Linear modeling in R

Kim Dill-McFarland, kadm@uw.edu

version March 08, 2022

Contents

Overview

In this workshop, we introduce linear modeling in R from a t-test to ANOVA to simple regression to mixed effects models. We also touch on tools for large modeling projects such as RNA-seq differential expression.

Video at https://youtu.be/UGtgeRPmc6I

Prior to the workshop

Please install R, RStudio, and the following packages.

```
#Install (once per computer)
# install.packages(c("tidyverse", "lme4", "car",
#                    "BiocManager","devtools"))
# BiocManager::install(c("limma","variancePartition))
# devtools::install_github("BIGslu/kimma")

#Load (every time you open a new R/RStudio session)
library(tidyverse)
library(broom)
library(lme4)
library(car)
library(limma)
library(variancePartition) #dream
library(kimma)
```

Load data

Briefly, these data are from RNA-sequencing of human dendritic cells cultured with and without virus. Samples are from 3 donors and a random subset of 1000 genes were selected. Expression data are in an limma EList object (named `example.voom`) containing expression (`E`), sample/patient metadata (`targets`), and gene metadata (`genes`). Expression is expressed as TMM-normalized log2 counts per million (CPM).

We combine the expression, sample, and gene metadata for 1 gene in a single table for use in modeling.

```
dat <- as.data.frame(example.voom$E) %>%
  rownames_to_column("geneName") %>%
  pivot_longer(-geneName, names_to = "libID") %>%
  inner_join(example.voom$targets, by = "libID") %>%
  inner_join(example.voom$genes, by = "geneName") %>%
  filter(hgnc_symbol == "ZNF439")

dat
```

```
## # A tibble: 12 x 15
##    geneName     libID value group lib.size norm.factors donorID median_cv_cover~
##    <chr>        <chr> <dbl> <fct>    <dbl>        <dbl> <chr>              <dbl>
##  1 ENSG0000017~ lib1   8.01 1       79646.        1.00  donor1             0.514
##  2 ENSG0000017~ lib2   7.15 1       88008.        0.951 donor1             0.435
##  3 ENSG0000017~ lib3   9.30 1      178020.        1.09  donor2             0.374
##  4 ENSG0000017~ lib4   5.07 1      133836.        0.943 donor2             0.388
##  5 ENSG0000017~ lib5   8.09 1      192547.        1.00  donor3             0.353
##  6 ENSG0000017~ lib6   8.67 1      175144.        0.974 donor3             0.349
##  7 ENSG0000017~ lib7   9.42 1      205377.        1.02  donor4             0.339
##  8 ENSG0000017~ lib8   7.81 1      149311.        0.995 donor4             0.350
##  9 ENSG0000017~ lib9   9.26 1      182080.        1.04  donor5             0.342
## 10 ENSG0000017~ lib10  8.84 1      181755.        0.980 donor5             0.330
## 11 ENSG0000017~ lib11  5.75 1      176991.        1.03  donor6             0.345
## 12 ENSG0000017~ lib12  5.39 1      131592.        0.977 donor6             0.356
## # ... with 7 more variables: virus <fct>, asthma <chr>, batch <dbl>,
## #   hgnc_symbol <chr>, `Previous symbols` <chr>, `Alias symbols` <chr>,
## #   gene_biotype <chr>
```

Introduction

Experimental design

We do not have time to extensively cover all that should go into experimental design *prior* to statistical modeling. However, some key aspects to consider are:

- Balance: balanced designs have roughly equal observations for each group.
- Randomization: subjects should be randomized to groups to help balance any unknown or unmeasured variables in the experiment. Randomization should support a balanced design as well.
- Blocking / repeated measures: when dealing with multiple observations from the same subject (like over time or pre/post treatment), this should be taken into account in models. Moreover, this type of design can add a lot of power to your analyses as it "cancels out" much of the subject-to-subject variation not usually of interest in your experiment.
- Assumptions: all statistical tests make assumptions about your data. You need to consider what test you're using and if your data fulfill these assumptions. We'll see this in practice later.

