FES 524 Winter 2022 Lab 3

Contents

Linear mixed models

In Lab 2 you fit a linear model with the only random effect as the observation-level random effect, which is the residual error term. In Lab 3 you will begin to work with models with additional random effects using linear mixed models.

Today you will see a scenario where we need to model additional random effects due to observations of the response variable being grouped together in space. We need to know the study design to help us figure out if observations are grouped.

Load packages for data manipulation and plotting

```
library(dplyr)
library(ggplot2)
```

Read in the dataset

Working with a CSV file

We will be working with the dataset `lab3.example.data.csv`, so make sure you have this file and have set your working directory appropriately. We will be reading this comma-delimited file in using `read.csv()`. You could use `read.table()` with the `sep` argument set to `","`, as well.

If you look at the help file for `read.csv()`, you'll see that by default the `header` argument is set to `TRUE`, so we will not need to include that argument when we have column names in our dataset.

```
?read.csv
```

As usual when we read in a dataset for the first time we'll take a look at it by looking at the top 6 lines and looking at the structure in the Environment pane.

```
planting = read.csv("lab3.example.data.csv")

head(planting)
```

```
  nursery plantdate growth
1       1     2-Jan  1.980
2       2     2-Jan  1.295
3       3     2-Jan  2.515
4       4     2-Jan  2.335
5       5     2-Jan  2.545
6       1    28-Jan  2.395
```

```
'data.frame':   15 obs. of  3 variables:
 $ nursery  : int  1 2 3 4 5 1 2 3 4 5 ...
 $ plantdate: chr  "2-Jan" "2-Jan" "2-Jan" "2-Jan" ...
 $ growth   : num  1.98 1.29 2.52 2.33 2.54 ...
```

Changing factor labels and levels

We can see that the variable `plantdate` is a character variable. We'll want this to be a factor and make sure the levels are in order by date before we fit the model. By default `25-Feb` will come before `28-Jan`, which doesn't make a lot of sense.

Using skills learned last week, we'll use `factor()` to change the order of the levels to be chronological with the `levels` argument. While we're at it, let's change the `labels`, purely for aesthetic reasons. See lab 2 for a review of setting the order of the levels and changing labels of factors.

```
planting$plantdate = factor(planting$plantdate,
                            levels = c("2-Jan", "28-Jan", "25-Feb"),
                            labels = c("Jan 2", "Jan 28", "Feb 25"))

planting$plantdate
```

```
 [1] Jan 2  Jan 2  Jan 2  Jan 2  Jan 2  Jan 28 Jan 28 Jan 28 Jan 28 Jan 28 Feb 25 Feb 25 Feb 25
[14] Feb 25 Feb 25
Levels: Jan 2 Jan 28 Feb 25
```

The variable `nursery` is an integer, which we saw when we looked at the structure of the dataset above. This variable is really categorical, so let's change this to a factor, as well.

```
planting$nursery = factor(planting$nursery)
```

Now we can look at the structure of the edited dataset in the Environment pane to make sure we made the changes correctly.

```
'data.frame':   15 obs. of  3 variables:
 $ nursery  : Factor w/ 5 levels "1","2","3","4",..: 1 2 3 4 5 1 2 3 4 5 ...
 $ plantdate: Factor w/ 3 levels "Jan 2","Jan 28",..: 1 1 1 1 1 2 2 2 2 2 ...
 $ growth   : num  1.98 1.29 2.52 2.33 2.54 ...
```

Initial data exploration

Summary statistics

As usual, we'll be looking at group summary statistics and creating some figures to explore our dataset graphically. We can get an initial feel for the dataset, including the number of observations in each group, by using the `summary()` function.

In this summary you can see that there are 5 observations for each `plantdate`. This makes sense, as each date was used in every nursery. In addition, you can see that there are 3 observations in every `nursery`. This also makes sense, since every nursery had each date. The researchers used a *blocked* study design, which is reflected in this summary.

```
summary(planting)
```

```
 nursery  plantdate     growth
 1:3     Jan 2 :5   Min.   :0.325
 2:3     Jan 28:5   1st Qu.:1.340
 3:3     Feb 25:5   Median :1.980
 4:3                Mean   :1.856
 5:3                3rd Qu.:2.518
                    Max.   :2.865
```

We can also make a table of summary statistics by `plantdate` using `group_by()` and `summarise()` from package **dplyr**. This will allow us to start thinking about our assumptions and what we might expect the answer to our research questions to be. Notice that the response `growth` is measured to the thousandths of a centimeter and so I round the summary table accordingly.

What do you note that is interesting in this summary?

```
( sumtable = summarise( group_by(planting, "Date" = plantdate),
         n = n(),
         mean = mean(growth),
         sd = round( sd(growth), 3),
         .groups = "drop") )
```

```
# A tibble: 3 x 4
  Date       n  mean    sd
  <fct>  <int> <dbl> <dbl>
1 Jan 2      5  2.13 0.52
2 Jan 28     5  2.34 0.580
3 Feb 25     5  1.09 0.529
```

Using the pipe with dplyr functions

For the last two weeks you've seen how to use **dplyr** functions for summarizing datasets by groups. We've been using what's called *nested* code, where we put one function inside another. This is fine, but the **dplyr** package was designed in such a way that it can be used with "pipes". The "pipe" in R is `%>%` and is pronounced *then*.

