Psych 610 Handout – R4
Two-way Between-Group ANOVA
Equal or Balanced N only!!!

Colleen Moore & Mike Amato
University of Wisconsin - Madison

This tutorial uses example data in Keppel & Wickens, 4th ed, from table 11.8. The dependent variable is number of errors made by monkeys in a search task. The independent variables are type of drug (drug: 1=control, 2=drug_x, 3=drug_y) and degree of food deprivation (food: 1=1 hour of deprivation, 2=24 hours of food deprivation). A text file of the data is on the Learn@UW site, named "Keppel2way.txt".

Note that the aov methods here work ONLY for balanced designs (equal cell n's will give you a balanced design).

1. Get the data into R

Save the data on your computer, and set R's working directory to the folder you saved it in. Use either of the following two commands to load the data into R.

```
>d1=read.table("filename.txt", header=T)  #specify the file to load
>d1=read.table(file.choose(),header=T)    #open a window and browse to the file
```

Check to make sure it read in correctly by typing the name of the data frame you just created.

```
> d1
```

Before anything else, tell R that our IVs are categoric factors, not continuous variables.

```
> d1$drug = as.factor(d1$drug)
> d1$food = as.factor(d1$food)
```

Make your life easier by giving the levels of each factor informative names.

```
> levels(d1$drug) = c("control", "drugX", "drugY")
> levels(d1$food) = c("1-hour", "24-hrs")
```

Now let's look at some descriptives. We'll use the familiar *tapply( )* function to get the means and standard deviations. But since we have two factors now, we'll have to use the *list( )* function to combine them and tell R that we want the mean and standard deviation for each cell in the design. We'll also use the *length( )* function to give us the number of observations in each cell, to make sure our design is balanced.

```
> tapply(d1$errors, list(d1$drug, d1$food), mean)
        1-hour 24-hrs
control      3     11
drugX       10     12
drugY       14     10

> tapply(d1$errors, list(d1$drug, d1$food), sd)
          1-hour   24-hrs
control 3.162278 3.915780
drugX   4.760952 5.477226
drugY   3.915780 4.082483

> tapply(d1$errors, list(d1$drug, d1$food), length)
        1-hour 24-hrs
control      4      4
drugX        4      4
drugY        4      4
```

2. Do the ANOVA and Make Some Graphs

Run the ANOVA using the *aov( )* function, just like with 1-way designs. The only difference is that now we want to include both factors and their interaction in our model. There are two equivalent ways to do this. The first way explicitly specifies each term; the second way is a shortcut. Take your pick.

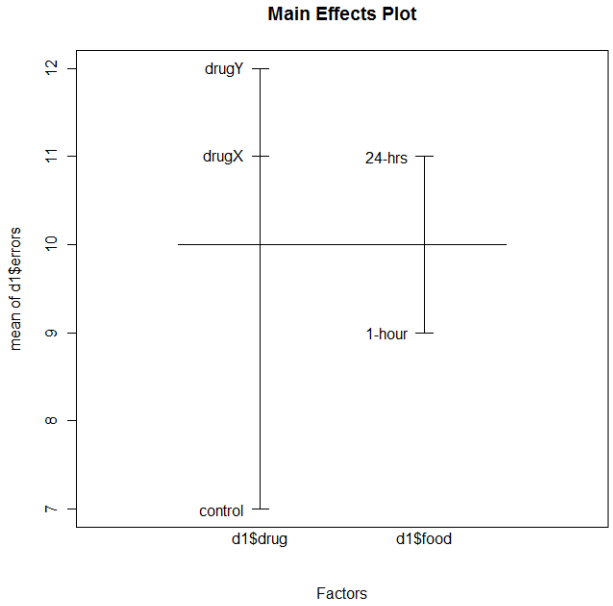
```
> omni = aov( errors ~ drug + food + drug:food, data=d1)  # option 1
> omni = aov( errors ~ drug*food, data=d1)                # option 2
> summary(omni)
            Df Sum Sq Mean Sq F value   Pr(>F)
(Intercept)  1   2400 2400.00 130.9091 1.083e-09 ***
drug         2    112   56.000   3.0545 0.07208 .
food         1     24   24.000   1.3091 0.26755
drug:food    2    144   72.000   3.9273 0.03843 *
Residuals   18    330   18.333
```

