IPS9 in R: Inference for Categorical Data (Chapter 9)

Shukry Zablah (szablah20@amherst.edu) and Nicholas Horton (nhorton@amherst.edu)

January 19, 2019

Introduction and background

These documents are intended to help describe how to undertake analyses introduced as examples in the Ninth Edition of *Introduction to the Practice of Statistics* (2017) by Moore, McCabe, and Craig.

More information about the book can be found here. The data used in these documents can be found under Data Sets in the Student Site. This file as well as the associated R Markdown reproducible analysis source file used to create it can be found at https://nhorton.people.amherst.edu/ips9/.

This work leverages initiatives undertaken by Project MOSAIC (http://www.mosaic-web.org), an NSF-funded effort to improve the teaching of statistics, calculus, science and computing in the undergraduate curriculum. In particular, we utilize the `mosaic` package, which was written to simplify the use of R for introductory statistics courses. A short summary of the R needed to teach introductory statistics can be found in the mosaic package vignettes (http://cran.r-project.org/web/packages/mosaic). A paper describing the mosaic approach was published in the *R Journal*: https://journal.r-project.org/archive/2017/RJ-2017-024.

Chapter 9: Inference for Categorical Data

This file replicates the analyses from Chapter 9: Inference for Categorical Data.

First, load the packages that will be needed for this document:

```
library(mosaic)
library(readr)
```

Section 9.1: Inference for two-way tables

To recreate the dataset that was used in Example 9.1, we will use a combination of several `do()` calls and `rbind()`. This will allow us to create the observations with the specific attributes based on the counts that appear in the table.

We recreate it like this.

```
#Ex9.1
Instag <- rbind(
  do(298) * data.frame(Sex = "Men",  User = "No"),
  do(209) * data.frame(Sex = "Women", User = "No"),
  do(234) * data.frame(Sex = "Men",  User = "Yes"),
  do(328) * data.frame(Sex = "Women", User = "Yes")
)
head(Instag)
```

```
##   Sex User .row .index
## 1 Men   No    1      1
## 2 Men   No    1      2
## 3 Men   No    1      3
## 4 Men   No    1      4
## 5 Men   No    1      5
```

```
## 6 Men   No    1      6
```

We take a small peek of the dataset with the `head()` function that returns the first few observations from a given dataset. Some useful columns were returned with our dataset. You don't have to worry about them now.

We will get some tables that summarize the information displayed in Ex9.1. We can use the `tally()` function for this.

```
#Ex9.1
tally(User ~ Sex, data = Instag, margin = TRUE)
```

```
##        Sex
## User    Men Women
##   No    298   209
##   Yes   234   328
##   Total 532   537
```

```
tally(User ~ Sex, data = Instag, format = "proportion", margin = TRUE)
```

```
##        Sex
## User          Men     Women
##   No    0.5601504 0.3891993
##   Yes   0.4398496 0.6108007
##   Total 1.0000000 1.0000000
```

Now take look at Example 9.2 in page 526. To recreate that table of counts we simply have to call the `tally()` function and it will make the 2-way table for us. We call it like this:

```
#Ex9.2
tally(~ User + Sex, data = Instag, margins = TRUE)
```

```
##        Sex
## User     Men Women Total
##   No     298   209   507
##   Yes    234   328   562
##   Total  532   537  1069
```

The `margins = TRUE` option makes sure that `tally()` ouputs the convenient Total columns just like in page 527. To understand the difference between our last two `tally()` calls, look at the Total column of our tables.

