You find it time-consuming to manually format, copy and paste output values to your report or manuscript? That time is over: the psycho package is here for you!

**The data**

Let’s take the example dataset included in the psycho package.

library(psycho)

library(tidyverse)

df <- psycho::emotion %>%

select(Participant\_ID,

Emotion\_Condition,

Subjective\_Valence,

Autobiographical\_Link)

summary(df)

Participant\_ID Emotion\_Condition Subjective\_Valence Autobiographical\_Link

10S : 48 Negative:456 Min. :-100.000 Min. : 0.00

11S : 48 Neutral :456 1st Qu.: -65.104 1st Qu.: 0.00

12S : 48 Median : -2.604 Median : 16.15

13S : 48 Mean : -18.900 Mean : 28.99

14S : 48 3rd Qu.: 7.000 3rd Qu.: 59.90

15S : 48 Max. : 100.000 Max. :100.00

(Other):624 NA's :1

Our dataframe (called df) contains data from several participants, exposed to neutral and negative pictures (the Emotion\_Condition column). Each row corresponds to a single trial. During each trial, the participant had to rate its emotional valence (Subjective\_Valence: positive – negative) experienced during the picture presentation and the amount of personal memories associated with the picture (Autobiographical\_Link).

Our dataframe contains, for each of the 48 trials, 4 variables: the **name of the participant** (Participant\_ID), the **emotion condition** (Emotion\_Condition), the **valence rating** (Subjective\_Valence) and the **Autobiographical Link** (Autobiographical\_Link).

**Fit the model**

**Let’s fit a linear mixed model to predict the autobiographical link with the condition and the subjective valence.**

library(lmerTest)

fit <- lmer(Autobiographical\_Link ~ Emotion\_Condition \* Subjective\_Valence + (1|Participant\_ID), data=df)

summary(fit)

Linear mixed model fit by REML. t-tests use Satterthwaite's method [

lmerModLmerTest]

Formula: Autobiographical\_Link ~ Emotion\_Condition \* Subjective\_Valence +

(1 | Participant\_ID)

Data: df

REML criterion at convergence: 8555.5

Scaled residuals:

Min 1Q Median 3Q Max

-2.2682 -0.6696 -0.2371 0.7052 3.2187

Random effects:

Groups Name Variance Std.Dev.

Participant\_ID (Intercept) 243.1 15.59

Residual 661.4 25.72

Number of obs: 911, groups: Participant\_ID, 19

Fixed effects:

Estimate Std. Error df

(Intercept) 25.52248 4.23991 31.49944

Emotion\_ConditionNeutral 6.13715 2.66993 895.13045

Subjective\_Valence 0.05772 0.03430 898.46616

Emotion\_ConditionNeutral:Subjective\_Valence 0.16140 0.05020 896.26695

t value Pr(>|t|)

(Intercept) 6.020 1.09e-06 \*\*\*

Emotion\_ConditionNeutral 2.299 0.02176 \*

Subjective\_Valence 1.683 0.09280 .

Emotion\_ConditionNeutral:Subjective\_Valence 3.215 0.00135 \*\*

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:

(Intr) Emt\_CN Sbjc\_V

Emtn\_CndtnN -0.459

Sbjctv\_Vlnc 0.455 -0.726

Emtn\_CN:S\_V -0.308 0.301 -0.676

**The analyze function**

The analyze function, available in the psycho package, transforms a model fit object into user-friendly outputs.

results <- analyze(fit, CI = 95)

**Summary**

Summarizing an analyzed object returns a dataframe, that can be easily saved and included in reports. It also includes standardized coefficients, as well as bootstrapped confidence intervals (CI) and effect sizes.

summary(results) %>%

mutate(p = psycho::format\_p(p))

| **Variable** | **Coef** | **SE** | **t** | **df** | **Coef.std** | **SE.std** | **p** | **Effect\_Size** | **CI\_lower** | **CI\_higher** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (Intercept) | 25.52 | 4.24 | 6.02 | 31.50 | 0.00 | 0.00 | < .001\*\*\* | Very Small | 17.16 | 33.93 |
| Emotion\_ConditionNeutral | 6.14 | 2.67 | 2.30 | 895.13 | 0.10 | 0.04 | < .05\* | Very Small | 0.91 | 11.37 |
| Subjective\_Valence | 0.06 | 0.03 | 1.68 | 898.47 | 0.09 | 0.06 | = 0.09° | Very Small | -0.01 | 0.12 |
| Emotion\_ConditionNeutral:Subjective\_Valence | 0.16 | 0.05 | 3.22 | 896.27 | 0.13 | 0.04 | < .01\*\* | Very Small | 0.06 | 0.26 |

**Print**

Moreover, the print method return a nicely formatted output that can be almost directly pasted into the manuscript.

print(results)

The overall model predicting Autobiographical\_Link (formula = Autobiographical\_Link ~ Emotion\_Condition \* Subjective\_Valence + (1 | Participant\_ID)) successfully converged and explained 32.48% of the variance of the endogen (the conditional R2). The variance explained by the fixed effects was of 7.66% (the marginal R2) and the one explained by the random effects of 24.82%. The model's intercept is at 25.52 (SE = 4.24, 95% CI [17.16, 33.93]). Within this model:

- The effect of Emotion\_ConditionNeutral is significant (beta = 6.14, SE = 2.67, 95% CI [0.91, 11.37], t(895.13) = 2.30, p < .05\*) and can be considered as very small (std. beta = 0.098, std. SE = 0.043).

- The effect of Subjective\_Valence is significant (beta = 0.058, SE = 0.034, 95% CI [-0.0097, 0.12], t(898.47) = 1.68, p = 0.09°) and can be considered as very small (std. beta = 0.095, std. SE = 0.056).

- The effect of Emotion\_ConditionNeutral:Subjective\_Valence is significant (beta = 0.16, SE = 0.050, 95% CI [0.063, 0.26], t(896.27) = 3.22, p < .01\*\*) and can be considered as very small (std. beta = 0.13, std. SE = 0.041).

The intercept (the baseline level) corresponds, here, to the negative condition with subjective valence at 0 (the average, as the data is standardized). Compared to that, changing the condition from negative to neutral does not induce any significant change to the outcome. However, in the negative condition, there is a trending linear relationship between valence and autobiographical memories: the more an item is positive the more it is related to memories. Finally, the interaction is significant: the relationship between valence autobiographical memories is stronger (more positive) in the neutral condition.

Code Chunks – Linear Mixed Model

Statistical models generally assume that

1. All observations are independent from each other
2. The distribution of the residuals follows \mathcal{N}(0, \sigma^2), irrespective of the values taken by the dependent variable *y*

When any of the two is not observed, more sophisticated modelling approaches are necessary. Let’s consider two hypothetical problems that violate the two respective assumptions, where *y* denotes the dependent variable:

**A.** Suppose you want to study the relationship between average income (*y*) and the educational level in the population of a town comprising four fully segregated blocks. You will sample 1,000 individuals irrespective of their blocks. If you model as such, you neglect dependencies among observations – individuals from the same block are not independent, yielding residuals that correlate within block.

**B.** Suppose you want to study the relationship between anxiety (*y*) and the levels of triglycerides and uric acid in blood samples from 1,000 people, measured 10 times in the course of 24 hours. If you model as such, you will likely find that the variance of *y*changes over time.

In **A.** we have a problem of dependency caused by spatial correlation, whereas in **B.** we have a problem of heterogeneous variance. As a result, classic linear models cannot help in these hypothetical problems, but both can be addressed using linear mixed-effect models (LMMs). In rigour though, you do not need LMMs to address the second problem.

LMMs are extraordinarily powerful, yet their complexity undermines the appreciation from a broader community. LMMs dissect hierarchical and / or longitudinal (*i.e.* time course) data by separating the variance due to random sampling from the main effects. In essence, on top of the fixed effects normally used in classic linear models, LMMs resolve *i*) correlated residuals by introducing random effects that account for differences among random samples, and *ii*) heterogeneous variance using specific variance functions, thereby improving the estimation accuracy and interpretation of fixed effects in one go.

**Mixed-effect linear models**

Whereas the classic linear model with *n* observational units and *p* predictors has the vectorized form

\mathbf{y} = \mathbf{X}\beta + \epsilon 

with the n \times (p+1) predictor matrix \mathbf{X} , the vector of *p* + 1 coefficient estimates \beta and the *n*-long vectors of the response \mathbf{y} and the residuals \epsilon , LMMs additionally accomodate separate variance components modelled with a set of random effects \mathbf{u} ,

\mathbf{y} = \mathbf{X}\beta + \mathbf{Z}\mathbf{u} + \epsilon 

where \mathbf{Z} and \mathbf{X} are design matrices that jointly represent the set of predictors. Random effects models include only an intercept as the fixed effect and a defined set of random effects. Random effects comprise random intercepts and / or random slopes. Also, random effects might be crossed and nested. In terms of estimation, the classic linear model can be easily solved using the least-squares method. For the LMM, however, we need methods that rather than estimating predict \mathbf{u} , such as maximum likelihood (ML) and restricted maximum likelihood (REML). Bear in mind that unlike ML, REML assumes that the fixed effects are not known, hence it is comparatively unbiased for more details). Unfortunately, LMMs too have underlying assumptions – **both residuals and random effects should be normally distributed**. Residuals in particular should also have a uniform variance over different values of the dependent variable, exactly as assumed in a classic linear model.

One of the most common doubts concerning LMMs is determining whether a variable is a random or fixed. First of all, an effect might be fixed, random or even both simultaneously – it largely depends on how you approach a given problem. Generally, you should consider all factors that qualify as sampling from a population as random effects (*e.g.* individuals in repeated measurements, cities within countries, field trials, plots, blocks, batches) and everything else as fixed. As a rule of thumb, *i*) factors with fewer than 5 levels should be considered fixed and conversely *ii*) factors with numerous levels should be considered random effects in order to increase the accuracy in the estimation of variance. Both points relate to the LMM assumption of having normally distributed random effects.

Best linear unbiased estimators (BLUEs) and predictors (BLUPs) correspond to the values of fixed and random effects, respectively. The usage of the so-called genomic BLUPs (GBLUPs), for instance, elucidates the genetic merit of animal or plant genotypes that are regarded as random effects when trial conditions, *e.g.* location and year of trials are considered fixed. However, many studies sought the opposite, *i.e.* using breeding values as fixed effects and trial conditions as random, when the levels of the latter outnumber the former, chiefly because of point *ii*) outlined above. In GWAS, LMMs aid in teasing out population structure from the phenotypic measures.

**Let’s get started with R**

We will follow a structure similar to the 10-step protocol outlined: *i*) fit a full ordinary least squares model and run the diagnostics in order to understand if and what is faulty about its fit; *ii*) fit an identical generalized linear model (GLM) estimated with ML, to serve as a reference for subsequent LMMs; *iii*) deploy the first LMM by introducing random effects and compare to the GLM, optimize the random structure in subsequent LMMs; *iv*) optimize the fixed structure by determining the significant of fixed effects, always using ML estimation; finally, *v*) use REML estimation on the optimal model and interpret the results.

You need to have

nlme

and

lme4

installed to proceed. We will firstly examine the structure of the Arabidopsis dataset.

# Install (if necessary) and load nlme and lme4

library(nlme)

library(lme4)

# Load dataset, inspect size and additional info

data(Arabidopsis)

dim(Arabidopsis) # 625 observations, 8 variables

?Arabidopsis

attach(Arabidopsis)

The Arabidopsis dataset describes 625 plants with respect to the the following 8 variables (transcript from R):

reg

region: a factor with 3 levels

NL

(Netherlands),

SP

(Spain),

SW

(Sweden)

popu

population: a factor with the form

n.R

representing a population in region

R

gen

genotype: a factor with 24 (numeric-valued) levels

rack

a nuisance factor with 2 levels, one for each of two greenhouse racks

nutrient

fertilization treatment/nutrient level (1, minimal nutrients or 8, added nutrients)

amd

simulated herbivory or “clipping” (apical meristem damage):

unclipped

(baseline) or

clipped

status

a nuisance factor for germination method (

Normal

,

Petri.Plate

, or

Transplant

)

total.fruits

total fruit set per plant (integer), henceforth TFPP for short.

We will now visualise the absolute frequencies in all 7 factors and the distribution for TFPP.

# Overview of the variables

par(mfrow = c(2,4))

barplot(table(reg), ylab = "Frequency", main = "Region")

barplot(table(popu), ylab = "Frequency", main = "Population")

barplot(table(gen), ylab = "Frequency", las = 2, main = "Genotype")

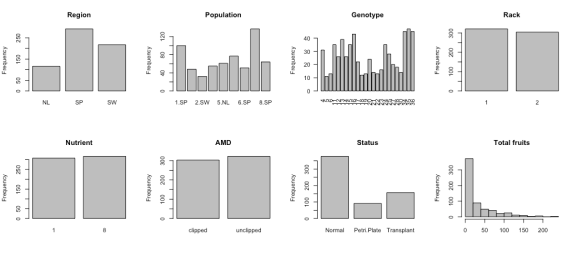
barplot(table(rack), ylab = "Frequency", main = "Rack")

barplot(table(nutrient), ylab = "Frequency", main = "Nutrient")

barplot(table(amd), ylab = "Frequency", main = "AMD")

barplot(table(status), ylab = "Frequency", main = "Status")

hist(total.fruits, col = "grey", main = "Total fruits", xlab = NULL)



The frequencies are overall balanced, perhaps except for

status

 (*i.e.* germination method). Genotype, greenhouse rack and fertilizer are incorrectly interpreted as quantitative variables. In addition, the distribution of TFPP is right-skewed. As such, we will encode these three variables as categorical variables and log-transform TFPP to approximate a Gaussian distribution (natural logarithm).

