6 Working with Tables in R

6.1 Intro

Tables are often essential for organzing and summarizing your data, especially with categorical variables. When creating a table in R, it considers your table as a specifc type of object (called “table”) which is very similar to a data frame. Though this may seem strange since datasets are stored as data frames, this means working with tables will be very easy since we have covered data frames in detail over the previous tutorials. In this chapter, we will discuss how to create various types of tables, and how to use various statistical methods to analyze tabular data. Throughout the chapter, the AOSI dataset will be used.

6.2 Creating Basic Tables: table() and xtabs()

A contingency table is a tabulation of counts and/or percentages for one or more variables. In R, these tables can be created using **table()** along with some of its variations. To use table(), simply add in the variables you want to tabulate separated by a comma. Note that table() does not have a data= argument like many other functions do (e.g., ggplot2 functions), so you much reference the variable using dataset$variable. Some examples are shown below. By default, missing values are excluded from the counts; if you want a count for these missing values you must specify the argument useNA=“ifany” or useNA=“always”. The below examples show how to use this function.

```
aosi_data <- read.csv("Data/cross-sec_aosi.csv", stringsAsFactors=FALSE, na.strings = ".

# Table for gender
table(aosi_data$Gender)
```

```
##
## Female   Male
##    235    352
```

```
# Table for study site
table(aosi_data$Study_Site)
```

```
##
## PHI SEA STL UNC
## 149 152 145 141
```

```
# Two-way table for gender and study site
table(aosi_data$Gender, aosi_data$Study_Site)
```

```
##
##          PHI SEA STL UNC
##   Female  55  67  60  53
##   Male    94  85  85  88
```

```
# Notice order matters: 1st variable is row variable, 2nd variable is column variable

# Let's try adding in the useNA argument
table(aosi_data$Gender, aosi_data$Study_Site, useNA = "ifany")
```

```
##
##          PHI SEA STL UNC
##   Female  55  67  60  53
##   Male    94  85  85  88
```

```
table(aosi_data$Gender, aosi_data$Study_Site, useNA = "always")
```

```
##
##          PHI SEA STL UNC <NA>
##   Female  55  67  60  53    0
##   Male    94  85  85  88    0
##   <NA>     0   0   0   0    0
```

```
# Let's save one of these tables to use for later examples
table_ex <- table(aosi_data$Gender, aosi_data$Study_Site)
```

Now let's add row and column labels to the gender by study site table. For a table object, these labels are referred to as "dimnames" (i.e., dimension names) which can be accessed using the **dimnames()** function. Note that this is similar to the **names()** function with lists, except that now our table has multiple dimensions, each of which can have its own set of names. For a table, dimnames are stored as a list, with each list entry holding the group labels for the variable corresponding to that dimension. The name for each of these list entries will specify the actual label to be used in the table. By default, these names are blank, hence why the default table has no row and column labels. We can change this by specifying these names, using **names()** with dimnames().

```
dimnames(table_ex)
```

```
## [[1]]
## [1] "Female" "Male"
##
## [[2]]
## [1] "PHI" "SEA" "STL" "UNC"
```

```
# we see the group labels.  Note that each set of group labels in unnamed (blanks next t
names(dimnames(table_ex))
```

```
## [1] "" ""
```

```
# Now, let's change these names and see how the table changes
names(dimnames(table_ex)) <- c("Gender", "Site")
names(dimnames(table_ex))
```

```
## [1] "Gender" "Site"
```

```
table_ex
```

```
##         Site
## Gender   PHI SEA STL UNC
##   Female  55  67  60  53
##   Male    94  85  85  88
```

```
# Now the row and column labels appear, making the table easier to understand
```

It also common to view these tabulations as percentages. This can be done by using **prop.table()**, which unlike table() takes in a table object as an argument and not the actual variables of interest. Note that any changes to dimnames that are done to the table object are kept when applying prop.table(). The output from prop.table() is also stored as an object of type table.

```
# 2 Way Proportion Table
prop_table_ex <- prop.table(table_ex)
prop_table_ex
```

```
##         Site
## Gender          PHI        SEA        STL        UNC
##   Female 0.09369676 0.11413969 0.10221465 0.09028961
##   Male   0.16013629 0.14480409 0.14480409 0.14991482
```

A second way of creating contingency tables is using the **xtabs()** function, which requires the **stats** package (which is included in R by default, though still load the package using **library()**). The function xtabs() creates a object of type xtabs and you will notice that the output of both xtabs() and tabel() is nearly identical. xtabs() has the following advantages: 1) row and column labels are included automatically, set to the variable names and 2) there is a data= argument, which means you just have to reference the variable names. With xtabs(), you do not list out the variables of interest separated by commas. Instead you use formula notation, which is ~variable1+variable2+… where variable1 and variable2 are the names of the variables of interest. You can add more then two variables (hence the …). See below for the two-way gender and site example.

```
library(stats)
table_ex_xtabs <- xtabs(~Gender+Study_Site, data=aosi_data)
table_ex_xtabs
```

```
##         Study_Site
## Gender   PHI SEA STL UNC
##   Female  55  67  60  53
##   Male    94  85  85  88
```

To create a table of proportions using xtab(), you first create the table of counts using xtab(), and then use the prop.table() function on this table object. This is exactly what was done when using table().

