STAT 113 Lab 2: Visualizing and Describing Variables

Colin Reimer Dawson

Last Revised September 7, 2017

What to Turn In

Only the exercises at the end labeled "Homework" need to be written up and turned in. The interspersed exercises are just there for you to do as you are working through the packet.

Log in to the RStudio Server (Skip if you're using your own copy of R/RStudio)

Instructions

1. In your web browser, visit `rstudio.oberlin.edu`.
2. Your username is your Obie ID. You should have changed your password last week, but if not it is also your Obie ID. See Lab 1 for instructions on changing it.

1 Describing and Visualizing Data

In this section we will look at how to create some basic visualizations and get some basic summary statistics in R.

As you work through the commands listed in this packet, I suggest typing each line into an R script so that you can come back and refer to it later:

Under the File menu, select New File, and choose R Script. used to create new files. Click here and select "R Script".

Remember that you can click "Run" to execute the current line or the highlighted region, and you can click "Source" to run the whole file (or press Cmd-A (Ctrl-A on Windows) to highlight the whole thing and press Cmd-Enter (Ctrl-Enter) to Run the selection).

Remember that any code preceded on the same line by a pound sign (#) is interpreted as a comment, and will be ignored when running that line or the whole script. It is a good idea to add comments to your script as you go, to remind yourself what each line is doing.

1.1 Warmup

Exercise 1 Examine the documentation for the `EmployedACS` dataset from the `Lock5Data` package, including the description of the values of each variable. Which ones are quantitative? Which are categorical? Which are binary?

1.2 Visualizing a Quantitative Variable

We are going to be using functions from the `mosaic` package, as well as using data from the `Lock5Data` package, so first load these packages into your R session.

```
library("mosaic")    # Needed for the various functions
library("Lock5Data") # Needed for the EmployedACS data
```

Now, load the `EmployedACS` dataset

```
data("EmployedACS")  # Loading the dataset
```

R Tip As a rule, any time you start a new script, you should load any libraries you need with `library()` at the top of the file. You should also load any datasets you need (with `data()` for built-in datasets, or `read.file()` for data that's in a file). Each of these lines must be present before any commands that try to access the functions or datasets in question, or you will get an error. If you have *installed* new packages (using `install.packages()`), you do *not* need to, nor should you, do this again, as the package now exists in your system.

For a quantitative variable like `HoursWk` (Hours worked per week), we can get a general sense of what values are in the data using a **dot plot** or a **histogram**.

```
## note the capital 'P' in dotPlot.  There is unfortunately another
## dot plot function with a lowercase p that works differently.
## This is a consequence of different
## people independently developing R packages.  The one we want is part of
## the mosaic package.
dotPlot(~HoursWk, data = EmployedACS)
```

You should see the plot appear in the lower-right "Plots" pane. Examine the plot. This is the same type of plot that we created in class for the Gettysburg Address sampling worksheet: dots represent cases, and they are stacked on an axis representing hours worked.

Let's examine the structure of the command you just typed. The first argument here (with the $\sim$ in front) is a "formula", with the name of the variable we are interested in plotting appearing on the right hand side of the tilde ($\sim$) symbol. We will see this format appear over and over again.

We will also see the `data=` argument appear over and over again. This indicates what data frame the variable we are interested in is in.

Likely the stacks of dots are rounded off at weird intervals, and the dots are too small to really see them individually, by default. We can modify the `dotPlot()` command to control the rounding that gets done and the size of the dots with the `width=` and `cex=` arguments, respectively.

```
dotPlot(~HoursWk, data = EmployedACS, width = 5, cex = 3)
```

This says to group the dots in bins spanning a range of 5 hours, and to magnify the default dot size by a factor of 3. Alternatively, we could use `nint=` instead of `width=` to control the *number* of bins instead of the width of each bin.

Alternatively, we could create a **histogram**:

```
histogram(~HoursWk, data = EmployedACS, type = "count")
```

You should again see a horizontal axis representing number of hours worked, but now instead of individual dots, there are bars. The height of each bar represents the number of cases in a particular range.

Note that the command has the same structure as `dotPlot()`, except that the function name is different, and there is an additional named argument, `type=`, which controls what we plot on the y-axis: here, the number of observations in each bin, or set of values of the quantitative variable. Other options are available to label the y-axis with proportions of the total instead of counts, for example.

The same options to control the bin width or number apply to histograms as well.

Optional: Controlling Color The default plot colors are kind of ugly, in my opinion. You can control the color of the dots or bars, etc. using the `col=` argument. For example:

```
histogram(~HoursWk, data =  EmployedACS, width = 5, col = "forestgreen")
```

Type `colors()` at the console to see a (long) list of available colors. Alternatively you can use a predefined color palette and then refer to colors by number. For example, try this:

```
library("RColorBrewer")  # a package with some decent palettes
## Sets the palette to the first 8 colors in "Set1"
## Type ?brewer.pal to see other palette options
palette(brewer.pal(n = 8, name = "Set1"))
```

Now if we give colors by number it will use the palette:

```
histogram(~HoursWk, data = EmployedACS, width = 5, col = 3)
```

Exercise 2 Before moving on, estimate the mean and median number of hours worked by looking at the histogram and/or the dot plot.