# Transform the three factor variables gen, rack and nutrient

Arabidopsis[,c("gen","rack","nutrient")] <- lapply(Arabidopsis[,c("gen","rack","nutrient")], factor)

str(Arabidopsis)

# Re-attach after correction, ignore warnings

attach(Arabidopsis)

# Add 1 to total fruits, otherwise log of 0 will prompt error

total.fruits <- log(1 + total.fruits)

A closer look into the variables shows that each genotype is exclusive to a single region. The data contain no missing values.

# gen x popu table

table(gen, popu)

# Any NAs?

any(is.na(Arabidopsis)) # FALSE

**Formula syntax basics**

At this point I hope you are familiar with the formula syntax in R. Note that interaction terms are denoted by

:

 and fully crossed effects with

\*

,so that

A\*B = A + B + A:B

. The following code example

lm(y ~ x1 + x2\*x3)

builds a linear model of *y*using x_1 , x_2 , x_3  and the interaction between x_2  and x_3 . In case you want to perform arithmetic operations inside the formula, use the function

I

. You can also introduce polynomial terms with the function

poly

. One handy trick I use to expand all pairwise interactions among predictors is

model.matrix(y ~ .\*., data = X)

provided a matrix *X* that gathers all predictors and *y.*You can also simply use

.\*.

 inside the

lm

 call, however you will likely need to preprocess the resulting interaction terms.

While the syntax of

lme

is identical to

lm

 for fixed effects, its random effects are specified under the argument

random

 as

random = ~intercept + fixed effect | random effect

and can be nested using

/

. In the following example

random = ~1 + C | A/B

the random effect B is nested within random effect A, altogether with random intercept and slope with respect to C. Therefore, not only will the groups defined by A and A/B have different intercepts, they will also be explained by different slight shifts of \beta from the fixed effect C.

**Classic linear model**

Ideally, you should start will a full model (*i.e.* including all independent variables). Here, however, we cannot use all descriptors in the classic linear model since the fit will be singular due to the redundancy in the levels of

reg

 and

popu

. For simplicity I will exclude these alongside

gen

, since it contains a lot of levels and also represents a random sample (from many other extant Arabidopsis genotypes). Additionally, I would rather use

rack

 and 

status

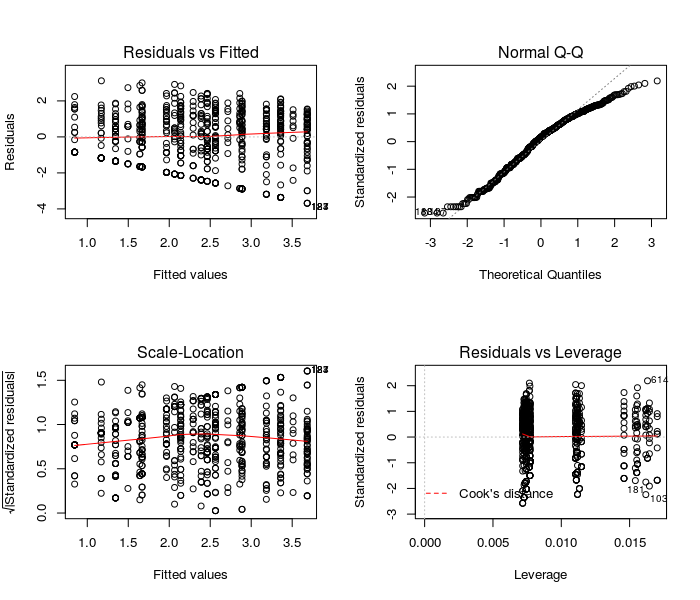
as random effects in the following models but note that having only two and three levels respectively, it is advisable to keep them as fixed.

LM <- lm(total.fruits ~ rack + nutrient + amd + status)

summary(LM)

par(mfrow = c(2,2))

plot(LM)



These diagnostic plots show that the residuals of the classic linear model poorly qualify as normally distributed. Because we have no obvious outliers, the leverage analysis provides acceptable results. We will try to improve the distribution of the residuals using LMMs.

**Generalized linear model**

We need to build a GLM as a benchmark for the subsequent LMMs. This model can be fit without random effects, just like a

lm

 but employing ML or REML estimation, using the

gls

 function. Hence, it can be used as a proper null model with respect to random effects. The GLM is also sufficient to tackle heterogeneous variance in the residuals by leveraging different types of variance and correlation functions, when no random effects are present (see arguments

correlation

 and

weights

).

GLM <- gls(total.fruits ~ rack + nutrient + amd + status,

method = "ML")

summary(GLM)

At this point you might consider comparing the GLM and the classic linear model and note they are identical. Also, you might wonder why are we using LM instead of REML – as hinted in the introduction, REML comparisons are meaningless in LMMs that differ in their fixed effects. Therefore, we will base all of our comparisons on LM and only use the REML estimation on the final, optimal model.

**Optimal random structure**

Let’s fit our first LMM with all fixed effects used in the GLM and introducing

reg

,

popu

,

gen

,

reg/popu

,

reg/gen

,

popu/gen

 and

reg/popu/gen

 as random intercepts, separately.

In order to compare LMMs (and GLM), we can use the function

anova

(note that it does not work for

lmer

 objects) to compute the likelihood ratio test (LRT). This test will determine if the models are significantly different with respect to goodness-of-fit, as weighted by the trade-off between variance explained and degrees-of-freedom. The model fits are also evaluated based on the Akaike (AIC) and Bayesian information criteria (BIC) – the smaller their value, the better the fit.

lmm1 <- lme(total.fruits ~ rack + nutrient + amd + status,

random = ~1|reg, method = "ML")

lmm2 <- lme(total.fruits ~ rack + nutrient + amd + status,

random = ~1|popu, method = "ML")

lmm3 <- lme(total.fruits ~ rack + nutrient + amd + status,

random = ~1|gen, method = "ML")

lmm4 <- lme(total.fruits ~ rack + nutrient + amd + status,

random = ~1|reg/popu, method = "ML")

lmm5 <- lme(total.fruits ~ rack + nutrient + amd + status,

random = ~1|reg/gen, method = "ML")

lmm6 <- lme(total.fruits ~ rack + nutrient + amd + status,

random = ~1|popu/gen, method = "ML")

lmm7 <- lme(total.fruits ~ rack + nutrient + amd + status,

random = ~1|reg/popu/gen, method = "ML")

anova(GLM, lmm1, lmm2, lmm3, lmm4, lmm5, lmm6, lmm7)

We could now base our selection on the AIC, BIC or log-likelihood. Considering most models are undistinguishable with respect to the goodness-of-fit, I will select

lmm6

 and

lmm7

 as the two best models so that we have more of a random structure to look at. Take a look into the distribution of the random effects with

plot(ranef(MODEL))

. We next proceed to incorporate random slopes.

There is the possibility that the different researchers from the different regions might have handled and fertilized plants differently, thereby exerting slightly different impacts. Let’s update

lmm6

 and

lmm7

to include random slopes with respect to

nutrient

. We first need to setup a control setting that ensures the new models converge.

# Set optimization pars

ctrl <- lmeControl(opt="optim")

lmm6.2 <- update(lmm6, .~., random = ~nutrient|popu/gen, control = ctrl)

lmm7.2 <- update(lmm7, .~., random = ~nutrient|reg/popu/gen, control = ctrl)

anova(lmm6, lmm6.2, lmm7, lmm7.2) # both models improved

anova(lmm6.2, lmm7.2) # similar fit; lmm6.2 more parsimonious

summary(lmm6.2)

Assuming a level of significance \alpha = 0.05 , the inclusion of random slopes with respect to

nutrient

 improved both

lmm6

 and

lmm7

. Comparing

lmm6.2

and

lmm7.2

 head-to-head provides no evidence for differences in fit, so we select the simpler model,

lmm6.2

. Let’s check how the random intercepts and slopes distribute in the highest level (*i.e.*

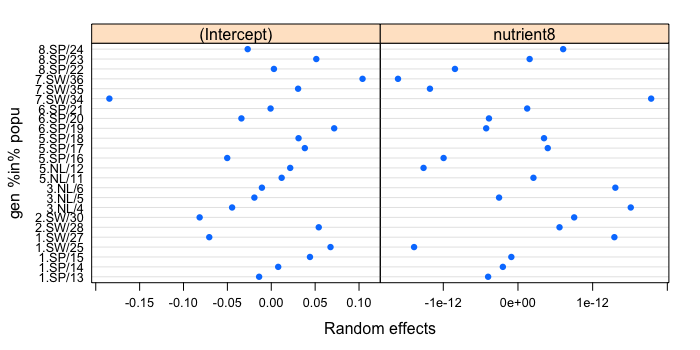
gen

within

popu

).

plot(ranef(lmm6.2, level = 2))



The random intercepts (left) appear to be normally distributed, except for genotype

34

, biased towards negative values. This could warrant repeating the entire analysis without this genotype. The random slopes (right), on the other hand, are rather normally distributed. Interestingly, there is a negative correlation of -0.61 between random intercepts and slopes, suggesting that genotypes with low baseline TFPP tend to respond better to fertilization. Try

plot(ranef(lmm6.2, level = 1))

 to observe the distributions at the level of

popu

 only. Next, we will use QQ plots to compare the residual distributions between the GLM and

lmm6.2

 to gauge the relevance of the random effects.

# QQ plots (drawn to the same scale!)

par(mfrow = c(1,2))

lims <- c(-3.5,3.5)

qqnorm(resid(GLM, type = "normalized"),

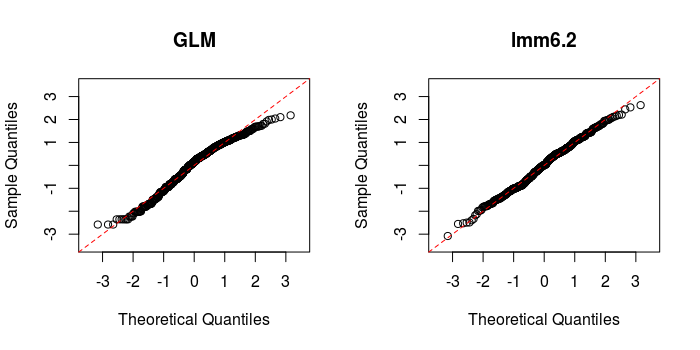
xlim = lims, ylim = lims,main = "GLM")

abline(0,1, col = "red", lty = 2)

qqnorm(resid(lmm6.2, type = "normalized"),

xlim = lims, ylim = lims, main = "lmm6.2")

abline(0,1, col = "red", lty = 2)



The improvement is clear. Bear in mind these results do not change with REML estimation. Try different arrangements of random effects with nesting and random slopes, explore as much as possible!

**Optimal fixed structure**

Now that we are happy with the random structure, we will look into the summary of the optimal model so far (*i.e.*

lmm6.2

) and determine if we need to modify the fixed structure.

summary(lmm6.2)

All effects are significant with \alpha = 0.05 , except for one of the levels from

status

 that represents transplanted plants. Given the significant effect from the other two levels, we will keep

status

 and all current fixed effects. Just for fun, let’s add the interaction term

nutrient:amd

 and see if there is any significant improvement in fit.

lmm8 <- update(lmm6.2, .~. + nutrient:amd)

summary(lmm8)

anova(lmm8, lmm6.2)

The addition of the interaction was non-significant with respect to both \beta and the goodness-of-fit, so we will drop it. Note that it is not a good idea to add new terms after optimizing the random structure, I did so only because otherwise there would be nothing to do with respect to the fixed structure.