The pipe works by passing the dataset in to the first argument of a function. This is the reason the **dplyr** functions have the dataset as the first argument.

Below is the same summary table made using pipes. The pipe can make code very readable compared to other coding methods.

To read this code out loud, we would say:

> Take the planting dataset and *then*
> group it by `plantdate` and *then*
> calculate summary statistics for each `plantdate` group

Note that the standard coding practice is to have line breaks after each line (ending with a pipe) rather than stringing everything together on a single row. This is for readability, the same reason we include spaces in code.

```
planting %>%
    group_by("Date" = plantdate) %>%
    summarise(n = n(),
            Mean = mean(growth),
            SD = round( sd(growth), 3),
            .groups = "drop")
```

```
# A tibble: 3 x 4
  Date       n  Mean    SD
  <fct>  <int> <dbl> <dbl>
1 Jan 2      5  2.13 0.52
2 Jan 28     5  2.34 0.580
3 Feb 25     5  1.09 0.529
```

The above code only printed to the Console. We need to assign a name to the object like we did with the nested coded.

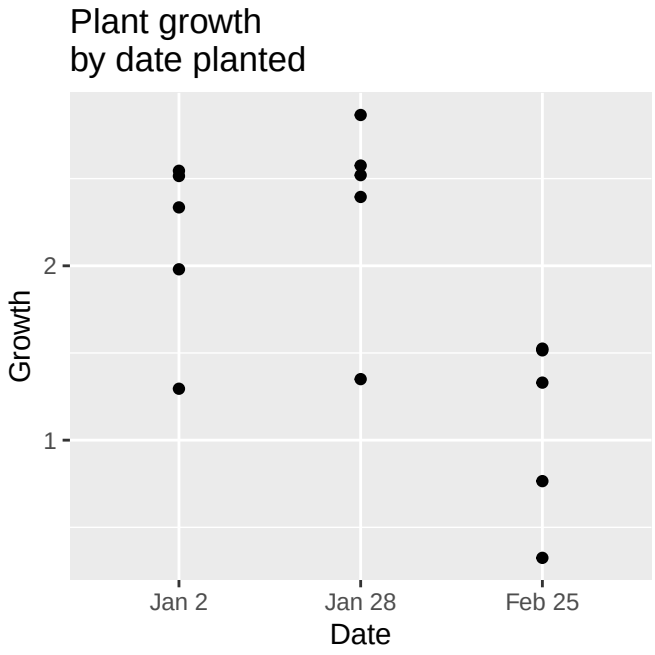
```
sumtable = planting %>%
    group_by("Date" = plantdate) %>%
    summarise(n = n(),
              Mean = mean(growth),
              SD = round( sd(growth), 3),
              .groups = "drop")
```

People often either love pipes or hate 'em. You will see them in example code for this class, and you can use them or not in your own work as you see fit.

Graphical data exploration

Our first plot will be a simple scatterplot with `growth` on the y-axis and our explanatory variable `plantdate` on the x-axis. This most interesting thing in this plot to my eye is that growth for February 25 was much lower than the other dates.

```
qplot(x = plantdate, y = growth,
      data = planting,
      xlab = "Date", ylab = "Growth",
      main = "Plant growth\nby date planted")
```

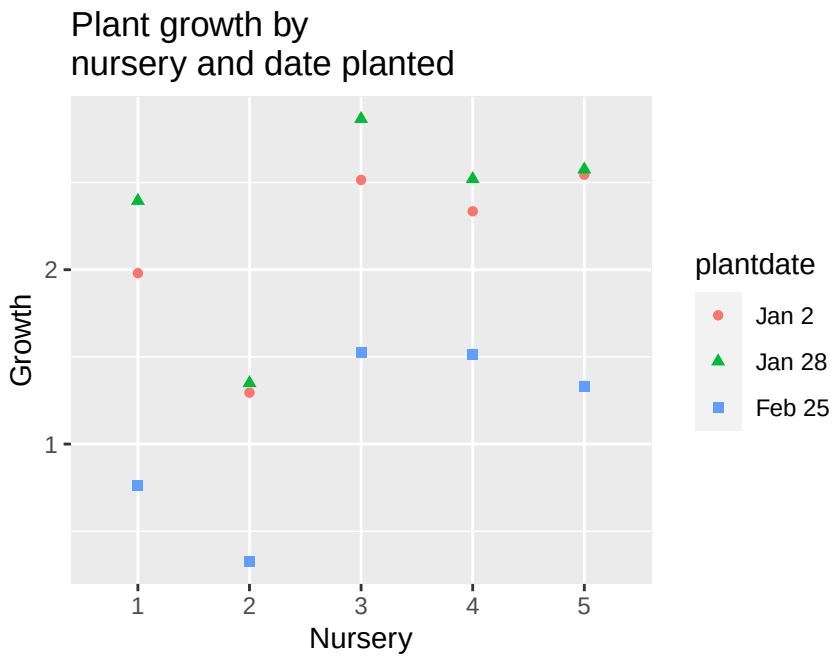


We have more variables to think about this week, though. While we don't have any questions specific to `nursery`, we will want to see how `growth` changes overall among nurseries. We can plot `nursery` as the `x` variable and color/shape the points by the `plantdate` variable to get a better feel for this.