We can see the marginal means, standard errors, and number of observations in each cell with the following:

```
> model.tables(omni, "means", se=T)
Tables of means
Grand mean

10  ← this is the grand mean

 drug
drug
control    drugX    drugY
      7       11       12  ← these are the marginal means for factor "drug"

 food
food
1-hour 24-hrs
     9     11   ← these are the marginal means for factor "food"

 drug:food
         food
drug      1-hour 24-hrs
  control  3     11     ← cell means
  drugX   10     12     ← cell means
  drugY   14     10     ← cell means

Standard errors for differences of means  ← SE for differences, NOT for means
         drug  food drug:food
        2.141 1.748     3.028    ← calculated as sqrt( MS_S/AB * 2/n )
replic.     8    12         4    ← marginal n
```

Graph the Main Effects

```
> plot.design(d1$errors~d1$drug*d1$food, main="Main Effects Plot")
```



With this handy plot we see :

- the grand mean
- the cell means
- a graphical representation of their spread

Which is useful. But it would also be nice to see the interactions and some error bars.

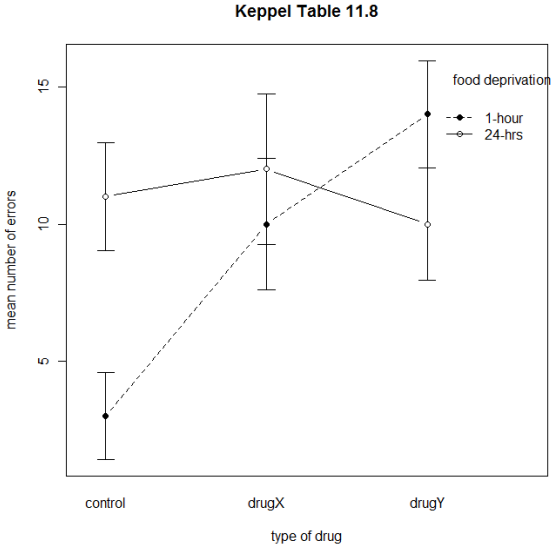
Graph the Interaction

The "sciplot" package has some handy graphing tools. The first time you use it you'll need to install it. Go to R's "Packages" menu, select "Install", choose a mirror site near you, and then scroll to "sciplot" and hit OK. Once the package is installed, load it for use during your current session with the following command:

```
> library (sciplot)
```

Then draw an interaction plot complete with automatically calculated error bars at +/- 1 SE:

```
> lineplot.CI(x.factor=d1$drug, response=d1$errors, group=d1$food,
  trace.label="food deprivation", xlab="type of drug", ylab="mean number
  of errors", main="Keppel Table 11.8")
```



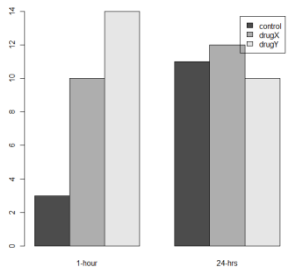
There are a bunch of optional parameters you can set, such as adjusting the y limits, or moving around the legend. We used the default values, but you can easily change them. See:

```
> help(lineplot.CI)
```

Remember that you can only use this type of plot after loading "sciplot" with the *library* command.

This kind of plot is useful for an initial inspection of the data. The lines make it easy to see the interaction, and it uses individual standard errors so we can eyeball homogeneity of variance. But you probably wouldn't want to use it in a publication.

Make a Snazzy Bar Graph



Create a matrix of the group means.

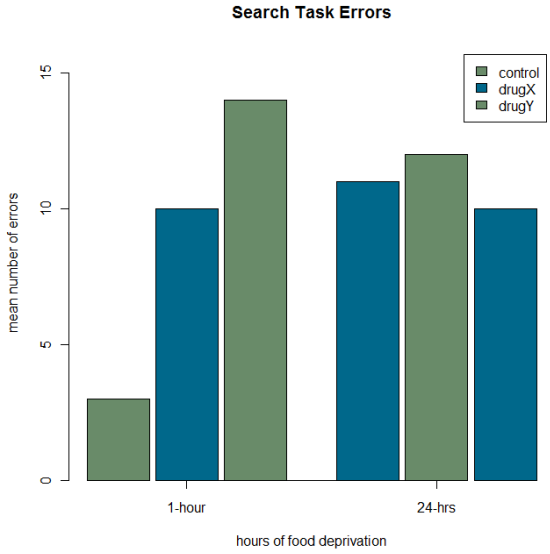
```
> mm = tapply(d1$errors, list(d1$drug, d1$food), mean)
```

