R Lab #6c - Chi-square test

Contents

Chi-squared goodness-of-fit test: baby birth day example

Chapter 8 covers how we use the chi-square goodness-of-fit test to analyze the frequency of births across the different days of the week. Here there are seven categories (days), so the binomial test cannot be used. The null hypothesis is that the probability of birth is the same on every day of the week.

Importing data and creating a frequency table

First step is to import the data. Download the file "chap08e1DayOfBirth2016.csv" and read it into R:

```
births <- read.csv("chap08e1DayOfBirth2016.csv", stringsAsFactors = T)
```

Notice that the file has a single column/variable called "day" that is populated with 180 entries of days (e.g., Sunday, Monday). The next step is to convert these data into a frequency table that stores counts for each day. This is done with the command table().

```
birthsTable <- table(births$day)
birthsTable
##
##    Friday    Monday  Saturday    Sunday  Thursday   Tuesday Wednesday
##        38        26        20        14        27        34        21
```

Re-ordering groups

Notice that by default R puts the different groups into alphabetically order (i.e., Friday, Monday, Saturday, etc.). This is not a good ordering in this case, as it would be weird to display these data as a table or a graph like this.

First let's see how R is reading/interpreting this variable. This is done with str(), which gives the structure of an object.

```
str(births)
## 'data.frame':    180 obs. of  1 variable:
##  $ day: Factor w/ 7 levels "Friday","Monday",..: 1 2 5 6 1 1 6 4 2 5 ...
```

Starting with R v4.0, text like we have for births$day is interpeted as characters (chr). If you're using R v3 then you might see factor instead. Factor is like a categorical variable with different groups called "levels" in

R terminology.

A good thing about a factor variable is that it is easy to change the order of its levels. So whether you're dealing with chr or factor, you can use the command() factor to change the order of the days:

```
births$day <- factor(births$day, levels = c("Sunday", "Monday",
    "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"))
str(births)
## 'data.frame':    180 obs. of  1 variable:
##  $ day: Factor w/ 7 levels "Sunday","Monday",..: 6 2 5 3 6 6 3 1 2 5 ...
```

Now you can see that the variable has been set as a factor and the levels follow a Sunday, Monday, ... pattern.

Now we can re-do the frequency table:

```
birthsTable <- table(births$day)
birthsTable
##
##    Sunday    Monday   Tuesday Wednesday  Thursday    Friday  Saturday
##        14        26        34        21        27        38        20
```

Defining expected probabilities

We now have the observed frequency for each group, and now we need to define the expected probabilities for each group. Note that the chi-square test needs probabilities as opposed to counts. This is analogous to the "Proportion of Days in 2016" column in Table 8.1-2.

In this case we know the total number of days in the year 2016 was 366, and we know the number of Sundays, Mondays, etc. in the year. So the probability for each day is the number of the particular day divided by 366.

Importantly, the *order of the probabilities needs to exactly match the order of the observed frequencies*. Since we changed our observed data to Sunday, Monday, Tuesday, etc. then the probabilities need to follow the same order.

```
c(52, 52, 52, 52, 52, 53, 53)/366
## [1] 0.1420765 0.1420765 0.1420765 0.1420765 0.1420765 0.1448087 0.1448087
```

Note that since these outcomes are mutually exclusive that these probabilities shoud sum to 1.

```
sum(c(52, 52, 52, 52, 52, 53, 53)/366)
## [1] 1
```

Running the chi-square test

Now we have the observed frequencies and the expected probabilities, so we are ready to run the chi-square test. The command is chisq.test() and takes two arguements: 1) observed frequencies and 2) expected probabilities (marked with p=.

```
chisq.test(birthsTable, p = c(52, 52, 52, 52, 52, 53, 53)/366)
##
##  Chi-squared test for given probabilities
##
## data:  birthsTable
## X-squared = 15.795, df = 6, p-value = 0.0149
```

The P-value is reported as 0.0149, which is below an alpha of 0.05. Therefore, we can reject the null hypothesis that births are equally likely on any day of the year. You can then go back to the data to see which groups have the biggest discrepancy between the observed and expected to infer the significant alternative.