Writing formulae

In R, formulae are written in the form `y ~ x` where the `~` functions like an `=` in what you'd traditionally see written in text like `y = x`. In the models we explore here, you will only have 1 `y` variable. However, you may have more than 1 `x` variable, and these are added to the model like `y ~ x1 + x2 ...`. You may also have interaction terms in a model such as the impact of `x1` within `x2` groups. This is written with a `:` as in `y ~ x1 + x2 + x1:x2` or the shorthand `*` which stands for all individual variables and all interactions. This would be `y ~ x1 * x2` which is equivalent to `y ~ x1 + x2 + x1:x2`.

Formulae are a specific type of R object. Similar to other types, you can check if what you have is of this type with `class( )`.

```
class(y ~ x)
```

```
## [1] "formula"
```

Importantly, this does not check that this is the most correct model for your data. It merely let's you know that you've formatted the syntax correctly for R to recognize a formula.

Simple linear regression

t-test

Under the hood, a t-test is simply a special case of a simple linear regression where there is only 1 categorical `x` variable with exactly 2 levels to predict 1 numeric `y` variable.

Here, we run a t-test of gene expression `value` in control `none` vs virus `HRV` infected samples for our gene of interest. We see that gene expression does not significantly change with viral infection (P = 0.22).

```
TT <- t.test(formula = value ~ virus, data = dat)
TT
```

```
##
##  Welch Two Sample t-test
##
## data:  value by virus
## t = 1.3151, df = 9.8045, p-value = 0.2184
## alternative hypothesis: true difference in means between group none and group HRV is not equal to 0
## 95 percent confidence interval:
##  -0.8027454  3.0999532
## sample estimates:
```

```
## mean in group none  mean in group HRV
##           8.304152           7.155548
```

ANOVA

Similarly, analysis of variance (ANOVA) is another special case of simple linear regression. In this case, it is when there are 1 or more categorical `x` variables with 2 or more levels to predict 1 numeric `y` variable.

So, we could run the same model as in our t-test and get the same result. Notice that the `aov` function does not automatically estimate the P-value. So, we use another function to extract that in a data frame.

```
AV <- aov(formula = value ~ virus, data = dat)
AV
```

```
## Call:
##    aov(formula = value ~ virus, data = dat)
##
## Terms:
##                     virus Residuals
## Sum of Squares   3.957873 22.885276
## Deg. of Freedom         1        10
##
## Residual standard error: 1.512788
## Estimated effects may be unbalanced
```

```
tidy(AV)
```

```
## # A tibble: 2 x 6
##   term         df sumsq meansq statistic p.value
##   <chr>     <dbl> <dbl>  <dbl>     <dbl>   <dbl>
## 1 virus         1  3.96   3.96      1.73   0.218
## 2 Residuals    10 22.9    2.29     NA     NA
```

In future, you may see other functions to extract p-values such as `summary( )`. These work here as well; I just like the `broom` package's `tidy` function as it puts results in a data frame with consistent column names. However, `tidy` does not work on all model types.

Now, let's move beyond 1 `x` variable. If we were to add another variable to our t-test, it will not run since it no longer fulfills the special case of 1 `x` variable.

```
t.test(formula = value ~ virus + asthma, data = dat)
```

```
## Error in t.test.formula(formula = value ~ virus + asthma, data = dat): 'formula' missing or incorrec
```

So, we must use an ANOVA. We see that gene expression does not change significantly with either virus or asthma. Note that the p-value for virus is slightly different that the first model `value ~ virus` because adding variables impacts degrees of freedom for **all** variables in the model.

```
AV <- aov(formula = value ~ virus + asthma, data = dat)
AV
```

```
## Call:
##    aov(formula = value ~ virus + asthma, data = dat)
##
## Terms:
##                     virus    asthma Residuals
## Sum of Squares   3.957873  0.002241 22.883035
## Deg. of Freedom         1         1         9
##
```

```
## Residual standard error: 1.594541
## Estimated effects may be unbalanced
```

```
tidy(AV)
```

```
## # A tibble: 3 x 6
##   term         df    sumsq  meansq statistic p.value
##   <chr>     <dbl>    <dbl>   <dbl>     <dbl>   <dbl>
## 1 virus         1  3.96    3.96     1.56       0.244
## 2 asthma        1  0.00224 0.00224  0.000881   0.977
## 3 Residuals     9 22.9     2.54    NA         NA
```

Linear regression

We continue to generalize our model to the heart of both t-tests and ANOVA. Linear regression is the most general version of these tests and can be used for any number of `x` variables of any type (numeric, categorical).

