Block example

Philip Dixon

9/13/2022

This document explains the R code in blockex.r. This analyzes a small study using an RCBD with 3 treatments and 10 blocks. This is the plant study used in the lecture notes. The three treatments are labeled T1, T2 (the two active treatments) and C (control). The blocks are numbered 1, 2, ... 10.

First, we read the data and create factor versions of the treatment variable and the blocking variable. Nothing new here.

Starting with R version 4.X, read.csv() leaves character variables as character variables, so specifying as.is=T is the default and adding it is no longer necessary.

The path (../data/) to the data file depends on where you saved the data file and what your working directory (setwd) is. You will almost certainly need to change this for your setup.

```
mead <- read.csv('../data/mead53b.csv')
mead$trt.f <- factor(mead$trt)
mead$block.f <- factor(mead$block)
```

The factor() function converts a variable to a factor. That is not strictly necessary for variables with character values, e.g., trt, because they can only be interpreted as factors.

My practice is to create a factor version of each variable so I know they are a factor. Others prefer to overwrite the original variable.

Converting block to a factor is critical, because blocks are identified by their number (1 to 10). If you do not convert it to a factor, block will be treated as a regression variable. Not what you want. You want each block to have its own mean. That requires block be a factor. See below for more information and an example.

Blocks as fixed effects The standard model for data from an RCBD is blocks + treatments. It is common to put blocks first, because R anova() calculates sequential sums of squares. We want to compare treatments after adjusting for differences among blocks, so blocks goes first in the model.

To fit the model and get an ANOVA table, use lm() to fit the model including both block and treatment effects. Multiple terms on the right-hand side of the model formula are separated by + signs. The anova() function creates the sequential ANOVA table. We'll talk about sequential tests when we talk about 2 way factorial designs.

```
mead.lm <- lm(height ~ block.f + trt.f, data=mead)
anova(mead.lm)
```

```
## Analysis of Variance Table
##
## Response: height
##           Df Sum Sq Mean Sq F value   Pr(>F)
## block.f    9 74.833  8.3148  2.2654 0.066801 .
## trt.f      2 51.267 25.6333  6.9839 0.005689 **
## Residuals 18 66.067  3.6704
## ---
```

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

After fitting the model, you would use emmeans functions to explore the means. That's material from week 2 and reviewed here.

```
mead.emm <- emmeans(mead.lm, 'trt.f')
mead.emm
```

```
##  trt.f emmean    SE df lower.CL upper.CL
##  C        7.2 0.606 18     5.93     8.47
##  T1       8.9 0.606 18     7.63    10.17
##  T2      10.4 0.606 18     9.13    11.67
##
## Results are averaged over the levels of: block.f
## Confidence level used: 0.95
```

The emmeans() function estimates the marginal means for the specified variable (in quotes), here trt.f. Marginal means are averaged over all other effects in the model. The result is the mean for each treatment, averaged over blocks.