We expect the relationships among the transplanting dates to be generally the same across all nurseries. You can see in the plot below that this is true; January 28 has the highest growth in every nursery.

We also might expect that overall some nurseries have greater or lower growth than others due to uncontrolled differences in, e.g., the location of a nursery or how a nursery operates. You could imagine differences in micro-climates or differences in amount and types of fertilizer used at individual nurseries causing overall differences in `growth` across all the planting dates. We see this sort of pattern in the graph, particularly for nursery 2.

```
qplot(x = nursery, y = growth,
      color = plantdate,
      shape = plantdate,
      data = planting,
      xlab = "Nursery", ylab = "Growth",
      main = "Plant growth by\nnursery and date planted")
```



Fitting a linear mixed model with `lme()`

In this study, `plantdate` is the factor variable of interest and the research questions are all about differences in mean growth increment between transplanting dates. However, the study was designed in such a way that we expect measurements of growth within a nursery to be more alike than measurements between nurseries. This is what I referred to as measurements being grouped in space at the beginning of this lab. Essentially, measurements within nurseries are not *independent* of each other so we need a model that addresses this nonindependence so the model *errors* are independent.

You may hear such designs called *nested* designs, where the observations are *nested* in nursery. The variable `nursery` can be considered a *blocking* variable, which you learned about in the lecture material.

In addition to the issue of nonindependence, we expect to have variation in growth due to nursery-to-nursery differences as discussed earlier when looking at the growth by nursery plot. We want to account for nursery-to-nursery variability separately from observation-to-observation variability.

Both the independence issue and the variability from different sources can be addressed with a linear mixed model (LMM). We will fit a LMM to these data to answer our research question, where `nursery` will be considered a random effect and `plantdate` a fixed effect. The *mixed* part of the model refers to mixing random and fixed effects in a single model.

In this class, we will be using function `lme()` from package **nlme** to fit linear mixed models for data from nested designs in R.

```
library(nlme)
```

When we fit a model with `lme()`, the fixed effects portion of the model is written using the same formula coding we used with `lm()`. The random portion of the model is defined using the `random` argument. You cannot fit a model with `lme()` without using the `random` argument; you would have to switch to the `gls()` function for that.

In this model we will allow for a random effect of nursery by using the one-sided formula `~1|nursery` for the `random` argument. The observation-level random effect of observations nested within nursery (the residual error term) is included automatically when fitting linear mixed models with `lme()`.

```
model1 = lme(growth ~ plantdate,
             random = ~1|nursery,
             data = planting)
```

The resulting model, which named `model1`, is called an **lme** object. It is generally useful to know which functions were modified to specifically work with **lme** objects. We can use `methods()` like we did last week for **lm** objects.

```
methods(class = "lme")
```

```
 [1] ACF             anova           augPred         coef            comparePred
```

```
 [6] confint          deviance         extractAIC       fitted           fixef
[11] formula          getData          getGroups        getGroupsFormula getResponse
[16] getVarCov        intervals        logLik           nobs             pairs
[21] plot             predict          print            qqnorm           ranef
[26] residuals        sigma            simulate         summary          update
[31] VarCorr          Variogram        vcov
see '?methods' for accessing help and source code
```

Checking model assumptions

As always, we'll need to check the assumptions of the model using the residual and fitted values from the model. We can add the residuals directly to the dataset `planting` for graphical checking of our assumptions.

Residuals vs fitted values graphs

Here we will use the `residuals()` function to pull out the residual values from the **lme** model object `model1` and add them to the dataset.

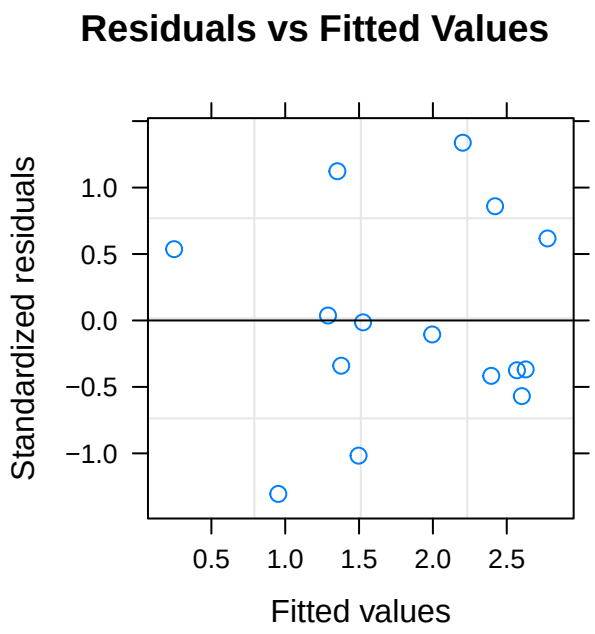
To look at the help page for the lme-specific version of these functions you need to add `.lme` to the end of the function name. The functions have different defaults than their `.lm` counterparts. For example, if we look at the help page at `?residuals.lme` we can see that there are different types of residuals to choose from and so we need to find find out what the default type is. The default `type` is the "response" residuals, which are fine for what we're doing today.

```
planting$res = residuals(model1)
```