We see that our last ANOVA model gave the same results as the `lm` model.

```
LM <- lm(formula = value ~ virus + asthma, data = dat)
LM
```

```
##
## Call:
## lm(formula = value ~ virus + asthma, data = dat)
##
## Coefficients:
##   (Intercept)       virusHRV  asthmahealthy
##       8.31782       -1.14860       -0.02733
```

```
tidy(LM)
```

```
## # A tibble: 3 x 5
##   term          estimate std.error statistic    p.value
##   <chr>            <dbl>     <dbl>     <dbl>      <dbl>
## 1 (Intercept)     8.32       0.797   10.4    0.00000251
## 2 virusHRV       -1.15       0.921   -1.25   0.244
## 3 asthmahealthy  -0.0273     0.921   -0.0297 0.977
```

Reference levels

All of our models thus far have compared categorical groups to a reference level. R assumes the reference is the first level alphabetically unless you code a factor to tell it otherwise. For example, `asthma` assumes the reference is "asthma" and then compares the other level, "healthy", to this reference. Hence why are see asthmahealthy in the model results.

```
sort(unique(dat$asthma))
```

```
## [1] "asthma"  "healthy"
```

In contrast, `virus` is a factor with levels forcing the reference to be "none" even though it is alphabetically after "HRV".

```
sort(unique(dat$virus))
```

```
## [1] none HRV
## Levels: none HRV
```

You could achieve this for the `asthma` variable re-coding it as a factor.

```
dat <- dat %>% mutate(asthma.fct = factor(asthma, levels = c("healthy", "asthma")))

tidy(lm(formula = value ~ virus + asthma.fct, data = dat))
```

```
## # A tibble: 3 x 5
##   term             estimate std.error statistic    p.value
##   <chr>               <dbl>     <dbl>     <dbl>      <dbl>
## 1 (Intercept)        8.29       0.797   10.4    0.00000258
## 2 virusHRV          -1.15       0.921   -1.25   0.244
## 3 asthma.fctasthma   0.0273     0.921    0.0297 0.977
```

Where did (Intercept) come from?

You may have noticed that we appear to have gained a variable with linear regression. This is the intercept of the fit line. In the case of categorical variables such as here, this variable is not meaningful. However, if you had a numeric `x` variable, you may wish to know what the estimated value of `y` is at `x` = 0 and if this significantly differs from (0,0). This variable does not appear for `t.test` or `aov` because they only work for categorical `x`.

You can remove the intercept from a linear model by starting with `0` such as below. However, even when the intercept isn't meaningful to your interpretation, this might not be what you want to do. In this case, removing the intercept means we have two results for `virus`, each asking if gene expression of that virus group is different from 0. We don't want this and instead, want the original model asking if none and HRV infected differ.

```
LM2 <- lm(formula = value ~ 0 + virus + asthma, data = dat)
LM2
```

```
##
## Call:
## lm(formula = value ~ 0 + virus + asthma, data = dat)
##
## Coefficients:
##     virusnone       virusHRV  asthmahealthy
##       8.31782        7.16921       -0.02733
```

```
tidy(LM2)
```

```
## # A tibble: 3 x 5
##   term          estimate std.error statistic    p.value
##   <chr>            <dbl>     <dbl>     <dbl>      <dbl>
## 1 virusnone       8.32       0.797   10.4    0.00000251
## 2 virusHRV        7.17       0.797    8.99   0.00000860
## 3 asthmahealthy  -0.0273     0.921   -0.0297 0.977
```

t-test or ANOVA or linear regression?

In practice in R, you never have to make this decision. You can simply use `lm( )` and it will take care of the cases where your data fit a t-test or ANOVA. You'll get the same result regardless of the function.