Failing to declare a numeric variable as a factor This code demonstrates the consequences of not converting a numeric block value to a factor. (Remember block is the original value; block.f was the factor version).

```
mead.lm3 <- lm(height ~ block + trt.f, data=mead)
anova(mead.lm3)
```

```
## Analysis of Variance Table
##
## Response: height
##           Df  Sum Sq Mean Sq F value  Pr(>F)
## block      1   4.268  4.2677  0.8121 0.37577
## trt.f      2  51.267 25.6333  4.8778 0.01589 *
## Residuals 26 136.632  5.2551
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

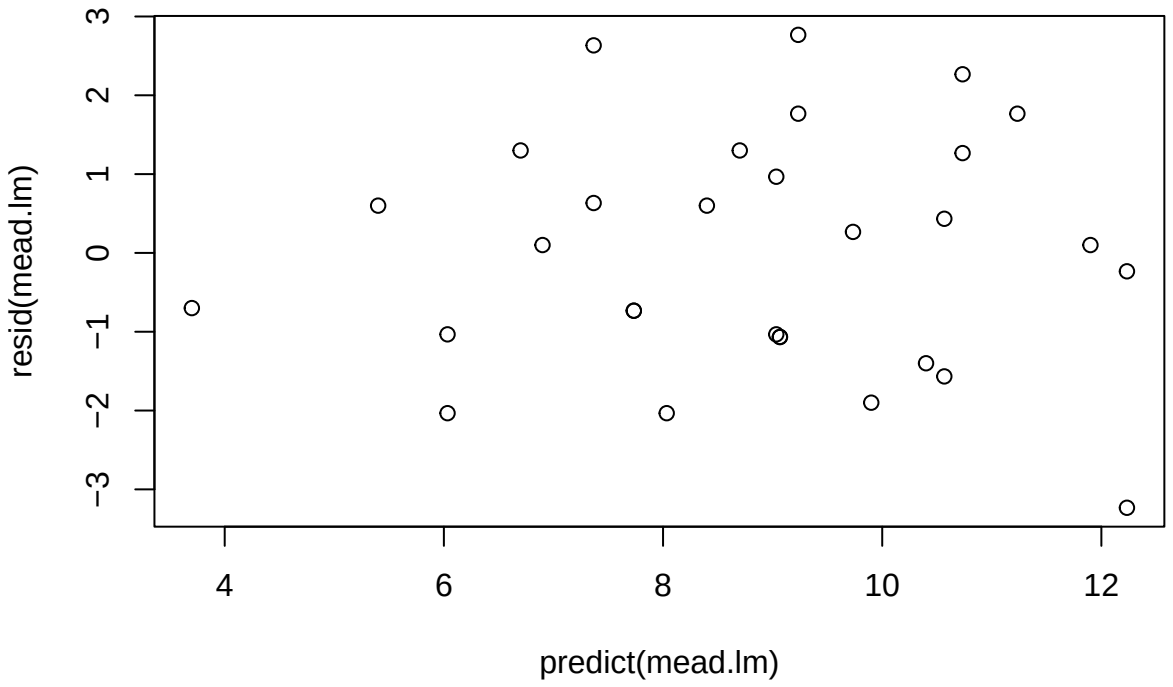
No warnings, no errors, but the results are wrong because you fit the wrong model.

The only sign that you fit the wrong model is the df for block. When you fit a regression, it is 1 df; when you fit a means model, it is #blocks - 1.

Advice: check the degrees of freedom for model terms. It can sometimes be the only way to check that you fit the model you wanted to.

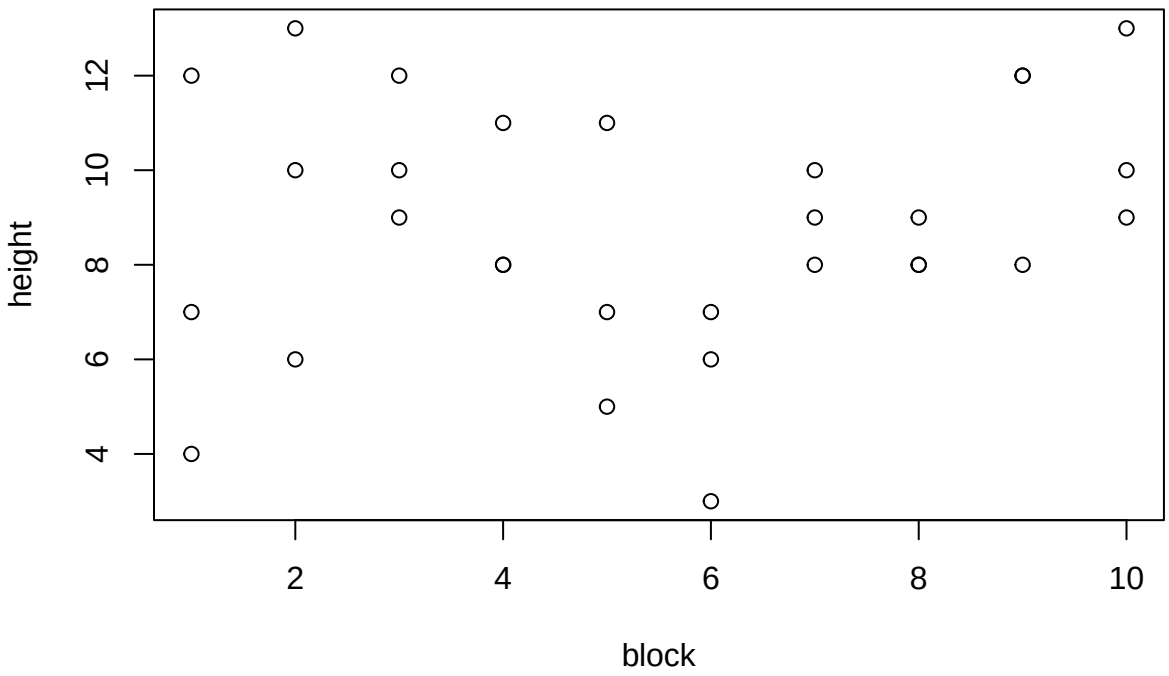
Checking assumptions It's almost always a good idea to check assumptions. My quick check is a plot of X=predicted values vs Y = residuals. You want a "flat fat sausage". A trumpet shape or an unusually large residual are signs of trouble (unequal variances or outlier). This plot looks fine to me.