Graphing frequencies

For this type of analysis it would be useful to make a grouped bar plot of observed and expected frequencies.

First, we need to make a data frame with day, observed frequency, and expected frequency.

The variable birthsTable has day and frequency (but in a table format). Now let's create a vector of expected frequencies. Note this is different than the expected probabilities created above. We need the expected probability for each day multiplied by the sample size of the study, 180.

```
expFreq <- (c(52, 52, 52, 52, 52, 53, 53)/366) * 180
expFreq
## [1] 25.57377 25.57377 25.57377 25.57377 25.57377 26.06557 26.06557
```

Now we can merge birthsTable and expFreq into a single data frame:

```
ObsExp <- data.frame(birthsTable, expFreq)
ObsExp
##        Var1 Freq  expFreq
## 1    Sunday   14 25.57377
## 2    Monday   26 25.57377
## 3   Tuesday   34 25.57377
## 4 Wednesday   21 25.57377
## 5  Thursday   27 25.57377
## 6    Friday   38 26.06557
## 7  Saturday   20 26.06557
```

Notice that ObsExp is a data frame in which the first column/variable is called Var1 and has the day of week, the second variable is called Freq and has the observed frequencies, and the third variable is called expFreq and has the expected frequencies.

The labels of the first two columns were created by R by default. We could clean this up and make the labels more relevant with the command setNames():

```
ObsExp <- setNames(ObsExp, c("day", "obsFreq", "expFreq"))
ObsExp
##         day obsFreq  expFreq
## 1    Sunday      14 25.57377
## 2    Monday      26 25.57377
## 3   Tuesday      34 25.57377
## 4 Wednesday      21 25.57377
## 5  Thursday      27 25.57377
## 6    Friday      38 26.06557
## 7  Saturday      20 26.06557
```