Then draw a graph with the minimum amount of information on it.

```
> graph1 = barplot(mm, beside=T, legend=T)
```

That graph gets the job done, but it's pretty bare bones. One of R's biggest strengths is the amount of control it allows over graphs. Users can customize nearly all aspects of the graph by setting optional parameters of the *barplot( )* function. The same parameters are also used for other types of graphs.

```
> graph2 = barplot(mm, beside=T, ylim=c(0,16), space=c(.1,.8),
  main="Search Task Errors", xlab="hours of food deprivation",
  ylab="mean number of errors", legend =T, axis.lty=1,
  col=c("darkseagreen4","deepskyblue4"))
```



beside = T	groups the bars next to each other
ylim = c(0,16)	specifies the y-axis min and max
space = c(.1,.8)	space between bars; as proportion of bar width *1st # is space between bars of same group* *2nd # is space between bars of different groups*
main	main title for the graph
xlab	label for the x-axis
ylab	label for the y-axis
legend = T	display a legend
axis.lty=1	draws a line for the x-axis *line type* *lowercase letter "L" after the dot* *number "one" after equal sign*
col=	colors for the bars

Calculate the pooled standard error of the means. $\sigma_M = \sqrt{MS_{S/AB} / n}$

```
> pooled.se = sqrt(18.33/4)
```

Define the superpose( ) function. Note that we only need to define it once, and then can use it to make as many graphs as we want.

```
> superpose.eb = function (x, y, ebl, ebu = ebl, length = 0.08, ...)
arrows(x, y + ebu, x, y - ebl, angle = 90, code = 3,
length = length, ...)
```

Draw error bars on the graph.

```
> superpose.eb(x=graph2, y=mm, ebl=pooled.se, col="black", lwd=2)
```

x =	x coordinates for the error bars, or simply the name of the graph
y =	y coordinates for the **center** of the error bars, which is the group means
ebl =	length of the error bars, which should be 1 se in each direction
col =	color for the error bars
lwd =	thickness of the error bar lines (*line width)*

3. Simple main effects. Test the simple main effect of drug, at both levels of food deprivation.

1. Separate the data into subsets, based on "food"
2. Run an ANOVA testing the effect of "drug" in each subset
3. Manually recalculate F and p for "drug" in each ANOVA, using $MS_{S/AB}$ from the omnibus ANOVA.

Make subsets of the data for each level of "food".

```
> deprived.1hour = subset(d1, food=="1-hour")
> deprived.24hours = subset(d1, food=="24-hrs")
```

Check that the number of observations in each subset is half of the total observations.

```
> nrow(deprived.1hour)    # nrow( ) returns the number of rows in a dataframe
[1] 12
> nrow(deprived.24hours)
[1] 12
> nrow(d1)
[1] 24
```

Conduct separate ANOVAs for the simple main effect of drug in each subset.

```
> drug.at.1hour = aov(errors~drug, data=deprived.1hour)
> summary(drug.at.1hour, intercept=T)
            Df Sum Sq Mean Sq F value    Pr(>F)
(Intercept)  1    972     972   60.75 2.724e-05 ***
drug         2    248     124    7.75   0.01104 *
Residuals    9    144      16

> drug.at.24hours = aov(errors~drug, data=deprived.24hours)
> summary(drug.at.24hours, intercept=T)
            Df Sum Sq Mean Sq F value    Pr(>F)
(Intercept)  1   1452 1452.00 70.2581 1.522e-05 ***
drug         2      8    4.00  0.1935    0.8274
Residuals    9    186   20.67
```

Stop and think! Look at the $MS_{residual}$ in these simple main effect ANOVAs. They differ slightly. Also, compare $df_{residual}$ to the original omnibus ANOVA on page 2.

The $df_{residual}$ are lower in the simple main effect ANOVAs. Since we are comfortable that homogeneity of variance is a reasonable assumption for our data (based on the plot of predicted values against residuals in the next section), it is probably safe to use the pooled error for these tests. We can do some calculations to use the pooled error from the overall omnibus ANOVA, and its larger number of df. Sometimes the increased power from having more df in the denominator of F changes the results dramatically.