**Fit optimal model with REML**

We could play a lot more with different model structures, but to keep it simple let’s finalize the analysis by fitting the

lmm6.2

model using REML and finally identifying and understanding the differences in the main effects caused by the introduction of random effects.

finalModel <- update(lmm6.2, .~., method = "REML")

summary(finalModel)

We will now contrast our REML-fitted final modelagainst a REML-fitted GLM and determine the impact of incorporating random intercept and slope, with respect to

nutrient

, at the level of

popu/gen

. Therefore, both will be given the same fixed effects and estimated using REML.

dev.off() # Reset previous graphical pars

# New GLM, updated from the first by estimating with REML

GLM2 <- update(GLM, .~., method = "REML")

# Plot side by side, beta with respective SEs

plot(coef(GLM2), xlab = "Fixed Effects", ylab = expression(beta), axes = F,

pch = 16, col = "black", ylim = c(-.9,2.2))

stdErrors <- coef(summary(GLM2))[,2]

segments(x0 = 1:6, x1 = 1:6, y0 = coef(GLM2) - stdErrors, y1 = coef(GLM2) + stdErrors,

col = "black")

axis(2)

abline(h = 0, col = "grey", lty = 2)

axis(1, at = 1:6,

labels = c("Intercept", "Rack", "Nutrient (Treated)","AMD (Unclipped)","Status (PP)",

"Status (Transplant)"), cex.axis = .7)

# LMM

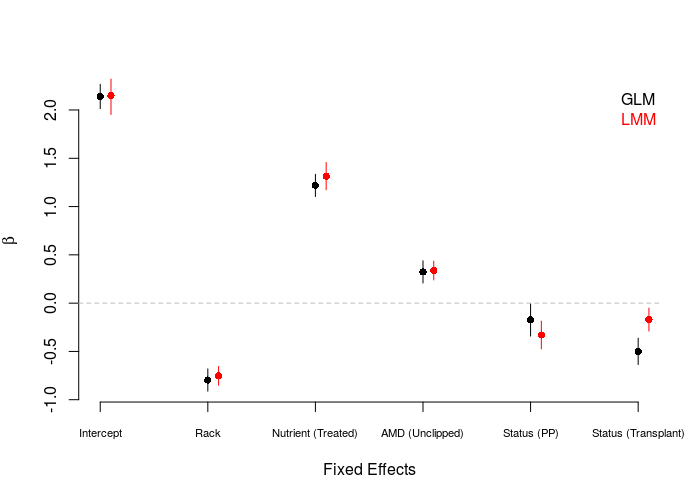
points(1:6 + .1, fixef(finalModel), pch = 16, col = "red")

stdErrorsLMM <- coef(summary(finalModel))[,2]

segments(x0 = 1:6 + .1, x1 = 1:6 + .1, y0 = fixef(finalModel) - stdErrorsLMM, y1 = fixef(finalModel) + stdErrorsLMM, col = "red")

# Legend

legend("topright", legend = c("GLM","LMM"), text.col = c("black","red"), bty = "n")



The figure above depicts the estimated \beta from the different fixed effects, including the intercept, for the GLM (black) and the final LMM (red). Error bars represent the corresponding standard errors (SE). Overall the results are similar but uncover two important differences. First, for all fixed effects except the intercept and

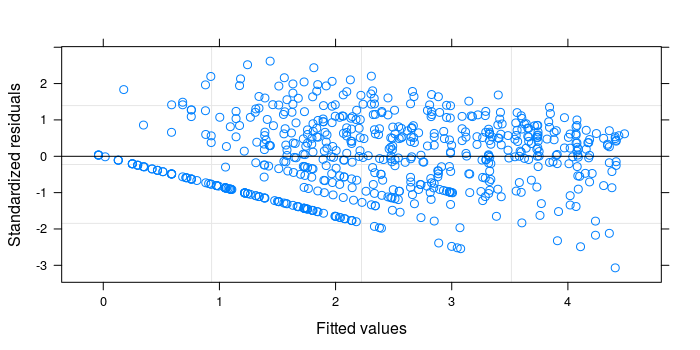
nutrient

, the SE is smaller in the LMM. Second, the relative effects from two levels of

status

 are opposite. With the consideration of random effects, the LMM estimated a more negative effect of culturing in Petri plates on TFPP, and conversely a less negative effect of transplantation.

plot(finalModel)

****

The distribution of the residuals as a function of the predicted TFPP values in the LMM is still similar to the first panel in the diagnostic plots of the classic linear model. The usage of additional predictors and generalized additive models would likely improve it.

**Conclusions**

Now that we account for genotype-within-region random effects, how do we interpret the LMM results?

Plants that were placed in the first rack, left unfertilized, clipped and grown normally have an average TFPP of 2.15. This is the value of the estimated grand mean (*i.e.* intercept), and the predicted TFPP when all other factors and levels do not apply. For example, a plant grown under the same conditions but placed in the second rack will be predicted to have a smaller yield, more precisely of ln(TFPP) = \beta_{intercept} + \beta_{Rack} = 2.15 + (-0.75) = 1.4 . To these reported yield values, we still need to add the random intercepts predicted for region and genotype within region (which are tiny values, by comparison; think of them as a small adjustment). Moreover, we can state that

* Fertilized plants produce more fruits than those kept unfertilized. This was the strongest main effect and represents a very sensible finding.
* Plants grown in the second rack produce less fruits than those in the first rack. This was the second strongest main effect identified. Could this be due to light / water availability?
* Simulated herbivory (AMD) negatively affects fruit yield. This is also a sensible finding – when plants are attacked, more energy is allocated to build up biochemical defence mechanisms against herbivores and pathogens, hence compromising growth and eventually fruit yield.
* Both culturing in Petri plates and transplantation, albeit indistinguishable, negatively affect fruit yield as opposed to normal growth. When conditions are radically changed, plants must adapt swiftly and this comes at a cost as well. Thus, these observations too make perfect sense.
* One important observation is that the genetic contribution to fruit yield, as gauged by

gen

, was normally distributed and adequately modelled as random. One single outlier could eventually be excluded from the analysis. This makes sense assuming plants have evolved universal mechanisms in response to both nutritional status and herbivory that overrule any minor genetic differences among individuals from the same species.

Code Chunks – Format and Interpret Linear Mixed Models

# 1. What is mixed effects modelling and why does it matter?

Ecological and biological data are often complex and messy. We can have different **grouping factors** like populations, species, sites where we collect the data, etc. **Sample sizes** might leave something to be desired too, especially if we are trying to fit complicated models with **many parameters**. On top of that, our data points might **not be truly independent**. For instance, we might be using quadrats within our sites to collect the data (and so there is structure to our data: quadrats are nested within the sites).

This is why **mixed models** were developed, to deal with such messy data and to allow us to use all our data, even when we have low sample sizes, structured data and many covariates to fit. Oh, and on top of all that, mixed models allow us to save degrees of freedom compared to running standard linear models! Sounds good, doesn’t it?

We will cover only linear mixed models here, but if you are trying to “extend” your linear model, fear not: there are generalised linear mixed effects models out there, too.

# 2. Explore the data

We are going to focus on a fictional study system, dragons, so that we don’t have to get too distracted with the specifics of this example. Imagine that we decided to train dragons and so we went out into the mountains and collected data on dragon intelligence (testScore) as a prerequisite. We sampled individuals with a range of body lengths across three sites in eight different mountain ranges. Start by loading the data and having a look at them.

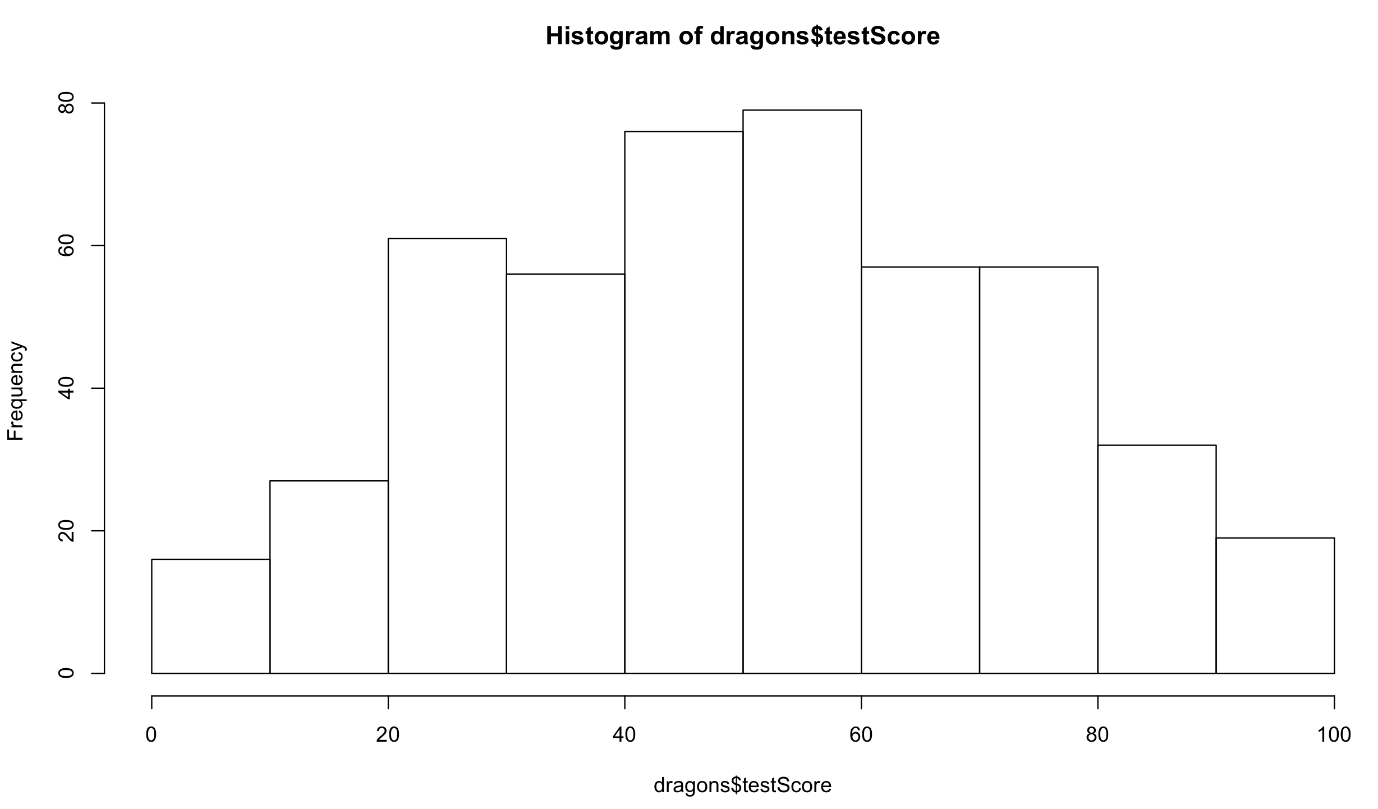
load("dragons.RData")

head(dragons)

Let’s say we want to know how the body length of the dragons affects their test scores.

You don’t need to worry about the distribution of your **explanatory** variables. Have a look at the distribution of the response variable:

hist(dragons$testScore) # seems close to a normal distribution - good!



It is good practice to **standardise** your explanatory variables before proceeding so that they have a mean of zero (“centering”) and standard deviation of one (“scaling”). It ensures that the estimated coefficients are all on the same scale, making it easier to compare effect sizes. You can use scale() to do that:

dragons$bodyLength2 <- scale(dragons$bodyLength, center = TRUE, scale = TRUE)

scale() centers the data (the column mean is subtracted from the values in the column) and then scales it (the centered column values are divided by the column’s standard deviation).

Back to our question: is the test score affected by body length?

# 3. Fit all data in one analysis

One way to analyse this data would be to fit a linear model to all our data, ignoring the sites and the mountain ranges for now.

Fit the model with testScore as the response and bodyLength2 as the predictor and have a look at the output:

basic.lm <- lm(testScore ~ bodyLength2, data = dragons)

summary(basic.lm)

Let’s plot the data with ggplot2.

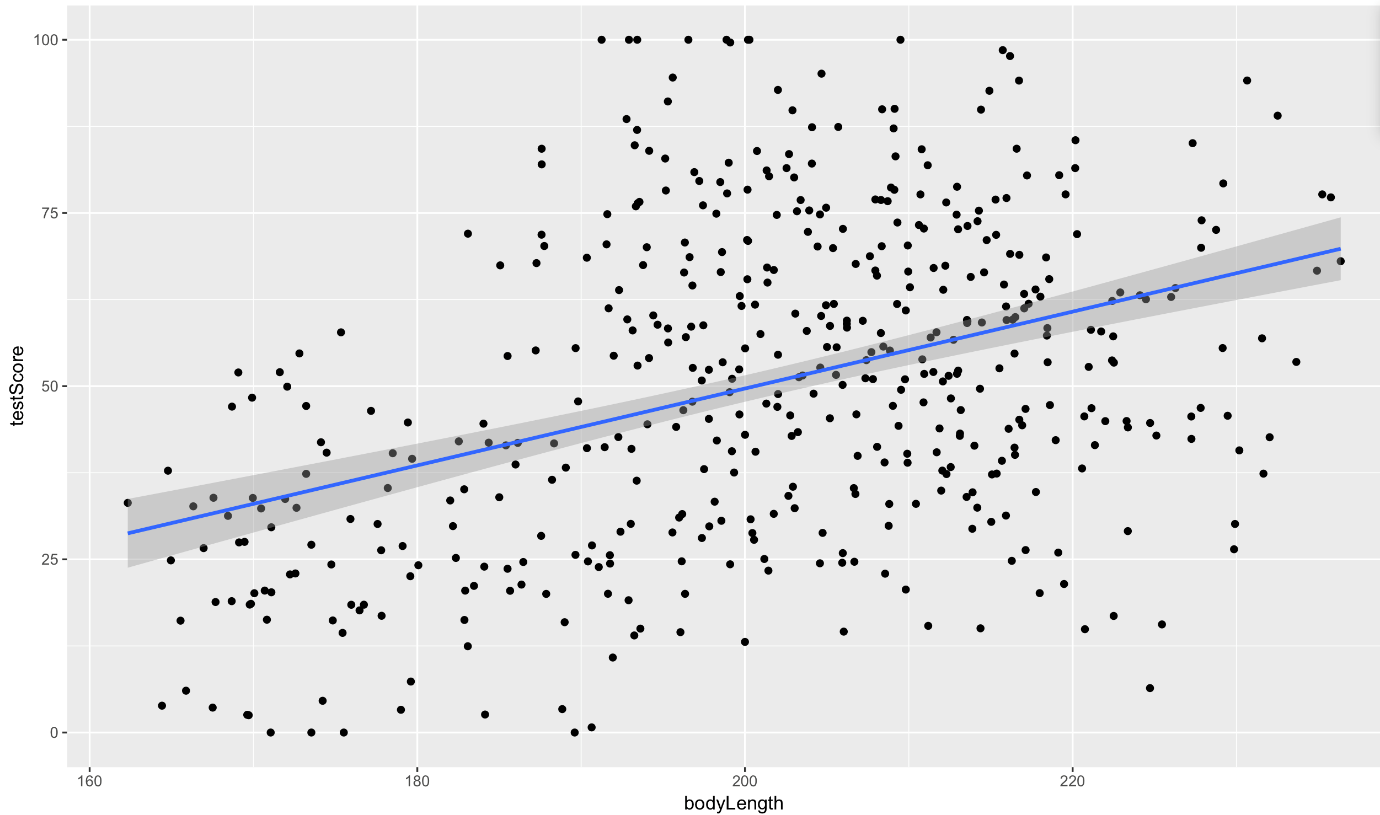
library(ggplot2) # load the package

(prelim\_plot <- ggplot(dragons, aes(x = bodyLength, y = testScore)) +

geom\_point() +

geom\_smooth(method = "lm"))

Note that putting your entire ggplot code in brackets () creates the graph and then shows it in the plot viewer. If you don’t have the brackets, you’ve only created the object, but haven’t visualised it. You would then have to call the object such that it will be displayed by just typing prelim\_plot after you’ve created the “prelim\_plot” object.