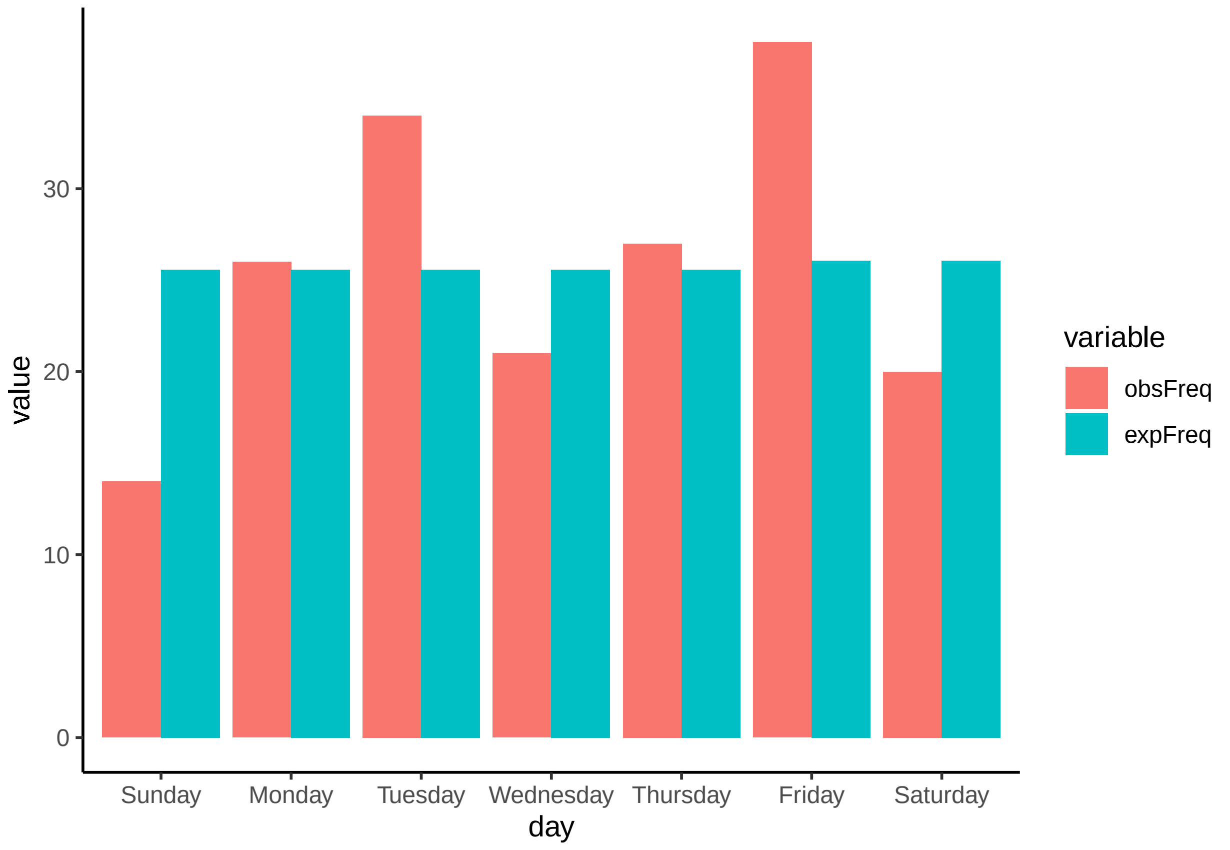
Now we are almost ready for a grouped bar plot. The remaining issue is that our values are split into two different columns, and to make a group bar plot we need them in one column. This can be done using a new command melt(), from the package reshape2. You will have to install the package first with install.packages("reshape2",dependencies=T). Then you can run melt()

```
library(reshape2)
ObsExp <- melt(ObsExp)
## Using day as id variables
ObsExp
##          day variable    value
## 1     Sunday  obsFreq 14.00000
## 2     Monday  obsFreq 26.00000
## 3    Tuesday  obsFreq 34.00000
## 4  Wednesday  obsFreq 21.00000
## 5   Thursday  obsFreq 27.00000
## 6     Friday  obsFreq 38.00000
## 7   Saturday  obsFreq 20.00000
## 8     Sunday  expFreq 25.57377
## 9     Monday  expFreq 25.57377
## 10   Tuesday  expFreq 25.57377
## 11 Wednesday  expFreq 25.57377
## 12  Thursday  expFreq 25.57377
## 13    Friday  expFreq 26.06557
## 14  Saturday  expFreq 26.06557
```

Notice that we now have a data frame where the days are repeated: one set for obsFreq and the other for expFreq. Also notice that the column names are day, variable, and value.

