Lab 3: Exploring And Manipulating Data in R

#Exploring and Graphing Birth Data

This is a lab designed to learn how to explore data in R. The data are from the fivethirtyeight package and describe how many births are on a given day. It builds on the material learned in chapter 4, but also adds a bit of information about working with data in R and graphing.

When you complete this lab, you should know the following. A lot of this, such as the aggregate command, is review from the last lab:

1. How to install and load packages into R
2. How to view a dataframe and see the types of variables in a dataframe
3. How to select rows and columns in a dataframe using row-column notation and selecting a variable using the dollar sign notation
4. How to use comparisons and `filter()` to select a subset of data
5. How to generate summary statistics using the `aggregate()` command

First, go ahead and make a new R notebook. Save your R notebook in its own folder. The data from this lab will come from a package, so there is no dataframe or dataset to download.

To complete this lab, we have to install two packages, one of which contains the data and another which contains extra commands for R. As discussed in Chapter 4 of the textbook, packages are add-ons to R that allow us to do extra things. Some packages have extra functions we might want to do, like the tidyverse package, whereas other packages contained built-in datasets to play with. You only have to install the packages once, so if you've installed these packages before, you don't have to do this. However, if you mistakenly try to reinstall a package, it isn't a problem.

The tidyverse package is a set of packages that go together in order to manipulate and visualize data. It includes a series of data manipulation tools that we will use in this lab.

There are two ways to install a package. The first way is to click Tools and then Install Packages and type the package you are interested in installing. In this case, yu should type the `tidyverse` package and the `fivethirtyeight` package. Once you install a package, it is on your computer. You do not have to reinstall it. However, every time you start R, you need to load the packages you use, and you need to load them in every notebook where you use them.

We can load them by using the library() command. Type the following in a code chunk to load the packages you installed.

```
library(fivethirtyeight)
library(tidyverse)
```

The data for this lab is contained in the `fivethirtyeight` package. The fivethirtyeight package is a package which has a lot of datasets built in. The birth data is built into the package with the name US_births_2000_2014. Since that is long to type, you can go ahead and add the data to your workspace as the dataframe `birth`. Type the command below:

```
birth = US_births_2000_2014
```

This dataframe has several columns. Here are the columns of interest. To view the data, you can select it from your Environment

- year: Year
- month: Month
- date_of_month: Day
- day_of_week: Abbreviation of day of week
- births: Number of births

When we view data, we also want to know what types of data are in each row. To see the types of data in each column, we can use the `str()` command:

```
str(birth)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    5479 obs. of  6 variables:
##  $ year         : int  2000 2000 2000 2000 2000 2000 2000 2000 2000 2000 ...
##  $ month        : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ date_of_month: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ date         : Date, format: "2000-01-01" "2000-01-02" ...
##  $ day_of_week  : Ord.factor w/ 7 levels "Sun"<"Mon"<"Tues"<..: 7 1 2 3 4 5 6 7 1 2 ...
##  $ births       : int  9083 8006 11363 13032 12558 12466 12516 8934 7949 11668 ...
```

From this output, we can see the classes of each variable. Year, month, date_of_month, and births are all "int" meaning integers. Date is a "Date" format, which means it is a special type of variable that holds dates. R has a series of packages designed to deal with all the problems involved with using dates. The last variable, "day_of_week" is a "Ord.factor" or ordinal factor with 7 levels. What this means is that it is a string variable that is also a factor. However, the variable is ordinal. In this case, the factors that are greater are the ones later in the week.

Why this is important is because you may need to know how to address all the data. Note that the variables in day_of_week are text. To enter text into R, we have to put it in quotes. This will be relevant later in this lab.

Selecting parts of data

When we have a long data frame, we may want to select parts of that data frame. There are a couple of ways of doing this.

Using row-column notation

Row-column notation allows us to select a part of a data frame by entering which rows and columns we want in brackets. To do this, we enter the data frame name, then brackets with row and column. For instance, to select the 5th row of the 4th column in the birth data, we would type:

```
birth[5,4]
```

```
## # A tibble: 1 x 1
##   date
##   <date>
## 1 2000-01-05
```

Go ahead and type these examples in your R notebook. You should also add annotations where you list what each of these examples should be doing.

