Part 1: Categorical data and contingency tables

To perform tests for independence, you will need a contingency table. This may be a 2x2 table or a larger R row by C column table.

The `table()` function takes factor variables as its input. If you provide one factor variable (e.g., `georoc$rock.type`) as the input, `table()` will count the number of items in each category (specifically, in each level of the factor) and will return a single-row contingency table.

If you provide two factor variables (separated by a comma) to the `table()` function – e.g. `table(factor1, factor2)` – it will generate a contingency table with factor1 as the rows and factor2 as the columns.

You may want to manually enter a contingency table with counts that aren’t generated from a larger data set. To do that, you’ll need the `matrix()` function. In R, a matrix variable has rows and columns of numbers, like a two-dimensional version of the numeric vector variable.

Data can be formed into a matrix in a variety of ways, so it’s best to be consistent when learning. One general format is `matrix(data, ncol = x, byrow = F)`, but let’s break that down into its parts.

First you need your data – the counts in all of the categories. This will be in the format of a numeric vector. Remember to use the `c()` function to indicate that all of the numbers should be grouped into a vector. For example, the data may look something like this: `c(5, 12, 18, 6)`.

You then need to decide on the number of columns your matrix should contain. The data length must be a multiple of the number of columns. In the example data given above, your number of columns could be 1, 2, or 4, but not 3.

Finally, you can specify whether to enter the data by rows, which means that it works across row 1 before moving to row 2 (like you would read words on a page) or by columns, which means that it works down column 1 before moving to column 2. Entry by columns is the default. For example,

```
matrix(c(5, 12, 18, 6), ncol = 2) #byrow = F is the default
```

```
[,1] [,2]
[1,]   5  18
[2,]  12   6
```

But `matrix(c(5, 12, 18, 6), ncol = 2, byrow = T)` gives you:

```
[,1] [,2]
[1,]   5   12
[2,]  18   6
```
Part 2: Statistical tests for categorical data

Fisher’s exact test (for independence)

Fisher’s exact test is most often performed and was specifically designed for 2x2 contingency tables. The function name is fisher.test() and it simply needs a single matrix variable as an input. That matrix will be your contingency table. Because the table is 2x2, the orientation (rows and columns) doesn’t matter.

For example, you can create the contingency table matrix and store it as a variable:

```r
obs_cts <- matrix(c(5, 12, 18, 6), ncol = 2)
```

Then run Fisher’s exact test:

```r
fisher.test(obs_cts)
```

Chi-squared test (for independence)

The R function chisq.test() can take input in several forms, and can do tests of goodness-of-fit as well, but we’ll just focus on one case (entering a contingency table matrix) for simplicity. The method tests for independence among categories and those categories should be the columns in the contingency table matrix.

Consider the following contingency table, with counts of mineral grains (based on a total count of 50 points in each sample) from three rock samples:

<table>
<thead>
<tr>
<th></th>
<th>Rock 1</th>
<th>Rock 2</th>
<th>Rock 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quartz</td>
<td>33</td>
<td>28</td>
<td>22</td>
</tr>
<tr>
<td>Biotite</td>
<td>6</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>Feldspar</td>
<td>11</td>
<td>14</td>
<td>26</td>
</tr>
</tbody>
</table>

You first need to set up the contingency table matrix:

```r
min_abund <- matrix(c(33, 6, 11, 28, 8, 14, 22, 2, 26), ncol = 3)
```

Then run the chi-squared test:

```r
chisq.test(min_abund)
```

You may get a warning if overall sample size or one or more of the counts is too small.

Extra: Chi-squared test for goodness of fit

You should try to use an exact test for goodness of fit (from last class), but if your sample size is large they won’t be feasible. The chi-squared test can test for goodness of fit instead, if you provide a numeric vector with observed counts and a numeric vector with expected probabilities.

```r
chisq.test(c(n1, n2, n3, ...), c(p1, p2, p3))
```