Okay, so both from the linear model and from the plot, it seems like bigger dragons do better in our intelligence test. That seems a bit odd: size shouldn’t really affect the test scores.

But… are the assumptions met?

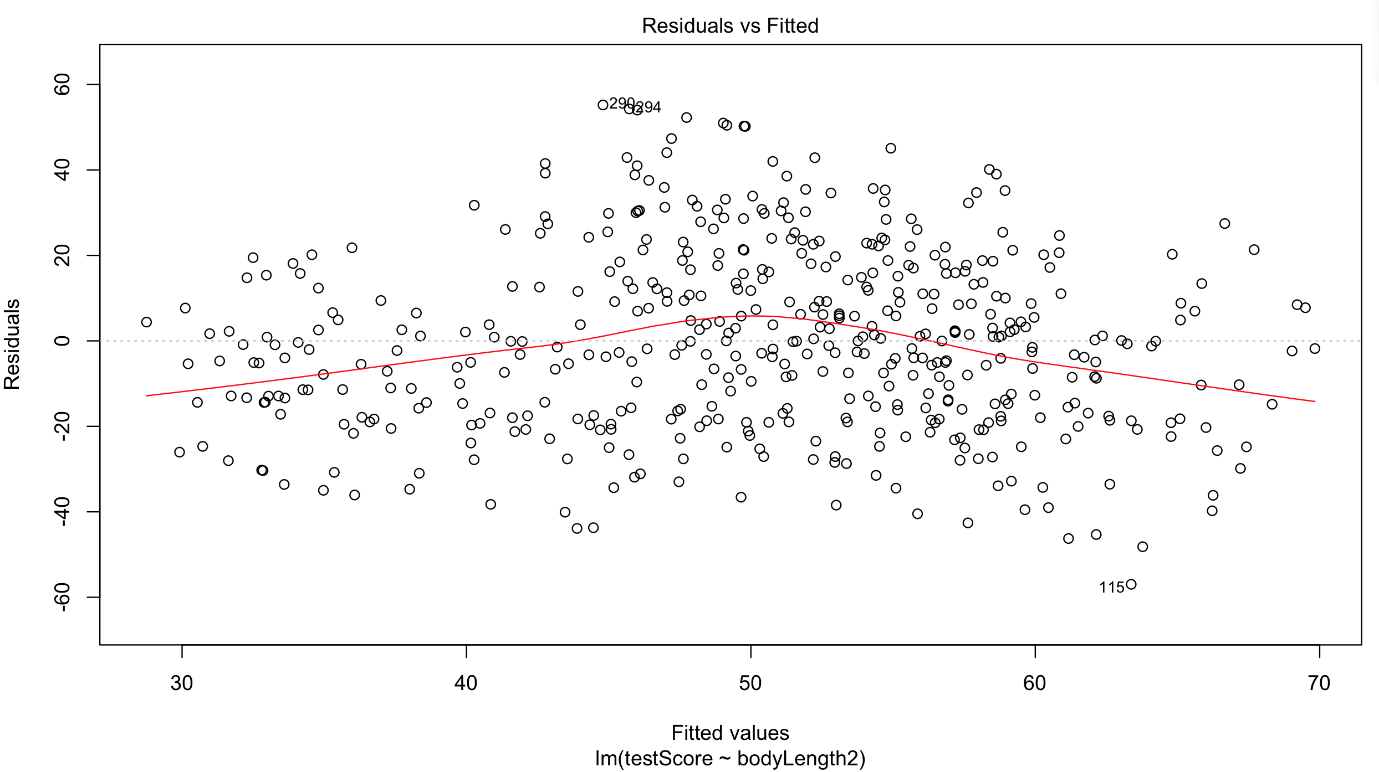
Plot the residuals: the red line should be nearly flat, like the dashed grey line:

plot(basic.lm, which = 1) # not perfect...

## but since this is a fictional example we will go with it

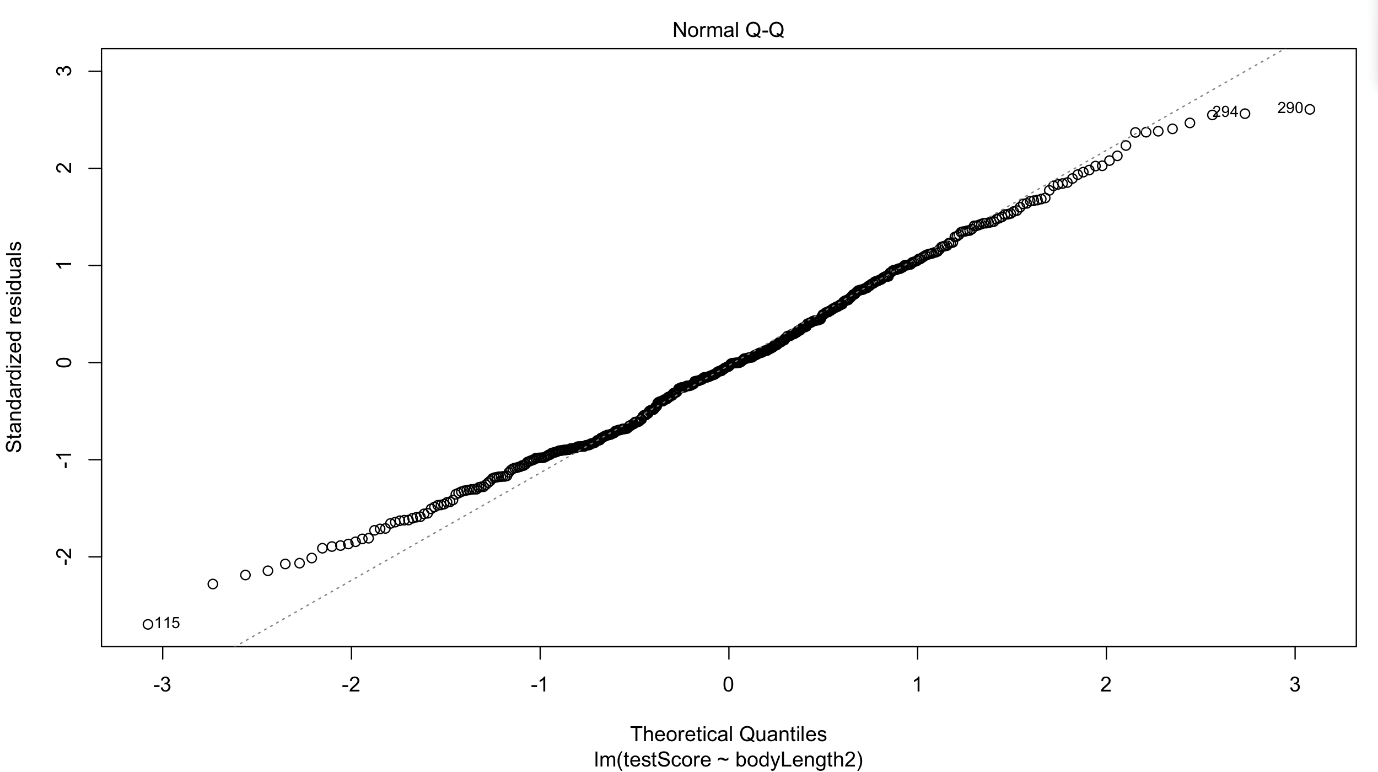
## for your own data be careful:

## the bigger the sample size, the less of a trend you'd expect to see



Have a quick look at the qqplot too: points should ideally fall onto the diagonal dashed line:

plot(basic.lm, which = 2) # a bit off at the extremes, but that's often the case; again doesn't look too bad

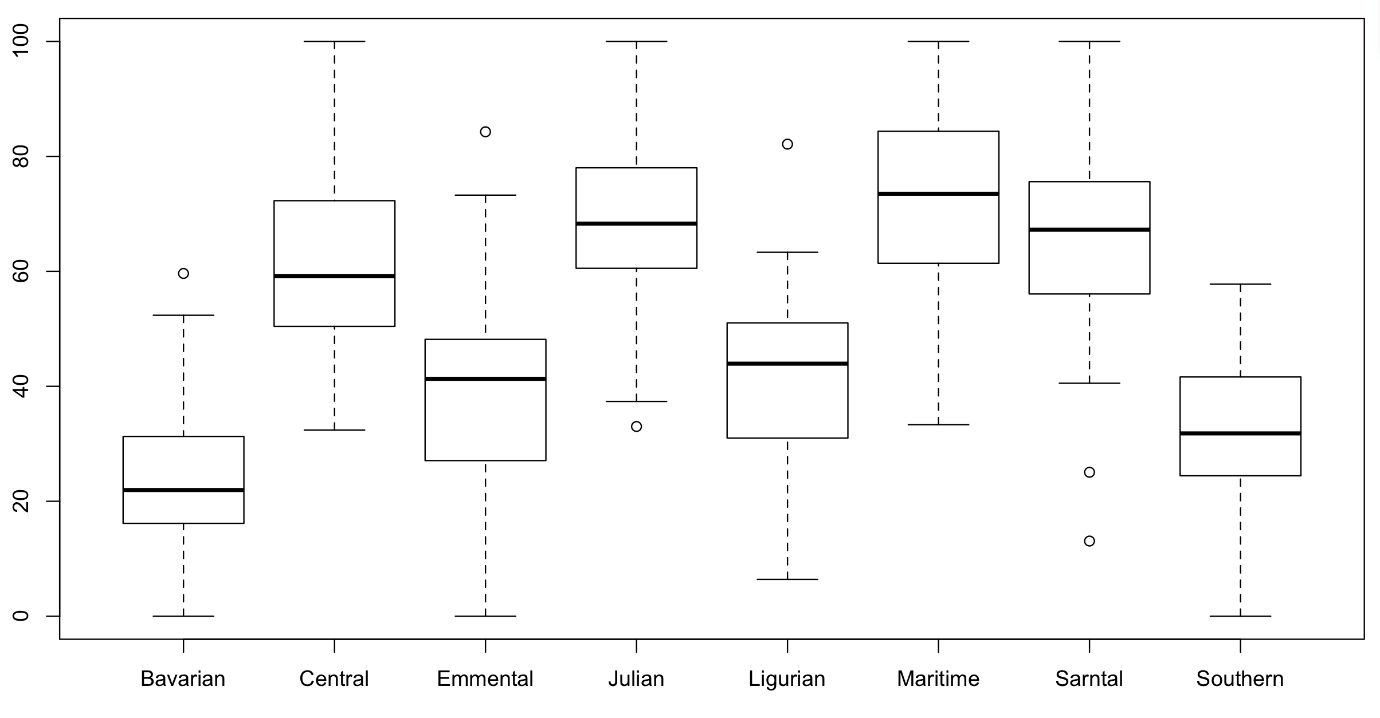


However, what about observation independence? Are our data independent?

We collected multiple samples from eight mountain ranges. It’s perfectly plausible that the data from within each mountain range are more similar to each other than the data from different mountain ranges: they are correlated.