Assumptions

Linear modeling makes a number of assumptions about your data. You can see a more complete list here but the assumptions of importance for these data are:

- Samples are balanced
- The variance of `y` is roughly equal in all groups (homoscedasticity)

- Samples are independent

Let's check each of these. We'll use functions in the tidyverse here. To learn more about working with data in the tidyverse, see previous workshops

Balance All groups are balanced with 6 observations per group.

```
dat %>% count(virus)
```

```
## # A tibble: 2 x 2
##   virus     n
##   <fct> <int>
## 1 none      6
## 2 HRV       6
```

```
dat %>% count(asthma)
```

```
## # A tibble: 2 x 2
##   asthma      n
##   <chr>   <int>
## 1 asthma      6
## 2 healthy     6
```

Variance Variance of the `y` variable are roughly equal. To be considered unequal, you'd expect differences in order of magnitude like 1 vs 10.

```
dat %>% group_by(virus) %>% summarise(variance = var(value))
```

```
## # A tibble: 2 x 2
##   virus variance
##   <fct>    <dbl>
## 1 none      1.97
## 2 HRV       2.61
```

```
dat %>% group_by(asthma) %>% summarise(variance = var(value))
```

```
## # A tibble: 2 x 2
##   asthma  variance
##   <chr>      <dbl>
## 1 asthma      3.17
## 2 healthy     2.20
```

Independence By design, our samples are not independent because we have a control and virus sample from each donor. Thus, this assumption of linear regression is broken and the results above are not valid.

```
dat %>% count(donorID)
```

```
## # A tibble: 6 x 2
##   donorID     n
##   <chr>   <int>
## 1 donor1      2
## 2 donor2      2
## 3 donor3      2
## 4 donor4      2
## 5 donor5      2
## 6 donor6      2
```

Linear mixed effects regression

But never fear! We can account for our paired sample design in our model. To do this, we move away from simple linear regression into mixed effects regression. This term refers to the use of both fixed effects (`x`) and random effects (have not seen yet) in one model.

To add a random effect, you use the syntax `(across | within)` such as expression across time within each donor `(time | donor)`. If you don't have an across variable (usually time), you fill in a 1 as a placeholder such as `(1 | donor)`. The syntax for random effects in a model can vary in different R packages. Here, we are using the `lme4` package and its associated syntax. If you use another package in future, be sure to check its syntax!

Thus, our model of interest is as follows. Notes that like `aov` and `lm`, `lmer` does not automatically give p-values. We must estimate them separately.

```
LME <- lmer(formula = value ~ virus + asthma + (1 | donorID), data = dat)
LME
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: value ~ virus + asthma + (1 | donorID)
##    Data: dat
## REML criterion at convergence: 37.412
## Random effects:
##  Groups   Name        Std.Dev.
##  donorID  (Intercept) 1.136
##  Residual             1.182
## Number of obs: 12, groups:  donorID, 6
## Fixed Effects:
##   (Intercept)       virusHRV  asthmahealthy
##       8.31782       -1.14860       -0.02733
```

```
tidy(car::Anova(LME))
```

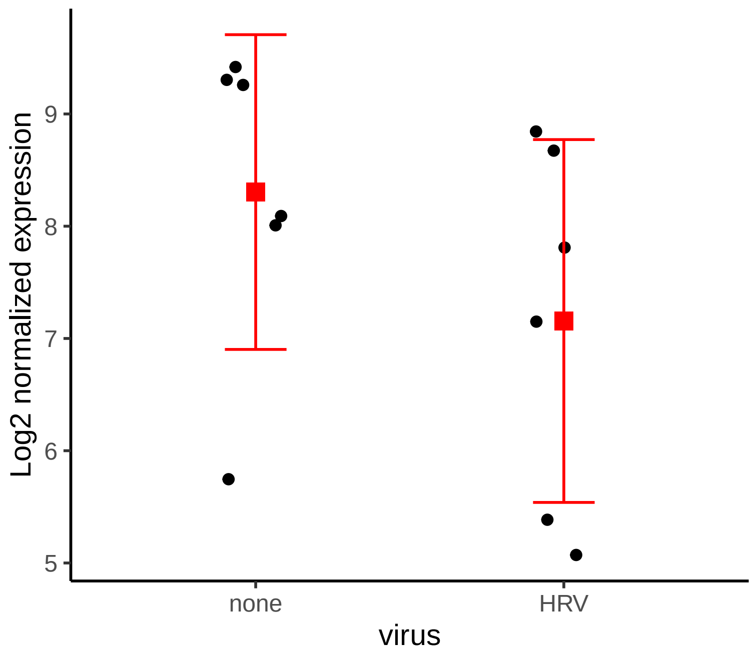
```
## # A tibble: 2 x 4
##   term   statistic    df p.value
##   <chr>      <dbl> <dbl>   <dbl>
## 1 virus   2.84         1  0.0922
## 2 asthma  0.000564     1  0.981
```

The power of paired sample design

What a change!! The p-value for virus is less than half what is was when we didn't include a random effect. This represents a fundamental difference in simple linear vs mixed effects regression.

