Comments and Hints
Assignment #3

General Comments

- *Please do not forget to refer to you statistics text.* You need to know what you are doing in order to use R correctly. This will also enable you to fill in correct documentation for your code (i.e., the "# ..." lines), and enable you to spot errors when they occur.
- Suppose you want an example of the use of a particular function. *Sometimes* (definitely *not very often*), a nice example appears in the R documentation pages for the function. You can read the examples there, or you can run them by running – for example, `example(table)`– in the console.
- When working with *data frames* or *lists*, take advantage of the `with` or `attach` functions, or, at a more basic level, take advantage of the `$` operator.

Othjer More Specific Comments

Problems 3 & 4: Contingency Tables

1. Suppose `Factor` represents a categorical variable. It should be clear from the R documentation pages that `table(Factor)` produces a 1-dimensional contingency table. Similarly, if `Factor1` and `Factor2` represent two categorical variables, one might try `table(Factor1, Factor2)`. Guess what you might try for three factors. What will the output look like? How many dimensions are you dealing with now?
2. How you order the factors in the `table` function has a big effect on how your output looks. Play around with the order of the factors, the goal should be to look for (the most) compact output – particularly when you go above 2-dimensions.

Problem 5: Constructing Tables of Statistics by Factors

In the exploratory exercises you are asked to look at three functions, `sapply`, `tapply`, and `by`. Here are some guidelines. Also, *do not forget to gain access to the variables in question* using the `attach` or `with` functions, or the `$` operator as needed.

`sapply`

Suppose you have k variables (samples), *all of which have the same measurement scale* (i.e., all are numeric, or categorical, or whatever). You can use this function to *apply a particular function to all k variables simultaneously.* For example, if all of the variables are numeric,

```
sapply(X = list(var1, var2, ..., vark), FUN = mean)
```

outputs the means for all k variables(samples) in a nice compact form. The samples (variables) need to be entered into the function as a list, and it is not necessary for all of the samples to have the same size (length). Note: The `lapply` function does exactly the same job, but doesn't always produce pretty output.

`tapply`

Suppose you have a numeric variable, say `score`, for which the entries are also classified by a categorical variable (factor), say `gender`. Now, if you want to calculate mean scores by gender (*without doing any additional work*, such as extracting all the male scores, and female scores, and then computing the means for each using the `sapply`, or more basic functions), then

```
tapply(X = score, INDEX = gender, FUN = mean)
```

produces what you want. For this function, both variables, in this case `score` and `gender` must have the same length. Additionally, *the* `INDEX` *must be a factor.*

by

Think of this as a supped-up version of `tapply`. Suppose you have a numeric variable, say `score`, for which the entries are classified by the categorical variable (factor) `gender`, *and* the categorical variable `course`. If you want to calculate mean scores by gender *and* course (*without doing any additional work*, such as extracting all the male scores for each course, and female scores for each course, and then computing the means for each using the `sapply`, or more basic functions), then

```
by(X = score, INDICES = list(gender, course) FUN = mean)
```

produces what you want. All three variables, `score`, `gender` and `course` must have the same length. Additionally, *the* `INDICES` *must be all be factors.*

A function that might help for Problem 5 is the `as.table` function. Try, also, running `example(by)`, and *look at the very last application that appears in the output.* This might help with Problem 5.

aggregate and apply

The output from `by` will usually not be very compact if there are two or more indices. If more compact (but very differently formatted) output is desired, another function that might find use is the `aggregate` function. Later, we will also encounter the `apply` function.

All of these functions perform the same task as for-loops (for those of you who have encountered these previously). The "apply-type" functions and the `aggregate` function do allow for compact programming, but they can get hairy very, very quickly when the task at hand is more complicated than the above simple applications. My personal feeling is that for-loops are a lot more intuitive. You will encounter for-loops in Assignment 4.