The `plot()` function for **lme** objects will give us a residuals vs fitted values plot. Like with `residuals.lme`, if you need to look at the help page you would need to type out `?plot.lme` instead of just `?plot`. Note that this plot has *standardized* residuals on the y axis; see the help page for `residuals.lme` to see the definition of these. We'll see these more as the term progresses.

What is interesting to you in this plot? Do any unusual patterns stand out? Certainly you should note the single point that has a small fitted value.

```
plot(model1, main = "Residuals vs Fitted Values")
```

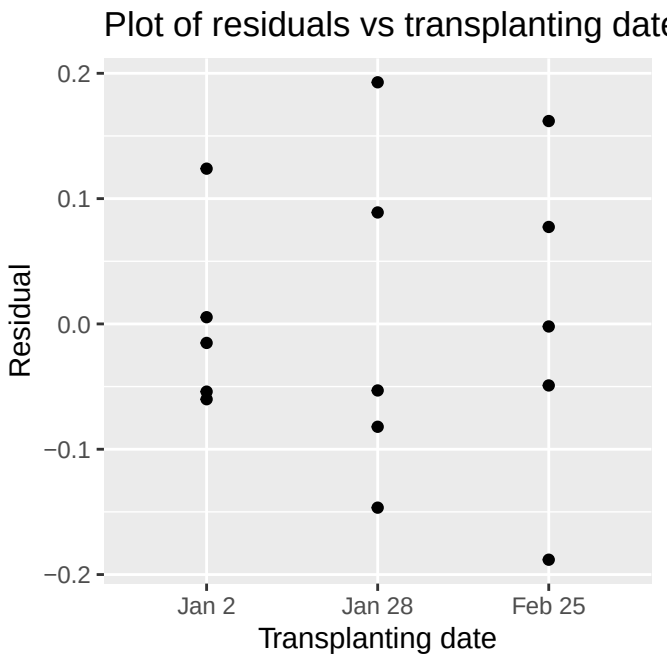


Residuals vs explanatory variables graphs

We can create a plot of residuals vs the explanatory variable using `qplot()`. We are looking for unusual values in these plots and thinking about the assumption of constant variance in particular. Plots of the residuals vs explanatory variables can help us detect what might be causing anything unusual we see in the residuals vs fitted values plot.

In this case we can see we have slightly less variation on January 2 compared to the other transplanting dates, although I ultimately decided this wasn't enough of a difference to matter. Compare the full spread of each group to help you decide if this is a problem or not; e.g., a group with a spread of 2 (residual values between -1 and 1) has twice as much variability as a group with a spread of 1 (residual values between -0.5 and 0.5).

```
qplot(x = plantdate, y = res, data = planting,
      main = "Plot of residuals vs transplanting date",
      xlab = "Transplanting date",
      ylab = "Residual")
```



Histograms and quantile-quantile plots

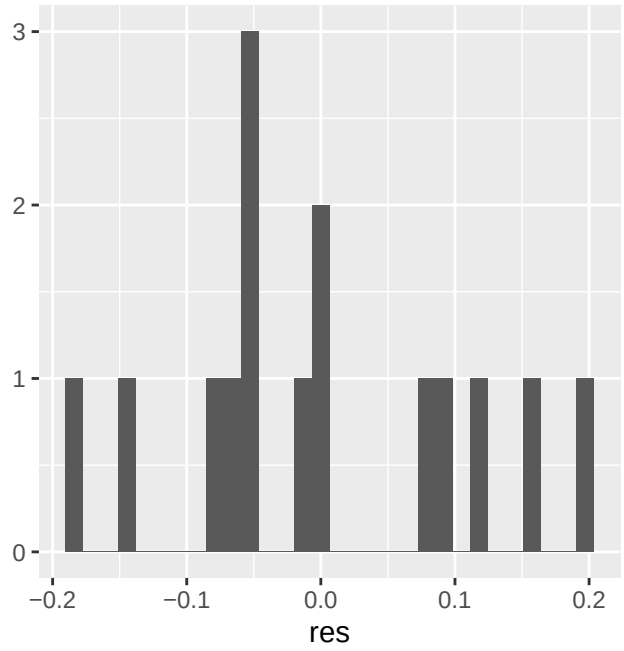
To check the assumption of normality/symmetry of the residuals we can use histograms or quantile-quantile plots.

A histogram may not work great here due to the small dataset. Also consider a boxplot as an alternative.

