fdr.r: Explanation of code

Goals of code:

- False discovery rate, starting with p-values.
- Subsetting a data frame, determining its size
- Running tests on many variables, saving p-values

False discovery rate adjustment: `p.adjust()`
The p.adjust() function does various adjustments to p-values. It starts with a vector of p-values and returns a vector of adjusted p-values. These can be stored, stored in a data frame, plotted, or printed just like any other column of information.

My code stores the adjusted p-values in the original data frame of tests.

Subsetting a data frame: `subset(tests, fdr < 0.2)`
There are (at least) two ways to extract a subset of rows that satisfy a specified condition. This is useful here when you only want to print (or save) rows with small adjusted p-values.

The simplest way to subset the rows of a data frame is to use the subset() function. The subset() function returns the subset of rows that meet the specified condition. The first argument is the name of the data frame to work with; the second argument is the condition that defines the desired subset. Variables in this condition are "looked up" in the data frame. So here, fdr<0.2 looks at `tests$fdr`, i.e., the fdr variable in the tests data frame.

R provides many logical operators: < and > should be obvious. Less than or equal to is <=, and >= is analogous. Equals is ==; that syntax was chosen by the R developers because it was not the same as = for passing an argument to a function. If R fusses with an obscure error message, check that you didn't accidentally use only one = when you needed two. Not equal is `!=`. You can also negate any expression using not, which is `!()`, where the thing you are complementing is inside the `()`. So, `!(fdr > 0.2)` is the same as `fdr <= 0.2`.

The result of subset() is saved as a new data frame. If you don't save it, it is printed.

There are other ways to subset either rows or columns of a data frame or to subset a vector. I will introduce those when / if we need them.

Determining the size of a data frame: `dim(signif)`
`dim()` returns the number of rows and number of columns in a data frame. Useful here because the number of rows in the subset is the number of tests with fdr < 0.2.

Running tests on many response variables; saving p-values
The `p.adjust()` function requires a vector with p-values, one for each test. Here is one way to get that vector.

If you just want to run my code, you create specific variables containing the necessary information. These include the data set (in dataset), the column number or name with the variable that defines the groups (in trt), the number of the first column with response variables (in first) and the number of the last column (in last). We want to analyze the manyY data set. Column 2 has the treatment name, columns 3 through 32 are the 30 response variables.

Comment or uncomment the line with the test you want. Then just run the code below the comment line. Don't change any else. `pval` is a vector with the p-values for each test. You can pass this into p.adjust() to get the fdr adjustment. If you want a nicer printout of the p-values, `pvalues` is the same information as in `pval`, saved as a data frame with two columns, the response variable name and the p-value.

How my code works: The concept is to write a for loop that runs the desired test on each variable. Inside the for loop, i is first, then first+1, then first+2, $\cdots$ through last. The squiggly braces, `{` and `}`, define what executed for each value of i.

Up to now, we have specified variables by name. You can also subset a specific column of the data frame by column number. That's what `dataset[,i]` does. The results of the test are saved (not printed). One component of the test result is called p.value. That has the 2 sided p-value.

To accumulate the p-values for all tests, we first create a vector to save all the results (`pval <- rep(NA, last)`, which repeats NA, the numeric missing value, last times). Then, inside the loop, `pval[i] <- temp$p.value`, extracts the pvalue and saves it in the i'th position of the vector.

When the loop finishes, there are p-values in pval[first], pval[first+1], $\cdots$ pval[last]. There are missing values in pval[1], pval[2], $\cdots$ pval[first-1]. `pval <- pval[first:last]` extracts the values from pval[first] to pval[last] and overwrites pval. The `names(pval)` line extracts the variable names for the response variables from the dataset and assigns them to the elements of pval. The `pvalues` creates a data frame with the information from pval.