Turn your attention to Example 9.3 now. After creating the dataset from the counts, we can use a similar `tally()` call to recreate the table and verify that our method to create the dataset is in fact accurate.

```
#Ex9.3
Vaccine <- rbind(
  do(729) * data.frame(Required = "Yes", Party = "Democratic"),
  do(479) * data.frame(Required = "Yes", Party = "Republican"),
  do(230) * data.frame(Required = "No",  Party = "Democratic"),
  do(258) * data.frame(Required = "No", Party = "Republican")
)

tally(~ Required + Party, data = Vaccine, margins = TRUE)
```

```
##         Party
## Required Democratic Republican Total
##    Yes          729        479  1208
##    No           230        258   488
##    Total        959        737  1696
```

Now we continue to explore our 2 way tables. In Example 9.5 we can see the marginal distribution of our Vaccine tables across political party preference. We recreate it with a call to `tally()` but this time we will use a new parameter too.

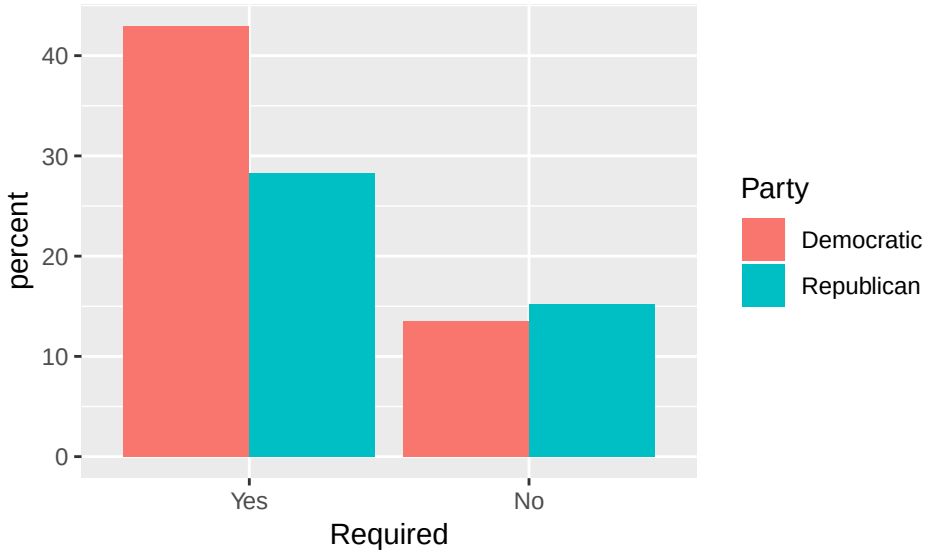
```
#Ex9.5
tally(Required ~ Party, data = Vaccine, margins = TRUE, format = "percent")
```

```
##          Party
## Required Democratic Republican
##    Yes     76.01668   64.99322
##    No      23.98332   35.00678
##    Total  100.00000  100.00000
```

The `format = "percent"` will nicely output the results in percentage form!

The output from `tally()` is good enough. However, a picture is worth a thousand words. Let's try to create a bar graph out of the Vaccine dataset.

```
#Ex9.6
gf_percents(~ Required, data = Vaccine, fill = ~ Party, position = "dodge")
```

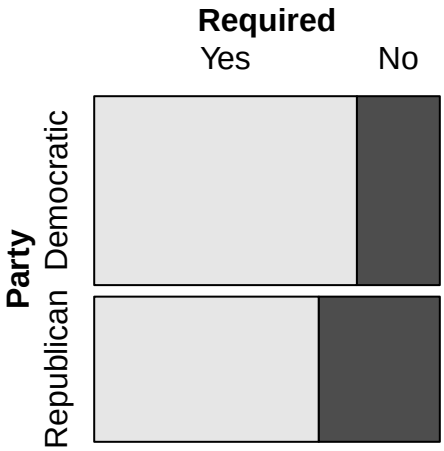


With the help of `gf_percents()` we can plot the percentage of each group (e.g. Democratic & "No") and compare them. This is a useful way to draw insights from two variables at once.

Note: This is not an equivalent bar graph but still provides the same useful information. The original bar graph in page 530 graphs the percentages across political party (i.e. adding both columns belonging to a party will give 100%).