Does this distribution look symmetric?

```
qplot(x = res, data = planting, geom = "histogram")
```

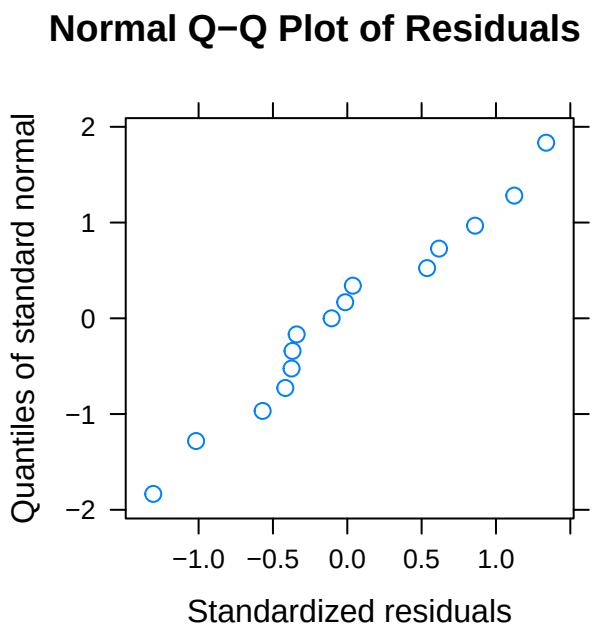
```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



As you saw above when we printed out the **lme** methods, there is a specific `qqnorm()` function that works directly with **lme** objects instead of the residuals. Notice the axes are different than the generic `qqnorm()` function we used last week and we don't get the theoretical line drawn on.

Don't make or report quantile-quantile plots if you don't know how to interpret the results. Using boxplots or histograms is most often sufficient for checking the symmetry of residuals.

```
qqnorm(model1, main= "Normal Q-Q Plot of Residuals")
```



Model results

Overall F-tests and the model summary

If the assumptions are reasonably met, we can report any model results of interest from `anova()` and/or `summary()`, both of which work directly on **lme** objects. For more help on these, see `?anova.lme` and `?summary.lme` for what the output means.

Take a look at the help page for `anova.lme` to compare it to `anova.lm` from last week; it does *not* return an ANOVA table

when used on **lme** objects but instead returns likelihood ratio tests.

Remember, never call the results from `anova.lme` "an ANOVA"; that implies you fit a linear model but here you fit a linear mixed model.

```
anova(model1)
```

```
            numDF denDF   F-value p-value
(Intercept)     1     8  61.11828   1e-04
plantdate       2     8 107.79680  <.0001
```

```
summary(model1)
```

```
Linear mixed-effects model fit by REML
  Data: planting
      AIC      BIC    logLik
  17.2214 19.64593 -3.610699

Random effects:
 Formula: ~1 | nursery
        (Intercept)  Residual
StdDev:   0.5241942 0.1441686

Fixed effects:  growth ~ plantdate
                 Value  Std.Error DF    t-value p-value
(Intercept)      2.134 0.24313130  8   8.777150  0.0000
plantdateJan 28  0.207 0.09118023  8   2.270229  0.0529
plantdateFeb 25 -1.042 0.09118023  8 -11.427916  0.0000
 Correlation:
                (Intr) plnJ28
plantdateJan 28 -0.188
plantdateFeb 25 -0.188  0.500

Standardized Within-Group Residuals:
       Min         Q1        Med         Q3        Max
-1.3051882 -0.3957695 -0.1052043  0.5769133  1.3375510

Number of Observations: 15
Number of Groups: 5
```

Estimating differences among transplanting dates

Using the `summary()` output This study was designed to estimate differences in mean growth increments among the three transplanting dates. Because this is a simple question, we could use the default summary output to answer our questions using `fixef()` and `intervals()` for **lme** objects much like we used `coef()` and `confint()` for **lm** objects in Lab 2.

```
# Coefficients
fixef(model1)
```

```
    (Intercept) plantdateJan 28 plantdateFeb 25
          2.134           0.207          -1.042
```

```
# Confidence intervals for all model effects (random and fixed)
intervals(model1)
```

```
Approximate 95% confidence intervals

 Fixed effects:
                       lower   est.     upper
(Intercept)      1.573338208  2.134  2.694662
plantdateJan 28 -0.003261977  0.207  0.417262
plantdateFeb 25 -1.252261977 -1.042 -0.831738
attr(,"label")
```

```
[1] "Fixed effects:"

 Random Effects:
  Level: nursery
                    lower      est.    upper
sd((Intercept)) 0.2575838 0.5241942 1.066758

 Within-group standard error:
     lower       est.      upper
0.08832235 0.14416860 0.23532643
```

```
# Confidence intervals for fixed effects
intervals(model1)$fixed
```

```
                        lower   est.     upper
(Intercept)      1.573338208  2.134  2.694662
plantdateJan 28 -0.003261977  0.207  0.417262
plantdateFeb 25 -1.252261977 -1.042 -0.831738
attr(,"label")
[1] "Fixed effects:"
```

Estimated differences in means to answer the research question with `emmeans()` However, as models get more complicated we are going to start using the helper function `emmeans()` from package **emmeans** to calculate the comparisons of interest for us. This package has a series of vignettes to show you some of the (many) options. See the basics vignette to get started. I also wrote a couple of posts on how to use **emmeans**, which you can see here.

Let's load the package to get started.

