Lecture 1 Tutorial

Data Visualization Using R

Leah Brooks

Spring 2018

Welcome! This tutorial assumes that you've successfully done everything in the "Get R Ready" tutorial.

A. The RStudio environment

1. Open up RStudio.
2. What you need to know about what you see
    a. Console: This is the window where you input R commands. You can input them one by one, or you can write a R program, we will do.
    b. Terminal: This window is a terminal window. On a Mac, it should be a unix interface; on a PC it is a DOS interface. I don't anticipate using this window, but you could use it to find the location of files and execute system commands.
    c. Environment: Ignore for now!
    d. History: This reports a history of your R commands. Hopefully you won't need to refer to this window since you won't be programming interactively.
    e. Connections: Igonore for now
    f. Files: Ignore for now
    g. Plots: Your plots will appear here – if we do well, we'll see one this class.
    h. Packages: This lists packages you have installed (they sit on your hard drive). You should see the package "rmarkdown" in this list.
    i. Help: Help for commands. Alternatively, you can type `help("command")` at the console prompt.
    j. Viewer: Ignore for now.

B. Hello world program

The very first program people often write is a very simple one to print "hello world" to the screen.
You should always write R code (or any code) in a program. Never program interactively! (This means typing things in one command at a time at the prompt.) Interactive programming is a way to lose track of what you're doing and make unreplicable work.

To write a R program, open RStudio, and choose File -> New File -> RScript.

You should see a new window open up.

In this new window, begin by writing a comment that says what your program does and when you worked on it. This is good practice, even though it may seem silly for this program.

When you write these comments, use a "#" sign in front of each line, so R will know that this is a comment and not code to evaluate.

My comments look like this

```
#
# this is my program to say "hello world"
#
# hiworld.r
#
# january 11, 2018

###### A. print hello world
```

After your comments, write the R command

```
print("Hello World!")
```

Save this file with a name you'll remember in a location you'll remember. Mine is saved as H:/pppa_data_viz/2018/assignments/lecture01_helloworld.R (If you type the file name, R will automatically add the .R extension to the name). This extension, .R,means it is an R program. Without the proper extension, the program will not work.

Now we want to run this program.

There are two ways to run this file (also known as a script)

1. While in the editor window, go to the Code menu -> Run Region -> Run All (or click the "run" bottom at the top of the window).
2. At the prompt in the console window, type the full name of your file and use the option "echo=TRUE" so that R will show all the individual commands that you're using.

```
source('H:/pppa_data_viz/2018/assignments/lecture01_helloworld.R', echo=TRUE)
```

Either way, you should see something like the below in the Console window:

```
print("Hello World!")
```

```
## [1] "Hello World!"
```

C. Write it up in R Markdown

For this and all lectures, you'll write and submit your homework using R Markdown, which is something like a word processor/code processor for R. You can see both text (like this), code, like the above, and the output of code.

1. Setting up

To get Rmarkdown to work, you need to install a package called knitr.

At the Console prompt type

```
install.packages("knitr", dependencies = TRUE)
```

This will install the package that "knits" your R Markdown code together. The dependencies option makes sure R intalls any other needed packages.

You need install packages only once – so you are now done installing this package for the rest of the course.

2. How it works

Text you write directly. See much more info here, or the R Markdown Cheat Sheet here.

For code to show up, but not be evaluated, you need to type ‘ three times then `{r, eval=FALSE}`, followed by code, then close with three more ‘. (I found it hard to get this to look correct: see examples here).

For output, you need to type the same, but put just {r} after the first three back ticks.

You can also see the underlying file (.Rmd file) for this handout posted next to the pdf version.

3. Try it: Start a new markdown file

Go to File -> New File -> R Markdown

Choose "document" as the format on the left, and the output format as "pdf" on the right.

This doc will have the extension .Rmd, which is an R Markdown file.

4. Type in the file

Get rid of everything in the file expect the yaml header which looks what's below (I added `urlcolor:blue` so that the links look ok).

```
---
title: "Lecture 1 Tutorial"
output: pdf_document
urlcolor: blue
---
```

Type some stuff. Probably a little text and a little unevaluated R code.

5. Knit the file

This is what R calls putting the file together into a pdf. Click the "knit" button at the top. Errors will appear in the Console window. If there are no errors, a pdf will pop up (but it may be a bit slow).

You will turn in homework as a R Markdown document. You can have it run the code, or you can have a separate script that runs the code.

D. Data for the first few classes

1. Codes

We will use datasets are from the Census and are by geography. States and counties are identified by FIPS (federal information processing) codes, which you can find at this website (and many others).

2. County data

These data are from the Decennial Census. The dataset has each existing US county in each census year from 1910 to 2010.

A complete list of variables and their definitions is here. Not all variables are available in all years.

Observations in this dataset are uniquely identified by the variables statefips, countyfips and year. This means that each row in this table has a unique combination of these variables, and that each row is a county in a year.