In simple regression, you are comparing **means** with error. In our model, this is comparing the two red square values (which is equivalent to 1 slope), keeping the overlap of error bars in mind. As is clear in the plot, the means are not very different and error bars overlap a lot between the two virus groups. Thus, our p-value is large.

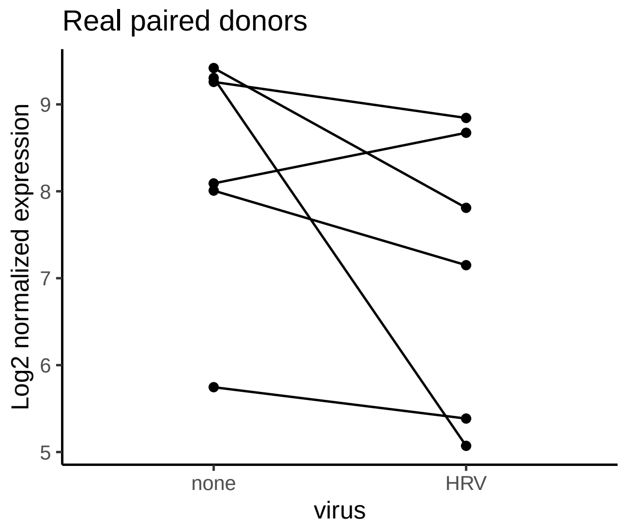
```
ggplot(dat, aes(x=virus, y=value)) +
  #individual data points
  geom_jitter(width=0.1, height=0) +
  #error bars
  stat_summary(fun.data=mean_sdl, fun.args = list(mult=1),
        geom="errorbar", color="red", width=0.2) +
  #mean
  stat_summary(fun=mean, geom="point", color="red", shape="square", size=3) +
  theme_classic() +
  labs(y="Log2 normalized expression")
```

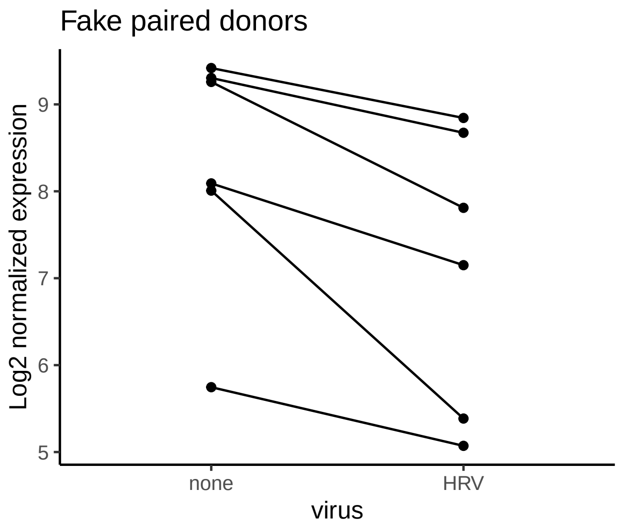


Note that `jitter` randomly distributes the dots along the width. Therefore, you're plot will not look exactly the same along x but the y-values will be consistent.

In contrast, incorporating donor as a random effect in a mixed effects model allows up to pair the appropriate samples and instead compare multiple **slopes**. In a sense, it corrects for the initial variation between donors and looks for a consistent **change** in expression instead of a difference in means. On the left, we see the real data where most (but not all) donors have decreased expression with virus, thus driving a lower p-value but not significant. If, instead, all donors showed a consistent trend, such as the fake example on the right, we would have seen an even smaller, significant p-value.

```
p1 <- ggplot(dat, aes(x=virus, y=value)) +
  #individual data points
  geom_point() +
  #connect points
   geom_line(aes(group=donorID)) +
  theme_classic() +
  labs(y="Log2 normalized expression", title="Real paired donors")
p2 <- dat %>%
  arrange(virus, value) %>%
  mutate(group = c(1:6,1:6)) %>%
  ggplot(aes(x=virus, y=value)) +
  #individual data points
  geom_point() +
  #connect points
   geom_line(aes(group=group)) +
  theme_classic() +
  labs(y="Log2 normalized expression", title="Fake paired donors")
p1+p2
```