Another way that we can visualize two categorical variables is to create a mosaic plot. We will use the `vcd` package's `mosaic()` function to plot the mosaic plot. Note the call resembles the same syntax of the `tally()` commands we made earlier.

```
#Ex9.7
vcd::mosaic(Required ~ Party, data = Vaccine, shade = TRUE)
```



Having multiple ways to visualize varibles will help you analyze your data more thoroughly and communicate your findings in a more intuitive way.

In Example 9.7 we are interested in getting the expected counts of our Vaccine data. In R you can take advantage of the `xchisq.test()` function and get the relevant output like this:

```
#Ex9.8 pg.533
chiSqVaccine <- xchisq.test(tally(Required ~ Party, data = Vaccine), correct = FALSE)
```

```
##
##  Pearson's Chi-squared test
##
## data:  x
## X-squared = 24.709, df = 1, p-value = 6.666e-07
##
##   729      479
## (683.06) (524.94)
## [ 3.09]  [ 4.02]
## < 1.76>  <-2.01>
##
##   230      258
## (275.94) (212.06)
## [ 7.65]  [ 9.95]
## <-2.77>  < 3.15>
##
## key:
##  observed
##  (expected)
##  [contribution to X-squared]
##  <Pearson residual>
```

```
with(chiSqVaccine, expected)
```

```
##         Party
## Required Democratic Republican
##      Yes   683.0613   524.9387
##      No    275.9387   212.0613
```

To understand what is going on in this code, break it down into its components. We are creating a variable called `chiSqVaccine` and we are assigning the output of the `xchisq.test()` call. The object stored in our variable will contain several useful fields as we will see. The first one is the expected values. To extract it from the object we use the `with()` function.

Note: We specify the `correct = FALSE` option to match the book's table. This option specifies that there should be no continuity correction applied to our test. You can see how the output changes by removing that option.

In a manner similar to the one above, we can get the observed counts we calculated with `tally()` before. We just retrieve the relevant field from our object with the `with()` function again.

```
#Ex9.8
with(chiSqVaccine, observed)
```

```
##          Party
## Required Democratic Republican
##      Yes        729        479
##      No         230        258
```

To see the output of the Chi-Square test discussed in Example 9.8 we just need to print the object we stored in our variable earlier.

```
#Ex9.8
chiSqVaccine
```

```
##
##  Pearson's Chi-squared test
##
## data:  x
## X-squared = 24.709, df = 1, p-value = 6.666e-07
```

All this useful features are already built into how R's `xchisq.test()` function works.

Note: There is an error in the χ^2 value in the book. While it showed the correct machine output, it specified the wrong χ^2 squared value.

We continue with Example 9.9 in page 537.

```
#Ex9.9
Health <- rbind(
  do(69) *  data.frame(PhysAct = "Low", FruitConsumption = "Low"),
  do(206) * data.frame(PhysAct = "Moderate", FruitConsumption = "Low"),
  do(294) * data.frame(PhysAct = "Vigorous",  FruitConsumption = "Low"),
  do(25) *  data.frame(PhysAct = "Low", FruitConsumption = "Medium"),
  do(126) * data.frame(PhysAct = "Moderate", FruitConsumption = "Medium"),
  do(170) * data.frame(PhysAct = "Vigorous", FruitConsumption = "Medium"),
  do(14) *  data.frame(PhysAct = "Low",  FruitConsumption = "High"),
  do(111) * data.frame(PhysAct = "Moderate", FruitConsumption = "High"),
  do(169) * data.frame(PhysAct = "Vigorous", FruitConsumption = "High")
)
```

You should already know what is happening in the code chunk above. We will store the dataset into a variable called Health.

