times 10^{-15}$

(1|stomdata$row): $p = 0.000156$

(1|stomdata$col): $p = 0.046$

Every one of the factors we had included in our max model is significant! Therefore, all of these factors should remain in our model. The "minimal" model, also called the "final" model, is the one that only includes ONLY significant terms. So, in this case, our max model is our min model. Most the time, there will be some non-significant terms in the max model. So there are a few more points to make about model selection.

1. If a non-significant term had been found in the first round of selection, or multiple non-significant terms discovered, we would have removed the most non-significant term. If there was a tie of "most non-significant", we would have to make a choice to remove a single non-significant term. Then, starting with a model with one less significant term, we would proceed through model selection again removing each term one-by-one. We would proceed through this until there were no more non-significant terms.
2. If we had removed non-significant terms, then the minimal model would have fewer terms than the max model. Another form of model selection you can do in this case is called "forward" selection. In forward selection, instead of beginning with the maximal model, you begin with the minimum model, and you add terms back one-by-one, comparing models with one additional term or not. This is a good way to "double check" the significance of the terms determined from backwards selection (or rather, all the removed terms should remain not significant when added back to the minimal model).
3. Why do we go to all this trouble of backwards and forwards selection, removing or adding terms only one at a time? The reason is because the ability of a term in a model to explain variance in the data is dependent upon other terms in the model. That is, removing terms in different orders and comparing models with give you different significance values! This is why backwards selection removes each term after the other before deciding whether or not to remove a term at all. Model selection can be a tricky process! Know your data well, explore the effects of removing things in different orders, and everything will be fine.

INTERPRETING A MODEL

Finally, after creating our final, minimal model, we should check how well our model fits to our data. That is, how far off is our model from the actual data in general? When asking this question, the key point is that certain groups or levels of data are not less explained by the model than others. Overall, the residuals, or the difference between the actual data from the model-fitted, predicted values of the data, should be normally distributed and show no biases. Julin's function that we created at the beginning of the tutorial, "modelcheck()", is designed to allow us to consider these aspects of the model.

Execute the function modelcheck() for our final, minimal model, model1:

```
modelcheck(model1)
```

Three plots should come up. The first two plot fitted values, which are the values predicted by the model, against the residuals or the square of the residuals. Ideally, there should be no biases in the residuals based on the value of the fitted values. There is a slight tendancy for the residuals to be greater in value the less extreme the fitted value, but this relationship is abolished when looking at the square root of the absolute value of residuals. Maybe the more telling graph is the QQ plot, which again compares the actual distribution

of residuals to their distribution had the data been truly normal. Except for some outliers at the extremes, the residuals appear rather normal. The model seems to have no particular biases in its ability to explain particular trait values (except for a handful of extreme values).

One of the best reasons to model data is to obtain the fitted values and estimates of the values we are trying to measure. In our case, we are interested in measuring the trait values of different ILs. Modeled values give us "ideal", "estimated" values of traits for ILs, if we sufficiently estimated the effects of other sources of systematic bias. To find the estimated IL trait values, simply use the summary() function:

```
summary(model1)
```

Looking under "Fixed effects", which refers to "il" in our case, you can find the estimated trait values for each IL. Remember, we puposefully made cvM82 first alphabetically. The result is that all ILs are compared back to M82. M82 is the "default", and everything else is measured as a change relative to it. Because we are dealing with linear models, M82 literally becomes an intercept, and the "(Intercept) Estimate" is the model fitted trait value for M82. The model predicts that the trans_abs_stom value for M82 is 5.3. Remember, this is a square root transformed value, so the actual trait value for M82 is (5.3)^2. The other ILs have much smaller estimated values than this, by magnitudes. This is because these values represent the DIFFERENCE between the estimated IL value with the intercept/M82. For instance, the predicted transformed trait value for IL1.1 is 5.3 + 3.3x10^-2. The predicted transformed trait value for IL1.1.3 is 5.3-1.3x10^-1.

What if we wanted to ask the question "what is the p value that a particular IL has a different trait value than M82?" This is a complicated question to address in mixed-effect linear models. So complicated, in fact, that people have written an entire package to help address this question . . . the languageR package (which was pointed out to me originally by DanF and Julin). I had some issues using languageR that had to do with the compatibility of the version of R I was using and the version of languageR. You can try to use languageR at your own risk! Hopefully whatever troubles were plaguing me will not hinder you. At this moment (and languageR is currently working for me) I am using R version 2.13.1 (which is very out of date but I keep to use languageR) and languageR version 1.2 (which is also out of date). To retrieve pvalues using languageR, execute the function below (it takes a little while to run):

```
#sepi_count_final_pvals <- pvals.fnc(model1, addPlot=FALSE, ndigits=16)
```

```
#sepi_count_final_pvals$fixed
```

Given are p-values, which should be interpreted as the p-value for the difference between each IL's trait value from that of M82. If we detect a significant difference between the trait of an IL relative to M82, it suggests that the genetic cause of that trait difference must lie within the introgressed region of the IL.

TRY IT YOURSELF!!!

OK, that's it. There is still one more trait for you to try the whole process yourself with: epi_count.

To review, some major steps of fitting models include:

1. Analyze how normal the distribution of your trait is and whether different levels of factors are heteroskedastic.
2. If necessary, try to transform your trait
3. Determine what are fixed effects and random effects
4. Create a maximum model
5. Remove one term at a time, and compare models to the maximum model
6. If at the end of that there are non-significant terms, remove the most non-significant term, or if a tie, just remove one of the most non-significant terms.

7. Repeat steps 5 & 6 until only significant terms remain. This is backwards selection.
8. Validate the insignificance of your terms through a forward selection process, adding terms one by one back to the minimal model
9. Now, with your minimal model, make sure your residuals are normally distributed. If not, try to track down which of your samples are not being fit by the model correctly.
10. Once you have a model that fits your data well, use it for what it is intended: finding estimated, model-fitted values and for finding significant differences in your dataset.

Good luck! Model fitting is usually much more complicated than presented here, so seek out help on the internet and from experience statisticians!