```
library(emmeans)
```

There are a variety of ways to use package **emmeans** to get estimated marginal means per group and comparisons of means among groups from models of many different types. I like to use the "formula" notation with the built-in "contrast methods", which I'll demonstrate today.

You can see the difference contrast methods via `?"contrast-methods"`. The ones I use most often are `pairwise` and `trt.vs.ctrl`.

To get started we need a fitted model object with at least one categorical explanatory variable, which we have, and a variable we want comparisons for. We ask for the comparisons via the `specs` argument. We'll request all pairwise comparisons among transplanting dates.

Notice that we get both the estimated means per group (the *estimated marginal means*) as well a the comparisons of interest (i.e., contrasts).

```
emmeans(model1, specs = pairwise ~ plantdate)
```

```
$emmeans
 plantdate emmean    SE df lower.CL upper.CL
 Jan 2       2.13 0.243  4    1.459     2.81
 Jan 28      2.34 0.243  4    1.666     3.02
 Feb 25      1.09 0.243  4    0.417     1.77

Degrees-of-freedom method: containment
Confidence level used: 0.95

$contrasts
 contrast        estimate     SE df t.ratio p.value
 Jan 2 - Jan 28    -0.207 0.0912  8  -2.270  0.1183
 Jan 2 - Feb 25     1.042 0.0912  8  11.428  <.0001
 Jan 28 - Feb 25    1.249 0.0912  8  13.698  <.0001

Degrees-of-freedom method: containment
P value adjustment: tukey method for comparing a family of 3 estimates
```

If we wanted to report the comparisons in the other direction we can use `revpairwise` instead.

```
emmeans(model1, specs = revpairwise ~ plantdate)
```

```
$emmeans
 plantdate emmean    SE df lower.CL upper.CL
 Jan 2       2.13 0.243  4    1.459     2.81
 Jan 28      2.34 0.243  4    1.666     3.02
 Feb 25      1.09 0.243  4    0.417     1.77

Degrees-of-freedom method: containment
Confidence level used: 0.95

$contrasts
 contrast        estimate     SE df t.ratio p.value
 Jan 28 - Jan 2     0.207 0.0912  8   2.270  0.1183
 Feb 25 - Jan 2    -1.042 0.0912  8 -11.428  <.0001
 Feb 25 - Jan 28   -1.249 0.0912  8 -13.698  <.0001

Degrees-of-freedom method: containment
P value adjustment: tukey method for comparing a family of 3 estimates
```

Thinking about multiple comparisons Note the default message

> P value adjustment: tukey method for comparing a family of 3 estimates

The **emmeans** package adjusts for multiple comparisons by default, defaulting to the Tukey adjustment when we are doing all pairwise comparisons with `pairwise`. We need to decide *a priori* if we'll be adjusting for multiple comparisons or not.

Given the danger of telling growers to change their practices based on this single small study, it's probably reasonable to do a multiple comparisons adjustment here (although the study is very small, which is another consideration).

You'll want to review the issues of multiple comparisons. See chapter 6 of the Statistical Sleuth for more details.

Extract comparisons from `emmeans()` output We haven't assigned our results from `emmeans()` to an object name yet, so let's do that. This is called an **emmGrid** object.

```
em_model1 = emmeans(model1, specs = pairwise ~ plantdate)
```

Since we'll be plotting just the comparisons, we'll need to pull these out of the results. This can be done via dollar sign notation, pulling out the `contrasts`.

```
em_model1$contrasts
```

```
 contrast        estimate     SE df t.ratio p.value
 Jan 2 - Jan 28    -0.207 0.0912  8  -2.270  0.1183
 Jan 2 - Feb 25     1.042 0.0912  8  11.428  <.0001
 Jan 28 - Feb 25    1.249 0.0912  8  13.698  <.0001

Degrees-of-freedom method: containment
P value adjustment: tukey method for comparing a family of 3 estimates
```

The comparisons have statistical hypothesis tests with them but don't have any confidence intervals. We want the latter, and can take or leave the former (depending on what we want to report). We can request confidence intervals with the `summary()` function for the **emmGrid** object. See `?summary.emmGrid` for a lot of arguments you can use, including the `adjust` argument to change the kind of multiple comparisons adjustment you want to use.

The `infer` argument is what we can use to get confidence intervals. It takes two logical values. The first controls whether or not we want CI and the second controls whether or not we want hypothesis tests. I'm going to use `c(TRUE, FALSE)` to get CI but not the test (use `c(TRUE, TRUE)` to keep the tests, as well).

Note that `level` is how we set the CI size, which defaults to 0.95 for 95% CI and are what I'm using today.

