4-by-4 factorial design

- predictors as factors
- analysis of variance
- testing main effects
- group means and treatment differences
    - using emmeans
- other functions and packages for comparisons
    - aov
    - general contrasts using package `multcomp`
    - multiple comparisons using package `DescTools`
- using specific contrasts as coefficients

In an experiment with soybean, micronutrients were added to a fertilizer: copper (Cu) and/or manganeze (Mn) (values in % of fertilizer). Yield was then measured (kg/acre).

```
soy = read.table("soybean.txt", header=T)
```

predictors as factors

Sometimes we will need to use `cu` as a numerical variable (to plot the data for instance), and sometimes as a factor with 4 categories (for the analysis), and similarly with `mn` . Below we define 2 new columns with the same data as in `cu` and `mn` , but considered as factors.

```
soy$cu_factor = factor(soy$cu)
soy$mn_factor = factor(soy$mn)
str(soy)
```

```
'data.frame':   32 obs. of  5 variables:
 $ cu       : int  1 1 1 1 3 3 3 3 5 5 ...
 $ mn       : int  20 50 80 110 20 50 80 110 20 50 ...
 $ yield    : int  1558 2003 2490 2830 1590 2020 2620 2860 1550 2010 ...
 $ cu_factor: Factor w/ 4 levels "1","3","5","7": 1 1 1 1 2 2 2 2 3 3 ...
 $ mn_factor: Factor w/ 4 levels "20","50","80",..: 1 2 3 4 1 2 3 4 1 2 ...
```

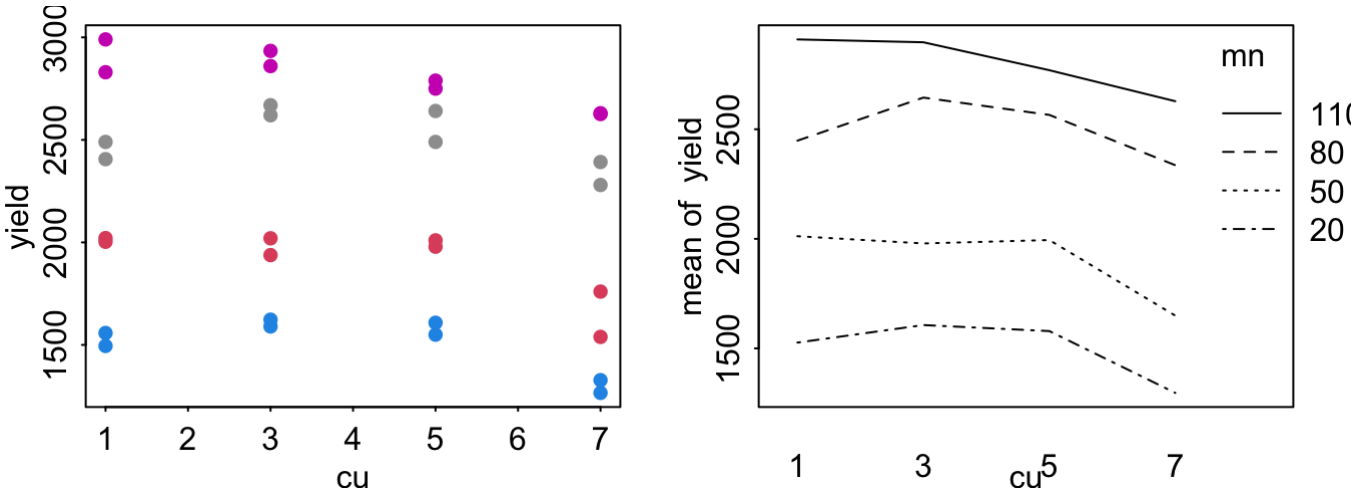
The experiments used 4 levels of cu addition, and 4 levels of mn addition, with 2 plots for each combination:

```
table(soy$cu, soy$mn)
```

```
   20 50 80 110
1   2  2  2   2
3   2  2  2   2
5   2  2  2   2
7   2  2  2   2
```

Let's look at the data. We can also visualize and calculate the means at each combination:

```
layout(matrix(1:2,1,2))
plot(yield ~ cu, col=mn, pch=16, data=soy)
with(soy, interaction.plot(cu, mn, yield))
```



