fdr.sas: Explanation of code

Goals of code:

- False discovery rate, starting with p-values.
- False discovery rate, starting with raw data.

False discovery rate adjustment: `proc multtest`
Proc multtest can do many different adjustments for multiple comparisons and multiple testing. It can be used in two ways:

it can do simple analyses (e.g., a two-sample t-test) from raw data, compute p-values, then produce the adjusted p-values.

it can start with p-values from any analysis, then produce adjusted p-values.

General principles:
My code illustrates how to start with p-values. The p-value data file contains one row for each test. One of the variables is the p-value for that test. proc multtest produces printed output with the row number (labelled test), the original p-value and the adjusted p-value. To find those tests that are significant with a false positive rate of 5%, tell multtest to use an fdr adjustment, then search through the output for any lines with False discovery rate less than 0.05.

To use the code in fdr, the only thing you need to change is the name of variable that contains the p-values. That is the name in `()` in the `inpvalues(ProbF)` part of proc multtest. This code looks for the p-values in the variable named `ProbF`. If the variable is called ProbF, you don't need to change anything. If it is called something else, change ProbF to the name of the appropriate variable.

The rest of the code (the two proc print's) are additional ways to get the results, which may simplify finding the results you want.

To help identify tests (especially if the row number is not very easy to interpret), multtest can store a data set with the all the information in the p-value data set and the adjusted p-values. The proc print shows all the information about each test, including the name of the response variable, the original p-value (in ProbF) and the adjusted p-value (in `fdr_p`).

To narrow that search by only printing observations that have a small value of `fdr_p`, you can subset the data set "on the fly". The second proc print does this. The where command only uses (in that proc step) the observations that satisfy the specified condition. `where fdr_p < 0.2;` in a `proc print` will print only those observations with values of `fdr_p` less than 0.2. You could also use a subsetting if (described in fdrfull.sas) to create a data set with only the desired subset, then print that data set. The where command does that in one step without requiring you to create a new data set.

The following are details, which can be ignored if you don't want to understand the code.

Calculating adjusted p-values:
`proc multtest inpvalues(ProbF)=tests2 fdr out=fdr;`
proc multtest calculates many different sorts of adjusted p-values. You can use multtest to do simple tests and then make adjustments. I have found it easier to do the analysis I want separately, save the p-values, then use multtest only to adjust the p-values. This statement calculates FDR adjusted p-values.

The `inpvalues(ProbF)=tests2` tells SAS where to find the p-values. The assumption is that this is a data set with one row for each test. The `=tests2` piece names the SAS data set containing the p-values. The `inpvalues` piece tells multtest that you are providing p-values as input. The `(ProbF)` specifies the name of the variable containing the p-values. If you look at the printout of the contents of tests2, you see that SAS used ProbF as the name for the variable containing the p-values (remember, SAS created this data set, by the ods output command, so you couldn't choose names).

The `fdr` option requests fdr adjustment. There are many others. See the documentation for the full list.

The `out=fdr` tells SAS to store the results in a new data set. You provide the name (just like all other out= commands). The example code includes saving the data set so we can print the subset of observations with small adjusted p-values.

The output from proc multtest is explained in the false discovery rate section, above.

You can also print the data set that stored the fdr results, which shows you more information about each test.

printing a subset of observations: `where fdr_p < 0.2;`
To print only observations that have a small value of `fdr_p`, you could use a subsetting if statement to create a new data set with only the subset (just like we created tests2 above). You can also subset the data set "on the fly" using a where command. The where command does the subsetting in one step without requiring you to create a new data set.

The second proc print illustrates this. The where command only uses observations that satisfy the specified condition. `where fdr_p < 0.2;` will print only those observations with values of `fdr_p` less than 0.2. If you want to use the subset multiple times, you probably want to create a data set with that subset. If you want to use the subset only once, you probably want to use where.

You can use a where command in any proc step. It does the same thing everywhere: restricts the action of the proc step to those observations that satisfy the where condition.

FDR adjustment, starting with raw data
My code demonstrates proc multtest's ability to conduct simple tests, then make the specified adjustments. As before `proc multtest fdr;` requests fdr adjustment. A t-test is specified by:
a `class` statement giving the name of variable that identifies groups,
`test mean()` requesting comparison of means, with variables inside the `()`. See next section for specifying a list of variables.
`contrast 'diff' 1 -1;` specifying the desired contrast, i.e., difference between two groups. As with proc glm contrast statements `'diff'` is a label that must be in quotes.

Specifying a list of variables:
The manyY data set has 30 response variables, called y1, y2, y3, $\cdots$ y30. We want to use all 30. This can be done in three different ways:
list the variables one by one: y1 y2 y3 y4 (keep on going) y30
list a numeric sequence of variables: `y1 - y30`. This requires all variables have the same "stem", i.e., y here.
an arbitrary sequence of variables: `y1 -- y30`. Two hyphens tells SAS to use all variables between the first and the last. The order of variables is the order (from left to right) when the data set is printed. This method can be used for any collection of variable names.
And you can combine approaches, so `y1-y6 a2-a10 bad good` specifies 17 variables.

For the manyY data set, `y1-y30` and `y1 -- y30` are equivalent, so I commented out one statement.

One line comments:
You may remember that `/*` and `*/` surrounding text are comments.

Another way to comment, especially useful for SAS code, is to put `*` as the first character on the line. Any text between * and ; is commented out. Both types of comments can span multiple lines.