The row-column notation can be powerful if we want to select several different rows or columns. To select a series of rows or columns, we write the first one and last one, separated by a colon. So if we want to select rows two through four, we would type 2:4. For instance, if we wanted to select rows 3:5 and columns 2:5 of our birth data, we would type:

```
birth[3:5,2:5]
```

```
## # A tibble: 3 x 4
##   month date_of_month date       day_of_week
##   <int>         <int> <date>     <ord>
## 1     1             3 2000-01-03 Mon
## 2     1             4 2000-01-04 Tues
```

```
## 3     1             5 2000-01-05 Wed
```

The tibble that this outputs has 3 rows and 4 columns. Note that the rows are listed in the output as rows 1, 2, and 3. This numbering is just listing the rows in the output, not the actual rows from the dataframe itself. If you look at the dates from this output, you can see that the data selected are rows 3-5 from the data, because it is the third through the fifth day of the year (Jan 3 - Jan 5).

If we want to select all the rows, we just omit a number, leaving all that part blank. So if we want to select all rows and column 4, we would type `[,4]`. Or if we want to select all columns and row 5, we would type `[5,]`. This is useful because selecting all columns and one row allows us to select one single observation. So if we want to see births on January 1, 2000, which is row 1, we would type:

```
birth[1,]
```

```
## # A tibble: 1 x 6
##    year month date_of_month date       day_of_week births
##   <int> <int>         <int> <date>     <ord>        <int>
## 1  2000     1             1 2000-01-01 Sat           9083
```

Likewise, if we wanted to see all the values for year, column 1, we could type:

```
birth[,1]
```

Note since there is a lot of data, the R notebook cuts off this output.

Another way to select a column is by using the column's name and the dollar sign, where you select a column by giving the data frame name and then the column name separated by a dollar sign ($). To select the column year from the data frame birth, just like in the example above, you would type:

```
birth$year
```

Finally, it can be helpful to select a series of rows or columns that are not contiguous. For instance, you can select rows 3 through 7 by typing birth[3:7,]. However, you may want to select rows 3, 6, and 8. You can do this by using the `c()` command and listing the row numbers in the `c()` command.

```
birth[c(3,6,8),]
```

```
## # A tibble: 3 x 6
##    year month date_of_month date       day_of_week births
##   <int> <int>         <int> <date>     <ord>        <int>
## 1  2000     1             3 2000-01-03 Mon          11363
## 2  2000     1             6 2000-01-06 Thurs        12466
## 3  2000     1             8 2000-01-08 Sat           8934
```

Questions: Create some code chunks to answer the following, and make sure you use annotations to tell me what question you're answering. 1. What would you type to select the day_of_week column? There are two possible ways to do this?

2. How would I select rows 21 through 92 and all columns?
3. How would I select the data for rows 3, 7, 24, and 97 and all columns?
4. How would I select rows 2-9 and columns 2 and 4?

Using comparisons and `filter()` from the tidyverse package

We can use comparison functions to select data based on whether data fits a certain condition. The `subset()` function in R does this well, but we can also use the `filter()` function from the tidyverse package (which includes the dplyr package that has the filter command). The filter function makes it easier to selet data

using multiple conditionals. The `filter()` command has two parts. The first thing we enter is the name of the data frame, then a comma, and then a list of one or more conditionals, each separated by commas. A conditional is a variable and then an equals, greater than, less than, or not equal (!=) some values. A few examples are below. Go ahead and type them and see what is outputted:

```
filter(birth, year == 2012)
filter(birth, day_of_week == "Mon", month < 4, year > 2010)
filter(birth,births > 10000, month != 11)
```

The `filter()` command selects data which fit each of the comparisons. For instance, the second row above selects all Mondays in months 1-3 where the year is after 2010. For a row to be selected, it has to fit every one of the conditional statements.

Conditionals are very powerful because they allow us to select parts of our data. For instance, if I wanted to create a new data frame called Monday which only has rows corresponding to Mondays, I would type:

```
Monday = filter(birth, day_of_week == "Mon")
```

Note that I put "Mon" in quotes because day_of_week is a string. I can also use multiple conditionals. If I wanted to select all the rows for birthdays on July 4th, I would type:

```
filter(birth, month == 7, date_of_month == 4)
```

We can also use the filter command to calculate statistics on a subgroup of the data. If we want to know the mean number of births on Friday, we could do the following:

```
Friday = filter(birth, day_of_week == "Fri")
mean(Friday$births)
```

```
## [1] 12596.16
```

Note in this example, I created a new dataframe named Friday and then calculated the mean of the births variable in the Friday dataframe. An easier way to do this may be using the `aggregate()` command we covered in the last lab and covered again below.

Now go ahead and make some code chunks in your notebook to answer the following questions, using the filter command:

5. How would I calculate the mean number of births in March?
6. How would I calculate the standard deviation of the births on Wednesday (standard deviation is the `sd()` command)?