Have a look at the data to see if above is true:

boxplot(testScore ~ mountainRange, data = dragons) # certainly looks like something is going on here



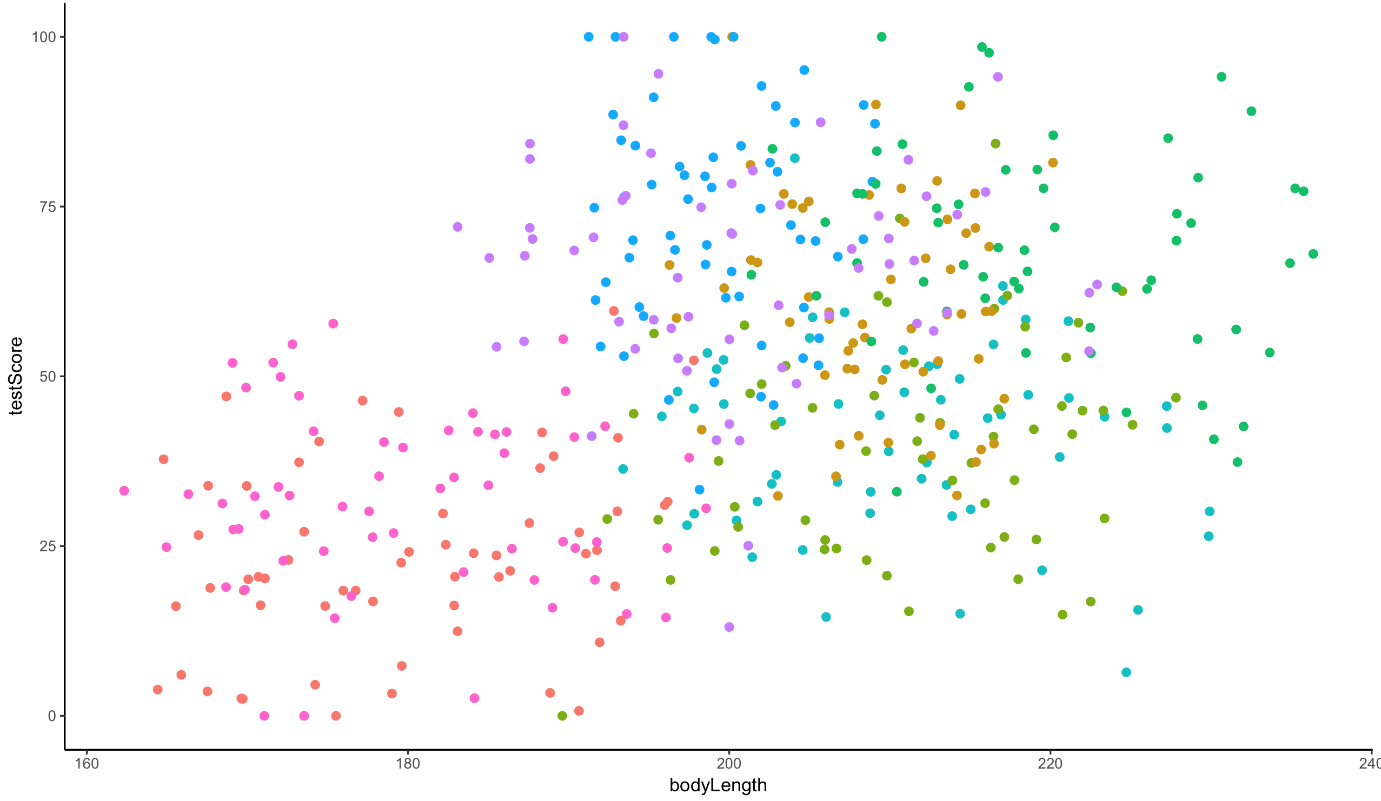
We could also plot it and colour points by mountain range:

(colour\_plot <- ggplot(dragons, aes(x = bodyLength, y = testScore, colour = mountainRange)) +

geom\_point(size = 2) +

theme\_classic() +

theme(legend.position = "none"))



From the above plots, it looks like our mountain ranges vary both in the dragon body length **AND** in their test scores. This confirms that our observations from within each of the ranges **aren’t independent**. We can’t ignore that: as we’re starting to see, it could lead to a completely erroneous conclusion.

So what do we do?

# 4. Run multiple analyses

We could run many separate analyses and fit a regression for each of the mountain ranges.

Lets have a quick look at the data split by mountain range. We use the facet\_wrap to do that:

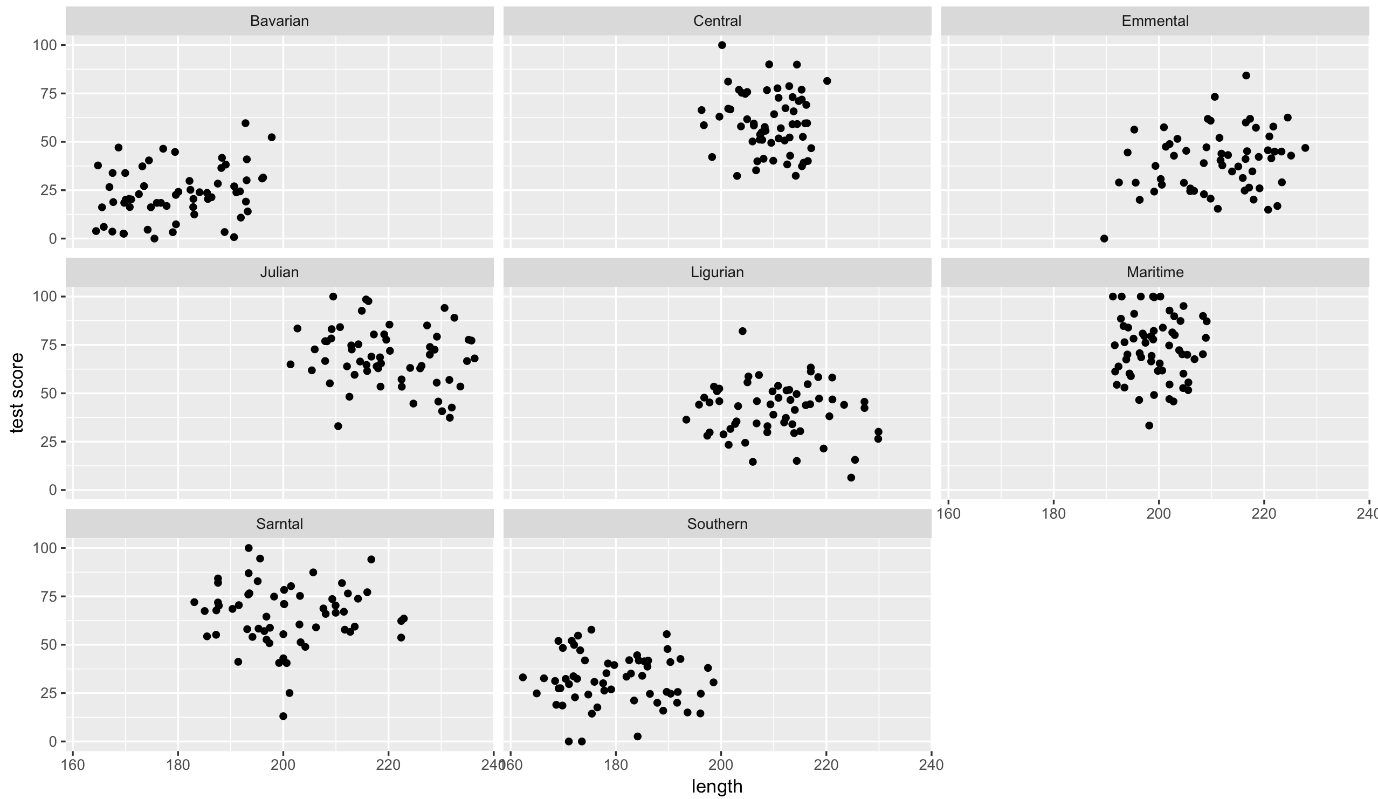
(split\_plot <- ggplot(aes(bodyLength, testScore), data = dragons) +

geom\_point() +

facet\_wrap(~ mountainRange) + # create a facet for each mountain range

xlab("length") +

ylab("test score"))



That’s eight analyses. Oh wait, we also have different sites in each mountain range, which similarly to mountain ranges aren’t independent… So we could run an analysis for each site in each range separately.

To do the above, we would have to estimate a slope and intercept parameter for **each regression**. That’s two parameters, three sites and eight mountain ranges, which means **48 parameter estimates** (2 x 3 x 8 = 48)! Moreover, the sample size for each analysis would be only 20 (dragons per site).

This presents problems: not only are we **hugely decreasing our sample size**, but we are also **increasing chances of a Type I Error (where you falsely reject the null hypothesis) by carrying out multiple comparisons**. Not ideal!

# 5. Modify the current model

We want to use all the data, but account for the data coming from different mountain ranges (let’s put sites on hold for a second to make things simpler).

Add mountain range as a fixed effect to our basic.lm

mountain.lm <- lm(testScore ~ bodyLength2 + mountainRange, data = dragons)

summary(mountain.lm)

Now body length is not significant. But let’s think about what we are doing here for a second. The above model is estimating the difference in test scores between the mountain ranges - we can see all of them in the model output returned by summary(). But we are not interested in quantifying test scores for each specific mountain range: we just want to know whether body length affects test scores and we want to simply **control for the variation** coming from mountain ranges.

This is what we refer to as **“random factors”** and so we arrive at mixed effects models. Ta-daa!

# 6. Mixed effects models

A mixed model is a good choice here: it will allow us to **use all the data we have** (higher sample size) and **account for the correlations between data** coming from the sites and mountain ranges. We will also **estimate fewer parameters** and **avoid problems with multiple comparisons** that we would encounter while using separate regressions.

We are going to work in lme4, so load the package (or use install.packages if you don’t have lme4 on your computer).

library(lme4)

## Fixed and random effects

Let’s talk a little about the difference between **fixed and random effects** first. It’s important to not that this difference has little to do with the variables themselves, and a lot to do with your research question! In many cases, the same variable could be considered either a random or a fixed effect (and sometimes even both at the same time!) so always refer to your questions and hypotheses to construct your models accordingly.

## Should my variables be fixed or random effects?

In broad terms, **fixed effects** are variables that we expect will have an effect on the dependent/response variable: they’re what you call **explanatory** variables in a standard linear regression. In our case, we are interested in making conclusions about how dragon body length impacts the dragon’s test score. So body length is a fixed effect and test score is the dependent variable.

On the other hand, **random effects** are usually **grouping factors** for which we are trying to control. They are always categorical, as you can’t force R to treat a continuous variable as a random effect. A lot of the time we are not specifically interested in their impact on the response variable, but we know that they might be influencing the patterns we see.

Additionally, the data for our random effect is just **a sample of all the possibilities**: with unlimited time and funding we might have sampled every mountain where dragons live, every school in the country, every chocolate in the box), but we usually tend to generalise results to a whole population based on representative sampling. We don’t care about estimating how much better pupils in school A have done compared to pupils in school B, but we know that their respective teachers might be a reason why their scores would be different, and we’d like to know how much variation is attributable to this when we predict scores for pupils in school Z.

test <- 1 + 3

In our particular case, we are looking to control for the effects of mountain range. We haven’t sampled all the mountain ranges in the world (we have eight) so our data are just a sample of all the existing mountain ranges. We are not really interested in the effect of each specific mountain range on the test score: we hope our model would also be generalisable to dragons from other mountain ranges! However, we know that the test scores from within the ranges might be correlated so we want to control for that.

If we specifically chose eight particular mountain ranges a priori and we were interested in those ranges and wanted to make predictions about them, then mountain range would be fitted as a fixed effect.

## More about random effects

Note that the golden rule is that you generally want your random effect to have **at least five levels**. So, for instance, if we wanted to control for the effects of dragon’s sex on intelligence, we would fit sex (a two level factor: male or female) **as a fixed, not random, effect**.

This is, put simply, because estimating variance on few data points is very imprecise. Mathematically you could, but you wouldn’t have a lot of confidence in it. If you only have two or three levels, the model will struggle to partition the variance - it will give you an output, but not necessarily one you can trust.

Finally, keep in mind that the name random doesn’t have much to do with mathematical randomness. Yes, it’s confusing. Just think about them as the grouping variables for now. Strictly speaking it’s all about making our models representative of our questions **and getting better estimates**. Hopefully, our next few examples will help you make sense of how and why they’re used.

**In the end, the big questions are:** what are you trying to do? What are you trying to make predictions about? What is just variation (a.k.a “noise”) that you need to control for?

## Let’s fit our first mixed model

Alright! Still with me? We have a response variable, the test score and we are attempting to **explain part of the variation** in test score through fitting body length as a fixed effect. But the response variable has some **residual variation** (i.e. unexplained variation) associated with mountain ranges. By using random effects, we are modeling that unexplained variation through **variance**.

[Sidenote: If you are confused between variation and variance: **variation** is a generic word, similar to dispersion or variability; **variance** is a particular measure of variation; it quantifies the dispersion, if you wish.]

Note that **our question changes slightly here**: while we still want to know whether there is an association between dragon’s body length and the test score, we want to know if that association exists **after** controlling for the variation in mountain ranges.

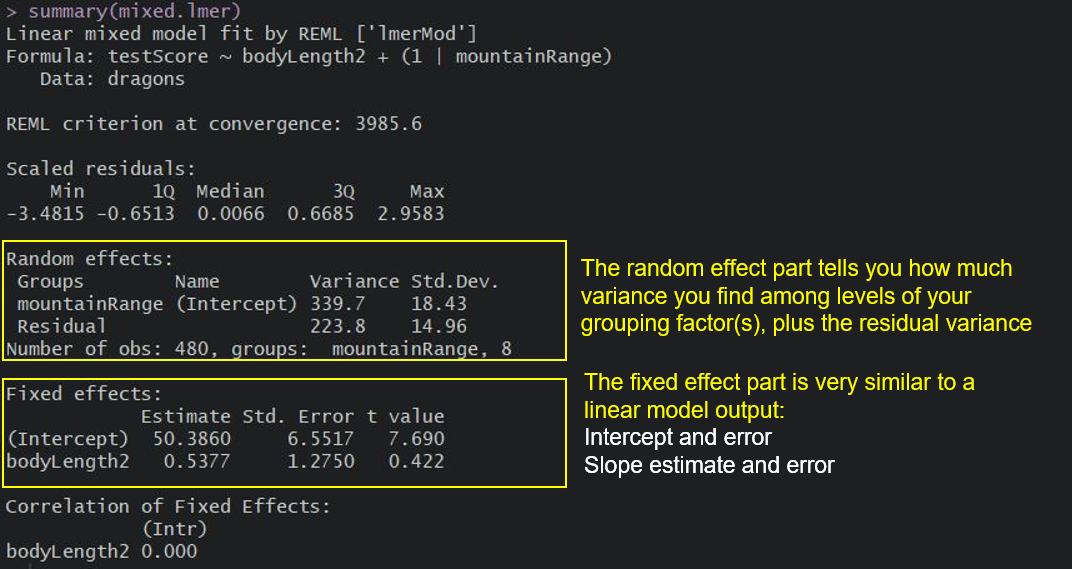
We will fit the random effect using the syntax (1|variableName):

mixed.lmer <- lmer(testScore ~ bodyLength2 + (1|mountainRange), data = dragons)

summary(mixed.lmer)

Once we account for the mountain ranges, it’s obvious that dragon body length doesn’t actually explain the differences in the test scores. How is it obvious? I hear you say?