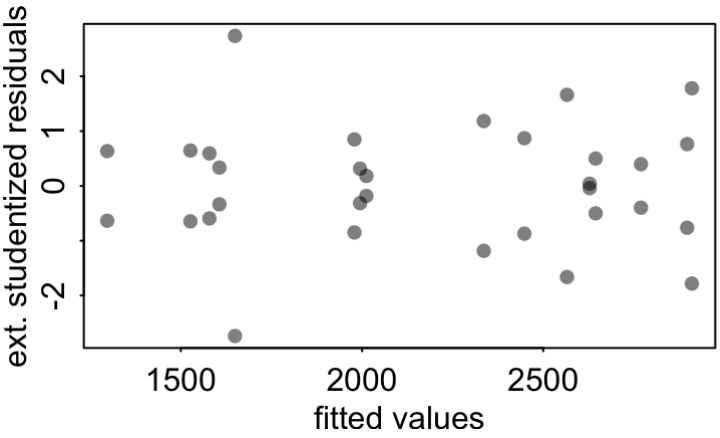

```
library(dplyr)
soy %>% group_by(cu, mn) %>% summarize(mean = mean(yield), sd = sd(yield))
```

```
# A tibble: 16 × 4
# Groups:   cu [4]
      cu    mn  mean     sd
   <int> <int> <dbl>  <dbl>
 1     1    20 1526.  44.5
 2     1    50 2012   12.7
 3     1    80 2448   59.4
 4     1   110 2910  113.
 5     3    20 1606.  23.3
 6     3    50 1979   58.0
 7     3    80 2644.  34.6
 8     3   110 2897   52.3
 9     5    20 1579   41.0
10     5    50 1994.  21.9
11     5    80 2566. 107.
12     5   110 2770.  27.6
13     7    20 1297   43.8
14     7    50 1650. 156.
15     7    80 2336   79.2
16     7   110 2628    2.83
```

analysis of variance

Now to the formal analysis. It is extremely important to consider our two predictors as factors, to let each group have its own mean. Otherwise, and the mean for group cu=3 would be constrained to be half-way between the mean for group cu=1 and for group cu=5.

```
fit = lm(yield ~ cu_factor * mn_factor, data=soy)
plot(rstudent(fit)~fitted(fit), col=rgb(0,0,0, 0.5), pch=16,
     xlab="fitted values", ylab="ext. studentized residuals")
```




```
anova(fit)
```

```
Analysis of Variance Table

Response: yield
                   Df  Sum Sq Mean Sq  F value    Pr(>F)
cu_factor           3  443636  147879  32.2943 5.084e-07
mn_factor           3 8161105 2720368 594.0845 < 2.2e-16
cu_factor:mn_factor 9   69190    7688   1.6789    0.1753
Residuals          16   73265    4579
```

In R, be really careful with this `anova` function: it returns the "type I" sums of squares, that is, when factors/predictors are added **sequentially**. Here, the design is balanced (n=2 in each treatment combination), so the type I and type III sums of squares are equal: the variance explained by each factor is the same whether it is added first or added last to the model. But in general, beware of the `anova` function: do not use it if you want to test a factor given that all others are in the model.

A safe way to get p-values based on type III sums of squares is using the `drop1` function, which drops only 1 term at a time:

```
drop1(fit, test="F")
```

```
Single term deletions

Model:
yield ~ cu_factor * mn_factor
                   Df Sum of Sq    RSS    AIC F value Pr(>F)
<none>                          73265 279.56
cu_factor:mn_factor  9     69190 142456 282.83  1.6789 0.1753
```

Unfortunately (or fortunately), `drop1` respects the hierarchy principle, and it won't drop the term for "cu" if an interaction involving "cu" is present in the model. So here we only get the test for the interaction term.

testing main effects

Why can't we test for main effects easily in R? We didn't get p-values for main effects with `drop1`, and we don't get them with `anova` unless the design is balanced (type I, or sequential SS). Here is why:

- if there is an interaction:
  - all terms in this interaction should be considered as having an effect: no need for further testing.
  - the "main" effects are not meaningful: the main effect of a factor does not represent any of its simple effects
- if there is no interaction, then we can drop the interaction term from the model, and testing main effects becomes easy.