Use pooled error to calculate F and p for the simple main effect of drug at 1 hour of food deprivation.

```
> Fdrug.1hr = 124/18.333     # MS_drug from "drug.at.1hour", divided by MS_residual from "omni"
> Fdrug.1hr
> pf(Fdrug.1hr, 2, 18, lower.tail=F) # ask R to calculate a p-value for the new F
[1] 0.006445319
```

Repeat the calculations for the simple main effect of drug at 24 hours of food deprivation.

```
> Fdrug.24hr = 4/18.333
> pf(Fdrug.24hr, 2, 18, lower.tail=F)
[1] 0.8060713
```

Notice that the p-values of the tests with pooled error is slightly smaller than for the tests using partitioned error, even though MS_{error} from the pooled analysis is slightly higher than the partitioned error for drug at 24 hours. The p-value is slightly lower because the df is so much larger. More df gives more statistical power.

The sum of SS_{drug} in the two partitions should equal $SS_{drug} + SS_{drugXfood}$ in the original ANOVA. If it does, we know we've done things correctly.

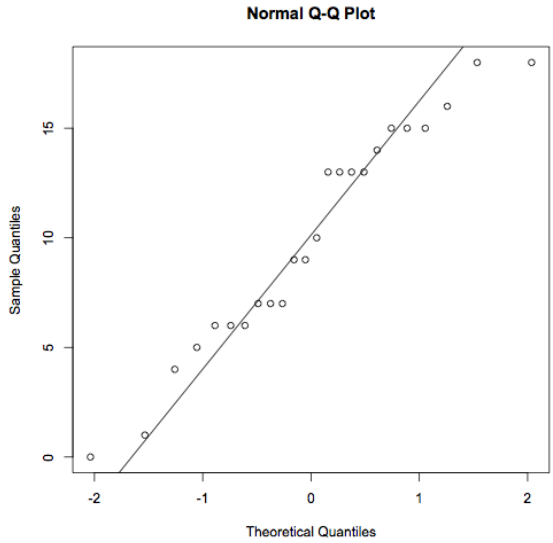
```
> 248 + 8 == 112 + 144
[1] TRUE
```

4. Check some assumptions

Make the normal QQ plot to check that our data is normally distributed. Susan Archambault from Wellesley College has this to say about the normal QQ plot:

"The normal Q-Q plot graphically compares the distribution of a given variable to the normal distribution (represented by a straight line). The straight line represents what our data would look like if it were perfectly normally distributed. Our actual data is represented by the [circles] plotted along this line. The closer the [circles] are to the line, the more normally distributed our data looks."

```
> qqnorm(d1$errors)
> qqline(d1$errors)
```



We want the data to be on the line.
Looks off in the tails, but not by much.
Nothing to get worked up about.

Plot predicted values and residuals.

Compute the predicted and residual values from the omnibus ANOVA, and save them as new columns in dateframe "d1". Note that "omni" is just the name we decided to give the ANOVA when we computed it on page 2. We could have called it anything.

```
> d1$pred = predict(omni)
> d1$res = residuals(omni)
```

Take a look at your error scores, the model's predicted values, and the residuals all together. Marvel at how neatly it all matches up.

```
> checkup = rbind(d1$errors, d1$pred, d1$res)
> checkup
```