Scaling up

We initially reduced this RNA-seq data set to 1 gene. In reality, we want to model expression of all genes, which is 1000 models! There are a number of R packages that help us achieve this in less time and with a lot less code that repeating what we've done previously.

limma linear regression

The limma package is wicked fast and great for simple linear regression. Syntax is a little different than what we've used previously. For the sake of time, an analysis of these data is summarized below.

```r
# Make model matrix. This is limma's version of a formula
mm <- model.matrix(~ virus + asthma, data = example.voom$targets)
# Fit linear regression. Similar to lm( )
fit <- lmFit(object = example.voom$E, design = mm)
# Estimate p-values. Similar to tidy( ) or Anova( )
efit <- eBayes(fit)
# Get results in a data frame
fdr <- extract_lmFit(design = mm, fit = efit)
```

The results contain all genes (1000) and all variables (3) = 3000 rows.

```r
nrow(fdr)
```

```
## [1] 3000
```

limma pseudo mixed effects regression

limma provides a shortcut to account for random effects by treating paired samples as pseudo replicates. It uses a single random effect estimate for all genes from a given donor based on correlation of expression between these "replicates". This is not a full mixed effects model but it can account for some of the random effect.

```r
# Make model matrix
mm <- model.matrix(~ virus + asthma, data = example.voom$targets)
# Estimate correlation of gene expression in paired (block) samples
dupCor <- duplicateCorrelation(object = example.voom$E,
                               design = mm,
                               block = example.voom$targets$donorID)
```

```
# Fit linear regression
fit2 <- lmFit(object = example.voom$E, design = mm,
                          block=example.voom$targets$donorID,
                          correlation=dupCor$consensus.correlation)
# Estimate p-values
efit2 <- eBayes(fit2)
# Get results in a data frame
fdr2 <- extract_lmFit(design = mm, fit = efit2)
```

dream mixed effects regression

dream is an extension to fit true mixed effects models of RNA-seq data. Instead of 1 random effect estimate per donor, dream allows different estimates for each gene within a donor. This is important as the impacts of repeated measures or paired sample design are not consistent across all genes.

Note that the weights used in this model (in `example.voom$weights`) are `voom` weights from limma, not the correct weights for dream. Thus, this example is to show you how to run dream but the results are not valid with these incorrect weights. See the dream tutorial in Resources for more info.

```
fit3 <- dream(exprObj = example.voom,
              formula = ~ virus + asthma + (1|donorID),
              data = example.voom$targets)
```

```
## Dividing work into 50 chunks...

##
## Total:24 s
```

```
efit3 <- eBayes(fit3)
fdr3 <- extract_lmFit(design = fit3$design, fit=efit3)
```

kimma mixed effects regression

kimma is an under-development package to run true linear mixed effects models on RNA-seq data. Under the hood, it uses `lmer( )` to fit models as we did earlier for the 1 gene. Unlike limma or dream, it also allows you to have a matrix of pairwise comparisons as a random effect such as kinship measures (not shown in this workshop).

```
fit4 <- kmFit(dat = example.voom, patientID = "donorID",
              model = "~ virus + asthma + (1|donorID)",
              run.lme = TRUE)
```

```
## lme/lmekin model: expression~virus+asthma+(1|donorID)

## Input: 12 libraries from 6 unique patients

## Model: 12 libraries

## Complete: 1000 genes

## Failed: 0 genes
```

```
fdr4 <- fit4$lme
```