In our example the interaction is not significant, so we could consider dropping it to test each factor individually:

```
fit.noint = lm(yield ~ cu_factor + mn_factor, data=soy)
drop1(fit.noint, test="F")
```

```
Single term deletions

Model:
yield ~ cu_factor + mn_factor
          Df Sum of Sq     RSS    AIC F value    Pr(>F)
<none>                  142456 282.83
cu_factor  3    443636  586092 322.10  25.952  7.58e-08
mn_factor  3   8161105 8303561 406.93 477.406 < 2.2e-16
```

However we changed the model: the interaction was removed, so the tests for the main effects have changed (notice the slightly different p-values): for instance, we tested "all α_i=0 | μ, β_j" instead of "all α_i=0 | μ, β_j, $(\alpha\beta)_{ij}$". Many researchers would argue **against** removing the interaction from the model, even if it is not significant, because the experiment was designed as a 2-factorial CRD.

group means and treatment differences

The estimated coefficients tell us about the estimated group means, but getting all the group means is not straightforward.

```
summary(fit)$coefficients
```

```
                      Estimate Std. Error    t value     Pr(>|t|)
(Intercept)             1526.5   47.84921 31.9023032 6.512077e-16
cu_factor3                80.0   67.66900  1.1822252 2.543885e-01
cu_factor5                52.5   67.66900  0.7758353 4.491629e-01
cu_factor7              -229.5   67.66900 -3.3915086 3.726336e-03
mn_factor50              485.5   67.66900  7.1746293 2.210244e-06
mn_factor80              921.5   67.66900 13.6177567 3.226611e-10
mn_factor110            1383.5   67.66900 20.4451074 6.813319e-13
cu_factor3:mn_factor50  -113.0   95.69842 -1.1807928 2.549412e-01
cu_factor5:mn_factor50   -70.0   95.69842 -0.7314645 4.750730e-01
cu_factor7:mn_factor50  -133.0   95.69842 -1.3897826 1.836308e-01
cu_factor3:mn_factor80   116.5   95.69842  1.2173660 2.411158e-01
cu_factor5:mn_factor80    65.0   95.69842  0.6792171 5.067079e-01
cu_factor7:mn_factor80   117.5   95.69842  1.2278155 2.372737e-01
cu_factor3:mn_factor110  -93.0   95.69842 -0.9718029 3.456119e-01
cu_factor5:mn_factor110 -193.0   95.69842 -2.0167522 6.082519e-02
cu_factor7:mn_factor110  -52.5   95.69842 -0.5485984 5.908547e-01
```

- For example, the levels **cu=1** and **mn=20** do not appear in the coefficient names. They are the baseline levels, with values of 0 for all dummy variables that represent cu and mn. Do `model.matrix(fit)` to see these dummy variables. So the mean yield in the baseline group, cu=1 and mn=20 is the intercept: 1526.50. Its standard error is given as 47.85, which would allow us to get a confidence interval for instance (using a multiplier from the t-distribution with dfError = 16 df).
- For the group mean in group **cu=3 and mn=50**, it's more complicated: we need to add the intercept, the coefficient for cu=3, and coefficient for mn=50, and the interaction coefficient for "cu=3 and mn=50": `1526.50+80+485.5+(-113)` = 1979. But the coefficient table does not give the SE for this estimate.

using emmeans

The package emmeans (https://cran.r-project.org/web/packages/emmeans/) is the best to do inference in treament means, their pairwise differences, and other contrasts. It has great documentation, is very flexible (e.g. to models with random effects), and easier syntax than other packages to specify contrats.

EMM stands for "estimated marginal means" (estimated from a fitted model), sometimes called adjusted means when the model has covariates. Let's look at treatment means first:

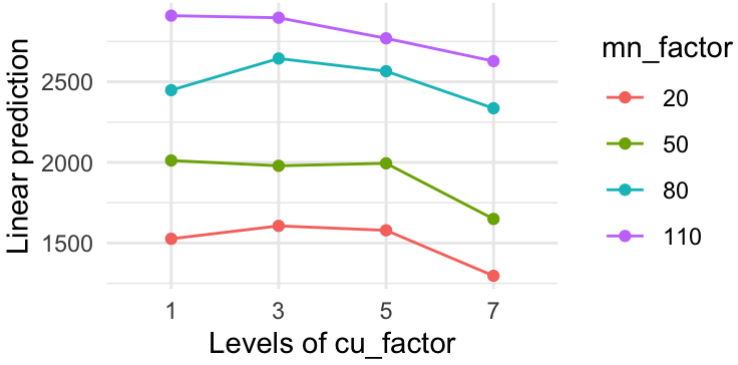
```
library(emmeans)
emmeans(fit, ~ mn_factor)
```

```
 mn_factor emmean   SE df lower.CL upper.CL
 20          1502 23.9 16     1452     1553
 50          1909 23.9 16     1858     1959
 80          2498 23.9 16     2448     2549
 110         2801 23.9 16     2750     2852

Results are averaged over the levels of: cu_factor
Confidence level used: 0.95
```