Now we recreate the table in page 537 as follows:

```
#Ex9.9
tally(~ FruitConsumption + PhysAct, data = Health, margins = TRUE)
```

```
##                 PhysAct
## FruitConsumption  Low Moderate Vigorous Total
##           Low      69      206      294   569
##           Medium   25      126      170   321
##           High     14      111      169   294
```

```
##           Total   108      443      633  1184
```

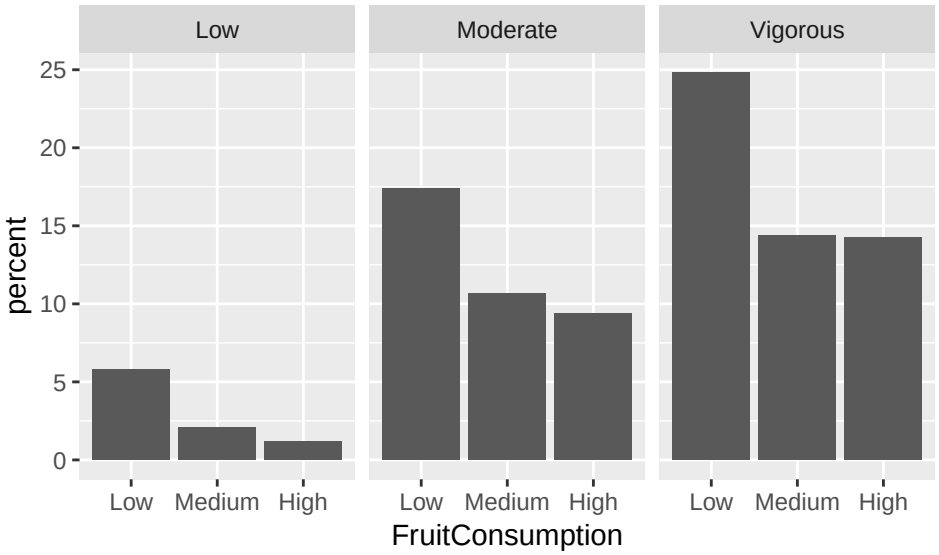
The table above is the 2 way table of counts from the Health data. To get the percentages instead we use the `format` parameter.

```
#Ex9.10
tally(~ FruitConsumption + PhysAct, data = Health, margins = TRUE, format = "percent")
```

```
##                 PhysAct
## FruitConsumption       Low   Moderate   Vigorous      Total
##           Low      5.827703  17.398649  24.831081  48.057432
##           Medium   2.111486  10.641892  14.358108  27.111486
##           High     1.182432   9.375000  14.273649  24.831081
##           Total    9.121622  37.415541  53.462838 100.000000
```

Again, visualizations will trump tables in all appropriate cases. We will create a faceted bar graph.

```
#Fig9.7
gf_percents(~ FruitConsumption | PhysAct, data = Health)
```



Note: These are not equivalent bar graphs to Figure 9.7, but they still provide the same useful information.

Now, let's get the expected counts for our Health data.

```
#Ex9.11
chiSqHealth <- xchisq.test(tally(FruitConsumption ~ PhysAct, data = Health), correct = FALSE)
```

```
##
##  Pearson's Chi-squared test
##
## data:  x
## X-squared = 14.152, df = 4, p-value = 0.006824
##
##    69      206      294
## ( 51.90) (212.89) (304.20)
## [5.6325] [0.2233] [0.3422]
## < 2.373> <-0.473> <-0.585>
##
##    25      126      170
## ( 29.28) (120.10) (171.62)
```

```
## [0.6257] [0.2895] [0.0152]
## <-0.791> < 0.538> <-0.123>
##
##    14      111      169
## ( 26.82) (110.00) (157.18)
## [6.1262] [0.0091] [0.8888]
## <-2.475> < 0.095> < 0.943>
##
## key:
##  observed
##  (expected)
##  [contribution to X-squared]
##  <Pearson residual>
```

```
with(chiSqHealth, expected)
```

```
##                 PhysAct
## FruitConsumption      Low Moderate Vigorous
##           Low    51.90203 212.8944 304.2035
##           Medium 29.28041 120.1039 171.6157
##           High   26.81757 110.0017 157.1807
```