```
summary( em_model1$contrasts, infer = c(TRUE, FALSE) )
```

```
 contrast        estimate     SE df lower.CL upper.CL
 Jan 2 - Jan 28    -0.207 0.0912  8   -0.468   0.0535
 Jan 2 - Feb 25     1.042 0.0912  8    0.781   1.3025
 Jan 28 - Feb 25    1.249 0.0912  8    0.988   1.5095

Degrees-of-freedom method: containment
Confidence level used: 0.95
Conf-level adjustment: tukey method for comparing a family of 3 estimates
```

The **emmeans** package makes things easy to extract the results into a data.frame for easy plotting, since using the `summary()` function automatically returns a data.frame.

Let's assign the resulting data.frame of comparisons a name so we can use it to plot the results.

```
em_dat = summary( em_model1$contrasts, infer = c(TRUE, FALSE) )
```

The advantage of accounting for nursery-level variation

Accounting for unexplained nursery-level variation actually helps us estimate differences among group means with more precision. Let's look at the results from a model where we *didn't* control for nursery-level variation. We can do this with the `lm()` function. Remember, we can't use the `lme()` function if we don't have random effects.

```
modelnaive = lm(growth ~ plantdate, data = planting)
```

Looking at the summary output we see that the estimates of means/differences in means are the same for the mixed model and this naive linear model.

```
# Note the estimated means are the same
summary(modelnaive)
```

```
Call:
lm(formula = growth ~ plantdate, data = planting)

Residuals:
    Min      1Q  Median      3Q     Max
-0.9910 -0.2405  0.2010  0.3960  0.5240

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)       2.1340     0.2431   8.777 1.44e-06 ***
plantdateJan 28   0.2070     0.3438   0.602   0.5584
plantdateFeb 25  -1.0420     0.3438  -3.030   0.0105 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5437 on 12 degrees of freedom
Multiple R-squared:  0.5582,    Adjusted R-squared:  0.4846
F-statistic:  7.58 on 2 and 12 DF,  p-value: 0.007437
```

We can use `emmeans()` to calculate the comparisons of interest from this naive model much like we did before. Notice I'm using a pipe here to get the confidence intervals from `summary()`.

```
emmeans(modelnaive, specs = pairwise ~ plantdate)$contrasts %>%
    summary(infer = c(TRUE, FALSE) )
```

```
 contrast        estimate    SE df lower.CL upper.CL
 Jan 2 - Jan 28    -0.207 0.344 12   -1.124     0.71
 Jan 2 - Feb 25     1.042 0.344 12    0.125     1.96
 Jan 28 - Feb 25    1.249 0.344 12    0.332     2.17
```

```
Confidence level used: 0.95
Conf-level adjustment: tukey method for comparing a family of 3 estimates
```

What differences do you see between the model controlling for the nursery-level variation and a naive model where we just ignore it? Let's compare.

The important practical difference in mean growth increment to growers is if one transplanting date has a mean growth increment of at least 0.25 centimeters more than another. As you can see, accounting for nursery-level variation helps us get a narrower range of plausible values around the differences and in some cases makes the results more conclusive. This means that using a linear mixed model following our study design has actually helped us compared to if we had ignored the nested structure and non-independence of observations in the study design.

Note they also have different degrees of freedom for the tests, since the naive model inappropriately treated all the groups within nurseries as the independent units.

```
em_dat
```

```
 contrast        estimate     SE df lower.CL upper.CL
 Jan 2 - Jan 28    -0.207 0.0912  8   -0.468   0.0535
 Jan 2 - Feb 25     1.042 0.0912  8    0.781   1.3025
 Jan 28 - Feb 25    1.249 0.0912  8    0.988   1.5095

Degrees-of-freedom method: containment
Confidence level used: 0.95
Conf-level adjustment: tukey method for comparing a family of 3 estimates
```

```
emmeans(modelnaive, specs = pairwise ~ plantdate)$contrasts %>%
    summary(infer = c(TRUE, FALSE) )
```

```
 contrast        estimate    SE df lower.CL upper.CL
 Jan 2 - Jan 28    -0.207 0.344 12   -1.124     0.71
 Jan 2 - Feb 25     1.042 0.344 12    0.125     1.96
 Jan 28 - Feb 25    1.249 0.344 12    0.332     2.17

Confidence level used: 0.95
Conf-level adjustment: tukey method for comparing a family of 3 estimates
```

Group means with `emmeans()`

We can get the estimated marginal means and confidence intervals for each transplanting date from `emmeans()`, as well, via dollar sign notation.

Group means and confidence intervals are not an efficient way to answer questions about differences, so we won't be using this approach in this class. However, it's not uncommon for folks to want to know how to do this or to make plots of the estimated marginal means.

```
em_model1$emmeans
```

```
 plantdate emmean    SE df lower.CL upper.CL
 Jan 2       2.13 0.243  4    1.459     2.81
 Jan 28      2.34 0.243  4    1.666     3.02
 Feb 25      1.09 0.243  4    0.417     1.77

Degrees-of-freedom method: containment
Confidence level used: 0.95
```

Wrapping up an analysis

Graphic of results

We can use the results we saved in `em_dat` in a graphic. The comparisons in the `contrast` variable are pretty good, but it might be clearer to replace `-` with the word `minus` to be clearer.

I set the order and the names of the levels of this variable so my graph will be in the order I want with the names on the axes I want.

This is what `contrast` looks like to start.