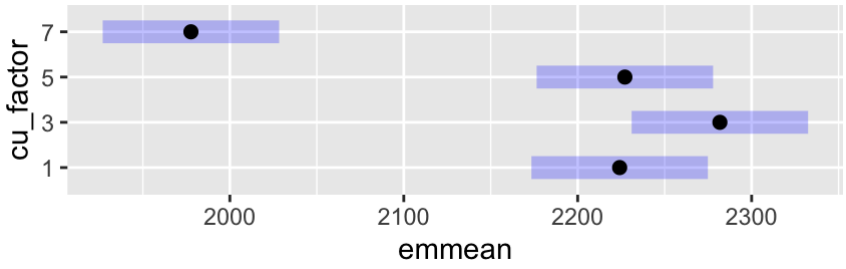
```
# emmeans(fit, ~ mn_factor | cu_factor) # same as below, presented differently
emmeans(fit, ~ mn_factor:cu_factor)
```

```
 mn_factor cu_factor emmean   SE df lower.CL upper.CL
 20        1            1526 47.8 16     1425     1628
 50        1            2012 47.8 16     1911     2113
 80        1            2448 47.8 16     2347     2549
 110       1            2910 47.8 16     2809     3011
 20        3            1606 47.8 16     1505     1708
 50        3            1979 47.8 16     1878     2080
 80        3            2644 47.8 16     2543     2746
 110       3            2897 47.8 16     2796     2998
 20        5            1579 47.8 16     1478     1680
 50        5            1994 47.8 16     1893     2096
 80        5            2566 47.8 16     2464     2667
 110       5            2770 47.8 16     2668     2871
 20        7            1297 47.8 16     1196     1398
 50        7            1650 47.8 16     1548     1751
 80        7            2336 47.8 16     2235     2437
 110       7            2628 47.8 16     2527     2729

Confidence level used: 0.95
```

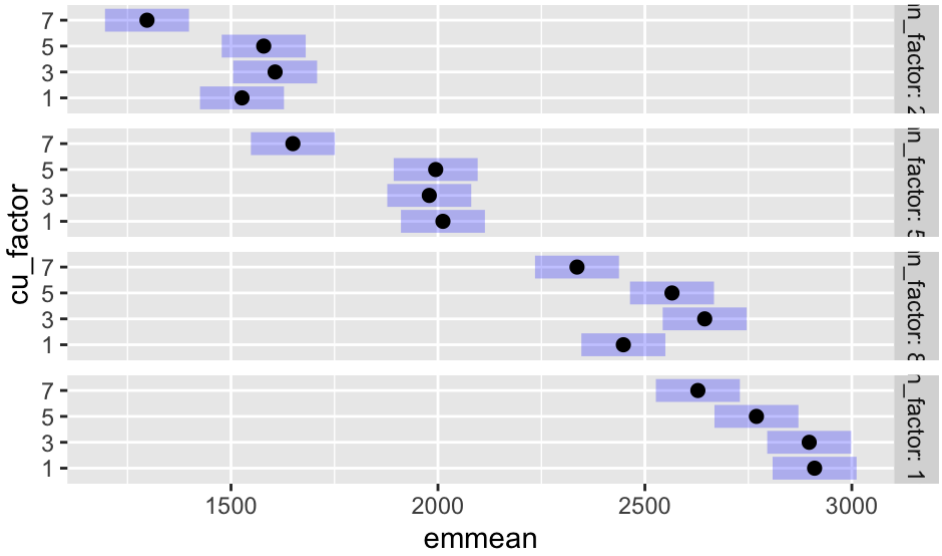
```
library(ggplot2)
emmip(fit, mn_factor ~ cu_factor) + theme_minimal()  # ip = interaction plot
```




```
plot(emmeans(fit, ~ cu_factor))
```



```
plot(emmeans(fit, ~ cu_factor | mn_factor), by = "mn_factor")
```