Distributions

When we look at data, we often want to see what kind of distribution the data has. This allows us to see whether we have outliers or any other issues. Creating a histogram in R is very simple, using the hist() command.

To see the distribution of birth data, type:

```
hist(birth$births)
```

What does this distribution look like? Is it unimodal, bimodal, etc? Is it skewed?

Sometimes, we may want to look at different distributions. One idea is that weekends will have a different pattern of births than weekdays because scheduled births will rarely be on weekends. Lets create two dataframes, one for weekdays and one for weekends and then look at the distribution for each.

Creating a dataframe which only contains the weekdays or the weekends can be a bit confusing because it is hard to use a single comparison. However, we can select data that fit one comparison OR another comparison

by separating each comparison using the vertical bar (hold shift and press the backslash (\) key, which should be the key above the Enter key).

This first command selects the variables where day_of_week equals "Sat" OR "Sun", so selecting the variables corresponding to the weekend.

```
weekend = filter(birth, birth$day_of_week == "Sat" | birth$day_of_week == "Sun")
```

This command selects the variables where day_of_week is equal to one of the five weekdays.

```
weekday = filter(birth, birth$day_of_week == "Mon"|
                   birth$day_of_week == "Tues" | birth$day_of_week == "Wed" |
                   birth$day_of_week == "Thu" | birth$day_of_week == "Fri" )
```

Now let's do a histogram for the variable births in each of these dataframes:

```
hist(weekday$births)
hist(weekend$births)
```

Make sure you include these histograms in your R notebook. In your annotations, answer the following question: What is different about each of these distributions, when comparing this to the histogram which contained all the data?

7. Create a histogram to visualize births in January. Use the filter command to select January and then create a histogram on that subset?

Aggregating Data

In the last lab

The `aggregate()` function in R allows us to take a function and calculate a statistic separately for each group, like calculating the mean for each day of the week.

Here is how we would calculate the mean births for each day of the week:

```
aggregate(births~day_of_week, birth, mean)
```

```
##   day_of_week    births
## 1         Sun  7518.377
## 2         Mon 11897.830
## 3        Tues 13122.444
## 4         Wed 12910.766
## 5       Thurs 12845.826
## 6         Fri 12596.162
## 7         Sat  8562.573
```

Like discussed in the last lab, we type three parts into the `aggregate()` function. The first part is called a "formula" and is a breakdown of which variables we want to use. We have to enter two or more column names here. The first column is the one that contains the Y variable or the variable that has the data we are interested in. This would be births in this example. Then we type a tilde (~) and then one or more grouping, or X variables, that we want to use to break down the data.

To look at births broken down by day of the week, we would type "births~day_of_week".

The second part is the data frame we are interested in, and the third part is which funciton we want to use to aggregate. We could look at sums, means, medians, standard deviations, and so forth. Means are probably most relevant here. Putting this all together, we would type:

So we see here, more babies are born during the week than during the weekend. We can also look at median, instead of mean, by changing the function at the end of `aggregate()`. This shows a similar pattern.

```
aggregate(births~day_of_week, birth, median)
```

```
##   day_of_week  births
## 1         Sun  7512.0
## 2         Mon 12087.0
## 3        Tues 13198.0
## 4         Wed 12932.0
## 5       Thurs 12958.0
## 6         Fri 12632.5
## 7         Sat  8574.0
```

Question: how would you look at the mean separated for each month (January, February, etc)? Which month has the most births?

Question: why would we want to look at a median versus a mean?

Here is a little more about the aggregate function that we did not discuss in the last lab. We can also have more than one grouping variable. For instance, we may want to see if the pattern for day of the week is different for each year. We can add more than one variable after the tilde in the first part of the aggregate formula by putting a plus sign between the variables. This gives a much larger data frame.

```
aggregate(births~day_of_week+year, birth, mean)
```

When you type this, what do you get?

Final problems:

Using what you know, create code chunks which do the following questions, and annotate what you are doing.

1. Select only those people born in October in a new dataframe called October. Then, use aggregate to get the mean number of people born in each date of the month, the date_of_month variable.

2. Use select to select people born on Wednesdays. Do a histogram based on these data.

3. Which month has the highest median number of births?

Summary

Go ahead and knit this R Notebook and submit it. After this lab, you should be comfortable selecting subsets of data and calculating summary statistics either `summary()`, `mean()`, `median()`, `sd()`, or `aggregate()`. Next time, we will cover how to do graphing, which builds on this!