```
plot(predict(mead.lm), resid(mead.lm))
```



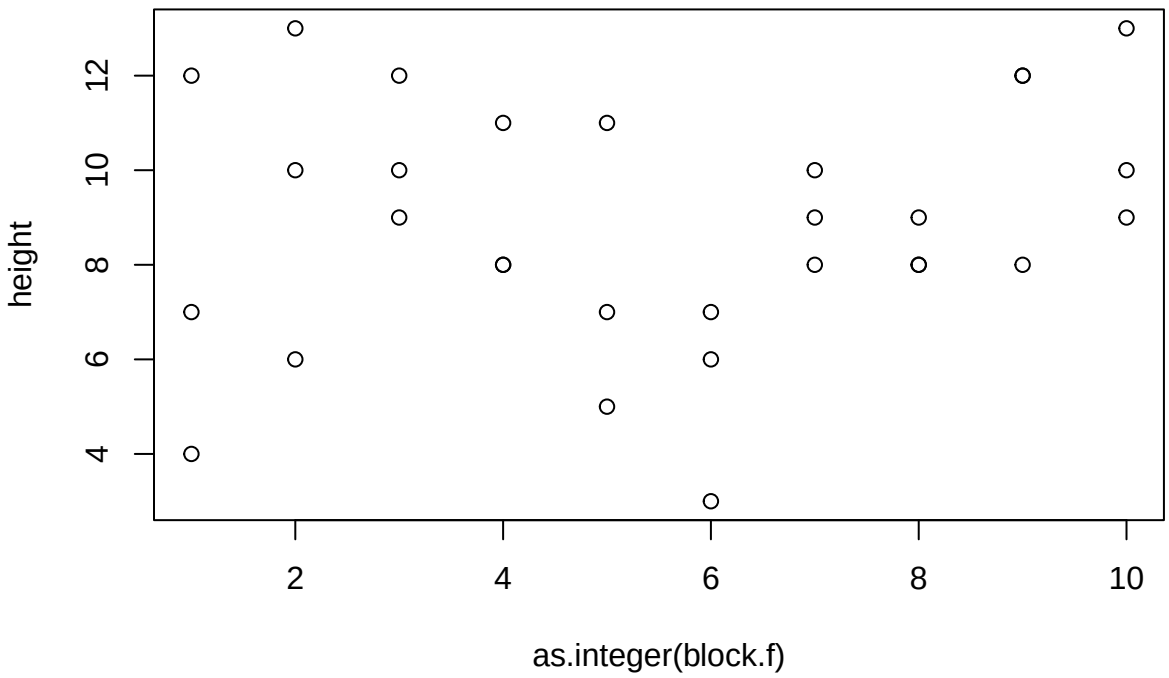
It can be useful to plot responses in each block. When blocks are numbered, this is easy, especially when numbered 1, 2, 3, ... If block is character, convert it to a factor (if you don't already have a factor version). Factors are actually stored as numbers (1, 2, ...) with an associated vector of labels. You can extract the numbers using as.integer(factor).

```
with(mead, plot(block, height) )
```



plots-1.pdf

```
with(mead, plot(as.integer(block.f), height) )
```



plots-2.pdf
These plots are often more useful if you plot responses vs block averages, instead of block number. That's demonstrated in the optional code at the end of the file.

Because the data are balanced (equal numbers of observations for each combination of block and treatment), the order of terms doesn't matter. This demonstrates that:

```
mead.lm2 <- lm(height ~ trt.f + block.f, data=mead)
anova(mead.lm2)
```

```
## Analysis of Variance Table
##
## Response: height
##           Df Sum Sq Mean Sq F value   Pr(>F)
## trt.f      2 51.267 25.6333  6.9839 0.005689 **
## block.f    9 74.833  8.3148  2.2654 0.066801 .
## Residuals 18 66.067  3.6704
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

blocks as a random effect When you define random effects, R only cares about the unique values, not whether it is factor, character, or numeric. So you can define random blocks using either the block variable (numeric) or block.f variable (factor). The random effect is defined in exactly the same way as we defined random effects in week 3. We need to fit the model using lmer() in the lme4 library (or you use one of the other libraries that fit linear mixed effects models).

The order of terms doesn't matter. And, as before, you use functions in the emmeans library to explore the means after fitting the model.

```
library(lme4)
mead.lme <- lmer(height ~ trt.f + (1|block), data=mead)
summary(mead.lme)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: height ~ trt.f + (1 | block)
##    Data: mead
##
## REML criterion at convergence: 126
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -1.33015 -0.72858  0.04051  0.63084  1.60555
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  block    (Intercept) 1.548    1.244
##  Residual             3.670    1.916
## Number of obs: 30, groups:  block, 10
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)   7.2000     0.7224   9.967
## trt.fT1       1.7000     0.8568   1.984
## trt.fT2       3.2000     0.8568   3.735
##
## Correlation of Fixed Effects:
##         (Intr) trt.T1
## trt.fT1 -0.593
## trt.fT2 -0.593  0.500
```

```
anova(mead.lme)
```

```
## Analysis of Variance Table
##       npar Sum Sq Mean Sq F value
## trt.f    2 51.267  25.633  6.9839
```

A variance component of 0 is bad news, just like last week. Check the output from summary() to make sure that didn't happen.