by the way: why are all intervals of the same length?
Now, let's look at pairwise comparisons:

```
em_cu = emmeans(fit, ~ cu_factor)
pairs(em_cu) # default: Tukey
```

```
 contrast estimate   SE df t.ratio p.value
 1 - 3       -57.6 33.8 16  -1.703  0.3542
 1 - 5        -3.0 33.8 16  -0.089  0.9997
 1 - 7       246.5 33.8 16   7.285  <.0001
 3 - 5        54.6 33.8 16   1.614  0.3987
 3 - 7       304.1 33.8 16   8.989  <.0001
 5 - 7       249.5 33.8 16   7.374  <.0001

Results are averaged over the levels of: mn_factor
P value adjustment: tukey method for comparing a family of 4 estimates
```

```
pairs(em_cu, adjust = "none") # for LSD
```

```
 contrast estimate   SE df t.ratio p.value
 1 - 3       -57.6 33.8 16  -1.703  0.1079
 1 - 5        -3.0 33.8 16  -0.089  0.9304
 1 - 7       246.5 33.8 16   7.285  <.0001
 3 - 5        54.6 33.8 16   1.614  0.1260
 3 - 7       304.1 33.8 16   8.989  <.0001
 5 - 7       249.5 33.8 16   7.374  <.0001

Results are averaged over the levels of: mn_factor
```

```
pwpm(em_cu, adjust = "none") # pw = pairwise pm = p-value matrix
```

```
       1      3      5      7
1 [2224] 0.1079 0.9304 <.0001
3  -57.6 [2282] 0.1260 <.0001
5   -3.0   54.6 [2227] <.0001
7  246.5  304.1  249.5 [1978]

Row and column labels: cu_factor
Upper triangle: P values
Diagonal: [Estimates] (emmean)
Lower triangle: Comparisons (estimate)   earlier vs. later
```

```
em_cuxmn = emmeans(fit, ~ cu_factor:mn_factor)[1:10] # first 10 only: to fit page width
pwpm(em_cuxmn)
```

```
        1 20    3 20    5 20    7 20    1 50    3 50    5 50    7 50    1 80   3 80
1 20  [1526]  0.9651  0.9980  0.0789  <.0001  0.0002  0.0001  0.7170  <.0001 <.0001
3 20   -80.0  [1606]  1.0000  0.0085  0.0006  0.0014  0.0009  0.9996  <.0001 <.0001
5 20   -52.5    27.5  [1579]  0.0185  0.0003  0.0007  0.0004  0.9842  <.0001 <.0001
7 20   229.5   309.5   282.0  [1297]  <.0001  <.0001  <.0001  0.0025  <.0001 <.0001
1 50  -485.5  -405.5  -433.0  -715.0  [2012]  1.0000  1.0000  0.0019  0.0003 <.0001
3 50  -452.5  -372.5  -400.0  -682.0    33.0  [1979]  1.0000  0.0048  0.0001 <.0001
5 50  -468.0  -388.0  -415.5  -697.5    17.5   -15.5  [1994]  0.0031  0.0002 <.0001
7 50  -123.0   -43.0   -70.5  -352.5   362.5   329.5   345.0  [1649]  <.0001 <.0001
1 80  -921.5  -841.5  -869.0 -1151.0  -436.0  -469.0  -453.5  -798.5  [2448] 0.1824
3 80 -1118.0 -1038.0 -1065.5 -1347.5  -632.5  -665.5  -650.0  -995.0  -196.5 [2644]

Row and column labels: cu_factor:mn_factor:.wgt.
Upper triangle: P values   adjust = "tukey"
Diagonal: [Estimates] (emmean)
Lower triangle: Comparisons (estimate)   earlier vs. later
```

Finally, let's consider a contrast other than a pairwise difference. Say, we want to see if the response at `cu=3` (averaged over the mn values) is the average between the mean at `cu=1` and at `cu=5` (again, averaged over the mn values). We find weak evidence against:

```
contrast(em_cu,  list(cu3_vs1and5 = c(-1/2, 1,-1/2,0)))
```

```
 contrast    estimate   SE df t.ratio p.value
 cu3_vs1and5     56.1 29.3 16   1.915  0.0735

Results are averaged over the levels of: mn_factor
```

other functions and packages for comparisons

Here are other tools that you may encounter, good to know about. But `emmeans` is the best so far, I think.

aov

The function `aov` gives us another way to fit the same model (and get the same ANOVA table), and has many tools for post-hoc group comparisons, like Tukey's honest significant differences. But it is limited to simple designs, and balanced designs.