```
em_dat$contrast
```

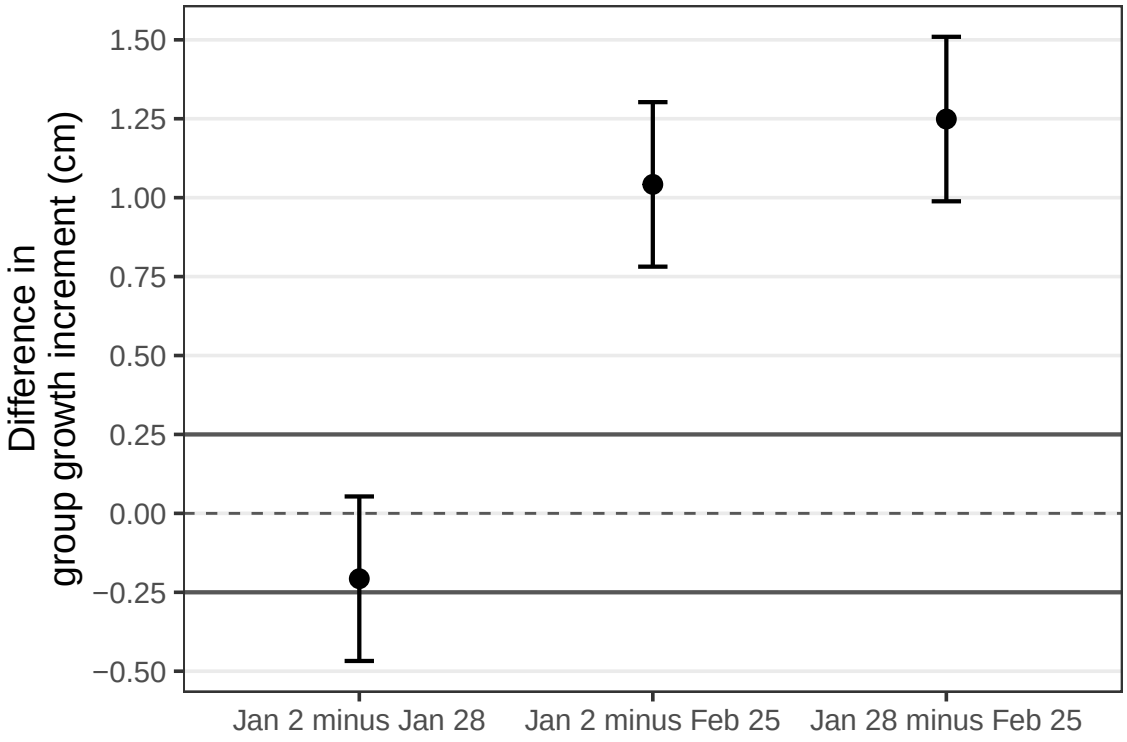
```
[1] "Jan 2 - Jan 28"  "Jan 2 - Feb 25"  "Jan 28 - Feb 25"
```

Here's how I'll change it.

```
em_dat$contrast = factor(em_dat$contrast,
                      levels = c("Jan 2 - Jan 28",
                                 "Jan 2 - Feb 25",
                                 "Jan 28 - Feb 25"),
                      labels = c("Jan 2 minus Jan 28",
                                 "Jan 2 minus Feb 25",
                                 "Jan 28 minus Feb 25") )
```

Here is a graphic showing the estimated differences and Tukey-adjusted 95% confidence intervals for a family of three comparisons.

```
( g1 = ggplot(em_dat, aes(x = contrast, y = estimate) ) +
    # Indicate dataset name and what's on each axis
    geom_hline(yintercept = 0, color="grey34", lty = 2) +  # Add horizontal line at 0 for reference
    geom_hline(yintercept = c(.25, -.25), color = "grey34", lwd = .75) +
    # Add horiz line at -.25 and .25 cm
    geom_errorbar(width = .1, lwd = .75, aes(ymin = lower.CL, ymax = upper.CL)) +
    # Add error bars with from lower and upper CI; change line width
    geom_point(size=3) + # Add point estimate, change point size
    theme_bw(base_size = 14) +  # Change graph to black and white
    labs(x = NULL, # Change labels on axes, leave x axis blank
        y = "Difference in\ngroup growth increment (cm)") +
    theme(panel.grid.major.x = element_blank(),
         # Remove vertical gridlines
         panel.grid.minor.y = element_blank() ) +
    # Remove minor y gridlines
    scale_y_continuous(breaks = seq(-.5, 1.5, by = .25) ) ) # Put in more y tick labels
```


Difference in
group growth increment (cm)
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
Jan 2 minus Jan 28
Jan 2 minus Feb 25
Jan 28 minus Feb 25