And our observed counts...

```
#Ex9.11
with(chiSqHealth, observed)
```

```
##                 PhysAct
## FruitConsumption Low Moderate Vigorous
##           Low     69      206      294
##           Medium  25      126      170
##           High    14      111      169
```

And finally our χ^2 statistic.

```
#Ex9.11
chiSqHealth
```

```
##
##  Pearson's Chi-squared test
##
## data:  x
## X-squared = 14.152, df = 4, p-value = 0.006824
```

Remember these are all possible thans to the functionality of the `xchisq.test()` function.

Section 9.2: Goodness of fit

We will be using data of the ACT from six different states. We recreate our dataset from the counts.

```
#Ex9.13
ACT <- rbind(
  do(167) * data.frame(State = "AZ", label = 1),
  do(257) * data.frame(State = "CA", label = 2),
  do(257) * data.frame(State = "HI", label = 3),
  do(297) * data.frame(State = "IN", label = 4),
  do(107) * data.frame(State = "NV", label = 5),
```

```
  do(482) * data.frame(State = "OH", label = 6)
)
```

To get a sense of the number of participants in the study (pg 546) we can quickly do a `tally()` call on the State column.

```
#Ex9.13
tally(~ State, data = ACT, margins = TRUE)
```

```
## State
##    AZ    CA    HI    IN    NV    OH Total
##   167   257   257   297   107   482  1567
```

We will now import the population proportions from a csv file. We will use these values to see how close our sample counts are to the population values.

```
#Ex9.13
ACTPopProp <- read_csv("https://nhorton.people.amherst.edu/ips9/data/chapter09/EG09-13ACT.csv")
ACTPopProp
```

```
## # A tibble: 6 x 4
##   State Label Count  Prob
##   <chr> <dbl> <dbl> <dbl>
## 1 AZ        1   167 0.105
## 2 CA        2   257 0.172
## 3 HI        3   257 0.164
## 4 IN        4   297 0.188
## 5 NV        5   107 0.07
## 6 OH        6   482 0.301
```

We will use the same `xchisq.test()` function from before. It is important to note the new behavior we expect from the function when we provide a vector of the population proportions. The expected counts and the χ^2 value will depend on this new parameter.

```
#Ex9.13
chisqACT <- xchisq.test(tally(~ State, data = ACT), p = c(0.105, .172, .164, .188, .07, .301), correct =
```

```
##
##  Chi-squared test for given probabilities
##
## data:  x
## X-squared = 0.93084, df = 5, p-value = 0.9679
##
##   167      257      257      297      107      482
## (164.53) (269.52) (256.99) (294.60) (109.69) (471.67)
## [3.7e-02] [5.8e-01] [5.6e-07] [2.0e-02] [6.6e-02] [2.3e-01]
## < 0.19217> <-0.76286> < 0.00075> < 0.14006> <-0.25684> < 0.47578>
##
## key:
##  observed
##  (expected)
##  [contribution to X-squared]
##  <Pearson residual>
```

Now that we have saved our object, we can access the expected counts and the test statistic just like before.

```
#Ex9.13
with(chisqACT, expected)
```

```
##      AZ      CA      HI      IN      NV      OH
## 164.535 269.524 256.988 294.596 109.690 471.667
```

```
#Ex9.14
chisqACT
```

```
##
##  Chi-squared test for given probabilities
##
## data:  x
## X-squared = 0.93084, df = 5, p-value = 0.9679
```

Another example of a field included in the return value of the `xchisq.test()` is the `residuals` field. Let's take a look.

```
#Ex9.15
with(chisqACT, residuals)
```

```
## State
##            AZ            CA            HI            IN            NV
##  0.1921709671 -0.7628591106  0.0007485569  0.1400622312 -0.2568436078
##            OH
##  0.4757827403
```

And just like that, let R help you with most of the computations when you are analyzing your categorical variables.