We can also create a **box-and-whisker plot** for a quantitative variable, which

depicts the median, the **first and third quartile** (the value that has 25% of the cases below it, in other words the 25th percentile, or the "median of the data below the median", and the value with 25% of the cases above it, in other words the 75th percentile, or the "median of the data above the median") as the edges of the box, and draws "whiskers" which are 1.5 times the width of the box. Points beyond the whiskers, identified as potential **outliers** are plotted individually. We'll talk about all this more formally in class next week.

```
### Again, be careful to use the function name exactly as written.
### There is an alternative box plot function called boxplot() that
### works slightly differently.
bwplot(~HoursWk, data = EmployedACS)
```

Exercise 3 Estimate the first and third quartile of number of hours worked per week from the box-and-whisker plot.

1.3 Summarizing a Quantitative Variable

We can get a set of useful summary statistics for our `HoursWk` variable using the `favstats()` function. The syntax is the same as for `histogram()`, in that we have a formula and a `data=` argument.

```
favstats(~HoursWk, data = EmployedACS)
```

Find the mean and median in the results. Were your estimates from above close? How about Q1 and Q3 (the first and third quartile)?

We can get individual summary statistics using the same format. For example,

```
mean(~HoursWk, data = EmployedACS)
median(~HoursWk, data = EmployedACS)
sd(~HoursWk, data = EmployedACS)      # Standard Deviation
var(~HoursWk, data = EmployedACS)     # Variance
```

If you don't know what standard deviation and variance are, don't worry; we will talk about them next week too.

1.4 Summarizing a Categorical Variable

Mean, median, etc. do not make sense for a categorical variable, since we cannot add the values together, and they are not ordered. Instead, we can summarize the variable with a frequency table:

```
tally(~Race, data = EmployedACS)
```

Or, we can get a relative frequency table by specifying `format = "proportion"`:

```
tally(~Race, data = EmployedACS, format = "proportion")
```

Exercise 4 What is the total percentage of the sample that is non-white? Preferably, use R as your calculator. (See the tip below for a way to do this "programmatically" if you are interested.)

Optional R Tip Sometimes when using commands like `tally()` that return multiple values at once, we want to be able to access specific entries as part of another calculation. We can do this by **indexing** into the variable (which is represented as an "array", or a list of values). First, let's store the results of `tally()` as a variable, so we can work with them:

```
race.stats <- tally(~Race, data = EmployedACS, format = "proportion")
```

We can access individual entries of this table (array) either by position or by label using square brackets after the name of the table. For example, if I want the first entry, I can do

```
race.stats[1]
```

(Coders: note that `R` uses indices that start with 1, not 0 like Python and many other languages.)

I can also access entries by their name, for example:

```
race.stats["asian"]
```

I can even access multiple entries at once. If I want consecutive positions, I can use a colon:

```
race.stats[1:3]
```

If I want arbitrary positions, or a set of labels, I can use the `c()` function to "combine" a list of entries separated by commas. For example:

```
race.stats[c(1,2,3)]
race.stats[c("asian", "black", "other")]
```

I can then use these values in arithmetic expressions, or pass them to functions such as `mean()`, `sum()`, etc.

1.5 Visualizing a Categorical Variable

We can plot counts or proportions using a **bar graph**.

```
## As is often the case, there are other bar graph functions with similar names.
## Be sure to type the name exactly as given.
bargraph(~Race, data = EmployedACS)
```

Or, with proportions:

```
bargraph(~Race, data = EmployedACS, type = "proportion")
```

(Note that the argument controlling the y-axis type has a different name for `bargraph` than it does for `tally`, annoyingly.)

Exercise 5 Create a bargraph showing the proportions of cases that do and do not have health insurance. Is there anything unsatisfying about the graph if you were going to put it in an article or paper?

Modify your graph by adding a label for the x-axis using the `xlab=` argument. Put the desired label in quotes as the argument value so the reader knows what the bars represent.