One useful function when creating tables is proportions is **round()**. As seen with the previous table of proportions, R will not round decimals by default. The round() function can be used for all types of R objects. The first argument is the object of values you want to round and the second argument is the number of decimal places to round to.

```r
prop_table_ex_xtabs <- prop.table(table_ex_xtabs)
prop_table_ex_xtabs
```

```
##         Study_Site
## Gender          PHI        SEA        STL        UNC
##   Female 0.09369676 0.11413969 0.10221465 0.09028961
##   Male   0.16013629 0.14480409 0.14480409 0.14991482
```

```r
prop_table_ex_xtabs <- round(prop_table_ex_xtabs, 2)
prop_table_ex_xtabs
```

```
##         Study_Site
## Gender    PHI  SEA  STL  UNC
##   Female 0.09 0.11 0.10 0.09
##   Male   0.16 0.14 0.14 0.15
```

```r
prop_table_ex <- round(prop_table_ex, 2)
prop_table_ex
```

```
##         Site
## Gender    PHI  SEA  STL  UNC
##   Female 0.09 0.11 0.10 0.09
##   Male   0.16 0.14 0.14 0.15
```

Lastly, we discuss how to add margin totals to your table. Whether using table() or xtab(), a simple way to add all margin totals to your table is with the function **addmargins()** from the stats package. Simply add your table or xtab object as the first argument to the addmargins() function, and a new table will be returned which includes these margin totals. This also works with tables of proportions.

```
table_ex <- addmargins(table_ex)
table_ex_xtabs <- addmargins(table_ex_xtabs)
prop_table_ex <- addmargins(prop_table_ex)
prop_table_ex_xtabs <- addmargins(prop_table_ex_xtabs)

table_ex
```

```
##         Site
## Gender   PHI SEA STL UNC Sum
##   Female  55  67  60  53 235
##   Male    94  85  85  88 352
##   Sum    149 152 145 141 587
```

```
table_ex_xtabs
```

```
##         Study_Site
## Gender   PHI SEA STL UNC Sum
##   Female  55  67  60  53 235
##   Male    94  85  85  88 352
##   Sum    149 152 145 141 587
```

```
prop_table_ex
```

```
##         Site
## Gender    PHI  SEA  STL  UNC  Sum
##   Female 0.09 0.11 0.10 0.09 0.39
##   Male   0.16 0.14 0.14 0.15 0.59
##   Sum    0.25 0.25 0.24 0.24 0.98
```

```
prop_table_ex_xtabs
```

```
##         Study_Site
## Gender     PHI  SEA  STL  UNC  Sum
##   Female 0.09 0.11 0.10 0.09 0.39
##   Male   0.16 0.14 0.14 0.15 0.59
##   Sum    0.25 0.25 0.24 0.24 0.98
```

There are many packages which you can install with more advanced tools for creating and customizing contingency tables. We will cover some in the Chapter 9, though table() and xtabs() should suffice for exploratory analyses.

6.3 Tabular Data Analysis

In this section, we detail some common statistical methods used to analyze contingency table data as well as how to implement these methods in R. These methods are defined and the statistics behind them are explained and then implementation in R is discussed and shown through examples.

6.3.1 Tests for Independence

6.3.1.1 Defining Independence

Suppose we have two categorical variables, denoted X and Y. Denote the joint distribution of X and Y by $f_{x,y}$, the distribution of X by f_x and the distribution of Y by f_y. Denote the distribution of X conditional on Y by $f_{x|y}$ and the distribution of Y conditional on X by $f_{y|x}$.

In statistics, X and Y are independent if $f_{x,y} = f_x * f_y$ (i.e., if the distribution of X and Y as a pair is equal to the distribution of X times the the distribution of Y). This criteria is the equivalent to $f_{x|y} = f_x$ and $f_{y|x} = f_y$ (i.e., if the distribution of X in the whole population is the same as the distribution of X in the sub-population defined by specific values of Y). As an example, suppose we were interested in seeing if a person voting in an election (X) is

independent of their sex at birth (Y). If these variables were independent, we would expect that the percentage of women in the total population is similar to the percentage of women among the people who vote in the election. This matches with our definition of independence in statistics.