Compare methods

```
#List 5 genes most signif for virus
top.genes <- fdr %>%
  filter(variable == "virusHRV") %>%
  top_n(5, -adj.P.Val) %>%
  distinct(geneName) %>% unlist(use.names = FALSE)

#Combine and format limma results
fdr.limma <- fdr %>%
  mutate(group = "limma LM") %>%
  bind_rows(mutate(fdr2, group="limma LME")) %>%
  #bind_rows(mutate(fdr3, group="dream LME")) %>%
  filter(geneName %in% top.genes & variable == "virusHRV") %>%
  select(group, geneName, adj.P.Val) %>%
  rename(gene=geneName, FDR=adj.P.Val)

#Add kimma results
fdr4 %>%
  mutate(group = "kimma LME") %>%
  filter(gene %in% top.genes & variable == "virus") %>%
  select(group, gene, FDR) %>%
  bind_rows(fdr.limma) %>%
  mutate(group = factor(group, levels=c("limma LM", "limma LME", #"dream LME",
                                        "kimma LME")),
         FDR = formatC(FDR, digits=2)) %>%
  arrange(group) %>%
  pivot_wider(names_from = group, values_from = FDR)
```

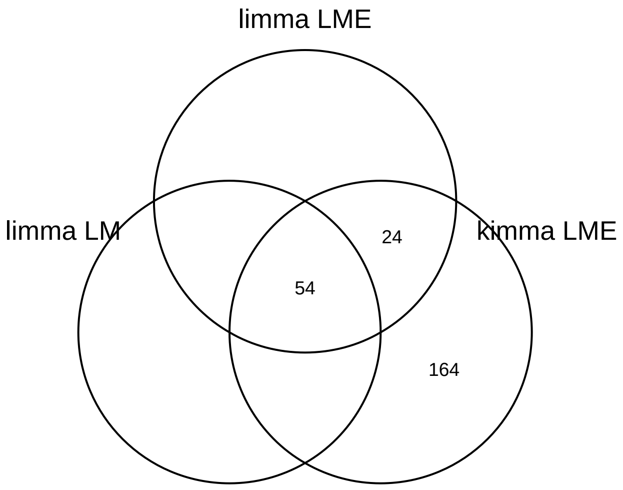
```
## # A tibble: 6 x 4
##   gene            `limma LM` `limma LME` `kimma LME`
##   <chr>           <chr>      <chr>       <chr>
## 1 ENSG00000175183 0.0013     0.00052     1.7e-22
## 2 ENSG00000164761 0.0013     0.00075     9.9e-15
## 3 ENSG00000128383 0.0037     0.0027      1.5e-10
## 4 ENSG00000135070 0.0037     0.0024      1.4e-12
## 5 ENSG00000119917 0.0045     0.0028      1e-09
## 6 ENSG00000169248 0.0045     0.0028      6e-09
```

Looking at 5 smallest FDR values from limma simple linear regression, we can see that the mixed effect methods result in changes in significance estimates. While all of these genes would be classified as differentially expression genes (DEG) at FDR < 0.01, differences exist in many genes as seen by the number of overlapping DEG at FDR < 0.05 in the Venn below. Thus, as always, it's important to use a model appropriate to your data.

Remember that the dream results were not valid because of the example data weights. Thus, it are not included here.

```
library(venn)
venn.ls <- list()
venn.ls [["limma LM"]] <- fdr %>%
  filter(variable == "virusHRV" & adj.P.Val < 0.05) %>%
  distinct(geneName) %>% unlist(use.names = FALSE)
venn.ls [["limma LME"]] <- fdr2 %>%
  filter(variable == "virusHRV" & adj.P.Val < 0.05) %>%
  distinct(geneName) %>% unlist(use.names = FALSE)
venn.ls [["kimma LME"]] <- fdr4 %>%
  filter(variable == "virus" & FDR < 0.05) %>%
  distinct(gene) %>% unlist(use.names = FALSE)
```

```r
venn(venn.ls, box = FALSE)
```



Resources

- The tidyverse
  - https://github.com/hawn-lab/workshops_UW_Seattle/tree/master/2021.06.21_IntroR
  - https://educe-ubc.github.io/workshops/introduction-to-the-tidyverse-1.html
- Linear modeling in R
  - https://educe-ubc.github.io/workshops/statistical-models-in-r-1.html
- RNA-seq data analysis
  - https://github.com/BIGslu/tutorials/tree/main/RNAseq
- limma
  - Chapter 15 https://www.bioconductor.org/packages/devel/bioc/vignettes/limma/inst/doc/usersguide.pdf
  - https://ucdavis-bioinformatics-training.github.io/2018-June-RNA-Seq-Workshop/thursday/DE.html
- dream
  - https://academic.oup.com/bioinformatics/article/37/2/192/5878955
  - https://bioconductor.org/packages/release/bioc/vignettes/variancePartition/inst/doc/dream.html
- kimma
  - https://github.com/BIGslu/tutorials/blob/main/RNAseq/3.Hawn_RNAseq_voom.to.DEG.pdf