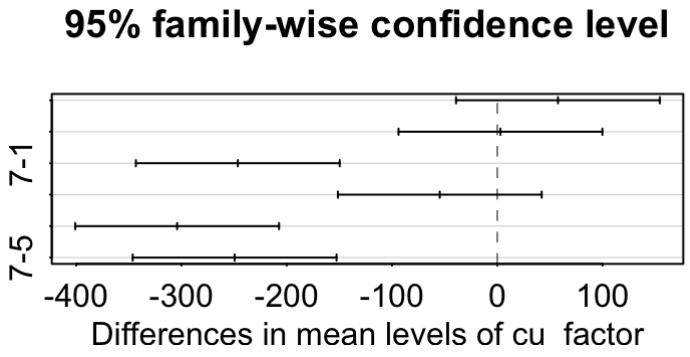
```
fit.aov = aov(yield ~ cu_factor * mn_factor, data=soy)
anova(fit.aov)
```

```
Analysis of Variance Table

Response: yield
                    Df  Sum Sq Mean Sq  F value    Pr(>F)
cu_factor            3  443636  147879  32.2943 5.084e-07
mn_factor            3 8161105 2720368 594.0845 < 2.2e-16
cu_factor:mn_factor  9   69190    7688   1.6789    0.1753
Residuals           16   73265    4579
```

```
TukeyHSD(fit.aov, "cu_factor")
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = yield ~ cu_factor * mn_factor, data = soy)

$cu_factor
        diff        lwr        upr     p adj
3-1   57.625  -39.17618  154.42618 0.3541571
5-1    3.000  -93.80118   99.80118 0.9997390
7-1 -246.500 -343.30118 -149.69882 0.0000100
5-3  -54.625 -151.42618   42.17618 0.3986692
7-3 -304.125 -400.92618 -207.32382 0.0000007
7-5 -249.500 -346.30118 -152.69882 0.0000086
```

```
plot(TukeyHSD(fit.aov, "cu_factor"))
```



Estimated group means are also easier to obtain after fitting the model with `aov` :

```
model.tables(fit.aov, type="means", se=TRUE)
```

```
Tables of means
Grand mean

2177.656

 cu_factor
cu_factor
     1      3      5      7
2224.1 2281.7 2227.1 1977.6

 mn_factor
mn_factor
    20     50     80    110
1502.2 1908.7 2498.5 2801.1

 cu_factor:mn_factor
         mn_factor
cu_factor 20     50     80     110
        1 1526.5 2012.0 2448.0 2910.0
        3 1606.5 1979.0 2644.5 2897.0
        5 1579.0 1994.5 2565.5 2769.5
        7 1297.0 1649.5 2336.0 2628.0

Standard errors for differences of means
        cu_factor mn_factor cu_factor:mn_factor
            33.83     33.83               67.67
replic.         8         8                   2
```

The last table gives SEs to calculate various things:

- the **LSD** (least significant difference) values: to compare the overall means among cu groups at the 5% level, for instance, the LSD is `33.83 * 2.12 = 71.72` . Group means that differ by more than 71.72 would be said statistically different (but recall that LSD is liberal). The 2.12 is the t multiplier for 95% confidence, from 16 df (dfError): `qt(.975, df=16)` = 2.12. So for instance, the means for `cu=1` and `cu=3` are not statistically different according to LSD: 2281.7 - 2224.1=57.6, which is less than 71.72.
- test for **differences between means**: to compare the mean at `cu=3` versus `cu=1` , we would divide the observed difference in means, 57.6, by the standard error of this difference: 33.83 (last table in the output above). Note that this SE can be derived manually using the MSError (4579) and the formula for the variance of a contrast: `sqrt(4579*(1/8+1/8))` = 33.83. So our t-value is 57.6/33.83 = 1.702, to be compared with a t-distribution with 16 df (error df): `pt(1.702, df=16, lower.tail=FALSE)*2` =0.13. Again, the difference is not significant.
- confidence interval for a **single mean**: we would follow a similar strategy. For the overal mean 2224.1 in groups `cu=1` , for example, the standard error of this mean is `sqrt(4579*(1/8))` = 23.92. The t multiplier for 95% confidence is still 2.12 (16 df), so our confidence interval has limits 2224.1 $\pm$ 2.12 * 23.92.