6.3.1.2 Chi-Square Test for Independence

To motivate the concept of testing for independence, let's consider the AOSI dataset. Let's see if study site and gender are independent. Recall the contingency table for these variables in the data was the following.

```
table_ex_xtabs
```

```
##         Study_Site
## Gender   PHI SEA STL UNC Sum
##   Female  55  67  60  53 235
##   Male    94  85  85  88 352
##   Sum    149 152 145 141 587
```

```
prop_table_ex_xtabs
```

```
##         Study_Site
## Gender    PHI  SEA  STL  UNC  Sum
##   Female 0.09 0.11 0.10 0.09 0.39
##   Male   0.16 0.14 0.14 0.15 0.59
##   Sum    0.25 0.25 0.24 0.24 0.98
```

From our definition of independence, it looks like gender and site are independent based on comparing the counts within each gender and site group as well as the population-level counts. Let's conduct a formal test to see if there is evidence for independence. First, we cover the Chi-Square test. For all of these tests the null hypothesis is that the variables are independent. Under this null hypothesis, we would expect the following contingency table.

```
expected_table <- table_ex_xtabs
sex_sums <- expected_table[,5]
site_sums <- expected_table[3,]
expected_table[1,1] <- 149*(235/587)
expected_table[2,1] <- 149*(352/587)

expected_table[1,2] <- 152*(235/587)
expected_table[2,2] <- 152*(352/587)

expected_table[1,3] <- 145*(235/587)
expected_table[2,3] <- 145*(352/587)

expected_table[1,4] <- 141*(235/587)
expected_table[2,4] <- 141*(352/587)
expected_table <- round(expected_table,2)
```

Where did these values come from? Take the Philadelphia study site column as an example (labeled PHI). As explained before, under independence, in Philadelphia we would expect the percentage of female participants to be the same as the percentage in the total sample There are 149 participants from Philadelphia, 235 females, and 587 total subjects in the sample. The total sample is about 40% female, so we would expect there to be approximately 0.40*149 or 59.6 females from the Philadelphia site and thus approximately 89.4 males. That is, the expected count is equal to (row total*column total)/sample size. All entries are calculated using this equation. Let's look at the differences between the counts from the AOSI data and the expected counts.

```
expected_table[-3,-5]-table_ex_xtabs[-3,-5]
```

```
##         Study_Site
## Gender     PHI   SEA   STL   UNC
##   Female  4.65 -6.15 -1.95  3.45
##   Male   -4.65  6.15  1.95 -3.45
```

We can see that the differences are small considering the study site margins, so it there no evidence to suggest dependence. However, let's do this more rigorously using a formal hypothesis test. For any hypothesis test, we create a test statistic and then calculate a p-value from this test statistic. Informally, the p-value measures the probability you would observe a test statistic value as or more extreme then the value observed in the dataset if the null hypothesis is true. For the Chi-Square test, the test statistic is equal to the sum of the squared differences between the observed and expected counts, divided by the expected counts. The distribution of this test statistic is approximately Chi-Square with $(r-1)*(c-1)$ degrees of freedom, where r is the number of row categories and c is the number of column categories. The approximation becomes more accurate as the large size grows larger and larger (to infinity). Thus, if the sample size is "large enough", we can accurately approximate the test statistics distribution with this Chi-Square distribution. This is what is referred to by "large sample" or "asymptotic" statistics. Let's conduct the Chi-Square test on AOSI dataset. This is done by using **summary()** with the contingency table object (created by table() or xtab()). Note that you cannot have the row and column margins in the table object when conducting this Chi-Square test, as R will consider these marginals are row and column categories.

```
table_ex_xtabs <- xtabs(~Gender+Study_Site, data=aosi_data)
summary(table_ex_xtabs)
```

```
## Call: xtabs(formula = ~Gender + Study_Site, data = aosi_data)
## Number of cases in table: 587
## Number of factors: 2
## Test for independence of all factors:
##  Chisq = 2.1011, df = 3, p-value = 0.5517
```

We see that the p-value is 0.55, which is very large and under a threshold of 0.05 is far from significance. Thus, we do not have evidence to reject the null hypothesis that gender and study site are independent.

6.3.1.3 Fisher's Exact Test

An alternative to the Chi-Square test is Fisher's Exact Test. This hypothesis test has the same null and alternative hypothesis as the Chi-Square test. However, its test statistic has a known

distribution for any finite sample size. Thus, no distributional approximation is required, unlike the Chi-Square test and thus it produces accurate p-values for any sample size. To conduct Fisher's Exact Test, use the function **fisher.test()** from the stats package with the table or xtab object. The drawback to Fisher's Exact Test is that it has a high computation time if the data has a large sample size; in that case, the approximation from the Chi-Square is likely accurate and this testing procedure should be used. Let's run Fisher's Exact Test on the gender by site contingency table.

```
fisher.test(table_ex_xtabs)
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  table_ex_xtabs
## p-value = 0.553
## alternative hypothesis: two.sided
```

Due to large sample size, we see that the p-value is very close to the p-value from the Chi-Square test, as expected.