PairedT.r: Explanation

Goals of code:

- Some useful tricks for working in R or R Studio
  - Executing multiple lines of code
  - Recalling a previous command
  - Suggestions for getting help
  - Specifying a default data frame for a command
  - R reference card
- Inference on difference in means from paired data
  - Using one-sample t-test
  - Using two paired variables
- Wilcoxon signed rank test
- Transforming variables

Executing multiple lines of code:
When you select multiple lines of code then type crtl-enter (R Studio) or crtl-r (R), the block of code is run.

Recalling / editing a previous command:
Sometimes you want to recall a command previously entered in the console window. This may be because you've changed the data set and want to rerun an analysis, or because you want to correct an error in that command without retyping all of it.

R keeps a history list of all the commands that have been executed. You can return to a previous command by activating the console window and typing the up arrow. This will display the previous command. If you type enter, it is executed (again). You can use the mouse or arrow keys to move the cursor within the line. If you then type delete or backspace, characters at the cursor are deleted. If you then type text, that is inserted. When done editing, hit enter and the command is executed.

Clearing the console window:
Sometimes, especially when code is not working as anticipated, you will have lots

of useless stuff in your console window. You can clear the entire window by making the window active and typing ctrl-l (Ctrl "El").

Suggestions for getting help:
The easiest way for us to debug R code is for us to see your code and exactly what R is fussing about. That means:

- If you are typing commands into the code window (top left window in RStudio), or using the code window to edit class code: Save the contents of the code window into a file. You can use File / Save from the main menu, the disk icon at the top of the code window, or ctrl-s (the hotkey for save). Then use the mouse to highlight the troublesome command and the error that R gives you (both from the console window) and copy that information to the clipboard. Paste that information into the **body** of an e-mail message and attach the command file to the e-mail. Add some text to the e-mail briefly explaining what you're trying to do and send us the e-mail.

- If you are using only the console (e.g., by pasting or typing commands directly into the console), use the mouse to select the contents of the console window and copy that to the clipboard. Paste into the body of an e-mail message. Add some text to the e-mail briefly explaining what you're trying to do and send us the e-mail.

  Note: If you've tried lots of things, you may have a lot of stuff in the console window. We can help you faster if you clear the console window (ctrl-l) then re-run the the critical stuff (reading the file, manipulating the data, doing the analysis), and paste the entire console window (ctrl-a ctrl-c) into the body of your e-mail.

Specifying a default data frame:
To specify a variable in a data frame, e.g. the score in the creativity data frame, you need to write `creativity$score`, where the first piece (before `$`) is the name of the data frame and the second piece is the name of the variable. This can get tedious if you have to name the data frame many times. The `with()` function specifies a default data set for the command inside the `()`. For example, creativity.r includes the command `boxplot(split(creativity$score, creativity$treatment))` to draw side-by-side box plots. The version using `with()` is
`with(creativity, boxplot(split(score, treatment)))`. The first argument to `with()` specifies the data frame to be used. The second is the function, or combination of functions to execute. Everyplace R expects a variable name, it looks

for that "inside" the specified data frame. This week's code, pairedt.r, includes another example of using `with()`.

R Reference card: `http://cran.r-project.org/doc/contrib/Short-refcard.pdf`
This is a link to an "R Reference Card", a four page succinct description of basic R functions. There is also a base R cheat sheet at `https://www.rstudio.com/wp-content/upl`
The R Studio folks have produced various very helpful cheat sheets emphasizing their libraries. They also host contributed cheat sheets. A full list of both is at `https://www.rstudio.com/resources/cheatsheets/`.

Pairedt.r: Read the data file:
`schiz <- read.table('case0202.txt', header=T, as.is=T)`
Identical to last week. Same arguments.

Pairedt.r: Print the first few observations: `head(schiz)`
It's useful to print out a bit of the data set to check that it was read as expected. `head()` prints the first 6 lines of thing inside `()`. If you want the last 6 observations, use `tail()`.

Pairedt.r: Calculate the difference: `schiz$diff <- schiz$unaff - schiz$aff`
Remember, `<-` assigns a result (right hand piece) to the specified variable (left hand piece). If that variable is in a data frame (e.g., `schiz$diff`), the data frame gains a variable. We can see the effect of the assignment by looking at the first 6 observations of the new version of the data frame.