- confidence interval for the group cu=1 *and* mn=20. Here we would use the estimated mean in that group: 1526.5. Its standard error is calculated on the basis of only 2 observation for that group: `sqrt(4579*(1/2))` = 47.85.
- difference between the means in group "cu=1 and mn=20" and group "cu=3 and mn=20": same strategy as before, but the standard error for the difference of interest is `sqrt(4579*(1/2 + 1/2))`, because each group only has 2 observations. In the output above, the last row of the last table is useful to determine the appropriate SE, based on which group(s) we are interested in and the sample size for that (those) groups.

We can also get the "effects" that we need to add to the overall mean (intercept in the model). Using the course notations, these effects are the the α_i, β_j and $(\alpha\beta)_{ij}$ terms. Note that they do sum to zero, in each row and each column:

```
model.tables(fit.aov) # default is type="effects"
```

```
Tables of effects

 cu_factor
cu_factor
      1       3       5       7
  46.47  104.09   49.47 -200.03

 mn_factor
mn_factor
    20     50     80    110
-675.4 -268.9  320.8  623.5

 cu_factor:mn_factor
         mn_factor
cu_factor 20     50     80     110
        1 -22.22  56.78 -96.97  62.41
        3   0.16 -33.84  41.91  -8.22
        5  27.28  36.28  17.53 -81.09
        7  -5.22 -59.22  37.53  26.91
```

general contrasts using package `multcomp`

```
library(multcomp)
```

First, let's look at a very specific contrast: at `mn=20`, is the response to `cu=3` in between those at `cu=1` and `cu=5`? Warning: the coefficient named `cu_factor3` is the difference between `cu=3` and `cu=1` (etc.), so the contrast coefficient is 0 for the intercept, below.

```
# coef(fit) # look at coefficients and what they mean, to define the constrast next
K = matrix(c(0,-2,1, rep(0,13)), 1)
t = glht(fit, linfct = K)
summary(t)
```

```
     Simultaneous Tests for General Linear Hypotheses

Fit: lm(formula = yield ~ cu_factor * mn_factor, data = soy)

Linear Hypotheses:
       Estimate Std. Error t value Pr(>|t|)
1 == 0   -107.5      117.2  -0.917    0.373
(Adjusted p values reported -- single-step method)
```

testing main effect of `cu` using Tukey's HSD or Fisher's LSD:

```
lh_cu = glht(fit, linfct = mcp(cu_factor="Tukey", interaction_average=T))
summary(lh_cu) # p-values: for Tukey's HSD
```

```
     Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts


Fit: lm(formula = yield ~ cu_factor * mn_factor, data = soy)

Linear Hypotheses:
           Estimate Std. Error t value Pr(>|t|)
3 - 1 == 0    57.62      33.84   1.703    0.354
5 - 1 == 0     3.00      33.84   0.089    1.000
7 - 1 == 0  -246.50      33.84  -7.285   <0.001
5 - 3 == 0   -54.62      33.84  -1.614    0.399
7 - 3 == 0  -304.12      33.84  -8.989   <0.001
7 - 5 == 0  -249.50      33.84  -7.374   <0.001
(Adjusted p values reported -- single-step method)
```

```
summary(lh_cu, test=univariate()) # no correction for multiple comparisons: Fisher LSD
```

```
     Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts


Fit: lm(formula = yield ~ cu_factor * mn_factor, data = soy)

Linear Hypotheses:
           Estimate Std. Error t value Pr(>|t|)
3 - 1 == 0    57.62      33.84   1.703    0.108
5 - 1 == 0     3.00      33.84   0.089    0.930
7 - 1 == 0  -246.50      33.84  -7.285 1.83e-06
5 - 3 == 0   -54.62      33.84  -1.614    0.126
7 - 3 == 0  -304.12      33.84  -8.989 1.19e-07
7 - 5 == 0  -249.50      33.84  -7.374 1.57e-06
(Univariate p values reported)
```