1.6 Visualizing Two-Way Contingency Tables

If we have a contingency table with two categorical variables, we can visualize it in a couple of ways. One is to use a grouped bar graph:

```
## Counts version
bargraph(~Race, groups = Sex, data = EmployedACS, auto.key = TRUE)
## auto.key = TRUE includes a key for the groups variable
## Note that groups= gives the variable to be distinguished by color
## whereas the variable after the ~ controls the 'clustering' variable

## Conditional proportion version
bargraph(~Race, groups = Sex, data = EmployedACS,
         auto.key = TRUE, type = "proportion")
## Note: proportions are computed within each level of the groups
## variable, not within clusters.
## Note: you can always break up commands into multiple lines for
## readability; if a line is incomplete R will wait for you to finish
## the command on the following line.
## Sometimes you will accidentally leave a line incomplete and see a
## + sign in the console.  If this happens, hit 'Esc' to abort the command.
```

Another option is a "mosaic plot", which takes the two way table and draws a rectangle for each cell whose area is proportional to the amount of data at that combination.

```
## col= controls color.  If you read the optional tip on
## controlling color, you may want to use it here
mosaicplot(Race ~ Sex, data = EmployedACS, col = c("forestgreen","darkorange"))
```

2 Summarizing and visualizing subsets of the data

2.1 Filtering and Piping

Sometimes we will want to analyze or visualize only some of the cases in the data. For example, suppose we want to look at a histogram of weekly hours worked only for female workers.

We can produce a "reduced" data frame with the `filter()` function, as in the following:

```
filter(EmployedACS, Sex == 0)
```

The first argument is the name of the full dataset, and the second is a "filter" condition. Note the double equals sign. This is used for a comparison, to distinguish from setting the value of an argument. If you try to use a single `=` you will get an error to the effect that there is no argument called `Sex`. (If we have a quantitative filtering variable, we can also define filters using `>` or `<` for greater-than or less-than, or `>=` or `<=` for greater-than-or-equal-to and less-than-or-equal-to).

The above command on its own will just print out the filtered dataset, just as if we had typed `EmployedACS` by itself. What we really want is to use the result of this function as the value of the `data=` argument of whatever plotting or summary function we want.

Essentially, we want to "chain" or "nest" the two commands.

We have three options, which are all equivalent; you should use whichever one you find most natural (this may depend on context).

1. Option 1 is to directly nest the two commands, substituting the whole `filter()` command directly into the `data=` argument for `histogram()`, say:

   ```
   histogram(~HoursWk, data = filter(EmployedACS, Sex == 0))
   ```

2. This can get messy if we have complicated filters; so another option is to "save" the filtered data as a new object and then use that in `histogram()`.

   ```
   EmployedACS.Females <- filter(EmployedACS, Sex == 0)
   histogram(~HoursWk, data = EmployedACS.Females)
   ```

3. A final option is to "chain" the commands together using the 'pipe' operator, which is written like `%>%`. This works like this:

   ```
   filter(EmployedACS, Sex == 0) %>%
       histogram(~HoursWk, data = .)
   ```

   where the `.` is a stand-in for the output of the "pipe". You can interpret this command as saying "first filter `EmployedACS`, keeping only those cases where `Sex == 0`, then use the resulting filtered data and create a histogram of the `HoursWk` variable.

The same structure works if we want a mean or variance or whatever else from the filtered data.

Again, all three of these approaches are equivalent ways of doing the same thing: we first use `filter()` to get only certain cases from the data, and then use `histogram()` to plot the variable of interest for only those cases.

2.2 Plotting and Summarizing by Group

If we want to create a plot or compute summary statistics for one variable for *every* different value of another, we can modify our formula expression so that the name of the variable we are summarizing is on the left of the ~, and the grouping variable is on the right:

```
bwplot(HoursWk ~ factor(Sex), data = EmployedACS)
```

The `factor()` function is used to make sure R knows that the grouping variable is categorical, even if it is represented in the data using a number (as it is here, as 1 or 0). In some contexts it can figure this out on its own, but it's often good practice to tell it explicitly anyway. (It makes it clearer for someone reading your code too.)

Exercise 6 Compute the mean income for workers with and without health insurance. Do this in two ways: first using `filter()` to select each subset and find the mean on each filtered dataset, and second using a grouping formula to find both means at once.

3 Homework

Write up answers to the following problems in an Rscript. Be sure that your verbal answers are 'commented out' so that the entire Rscript will run.

(1) Create a dot plot or histogram, and a box plot of the `Income` variable from the `EmployedACS` dataset. Estimate the mean and median. You do not need to upload plots, but include the code you used to create them.

(2) Filter out any cases with income above half a million dollars ($500K). What percentage of cases remain? (Hint: we saw a function last week that counts the number of cases/rows in a dataset. Use this to count the cases in both the filtered and unfiltered data.) How does this filtering affect the mean? How does it affect the median? How does it affect the standard deviation? How does it affect the **interquartile range** (the difference between the first and third quantiles). You can use the `IQR` function to get this value, or just subtract the quartiles returned by `favstats()`

(3) Produce side-by-side boxplots for income grouped by whether the person has health insurance (you will want to use `factor()` to tell R to treat the grouping variable as categorical). Suppose a new person (not in this dataset) filled out the survey and reported income of $80K but neglected to indicate whether they have health insurance. Based on the side-by-side boxplots, would you guess that they have health insurance or not? How confident would you be? Explain.