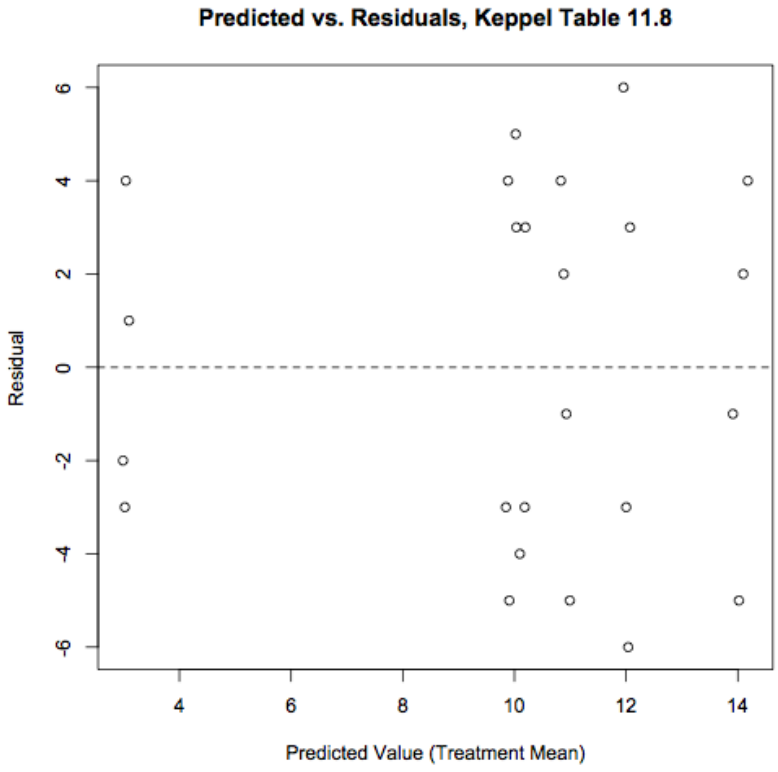
```
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24]
   1    4    0    7   13    5    7   15    9    16    18    13    15     6    10    13     6    18     9    15    14     7     6    13
   3    3    3    3   10   10   10   10   14    14    14    14    11    11    11    11    12    12    12    12    10    10    10    10
  -2    1   -3    4    3   -5   -3    5   -5     2     4    -1     4    -5    -1     2    -6     6    -3     3     4    -3    -4     3
```

Now plot the residuals as a function of their predicted cell mean. Tell R to "jitter" the predicted values a little bit so we can see if there are more than one on top of each other.

```
> plot(jitter(d1$pred), d1$res, xlab="Predicted Value (Treatment Mean)",
  ylab="Residual", main="Predicted vs. Residuals, Keppel Table 11.8")
```

Put a dotted line at zero so we can judge the spread of the residuals more easily.

```
> abline(a=0,b=0, lty=2)    # "lty=2" makes the line dotted. It means "line type"
```

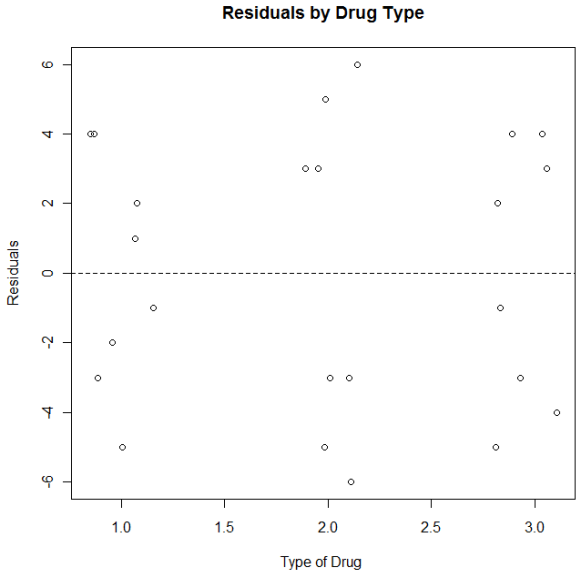


Plotting "pred" against "res" separates the groups along the x-axis in this example, since the only thing the model is using to predict the value is the cell mean. In this example, two of the groups have similar treatment means (at 10), so they are overlapping.

Looks like the treatment group with the lowest mean has the smallest variance, but the differences are not too dramatic.

It's also useful to group the residuals by a single factor, rather than the group mean like we did above. We can look at residuals for each group of factor "drug" just like we did on the earlier handout. This time though we'll jitter the x-values for each point, so they don't end up on top of each other. Note that we are treating "drug" as if it were a continuous variable, rather than a categoric factor, so we can get this scatterplot.

```
> plot( jitter(as.numeric(d1$drug)), d1$res, xlab="Type of Drug ",
  ylab="Residuals", main="Residuals by Drug Type")

> abline(a=0,b=0,lty=2)
```



What happens if we don't treat "drug" as a continuous factor? We get a graph like the following, which is actually quite useful.

```
> plot( d1$drug, d1$res, xlab="Type of Drug ", ylab="Residuals",
  main="Residuals by Drug Type")

> abline(a=0,b=0,lty=2)
```