vitC.r: Explanation of code

Goals of code:

- Reading a contingency table in “count” form
- Contingency table tests, proportions, and measures of association
- Reading event data in “individual” form
- Exact tests

Note: There are **many** different methods to analyze contingency table data. Everything we've done previously in the course has had alternative methods. Contingency table data has **a lot** of alternatives. The answers are usually only slightly different, especially with moderate to large sample sizes. I show how to get large sample results and one of the small sample (“exact”) results.

Reading “count form” data: `vitc.txt`
When the counts in a contingency table are pre-tabulated, the easiest way to do stats on that table is to enter the table with one row of data for each cell (combination of row and column) in the table. The key piece of information is the number of observations in that cell. Here we store that in the n variable. The row and column identifiers can be stored as character or numeric variables. n must be numeric.

Contingency table computations: `chisq.test()`
Most computations want the contingency to be represented by a table. The `matrix()` function converts the vector of counts in `vitc$n` into two rows and two columns. The arguments are the vector of counts, information about the size of the matrix, and information about the ordering of numbers. The `nrow=` argument specifies the number of rows. You can also specify `ncol=` or both. If you only specify one, the missing value is determined from length of the starting vector and the specified number. Here, there are four counts in the vitc data frame and we specify 2 rows. Hence, there are 2 columns. If there had been 6 counts, we would have gotten 3 rows.

The `byrow=T` specifies how R is to fill the matrix. By default, matrices are filled going “down the columns”, so the vector 1, 2, 3, 4 produces the matrix:

$$\begin{array}{cc} 1 & 3 \\ 2 & 4 \end{array}$$

It is customary to put groups or treatments in rows and the different outcomes in columns. That means the values in the vitamin C data frame should fill the matrix “going across”, that is 1, 2, 3, 4 should become

$$\begin{array}{cc} 1 & 2 \\ 3 & 4 \end{array}$$

This is specified by adding `byrow=T`.

The Chi-square test is provided by the `chisq.test()` function. The input is the r x c matrix of counts in the contingency table. The output includes the χ^2 statistic, its df, and the p-value. In lecture, I briefly discuss the continuity correction and why I prefer to not use it. The `correct=F` argument turns off that correction.

Fisher's exact test is provided by the `fisher.test()` function. Again, the input is the matrix of counts. This can be used for any size contingency table, not just a 2x2 table (although Fisher only described the 2x2 analysis).

Manipulating tables of counts:
The matrix `counts` is just the numbers. Sometimes, you want to reverse the rows (or the columns). This can be done various ways. One is to subscript rows in the desired new order. That's what `counts[c(2,1),]` does. It is acting on rows because the vector of subscripts are before the comma. If subscripts are after the comma, i.e., `counts[, c(2,1)]`, that would switch the columns.

Sometimes it helps to have names for the rows and columns. For a matrix, these are stored as the dimnames attribute. This is a list. The first part of the list is the vector of row names; the second is the vector of column names. Adding row and column names is optional. It does not have to be done and does not change any of the numerical results.

Odds ratios: `oddsratio()`
The oddsratio function in the epitools library calculates the odds ratio. You can give it the matrix of counts (first two calls of oddsratio) or a vector of counts that will be converted to a matrix (last two calls of oddsratio). If you have defined row and column names, those are used to label the output.

The `method='wald'` argument specifies how to calculate confidence intervals for the odds ratio. The Wald method is the one we've discussed in class.

The output includes the contingency table with marginal totals (the $data object), the odds ratio with 95% confidence interval (in the $measure object), and p-values for three different tests of odds ratio = 1. The chi.square number is the p-value for the usual chi-square test.

If you want to reverse the rows or columns, which inverts the odds ratio, add rev='rows' or rev='columns'.

Analyzing data in "long" format: `vitcLong`
The vitclong.csv file contains one row for each individual in the Vitamin C study. This data format is likely when the data are stored as individual records.

The contigency table (# individuals with each combination of trt and cold) can be produced by table(). You can either explicitly specify two columns of a data frame or use with() to indicate the data frame once then reference the columns. The two versions of table produce the same results.

When chisq.test() is given the vectors of group and response, it will count the number of events in each group and then apply the Chi-square test. The results from analyzing the long format data are the same as the results analyzing the matrix of counts.

Fisher's exact test: `fisher.test()`
For a 2x2 contingency table, the Fisher exact test is obtained by fisher.test() with the data as either a 2x2 matrix (summary format) or two columns (long format).

Exact tests with tables larger than 2x2: `exact chisq`
When the contingency table is larger than 2x2, the Fisher test isn't defined, but randomization can be used with the Chi-square statistic to get an exact p-value. This exact test would be more appropriate when sample sizes are small and necessary when sample sizes are really small (e.g., most expected counts < 5).

chisq.test() allows you to estimate the p-value by simulation, instead of using a theoretical distribution or trying to enumerate all possible contingency tables. This is requested by adding simulate.p.value=T to chisq.test().