Take a look at the summary output: notice how the **model estimate** is smaller than its associated error? That means that the effect, or slope, cannot be distinguised from zero.



Keep in mind that the random effect of the mountain range is **meant to capture all the influences of mountain ranges on dragon test scores** - whether we observed those influences explicitly or not, whether those influences are big or small etc. It could be many, many teeny-tiny influences that, when combined, affect the test scores and that’s what we are hoping to control for.

We can see the variance for mountainRange = 339.7. Mountain ranges are clearly important: they explain a lot of variation. How do we know that? We can take the variance for the mountainRange and divide it by the total variance:

339.7/(339.7 + 223.8) # ~60 %

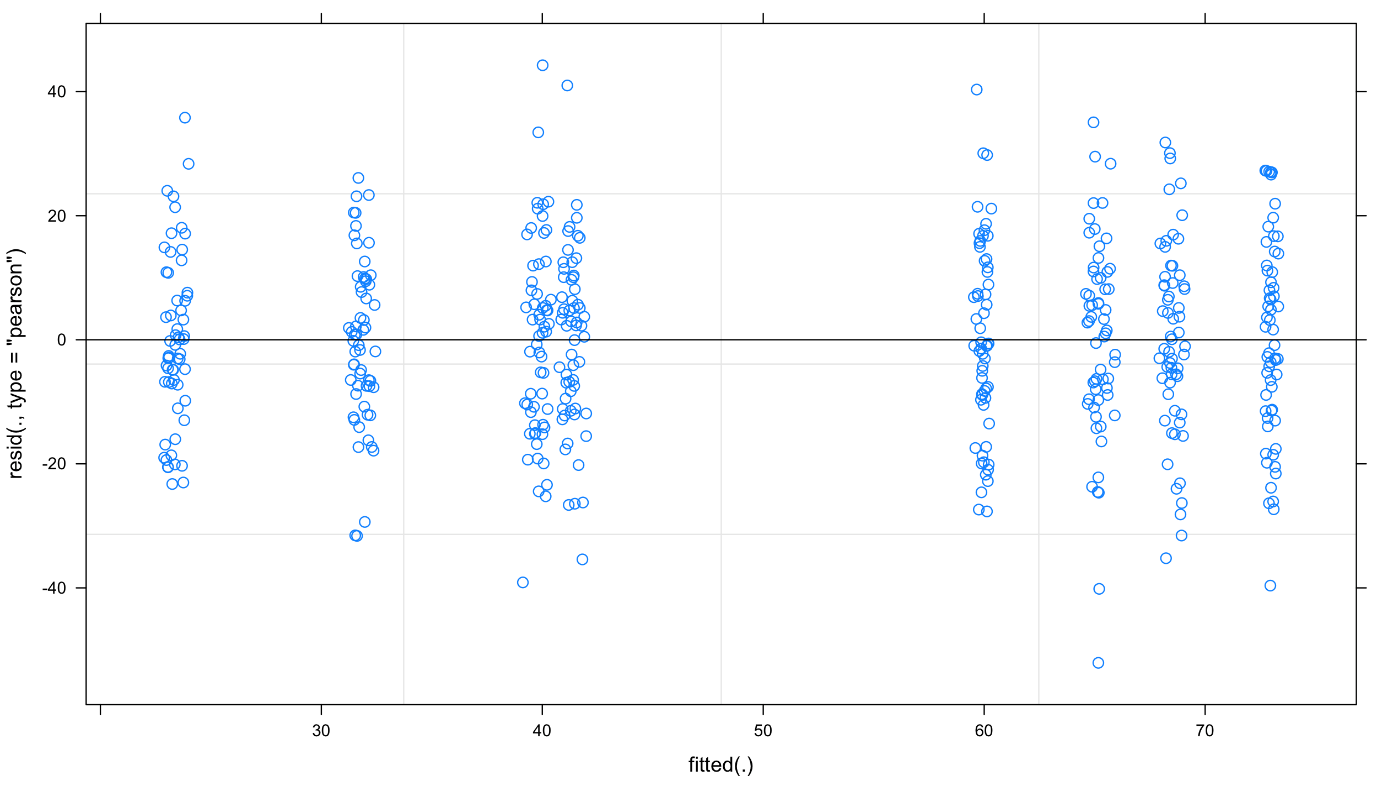
So the differences between mountain ranges explain ~60% of the variance that’s “left over” after the variance explained by our fixed effects.

## More reading on random effects

Still confused about interpreting random effects? These links have neat demonstrations and explanations:

As always, it’s good practice to have a look at the plots to check our assumptions:

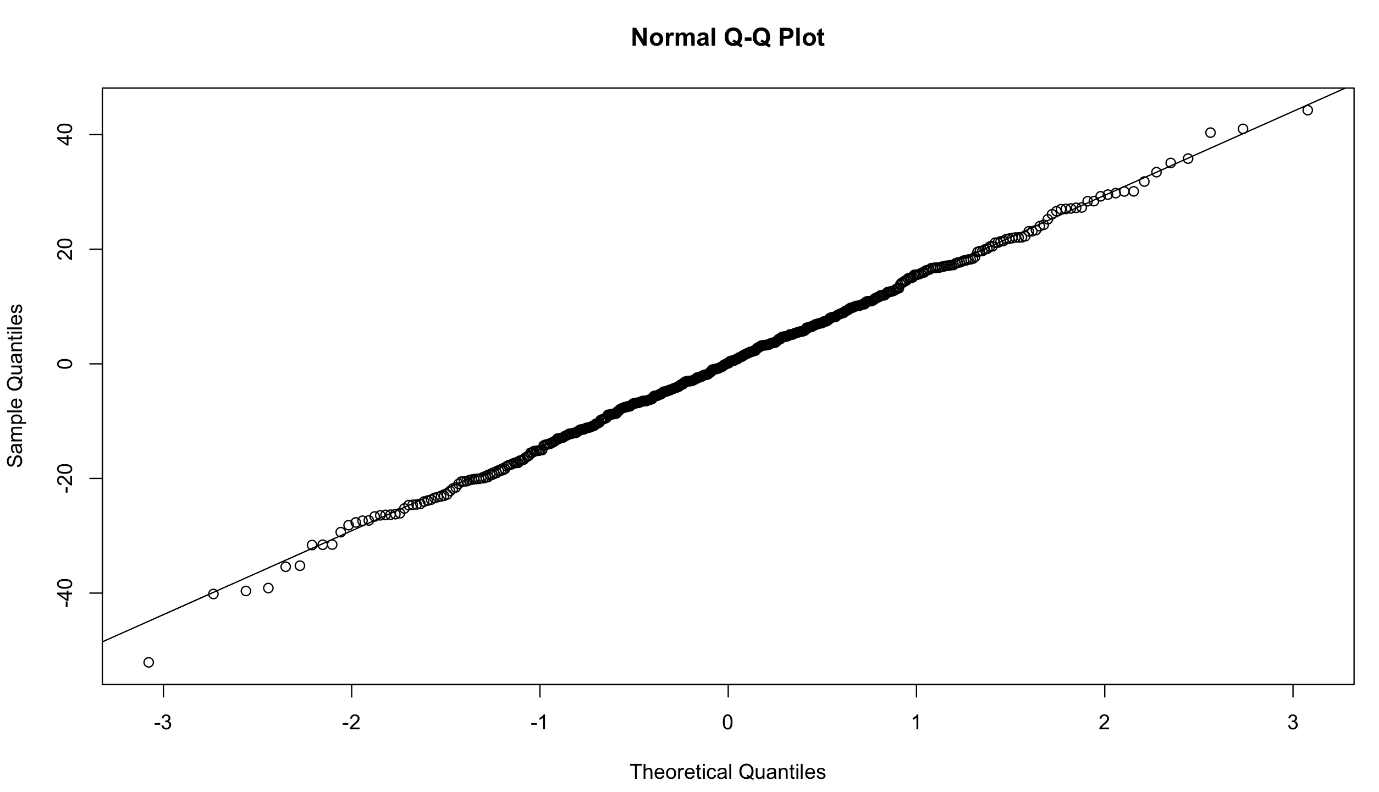
plot(mixed.lmer) # looks alright, no patterns evident



and qqplot:

qqnorm(resid(mixed.lmer))

qqline(resid(mixed.lmer)) # points fall nicely onto the line - good!



## Types of random effects

Before we go any further, let’s review the syntax above and chat about crossed and nested random effects. It’s useful to get those clear in your head.

**Reminder**: a factor is just any categorical independent variable.

Above, we used (1|mountainRange) to fit our random effect. Whatever is on the right side of the | operator is a factor and referred to as a “grouping factor” for the term.

**Random effects (factors) can be crossed or nested** - it depends on the relationship between the variables. Let’s have a look.

### Crossed random effects

Be careful with the nomenclature. There are **“hierarchical linear models”** (HLMs) or **“multilevel models”** out there, but while all HLMs are mixed models, **not all mixed models are hierarchical**. That’s because you can have **crossed (or partially crossed) random factors** that do not represent levels in a hierarchy.

Think for instance about our study where you monitor dragons (subject) across different mountain ranges (context) and imagine that we collect **multiple observations per dragon** by giving it the test multiple times (and risking **pseudoreplication** - but more on that later). Since our dragons can fly, it’s easy to imagine that **we might observe the same dragon across different mountain ranges**, but also that we might not see all the dragons visiting all of the mountain ranges. Therefore, we can potentially observe every dragon in every mountain range (**crossed**) or at least observe some dragons across some of the mountain ranges (**partially crossed**). We would then fit the identity of the dragon and mountain range as (partially) crossed random effects.

Let’s repeat with another example: an effect is **(fully) crossed** when all the subjects have experienced all the levels of that effect. For instance, if you had a fertilisation experiment on seedlings growing in a seasonal forest and took repeated measurements over time (say 3 years) in each season, you may want to have a crossed factor called season (Summer1, Autumn1, Winter1, Spring1, Summer2, …, Spring3), i.e. a factor for each season of each year. This grouping factor would account for the fact that all plants in the experiment, regardless of the fixed (treatment) effect (i.e. fertilised or not), may have experienced a very hot summer in the second year, or a very rainy spring in the third year, and those conditions could cause interference in the expected patterns. You don’t even need to have associated climate data to account for it! You just know that all observations from spring 3 may be more similar to each other because they experienced the same environmental quirks rather than because they’re responding to your treatment.

If this sounds confusing, not to worry - lme4 handles partially and fully crossed factors well. Now, let’s look at **nested** random effects and how to specify them.

## Nested random effects

If you’re not sure what nested random effects are, think of those Russian nesting dolls. We’ve already hinted that we call these models **hierarchical**: there’s often an element of scale, or sampling stratification in there.

Take our fertilisation experiment example again; let’s say you have 50 seedlings in each bed, with 10 control and 10 experimental beds. That’s 1000 seedlings altogether. And let’s say you went out collecting once in each season in each of the 3 years. On each plant, you measure the length of 5 leaves. That’s….(lots of maths)…5 leaves x 50 plants x 20 beds x 4 seasons x 3 years….. 60 000 measurements!

But if you were to run the analysis using a simple linear regression, eg. leafLength ~ treatment , you would be committing the crime (!!) of **pseudoreplication**, or massively increasing your sampling size by using non-independent data. With a sample size of 60,000 you would almost certainly get a “significant” effect of treatment which may have no ecological meaning at all. And it violates the **assumption of independance of observations** that is central to linear regression.

This is where our nesting dolls come in; leaves within a plant and plants within a bed may be more similar to each other (e.g. for genetic and environmental reasons, respectively). You could therefore add a random effect structure that accounts for this nesting:

leafLength ~ treatment + (1|Bed/Plant/Leaf)

This way, the model will account for non independence in the data: the same leaves have been sampled repeatedly, multiple leaves were measured on an individual, and plants are grouped into beds which may receive different amounts of sun, etc.

What about the crossed effects we mentioned earlier? If all the leaves have been measured in all seasons, then your model would become something like:

leafLength ~ treatment + (1|Bed/Plant/Leaf) + (1|Season)

Phew!

### Implicit vs. explicit nesting

To make things easier for yourself, code your data properly and **avoid implicit nesting**.

To tackle this, let’s look at another aspect of our study: we collected the data on dragons not only across multiple mountain ranges, but also across several sites within those mountain ranges. If you don’t remember have another look at the data:

head(dragons) # we have site and mountainRange

str(dragons) # we took samples from three sites per mountain range and eight mountain ranges in total

Just like we did with the mountain ranges, we have to assume that data collected within our sites might be **correlated** and so we should include sites as **an additional random effect** in our model.

Our site variable is a three-level factor, with sites called a, b and c. The nesting of the site within the mountain range is **implicit** - our sites are meaningless without being assigned to specific mountain ranges, i.e. there is nothing linking site b of the Bavarian mountain range with site b of the Central mountain range. To avoid future confusion we should create a new variable that is **explicitly nested**. Let’s call it sample:

dragons <- within(dragons, sample <- factor(mountainRange:site))

Now it’s obvious that we have 24 samples (8 mountain ranges x 3 sites) and not just 3: our sample is a 24-level factor and we should use that instead of using site in our models: each site belongs to a specific mountain range.