R session

```r
sessionInfo()
```

```
## R version 4.1.2 (2021-11-01)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur 10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRlapack.dylib
##
## locale:
```

```
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] venn_1.10              kimma_1.1.1            variancePartition_1.24.0
##  [4] BiocParallel_1.28.3    limma_3.50.1           car_3.0-12
##  [7] carData_3.0-5          lme4_1.1-28            Matrix_1.4-0
## [10] broom_0.7.12           forcats_0.5.1          stringr_1.4.0
## [13] dplyr_1.0.8            purrr_0.3.4            readr_2.1.2
## [16] tidyr_1.2.0            tibble_3.1.6           ggplot2_3.3.5
## [19] tidyverse_1.3.1        patchwork_1.1.1
##
## loaded via a namespace (and not attached):
##   [1] minqa_1.2.4         colorspace_2.0-3    ellipsis_0.3.2
##   [4] htmlTable_2.4.0     base64enc_0.1-3     fs_1.5.2
##   [7] rstudioapi_0.13     farver_2.1.0        fansi_1.0.2
##  [10] lubridate_1.8.0     xml2_1.3.3          codetools_0.2-18
##  [13] splines_4.1.2       doParallel_1.0.17   knitr_1.37
##  [16] Formula_1.2-4       jsonlite_1.8.0      nloptr_2.0.0
##  [19] pbkrtest_0.5.1      cluster_2.1.2       dbplyr_2.1.1
##  [22] png_0.1-7           compiler_4.1.2      httr_1.4.2
##  [25] backports_1.4.1     assertthat_0.2.1    fastmap_1.1.0
##  [28] cli_3.2.0           admisc_0.24         htmltools_0.5.2
##  [31] prettyunits_1.1.1   tools_4.1.2         lmerTest_3.1-3
##  [34] gtable_0.3.0        glue_1.6.1          reshape2_1.4.4
##  [37] Rcpp_1.0.8          Biobase_2.54.0      cellranger_1.1.0
##  [40] vctrs_0.3.8         nlme_3.1-155        iterators_1.0.14
##  [43] xfun_0.29           rvest_1.0.2         lifecycle_1.0.1
##  [46] gtools_3.9.2        statmod_1.4.36      MASS_7.3-55
##  [49] scales_1.1.1        hms_1.1.1           parallel_4.1.2
##  [52] RColorBrewer_1.1-2  yaml_2.3.5          gridExtra_2.3
##  [55] rpart_4.1.16        latticeExtra_0.6-29 stringi_1.7.6
##  [58] highr_0.9           foreach_1.5.2       checkmate_2.0.0
##  [61] caTools_1.18.2      BiocGenerics_0.40.0 boot_1.3-28
##  [64] rlang_1.0.1         pkgconfig_2.0.3     bitops_1.0-7
##  [67] evaluate_0.15       lattice_0.20-45     labeling_0.4.2
##  [70] htmlwidgets_1.5.4   tidyselect_1.1.2    plyr_1.8.6
##  [73] magrittr_2.0.2      R6_2.5.1            gplots_3.1.1
##  [76] generics_0.1.2      Hmisc_4.6-0         DBI_1.1.2
##  [79] pillar_1.7.0        haven_2.4.3         foreign_0.8-82
##  [82] withr_2.4.3         survival_3.2-13     abind_1.4-5
##  [85] nnet_7.3-17         modelr_0.1.8        crayon_1.5.0
##  [88] KernSmooth_2.23-20  utf8_1.2.2          tzdb_0.2.0
##  [91] rmarkdown_2.11      jpeg_0.1-9          progress_1.2.2
##  [94] grid_4.1.2          readxl_1.3.1        data.table_1.14.2
##  [97] reprex_2.0.1        digest_0.6.29       numDeriv_2016.8-1.1
## [100] munsell_0.5.0
```