The residual vs predicted value plot from the random blocks analysis doesn't look exactly the same as that from the fixed block analysis. That's because block effects are calculated differently in the two models. Usually, it doesn't make any difference to the interpretation of the plots. Again, this plot looks fine to me.

Optional code to plot responses against block averages. To do this, you need to compute block averages, merge them into the data file, then do the plot. I find it easiest to use dplyr functions for these operations. group_by() defines groups in the data set. summarize() computes summary statistics. When the data are grouped first, you get summary statistics for each group.

left_join() merges the second data set with the first and keeps one row for each row in the first data set. There are many other possible ways to merge two data sets and dplyr provides a lot of merge functions. When one data set has multiple observations with the same block number, left_join() does a many-to-one merge. The value in the second data set is duplicated for each matching observation in the first.

```
library(dplyr)   # if not already done
```

```
##
```

```
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

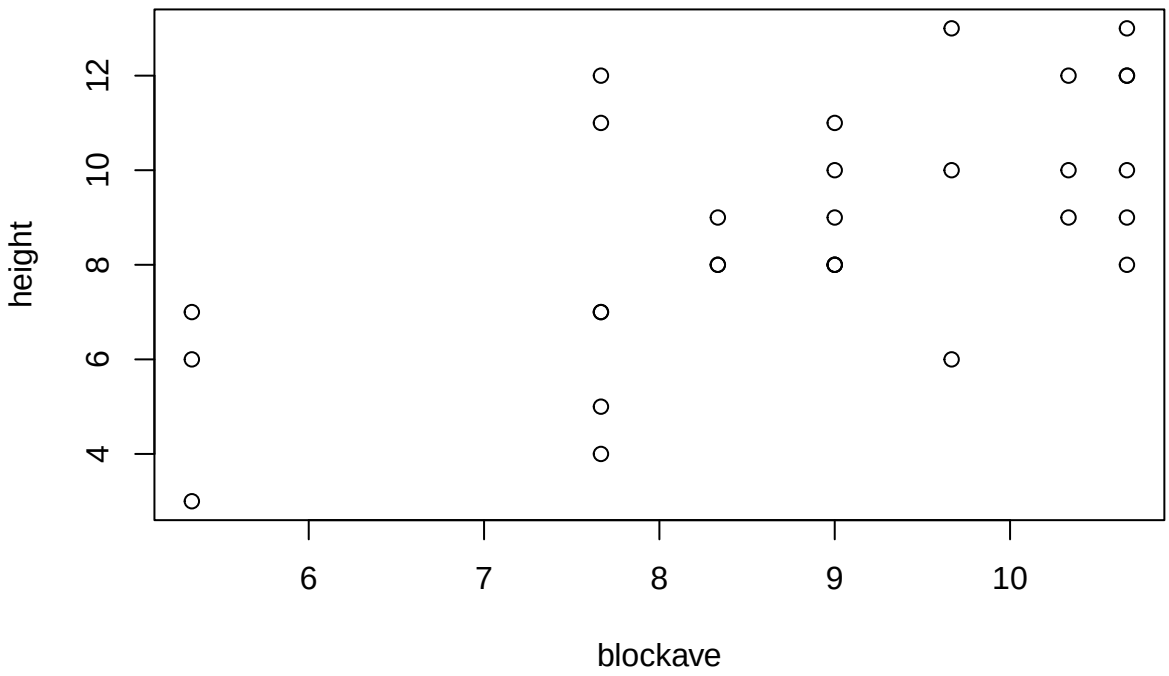
```
meadave <- mead %>% group_by(block) %>%
  summarize(blockave = mean(height))
meadave
```

```
## # A tibble: 10 x 2
##    block blockave
##    <int>    <dbl>
##  1     1     7.67
##  2     2     9.67
##  3     3    10.3
##  4     4     9
##  5     5     7.67
##  6     6     5.33
##  7     7     9
##  8     8     8.33
##  9     9    10.7
## 10    10    10.7
```

```
mead <- left_join(mead, meadave, by='block')
head(mead)
```

```
##   block trt height trt.f block.f blockave
## 1     1   C      4     C       1 7.666667
## 2     1  T1      7    T1       1 7.666667
## 3     1  T2     12    T2       1 7.666667
## 4     2   C      6     C       2 9.666667
## 5     2  T1     10    T1       2 9.666667
## 6     2  T2     13    T2       2 9.666667
```

```
with(mead, plot(blockave, height))
```


height
4
6
8
10
12
6
7
8
9
10
blockave