Pairedt.r: Calculate the standard error:
This is an extended explanation of what I provided last week

`sd(schiz$diff)/sqrt(length(schiz$diff))`
There is not a built-in function to compute the standard error. The easiest way to calculate it is by telling R the formula: sd / sqrt(n). The `sd()` function calculates the standard deviation of the specified variable. The `length()` function gives the number of values, i.e., the length, of the specified variable. The `sqrt()` function calculates the square root of the specified variable or number.

Note: If the result is `NA`, that means the data set includes missing values. R's missing value code is `NA`. If you attempt to calculate a mean, median, variance, or standard deviation of data that includes a missing value, the default behaviour is to return a result of `NA`. If this happens to you, figure out why the data set includes

missing values.

Define a function to calculate the standard error: 4 lines starting with `se <-`
Note: This is optional and assumes a bit of familiarity with computer programming. If my description doesn't make sense, ignore this section. Or, try it out, experiment, and ask questions to discover what it does.

One of the strengths of R is your ability to add functions. By default, R does not include a function to calculate standard error, but you can define a new function that does that. Basically, you teach R how to calculate the standard error. Here are the pieces:

- `se <- function(x) {`: This says you are defining a function called se that has one input argument (x). The body of the function (the definition of `se()`) is the stuff between the braces, `{` to `}`. Since a function should work for all sorts of data given to it, we want to remove any missing values. That is what `x2 <- na.omit(x)` does.
- `sd(x)/sqrt(length(x))`: This computes the se of the variable `x`. When the function is called, `x` stands for the argument used in the call (see next bold point for more explanation). This value is not assigned to anything. This is deliberate, see next item.
- `}`: Ends the function definition. At the console, the previous line, without an assignment, would print a value as output. Inside a function, this "printed" value is returned as the result from the function.

Pairedt.r: Use the se() function: `se(schiz$diff)`
This executes the `se()` function with the argument, `schiz$diff`, copied to `x` for the duration of the function.

Note: defining a new function is especially useful if you want to calculate the standard error for subgroups of observations. You now have a function that you can use in `tapply()` (see creativity1.r for explanation of `tapply()`).

Pairedt.r: Paired t-test as a one-sample t-test: `t.test(schiz$diff)`
There are two ways to calculate the paired t-test. One is to compute the difference, then do a one-sample t-test. Specifying one argument to `t.test()` does the second part.

Pairedt.r: Paired t-test as a two paired values:
`t.test(schiz$unaff, schiz$aff, paired=T)`
You can also specify the two variables to `t.test()` and specify that the observations are paired (by `paired=T`). The results are exactly the same as the one-sample results.

The next line of code, using `with()` does exactly the same thing as the current line.

Non-parametric (Wilcoxon signed rank) test for paired data:
`wilcox.test(schiz$diff, correct=F)`

The one-sample Wilcoxon test is performed using the `wilcox.test()` function. You provide one vector of differences between the two groups. By default, R will enumerate all permutations if n <= 50 and use a normal approximation (instead of drawing a random sample) when n > 50.

The `correct=F` argument turns off something called the continuity correction. This only affects the normal approximation (n > 50). I don't think it is useful, especially for large n. Tradition in some fields is to always use it; if that's your tradition, omit this argument.

Transforming variables:
Transformations are done by manipulating variables, just like we computed the difference. R has many mathematical functions. The Cheat sheet has a listing of all of them in base R. The code in pairedT illustrates three possible transformations.
PairedT.r: `schiz$logaff <- log(schiz$aff);`
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. The next two lines repeat that computation for the unaff variable, then calculate the difference in the log values. You can choose any name you like to store the result, but it helps to choose a name that you can remember and that gives some idea of the contents.

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

interface and data=.

PairedT.r: `# stuff`
Any text after `#` is a comment. All that text between the `#` and the end of the line (enter character) is ignored by R.

PairedT.r: `log10aff = log10(aff); sqrtaff=sqrt(aff);`
These two lines give you the functions for the log base 10 transformation (e.g. pH) and the square-root transformation. Again, you can choose any variable name you like.