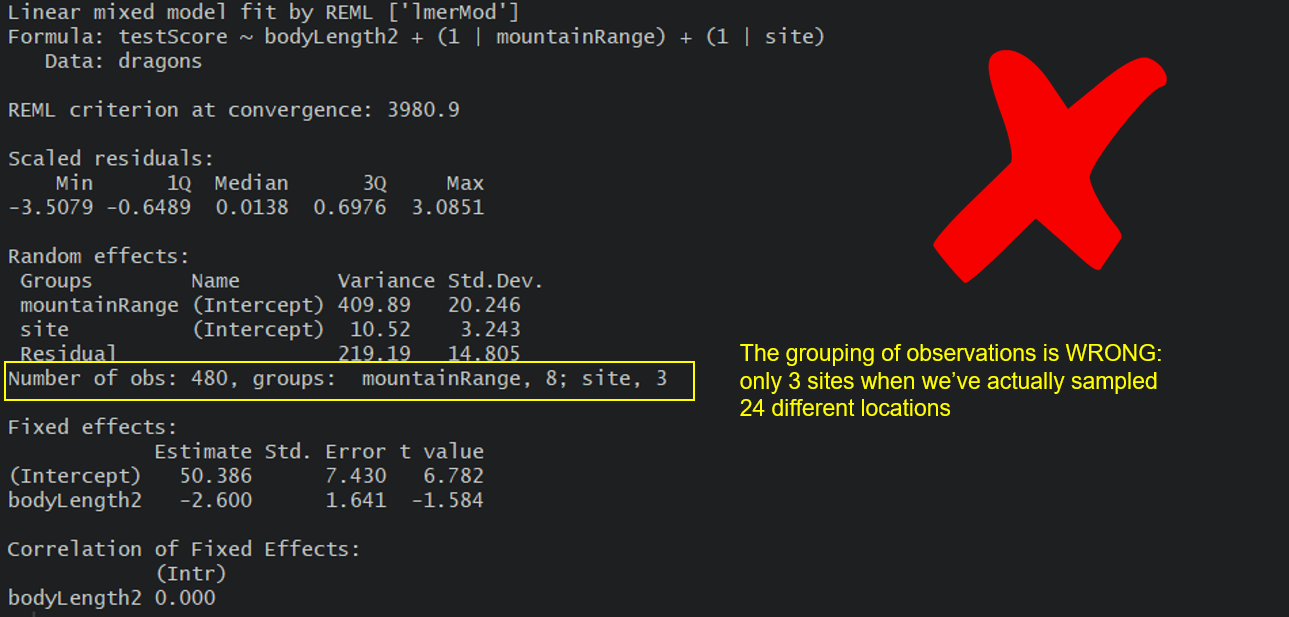
**To sum up:** for **nested random effects**, the factor appears **ONLY** within a particular level of another factor (each site belongs to a specific mountain range and only to that range); for **crossed effects**, a given factor appears in more than one level of another factor (dragons appearing within more than one mountain range). **Or you can just remember that if your random effects aren’t nested, then they are crossed!**

## Our second mixed model

Based on the above, using following specification would be **\*\*wrong\*\***, as it would imply that there are only three sites with observations at each of the 8 mountain ranges (crossed):

mixed.WRONG <- lmer(testScore ~ bodyLength2 + (1|mountainRange) + (1|site), data = dragons) # treats the two random effects as if they are crossed

summary(mixed.WRONG)

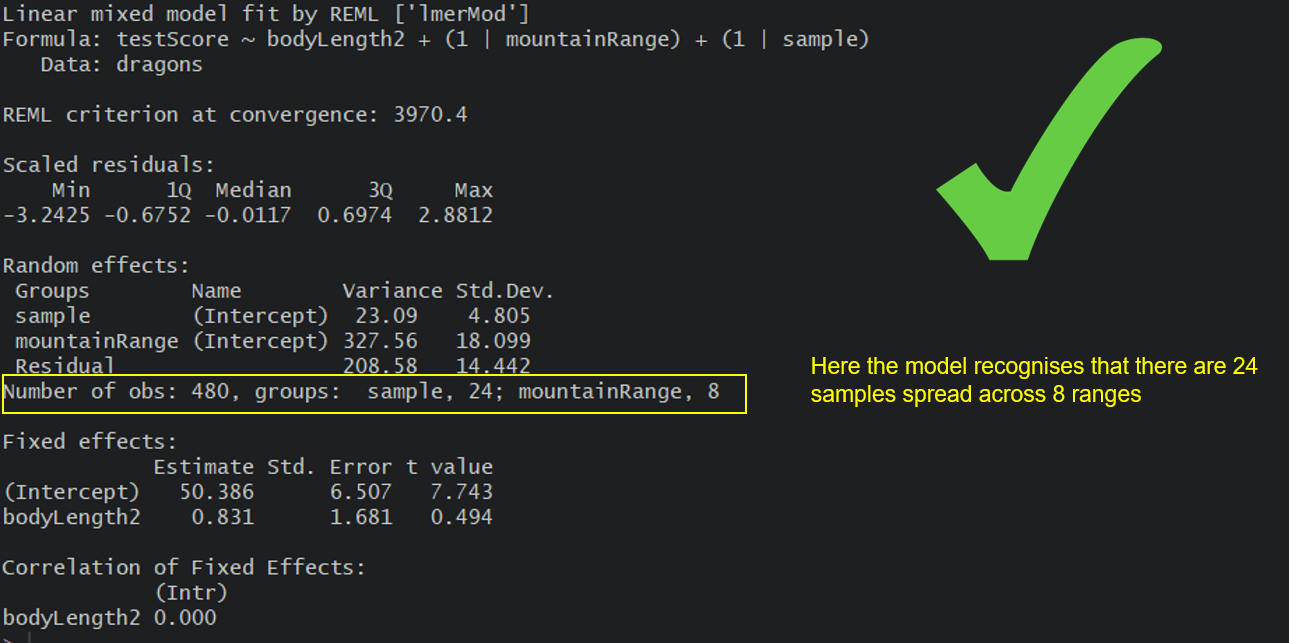


But we can go ahead and fit a new model, one that takes into account both the differences between the mountain ranges, as well as the differences between the sites within those mountain ranges by using our sample variable.

Our question gets **adjusted slightly again**: Is there an association between body length and intelligence in dragons **after** controlling for variation in mountain ranges and sites within mountain ranges?

mixed.lmer2 <- lmer(testScore ~ bodyLength2 + (1|mountainRange) + (1|sample), data = dragons) # the syntax stays the same, but now the nesting is taken into account

summary(mixed.lmer2)



Here, we are trying to account for **all the mountain-range-level** and **all the site-level influences** and we are hoping that our random effects have soaked up all these influences so we can control for them in the model.

For the record, you could also use the below syntax, and you will often come across it if you read more about mixed models:

(1|mountainRange/site) or even (1|mountainRange) + (1|mountainRange:site)

However, it is advisable to set out your variables properly and make sure nesting is stated explicitly within them, that way you don’t have to remember to specify the nesting.

Let’s plot this again - visualising what’s going on is always helpful. You should be able to see eight mountain ranges with three sites (different colour points) within them, with a line fitted through each site.

(mm\_plot <- ggplot(dragons, aes(x = bodyLength, y = testScore, colour = site)) +

facet\_wrap(~mountainRange, nrow=2) + # a panel for each mountain range

geom\_point(alpha = 0.5) +

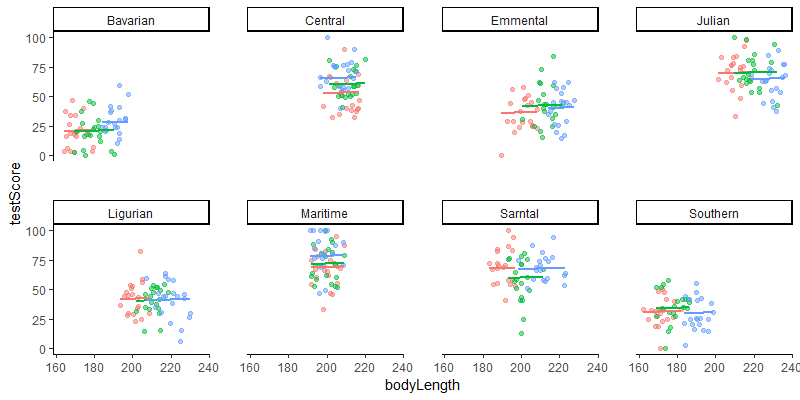
theme\_classic() +

geom\_line(data = cbind(dragons, pred = predict(mixed.lmer2)), aes(y = pred), size = 1) + # adding predicted line from mixed model

theme(legend.position = "none",

panel.spacing = unit(2, "lines")) # adding space between panels

)



## Introducing random slopes

You might have noticed that all the lines on the above figure are parallel: that’s because so far, we have only fitted **random-intercept models**. A random-intercept model allows the intercept to vary for each level of the random effects, but keeps the slope constant among them. So in our case, using this model means that we expect dragons in all mountain ranges to exhibit the same relationship between body length and intelligence (fixed slope), although we acknowledge that some populations may be smarter or dumber to begin with (random intercept).

Now, in the life sciences, we perhaps more often assume that not all populations would show the exact same relationship, for instance if your study sites/populations are very far apart and have some relatively important environmental, genetic, etc differences. Therefore, we often want to fit a **random-slope and random-intercept model**. Maybe the dragons in a very cold vs a very warm mountain range have evolved different body forms for heat conservation and may therefore be smart even if they’re smaller than average.

We only need to make one change to our model to allow for random slopes as well as intercept, and that’s adding the fixed variable into the random effect brackets:

mixed.ranslope <- lmer(testScore ~ bodyLength2 + (1 + bodyLength2|mountainRange/site), data = dragons)

summary(mixed.ranslope)

Here, we’re saying, let’s model the intelligence of dragons as a function of body length, knowing that populations have different intelligence baselines **and** that the relationship may vary among populations.

Let’s see that with a quick plot (we’ll plot predictions in more detail in the next section). Notice how the slopes for the different sites and mountain ranges are not parallel anymore?

### plot

(mm\_plot <- ggplot(dragons, aes(x = bodyLength, y = testScore, colour = site)) +

facet\_wrap(~mountainRange, nrow=2) + # a panel for each mountain range

geom\_point(alpha = 0.5) +

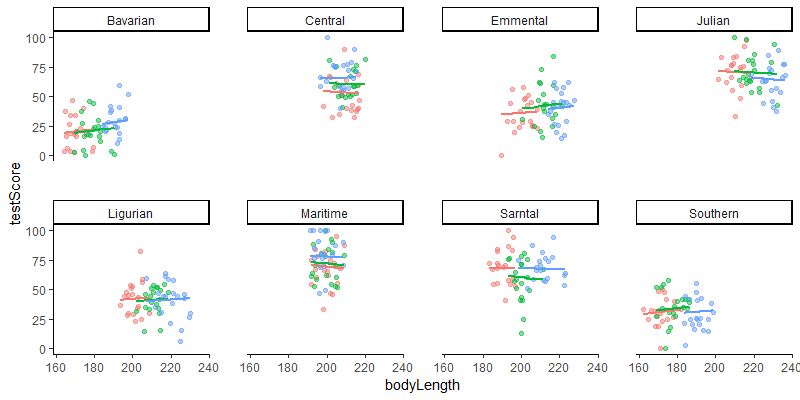
theme\_classic() +

geom\_line(data = cbind(dragons, pred = predict(mixed.ranslope)), aes(y = pred), size = 1) + # adding predicted line from mixed model

theme(legend.position = "none",

panel.spacing = unit(2, "lines")) # adding space between panels

)



**Well done for getting here!** You have now fitted random-intercept and random-slopes, random-intercept mixed models and you know how to account for hierarchical and crossed random effects. You saw that failing to account for the correlation in data might lead to misleading results - it seemed that body length affected the test score until we accounted for the variation coming from mountain ranges. We can see now that body length doesn’t influence the test scores - great! We can pick smaller dragons for any future training - smaller ones should be more manageable!

If you are particularly keen, the next section gives you a few options when it comes to **presenting your model results** and in the last “extra” section you can learn about the **model selection conundrum**. There is just a little bit more code there to get through if you fancy those.

## Presenting your model results

Once you get your model, you have to **present** it in a nicer form.

### Plotting model predictions

Often you will want to visualise your model as a regression line with some error around it, just like you would a simple linear model. However, ggplot2 stats options are not designed to estimate mixed-effect model objects correctly, so we will use the ggeffects package to help us draw the plots.

library(ggeffects) # install the package first if you haven't already, then load it

# Extract the prediction data frame

pred.mm <- ggpredict(mixed.lmer2, terms = c("bodyLength2")) # this gives overall predictions for the model

# Plot the predictions

(ggplot(pred.mm) +

geom\_line(aes(x = x, y = predicted)) + # slope

geom\_ribbon(aes(x = x, ymin = predicted - std.error, ymax = predicted + std.error),

fill = "lightgrey", alpha = 0.5) + # error band

geom\_point(data = dragons, # adding the raw data (scaled values)

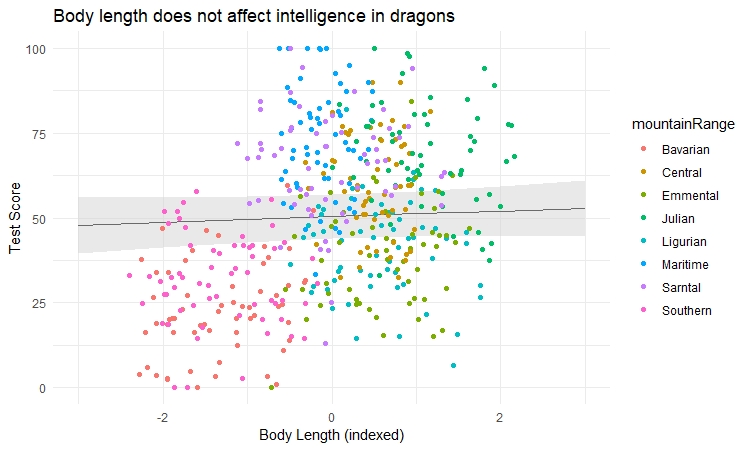
aes(x = bodyLength2, y = testScore, colour = mountainRange)) +

labs(x = "Body Length (indexed)", y = "Test Score",

title = "Body length does not affect intelligence in dragons") +

theme\_minimal()

)



What if you want to visualise how the relationships vary according to different levels of random effects? You can specify type = "re" (for “random effects”) in the ggpredict() function, and add the random effect name to the terms argument.

