resids.r: Explanation of code

Goals of code:

- Compute residuals for each observation
- Split data into groups so can look at each individually

Note: the explanation focuses on the new things in this code. Ask if you have any questions about reading in the data file.

We use the hamburger data in hamburger.csv to illustrate fitting a model and extracting residuals.

Calculating residuals
To calculate residuals, we need to use a general purpose model fitting function, lm(). We will use this extensively over the next few weeks to fit both regression models and ANOVA models. It can also be applied to fit the t-test model. The t-test is an ANOVA applied to data with only two groups.

factors Fit an ANOVA model or a regression model
lm() knows whether to fit an ANOVA model or a regression model by looking at whether a variable is defined as a factor or not. A variable that is defined as a factor defines groups of observations; lm() will fit an ANOVA model that compares the mean for each group. A variable that is not a factor defines a regression line; lm() will estimate the intercept and slope for that regression.

Advanced note: If you omit the as.is=T when you read a data file (using either read.csv() or read.txt()), R tries to guess whether the variable defines groups or a line. A variable with character values is turned into a factor for you. A variable with numbers is left as a numeric variable. Fine until groups are numbered (e.g. 1, 2, ...). You want a factor; R's default is a number. That's why my habit is to leave the variables as is and create the factor versions when needed. There are more reasons, mostly concerning something called ghost levels when you subset data, why I prefer to create a factor variable just before it is used.

`hb$trt.f <- factor(hb$treatment)` Create a factor variable
My habit is to explicitly indicate when I want a variable to be a factor. I also define a new variable that is the factor version. The factor() function creates the factor version of a variable. Here, we want to create a factor version of the trt variable. You could store that in the original variable, but I prefer to create a new variable, with a name related to the original variable. trt.f reminds me that the variable is the factor version of trt.

The next two statements print the original variable and its factor version. Notice the additional Levels piece of information for the factor variable. We will talk more later about levels of a factor.

`hb.lm <- lm(cfu ~ trt.f, data=hb)` Fit the t-test model

The lm() function does the actual fitting of the t-test model. The important piece is the first argument, which specifies the formula to be fit. The first piece, to the left of the ∼, is the name of the response variable. The piece to the right of the ∼ specifies the model to be fit. Here, we want to fit a mean to each treatment group, that is each level of the factor trt.f. The data= tells lm() where to find the variables; here, that is in the means data frame. The result is stored in the patty.lm object.

resid() and predict() Calculating residuals and predicted values
The resid() function takes the object created by lm() and extracts the residuals. The predict() function extracts the predicted values. In both cases, these are being stored back in the original data frame with variable names that indicate their contents.

These variables can then be manipulated using other functions, e.g. boxplots, as indicated using the boxplot function. You can use a formula interface to boxplot (first boxplot example) or do what we did earlier and split the data (second boxplot example).

Drawing a QQ plot to assess normality

`qqnorm(hb$resid); qqline(hb$resid)` One QQ plot for a vector of values
The resid variable in the hb data frame is where you just stored the residuals. This variable contains the residual for each of the 12 observations. `qqnorm()` draws a QQ plot, assuming normality, for a column of values. `qqline()` adds a line to indicate what the QQ plot should be if the observations are normally distributed. This line is based on the observations in the middle of the data set; the details of how this line is determined are not important (for this class).

Splitting the residuals by treatment
There are many ways to look at the residuals separately for each group. Here is one way:

`hb.both <- split(hb$resid, hb$treatment)` Split the data into two parts, one for each value of treatment. The next line shows the result. It is a list with two components. The next two lines give you information about hb.both. The first indicates its structure; the second just gives you the names of the components. Components of a list can be accessed individually just like variables in a data frame: by list name $ component name.

The next two pairs of lines plot a QQ plot for just the residuals from the active group, then just residuals from the control group.