applying Tukey's HSD to compare *all* 16 treatment means (try for yourself):

```
soy$cuxmn = with(soy, interaction(cu, mn, sep = "x"))
head(soy$cuxmn)
fit2 = lm(yield ~ cuxmn, data=soy)
lh2 = glht(fit2, linfct = mcp(cuxmn="Tukey"))
summary(lh2)
```

multiple comparisons using package `DescTools`

and `aov` fit (try for yourself):

```
library(DescTools)
PostHocTest(fit.aov, method = "lsd")
PostHocTest(fit.aov, method = "scheffe")
PostHocTest(fit.aov, method = "hsd")
```

using specific contrasts as coefficients

In our model `fit` , all levels except the first have their own coefficients. For each factor, the first level is the "base" level. The intercept corresponds to the mean yield when all the factors are at their "base" level. This parametrization of the design matrix and coefficient is said to use what's called, in R, the "treatment" contrasts. It facilitates pairwise comparisons between treatments.

```
options()$contrasts # default is "contr.treatment"
```

```
        unordered           ordered
"contr.treatment"      "contr.poly"
```

A different parametrization is with the "sum" contrasts, where the `cu` coefficients are α_i with $\Sigma\alpha_i = 0$ for example. Because of the constraint that α's (and β's, γ's etc) sum to 0, R does not estimate the last α_i, here the coefficient associated with level `cu=7` . It takes it as the negative sum of the others: $\alpha_7 = -(\alpha_1 + \alpha_3 + \alpha_5)$, using here the levels of `cu` to index the α coefficents.

```
fit.sum = lm(yield ~ cu_factor * mn_factor, data=soy,
             contrasts = list(cu_factor=contr.sum,
                              mn_factor=contr.sum))
summary(fit.sum)
```

```

Call:
lm(formula = yield ~ cu_factor * mn_factor, data = soy, contrasts = list(cu_factor = con
tr.sum,
    mn_factor = contr.sum))

Residuals:
    Min      1Q  Median      3Q     Max
-110.50  -31.12    0.00   31.12  110.50

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)           2177.6562    11.9623 182.043  < 2e-16
cu_factor1              46.4687    20.7193   2.243 0.039426
cu_factor2             104.0938    20.7193   5.024 0.000125
cu_factor3              49.4688    20.7193   2.388 0.029643
mn_factor1            -675.4063    20.7193 -32.598 4.63e-16
mn_factor2            -268.9062    20.7193 -12.979 6.56e-10
mn_factor3             320.8438    20.7193  15.485 4.74e-11
cu_factor1:mn_factor1  -22.2188    35.8869  -0.619 0.544545
cu_factor2:mn_factor1    0.1563    35.8869   0.004 0.996580
cu_factor3:mn_factor1   27.2813    35.8869   0.760 0.458191
cu_factor1:mn_factor2   56.7813    35.8869   1.582 0.133162
cu_factor2:mn_factor2  -33.8438    35.8869  -0.943 0.359670
cu_factor3:mn_factor2   36.2812    35.8869   1.011 0.327067
cu_factor1:mn_factor3  -96.9687    35.8869  -2.702 0.015705
cu_factor2:mn_factor3   41.9062    35.8869   1.168 0.260023
cu_factor3:mn_factor3   17.5312    35.8869   0.489 0.631814

Residual standard error: 67.67 on 16 degrees of freedom
Multiple R-squared:  0.9916,    Adjusted R-squared:  0.9838
F-statistic: 126.3 on 15 and 16 DF,  p-value: 1.024e-13
```

With these "sum" contrasts, the intercept represents the mean of all group means, which is very nice. But getting individual group means is much more tedious. For example, to get the mean at `cu=3` (second level) and `mn=80` (third level):

```
2177.6562 + 104.0938 + 320.8438 + 41.9062 # 2644.5
```

Group means involving the last level for one or more factors are more complicated. For example, the mean yield at `cu=7` (last level) and `mn=20` (first level) is:

```
2177.6562 -(46.4687+104.0938+49.4688) + -675.4063 -(-22.2188+0.1563+27.2813) # 1297
```

An alternative way to fit the model with the “sum” contrasts is to change the global default contrast settings, and the “sum” contrasts will be use for all factors, now and later:

```
options(contrasts = c("contr.sum", "contr.poly"))
fit.sum = lm(yield ~ cu_factor * mn_factor, data=soy)
summary(fit.sum) # same as above
```