Download the data from here

E. Load the data

1. Data format

The dataset is in .csv format, which is comma separated values.

A .csv file has the form

var1, var2, var3

a, b, c

d, e, f

g, h, i

where each row in this example is an observation and has three variables.

Variables are separated by commas.

2. Download data

Save the county data to your computer. Note where you save the data!

3. Load the data

To do this, edit your program from before.

Make a comment that you are loading .csv data, and bring it in. I write

```
counties = read.csv("h:/pppa_data_viz/2018/assignments/lecture01/counties_1910to2010_20180115.csv")
```

Note that R uses a forward slash to denote directories, even though Windows usually uses a blackslash.

This command creates a dataframe (R's version of a dataset) that contains the input csv file.

F. Explore the data

1. What variables are here?

```
names(counties)
```

Data frames have variable names, which you can see with the command above.

Refer to specific variables in a dataframe with the dollar sign. So population would be

```
counties$cv1
```

2. Print a few observations

To -see- the dataframe, try these commands:

```
counties[1:5]
counties[1]
counties[1,1]
counties$cv1[1:10]
```

But this actually prints out too much for this handout. So look at them on your screen, and I'll report the final one:

```
counties$cv1[1:10]
```

```
##  [1] 20038 18178 32728 22791 21456 30196 29030 39115 36056 20226
```

3. How many observations in total?

You can see how many rows and columns the data have by typing the below.

```
dim(counties)
```

```
## [1] 34149    68
```

The output tells us that the dataset has 34,149 observations (roughly 3000 counties * 11 years).

You can see what types of variables this dataframe has and get a sense of what they look like by typing

```
str(counties)
```

I don't print the output since it's quite long.

We learn that there are many variables (cv1, cv3, ...) and many missing values (NA). Recall that this dataset is unique by statefips/countyfips/year. Another variable is cv1, which I know is population.

You can see values of the variables for the first few observations.

4. What years are in this dataset?

Create a new object that tells us the frequency of the year variable, and then print it. Note that R uses <- for assigning values.

```
years.freq <- table(counties$year)
years.freq
```

```
##
## 1910 1920 1930 1940 1950 1960 1970 1980 1990 2000 2010
## 2955 3070 3104 3102 3106 3109 3141 3137 3141 3141 3143
```

We observe 2,955 counties in 1910, and 3,143 counties in 2010.

5. Which and how many states are in this dataset?

```
# which states?
states.freq <- table(counties$statefips)
states.freq
```

```
##
##    1    2    4    5    6    8    9   10   11   12   13   15   16   17   18
##  737  133  156  825  638  691   88   33   11  704 1734   51  463 1122 1012
##   19   20   21   22   23   24   25   26   27   28   29   30   31   32   33
## 1089 1155 1319  700  176  264  154  913  955  899 1265  590 1022  185  110
##   34   35   36   37   38   39   40   41   42   44   45   46   47   48   49
##  231  344  681 1098  579  968  846  394  737   55  503  734 1045 2784  317
##   50   51   53   54   55   56
##  154 1418  428  605  786  248
```

The output here tells us that we have 737 observations from the state of Alabama (statefips = 1), and 1,418 from Virginia (statefips = 51). Recall that each row in this dataframe is a state in a year, so there are not 737 counties in Alabama.

```
# how many states?
num.states <- length(states.freq)
num.states
```

```
## [1] 51
```

Here we count the number of elements in the `states.freq` object we created.

Is 51 reasonable? Why or why not?

6. What is the average county population by state?

Doing basic mathematical operations is one of the most complicated parts of R (in my opinion). We will do some this class and will surely revisit this topic in weeks to come.

This section requires a new package. So type

```
install.packages("dplyr", dependencies = TRUE)
```

This installs the dplyr package, and makes sure the package has all the other packages it needs to work properly.

```
# average county population by year
library(dplyr)
county.mn.pop <- aggregate(counties$cv1, list(counties$year), mean, na.rm=TRUE)
county.mn.pop
```

The first part of this command loads the dplyr library. You still need to do this every time you run the program after you've installed the package – installing the package is not equivalent to having the library available.

The second line creates a new object county.mn.pop that is separate from the counties dataframe. It is created by taking the population variable from the counties dataframe (counties$cv1) and taking the average by year. R needs to know which years it should tabulate by. To do this, we by a list of counties$year.

The third element of the second line says we want to take the average.

The fourth element of the second line says that if there are missing values in the calculation, R should proceed without them.

You should see

```
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
##    Group.1        x
## 1     1910 31195.60
## 2     1920 34527.54
## 3     1930 39671.89
## 4     1940 42582.29
## 5     1950 48679.06
## 6     1960 57605.99
## 7     1970 64696.57
## 8     1980 72217.34
## 9     1990 79181.75
## 10    2000 89596.28
## 11    2010 98232.75
```

7. By state and year?

Same idea as the previous, but one more index:

```
# average county population by state and year
library(dplyr)
county.mn.pop <- aggregate(counties$cv1, list(counties$year, counties$statefips), mean, na.rm=TRUE)
county.mn.pop
```

I omit the output here since it is quite long. You should inspect your output and see if it makes sense.