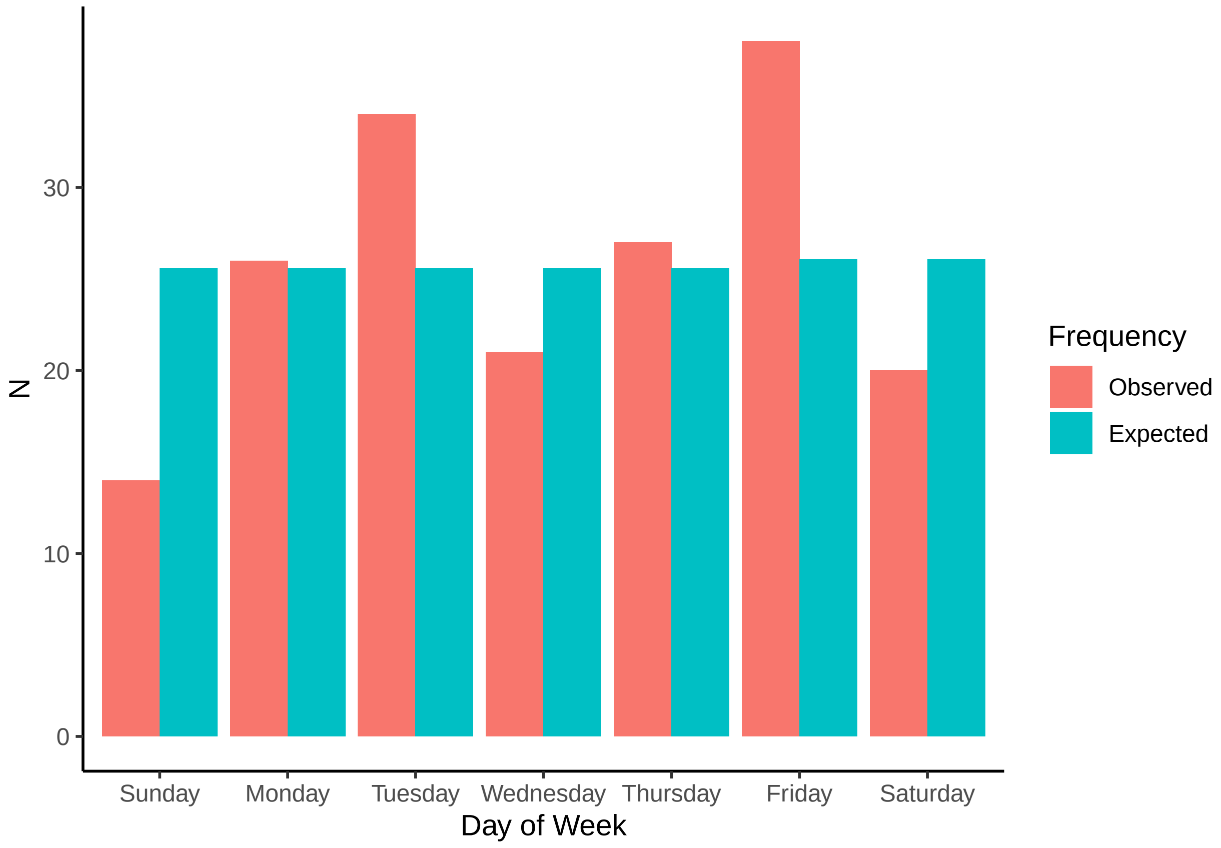
Now we are ready to try the plot. The first command will be ggplot() as usual, but for aes() we will have x (day), y (value), and fill (variable). The new fill option gives the column name to use to split values into different groups. The second command will be geom_bar(). We will need stat="identity" because we are supplying the y values, and we also need position=position_dodge(). The latter tells R to "dodge" the bars and make them side-by-side rather than the default of stacked on top of each other.

```
library(ggplot2)
ggplot(ObsExp, aes(x = day, y = value, fill = variable)) + geom_bar(stat = "identity",
    position = position_dodge()) + theme_classic()
```



Not bad! We could further tweak things a bit by making the axis and legend labels a bit better. We already know how to use xlab() and ylab() for the axis labels. For the legend title and labels we can use scale_fill_discrete():

```r
ggplot(ObsExp, aes(x = day, y = value, fill = variable)) + geom_bar(stat = "identity",
    position = position_dodge()) + theme_classic() + xlab("Day of Week") +
    ylab("N") + scale_fill_discrete(name = "Frequency", labels = c("Observed",
    "Expected"))
```



Now you can clearly visualize that the sample has an excess of births on Tuesday and Friday and a deficiency of births on Saturday and Sunday.

R commands summary

- **Table of observed frequencies**
  - table(factor)
- **Chi-squared goodness-of-fit test**
  - chisq.test(observed_freqs,p=expected_proportions)
- **Create data frame**
  - data.frame(data)
- **Set names on an object**
  - setNames(object)
- **Convert object into molten data frame**
  - melt(data)
- **ggplot2 bar plot**
  - geom_bar()