calcmeans.r: Explanation of code

Goals of code:

- Comments in R code
- Compute average response for each experimental unit
- Reminder: log transformations

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

The patty data set has 3 plate counts for each hamburger patty. The bacterial concentration was measured three times for each patty. In the language to be used shortly, the experimental unit is the patty; the observational unit (in the original data) is plate count. This is a violation of independence. A more appropriate analysis is based on the average count per patty. After averaging, the observational unit is the patty (because one row of data per patty), which is the same as the experimental unit (good).

The strategy is to calculate the averages for each patty and save them in a new data set. Then, we will analyze that new data set. There are a couple of ways to do this in R. The most straightforward is to use functions in the dplyr library.

install.packages('dplyr') To download and install a new library
Note that the name of the package is in quotes.
This is commented out because it is not needed everytime you run the code.

The dplyr library provides powerful tools to manipulate data sets and especially to create summary statistics for subsets of data. This library is automatically included with RStudio. If you are using "vanilla" R on an ISU server, the library should already be installed.

If you are using "vanilla" R on your computer, you need use install.packages() to download the package and save it on your computer. You are asked which mirror site to use. I suggest US (IA). A copy of the package is installed in your files so you don't need admin privileges . There is also a packages / install packages option from the R menu; this brings up a dropdown menu listing all 10,000+ packages. You only need to install a package once (until you update your R version).

library(dplyr) To calculate means for groups of observations.
In both RStudio and R, you need the library(dplyr) statement once each time you restart R or RStudio. This provides access to the functions in the library. This is necessary before R can use either of the next two statements, which use dplyr functions.

Note: If you get "Error ... could not find function ...", OR the function does something very different from what you expected, you probably forget the `library()` statement. This is a very common problem. Good R programming practice is to put all the necessary library statements at the beginning of your code. If a library is already loaded, there is no issue with a duplicate library statement, at least 99.99% of the time.

`pattyGr <- group_by(patty, trt, rep)` Define grouping variables
The `group_by()` function has two or more arguments. The first is the name of a data frame. The rest are the names of one or more variables that identify groups of observations. For the hamburger patties, you need the treatment name and the rep number to identify the three observations from a specific patty. The data frame with additional grouping information is stored in a new object, for each I chose the name pattyGr.

`means <- summarize(pattyGr, meancfu=mean(cfu))` Calculate the mean for each group
The summarize() function does the actual work of calculating the means. The first argument is the name of the object with grouping information. The rest of the pieces (one or more) are pairs of variable names and a function applied to a variable in the object. The code here calculates the mean, specified by mean(), of the cfu variable, because it is mean(cfu), and stores that in the variable named to the left of the =. That's meancfu in this example. You can have multiple requests, each separated by a comma, if you want multiple summary statistics.

The result is a tibble, a tidy table. This is an extension of a data frame. Everything we've done with data frames can be done with tibbles, plus there are some useful additional commands.

The dplyr and related tidyr libraries provides a large set of functions to manipulate data. A summary is on the data wrangling cheat sheet available at:
`https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf`.

Note: If you end up with only one observation in the result, you applied summarize() to the original data frame (no grouping information) instead of the object with grouping information.

means Print the result so you can see what summarize did

head(patty): If you want to see the first few lines of a data frame (or tibble), head() does that.

View(patty): If you want to bring up a speadsheet-like display of a data frame (or tibble), View() does that.
Note: Capitalization matters to R. This function name is capitalized.

Transforming variables: a reminder - details in pairedt.r
Transformations are done by manipulating variables, just like we computed the difference. R has many mathematical functions. The code in pairedT illustrates three possible transformations. This week we need the log transformation.

`means$logmeancfu <- log(means$meancfu)`
This command does the actual work: computing the log of the value in aff and storing it in the variable logaff. `log()` is the function that computes the natural log (base e), which is the commonly

used log transformation.

Note: There is a difference between `logmeancfu` and `means$logmeancfu`. The first is a stand-alone variable; the second is a new column in the means data frame (or tibble). You could work with stand-alone variables, but many things are made easier by putting the new variable into a data frame. One example is anything using the formula interface and data=.