G. Prepare the data to make graphs

0. We want to make two graphs
   - total population over time
   - 2010, all counties: share with college education or more versus median income
1. Find total population by year

Use a very similar technique to our previous work taking the average to find the total:

```
# total population by year
county.tot.pop <- aggregate(counties$cv1, list(counties$year), sum, na.rm=TRUE)
county.tot.pop
```

```
##    Group.1         x
## 1     1910  92151814
## 2     1920 105965011
## 3     1930 123141555
## 4     1940 132090256
## 5     1950 151197155
## 6     1960 179097008
## 7     1970 203211926
## 8     1980 226545805
## 9     1990 248709873
## 10    2000 281421906
## 11    2010 308745538
```

This code does the calculate and then looks at the object you've created. Does it seem reasonable?

2. Make a 2010 dataset with new variables

```
# make just 2010
counties.2010 <- subset(counties, year == 2010)
# check it is the right size
dim(counties.2010)
```

```
## [1] 3143   68
```

This first command makes a new dataframe with only observations where year is 2010. Note the double equals sign. In R, and other languages, single = means assignment, while == means a logical comparison (thanks to this page.

But you should also make sure what you're doing works. So I look at the dimension of the new dataframe. It should be about 3000 observations, which is the number of counties in one year.

To make new variables, you should first refer to the variable list to figure out how to create the share of people with college education.

```
# make both parts of the fraction
counties.2010$num.hged <- mapply(sum, counties.2010$cv15, counties.2010$cv16, counties.2010$cv17,
                                 counties.2010$cv25, counties.2010$cv26, counties.2010$cv27,
                                 na.rm = TRUE)
counties.2010$num.ed <- mapply(sum, counties.2010$cv8, counties.2010$cv9, counties.2010$cv10,
                               counties.2010$cv11,counties.2010$cv12, counties.2010$cv13,
                               counties.2010$cv14, counties.2010$cv15, counties.2010$cv16,
                               counties.2010$cv17, counties.2010$cv18, counties.2010$cv19,
                               counties.2010$cv20, counties.2010$cv21, counties.2010$cv22,
                               counties.2010$cv23, counties.2010$cv24, counties.2010$cv25,
                               counties.2010$cv26, counties.2010$cv27,
                               na.rm = TRUE)
# divide them
counties.2010 <- transform(counties.2010, shr.hged = num.hged / num.ed, na.rm = TRUE)
# check it a bit
counties.2010$shr.hged[1:10]
```

```
##  [1] 0.5132597 0.5956109 0.4029607 0.3453409 0.3950598 0.3428383 0.3747603
##  [8] 0.4530605 0.3873672 0.4353889
```

The first line makes a new variable in the dataframe counties.2010 (the one you just created) called num.hged, and it is the sum of counties.2010 variables cv15, cv16, cv17, cv25, cv26 and cv27. The firm term says ignore missings (without it, a sum with any missing values would itself be a missing value).

The second line does something similar, but adds up all people for whom the Census counts education.

The third line divides the two. You refer to the dataframe in the first argument, and then the function of the variables, and then say to calculate ignoring missing values.

The last line checks that the think you created looks like a share should. Does it?

H. Making two initial charts

0. Install package we'll use for graphics throughout (ggplot2).

```
install.packages("ggplot2", dependencies = TRUE)
```

1. Make a line graph of total population over time and save it.

```
# load the library so you can use the graph commands
library(ggplot2)
```

```
# make picture
ggplot(county.tot.pop, aes(x=Group.1, y=x)) + geom_line()
# save it
ggsave("h:/pppa_data_viz/2018/assignments/lecture01/totpop_line_overtime.jpg",
       plot = last_plot(), device="jpeg")
```

You should see the picture in the -plots- window. This command will also save it to the location specified. You'll need to change the path where you save it, since mine will not work for you!

This will only look so-so. The rest of the course will be making better looking graphs.

2. Make a scatter plot of share with college education versus median household income

```
# make a scatter plot
ggplot(counties.2010, aes(x=counties.2010$shr.hged, y=counties.2010$cv92)) + geom_point()
# save it
ggsave("h:/pppa_data_viz/2018/assignments/lecture01/2010county_scatter",
       plot = last_plot(), device="jpeg")
```

I. At home: PS 1

Use these data to

a. Find total total population by census region or division (your choice) in 2010. See this page for states by division.

Hint: you first need to make a new variable that assigns the census division (one option is to use R's ifelse command). Then aggregate as we did today.

b. Make a scatter plot of two different variables, and explain the relationship you see.