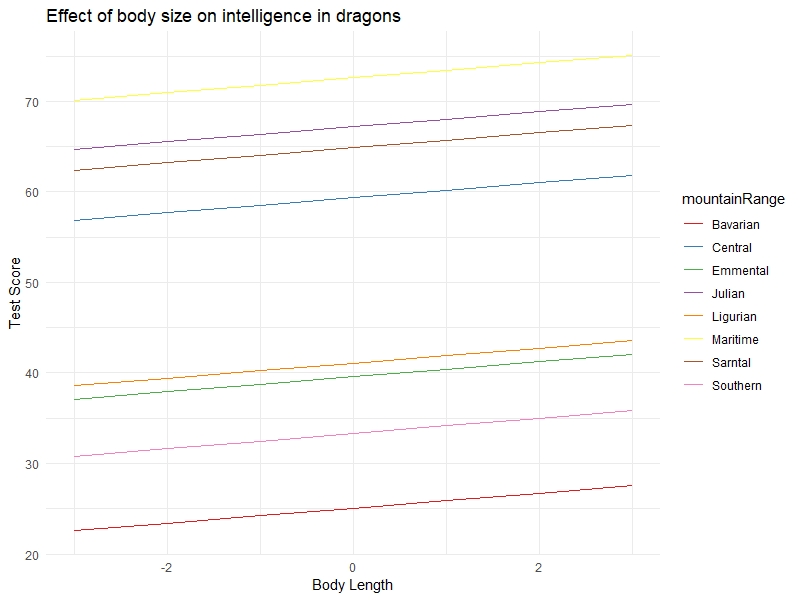
We also demonstrate a way to plot the graph quicker with the plot() function of ggEffects:

ggpredict(mixed.lmer2, terms = c("bodyLength2", "mountainRange"), type = "re") %>%

plot() +

labs(x = "Body Length", y = "Test Score", title = "Effect of body size on intelligence in dragons") +

theme\_minimal()



You can clearly see the random intercepts and fixed slopes from this graph. When assessing the quality of your model, it’s always a good idea to look at the raw data, the summary output, and the predictions all together to make sure you understand what is going on (and that you have specified the model correctly).

Another way to visualise mixed model results, if you are interested in showing the variation among levels of your random effects, is to plot the departure from the overall model estimate for intercepts - and slopes, if you have a random slope model:

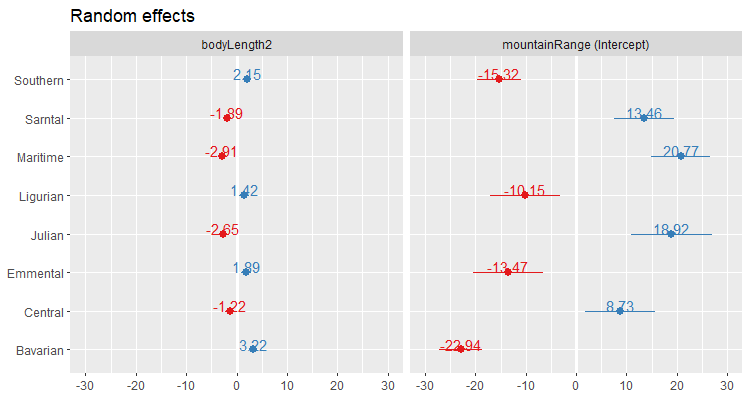
library(sjPlot)

# Visualise random effects

(re.effects <- plot\_model(mixed.ranslope, type = "re", show.values = TRUE))

# show summary

summary(mixed.ranslope)



**Careful here!** The values you see are **NOT** actual values, but rather the difference between the general intercept or slope value found in your model summary and the estimate for this specific level of random effect. For instance, the relationship for dragons in the Maritime mountain range would have a slope of (-2.91 + 0.67) = -2.24 and an intercept of (20.77 + 51.43) = 72.20.

### Tables

For lme4, if you are looking for a table, I’d recommend that you have a look at the stargazer package.

library(stargazer)

Here is a quick example - simply plug in your model name, in this case mixed.lmer2 into the stargazer function. I set type to "text" so that you can see the table in your console. I usually tweak the table like this until I’m happy with it and then export it using type = "latex", but "html" might be more useful for you if you are not a LaTeX user.

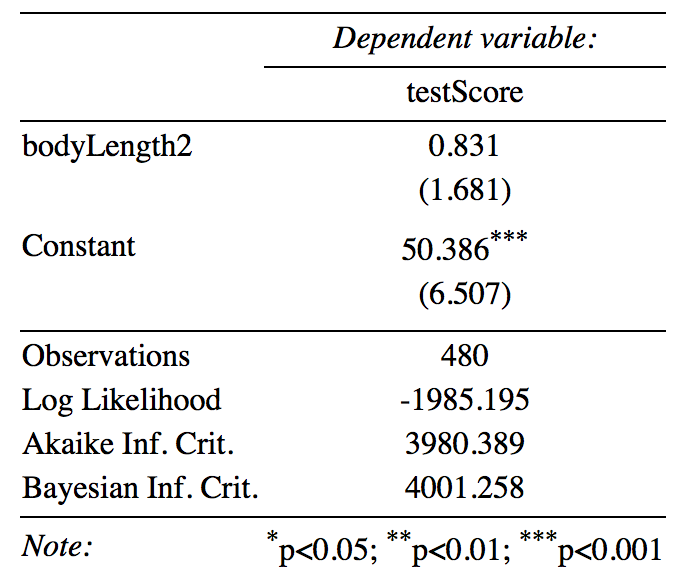
If you are keen, explore this table a little further - what would you change? What would you get rid off?

stargazer(mixed.lmer2, type = "text",

digits = 3,

star.cutoffs = c(0.05, 0.01, 0.001),

digit.separator = "")



### Further processing

If you’d like to be able **to do more with your model results**, for instance process them further, collate model results from multiple models or plot, them have a look at the broom package.

## EXTRA: P-values and model selection

Please be **very, very careful** when it comes to model selection. Focus on your **question**, don’t just plug in and drop variables from a model haphazardly until you make something “significant”. Always choose variables based on biology/ecology: I might use model selection to check a couple of non-focal parameters, but I keep the “core” of the model untouched in most cases. **Define your goals and questions and focus on that.** Also, don’t just put all possible variables in (i.e. don’t **overfit**). Remember that as a rule of thumb, **you need 10 times more data than parameters** you are trying to estimate.

## Fixed effects structure

**Before we start, again: think twice before trusting model selection!**

Most of you are probably going to be predominantly interested in your fixed effects, so let’s start here. lme4 doesn’t spit out p-values for the parameters by default. This is a conscious choice made by the authors of the package, as there are many problems with p-values (I’m sure you are aware of the debates!).

You will inevitably look for a way to assess your model though so here are a few solutions on how to go about hypothesis testing in linear mixed models (LMMs):

**From worst to best:**

* Wald Z-tests
* Wald t-tests (but LMMs need to be balanced and nested)
* Likelihood ratio tests (via anova() or drop1())
* MCMC or parametric bootstrap confidence intervals

I think that MCMC and bootstrapping are a bit out of our reach for this workshop so let’s have a quick go at **likelihood ratio tests** using anova(). With large sample sizes, p-values based on the likelihood ratio are generally considered okay. **NOTE:** With small sample sizes, you might want to look into deriving p-values using the Kenward-Roger or Satterthwaite approximations (for REML models). Check out the pbkrtest package.

Fit the models, a full model and a reduced model in which we dropped our fixed effect (bodyLength2):

full.lmer <- lmer(testScore ~ bodyLength2 + (1|mountainRange) + (1|sample),

data = dragons, REML = FALSE)

reduced.lmer <- lmer(testScore ~ 1 + (1|mountainRange) + (1|sample),

data = dragons, REML = FALSE)

Compare them:

anova(reduced.lmer, full.lmer) # the two models are not significantly different

Notice that we have fitted our models with REML = FALSE.

**REML** stands for **restricted (or “residual”) maximum likelihood** and it is the default parameter estimation criterion for linear mixed models. As you probably guessed, **ML** stands for **maximum likelihood** - you can set REML = FALSE in your call to lmer to use ML estimates. However, **ML estimates are known to be biased** and with REML being usually less biased, **REML estimates of variance components are generally preferred.** This is why in our previous models we skipped setting REML - we just left it as default (i.e. REML = TRUE).

**REML** assumes that the fixed effects structure is correct. You **should use maximum likelihood when comparing models with different fixed effects**, as **ML** doesn’t rely on the coefficients of the fixed effects - and that’s why we are refitting our full and reduced models above with the addition of REML = FALSE in the call.

Even though you **use ML to compare models**, you should **report parameter estimates from your final “best” REML model**, as ML may underestimate variance of the random effects.

**NOTE 2:** Models can also be compared using the AICc function from the AICcmodavg package. The Akaike Information Criterion (AIC) is a measure of model quality. AICc corrects for bias created by small sample size when estimating AIC. Generally, if models are within 2 AICc units of each other they are very similar. Within 5 units they are quite similar, over 10 units difference and you can probably be happy with the model with lower AICc. As with p-values though, there is no “hard line” that’s always correct.

**NOTE 3:** There isn’t really an agreed upon way of dealing with the variance from the random effects in mixed models when it comes to assessing significance. Both **p-values** and **effect sizes** have issues, although from what I gather, p-values seem to cause more disagreement than effect sizes, at least in the R community.

### Random effects structure

Now you might wonder about selecting your random effects. In general, I’d advise you to think about your **experimental design, your system and data collected, as well as your questions**.

If your random effects are there to deal with **pseudoreplication**, then it doesn’t really matter whether they are “significant” or not: they **are part of your design** and have to be included. Imagine we tested our dragons multiple times - we then have to fit dragon identity as a random effect.

On the other hand, if you are trying to account for other variability that you think might be important, it becomes a bit harder. Imagine we measured the mass of our dragons over their lifespans (let’s say 100 years). We might then want to fit year as a random effect to account for any temporal variation - maybe some years were affected by drought, the resources were scarce and so dragon mass was negatively impacted. Year would definitely be a sensible random effect, although strictly speaking not a must.

When it comes to such random effects you can use **model selection** to help you decide what to keep in. Following Zuur’s advice, we **use REML estimators for comparison of models with different random effects** (we keep fixed effects constant). (Zuur: “Two models with nested random structures cannot be done with ML because the estimators for the variance terms are biased.” )

**NOTE:** Do **NOT** vary random and fixed effects at the same time - either deal with your random effects structure or with your fixed effects structure at any given point.

**NOTE 2:** Do **NOT** compare lmer models with lm models (or glmer with glm).

### Entire model selection

A few notes on the process of model selection. There are two ways here: (i) **“top-down”**, where you start with a complex model and gradually reduce it, and (ii) **“step up”**, where you start with a simple model and add new variables to it. Unfortunately, you might arrive at different final models by using those strategies and so you need to be careful.

The model selection process recommended by Zuur et al. (2009) is a top-down strategy and goes as follows:

1. fit a **full model** (he even recommends “beyond optimal” i.e. more complex than you’d expect or want it to be)
2. sort out the **random effects structure** (use REML likelihoods or REML AIC or BIC)
3. sort out **fixed effects structure** (either use REML the F-statistic or the t-statistic or compare nested ML models - keep your random effects constant)
4. once you arrive at the **final model present it using REML estimation**

**NOTE:** At the risk of sounding like a broken record: I think it’s best to decide on what your model is based on biology/ecology/data structure etc. than through following model selection blindly. Additionally, just because something is non-significant doesn’t necessarily mean you should always get rid of it.

**Wrap-up**

* Always check the residuals and the random effects! While both linear models and LMMs require normally distributed residuals with homogeneous variance, the former assumes independence among observations and the latter normally distributed random effects. Use normalized residuals to establish comparisons.
* One key additional advantage of LMMs we did not discuss is that they can handle missing values.
* Wide format data should be first converted to long format, using *e.g.* the R package reshape.
* Variograms are very helpful in determining spatial or temporal dependence in the residuals. In the case of spatial dependence, bubble plots nicely represent residuals in the space the observations were drown from
* REML estimation is unbiased but does not allow for comparing models with different fixed structures. Only use the REML estimation on the optimal model.

With respect to this particular set of results:

* The analysis outlined here is not as exhaustive as it should be. Among other things, we did neither initially consider interaction terms among fixed effects nor investigate in sufficient depth the random effects from the optimal model.
* The dependent variable (total fruit set per plant) was highly right-skewed and required a log-transformation for basic modeling. The large amount of zeros would in rigour require zero inflated GLMs or similar approaches.
* All predictors used in the analysis were categorical factors. We could similarly use an ANOVA model. LMMs are likely more relevant in the presence of quantitative or mixed types of predictors.