Lecture 2 Tutorial: Manipulating Data in R

Leah Brooks

January 24, 2018

In order to produce graphics, you will need to be able to manipulate data to get to graphics production. It is this manipulation that we focus on today – and next class we will turn to the actual production of graphics.

A. Goals for today

Our goal for this tutorial is to get you more proficient in manipulating data in R. Without some basic ability to manipulate data, you will not be able to shape your data in the way you would prefer to make a graph.

In particular, after today I hope you will be able to

- make basic arithmetic transformations of data (e.g., add or divide columns of data)
- merge datasets
- summarize datasets
- check values of variables

In preparing this tutorial, I found it easier to write code in a R script (.R file) and then do the write-up after. Do whichever is easier for you, but the final product should be a R Markdown file with explanations.

B. More data

This lecture we'll use the county-level dataset from Lecture 1 and an additional data set at the block group level.

These data statsitics on people and housing for all block groups in the United States. A block group is a neighborhood of typically between 600 to 3,000 people. You can find examples of block groups by looking here. These data come from surveys conducted by the Census Bureau and are 5-year averages from 2008-2012.

For each block group, we observe a variety of characteristics. Use this Census provided dictionary to understand variables. Look at this file and you will see what the Census calls "tables." Table B00001 is total population, and the variable in the dataset that relates to this table is called B00001e1 (e is for estimate). Similarly, B00002e1 is the number of housing units, and B01001e2 is number of males under 5 years of age.

Be aware that

- the file I created does not include all of these variables (there are more than 10,000)
- I loaded variables from sequence numbers 1, 4, 9, 19, 41, 43, 58, 59, 62, 63, 64, 78, 81, 83, 105, and 106
- not all variables are available at the block group level. This file tells you whether variables are available at the block group level.

For further information, consult documentation from the American Community Survey.

A block group is uniquely identified by the variables state + county + tract + blkgrp.

This file contains just data from VA, MD and DC so as to be of a manageable size. If at some point you want the whole file (6G), let me know.

Download the data from here

C. Load and look at the block group data

1. Load the data

```
block.groups <- read.csv("h:/pppa_data_viz/2018/assignments/lecture02/acs_bgs20082012_dmv_20180123.csv"
```

2. How big is it?

```
dim(block.groups)
```

```
## [1] 9708 3063
```

This tells us that the dataset has about 9,000 rows (or block groups) and 3,063 variables.

3. What variables does it have?

```
str(block.groups)
```

```
## 'data.frame':    9708 obs. of  3063 variables:
##  $ FILEID     : Factor w/ 1 level "ACSSF": 1 1 1 1 1 1 1 1 1 1 ...
##  $ STUSAB     : Factor w/ 3 levels "dc","md","va": 1 1 1 1 1 1 1 1 1 1 ...
##  $ SUMLEVEL   : int  150 150 150 150 150 150 150 150 150 150 ...
##  $ COMPONENT  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ LOGRECNO   : int  367 368 369 370 371 372 373 374 375 376 ...
##  $ US         : logi  NA NA NA NA NA NA ...
##  $ REGION     : logi  NA NA NA NA NA NA ...
##  $ DIVISION   : logi  NA NA NA NA NA NA ...
##  $ STATECE    : logi  NA NA NA NA NA NA ...
##  $ STATE      : int  11 11 11 11 11 11 11 11 11 11 ...
##  $ COUNTY     : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ COUSUB     : logi  NA NA NA NA NA NA ...
##  $ PLACE      : logi  NA NA NA NA NA NA ...
##  $ TRACT      : int  100 100 100 100 201 202 202 202 202 300 ...
##  $ BLKGRP     : int  1 2 3 4 1 1 2 3 4 1 ...
##  $ CONCIT     : logi  NA NA NA NA NA NA ...
##  $ CSA        : logi  NA NA NA NA NA NA ...
##  $ METDIV     : logi  NA NA NA NA NA NA ...
##  $ UA         : logi  NA NA NA NA NA NA ...
##  $ UACP       : logi  NA NA NA NA NA NA ...
##  $ VTD        : logi  NA NA NA NA NA NA ...
##  $ ZCTA3      : logi  NA NA NA NA NA NA ...
##  $ SUBMCD     : logi  NA NA NA NA NA NA ...
##  $ SDELM      : logi  NA NA NA NA NA NA ...
##  $ SDSEC      : logi  NA NA NA NA NA NA ...
##  $ SDUNI      : logi  NA NA NA NA NA NA ...
##  $ UR         : logi  NA NA NA NA NA NA ...
##  $ PCI        : logi  NA NA NA NA NA NA ...
##  $ TAZ        : logi  NA NA NA NA NA NA ...
##  $ UGA        : logi  NA NA NA NA NA NA ...
##  $ GEOID      : Factor w/ 9708 levels "15000US110010001001",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ NAME       : Factor w/ 9708 levels "Block Group 0, Census Tract 751.01, Suffolk city, Virginia",.
##  $ AIANHH     : logi  NA NA NA NA NA NA ...
##  $ AIANHHFP   : logi  NA NA NA NA NA NA ...
##  $ AIHHTLI    : logi  NA NA NA NA NA NA ...
##  $ AITSCE     : logi  NA NA NA NA NA NA ...
##  $ AITS       : logi  NA NA NA NA NA NA ...
##  $ ANRC       : logi  NA NA NA NA NA NA ...
##  $ CBSA       : logi  NA NA NA NA NA NA ...
##  $ MACC       : logi  NA NA NA NA NA NA ...
##  $ MEMI       : logi  NA NA NA NA NA NA ...
##  $ NECTA      : logi  NA NA NA NA NA NA ...
##  $ CNECTA     : logi  NA NA NA NA NA NA ...
##  $ NECTADIV   : logi  NA NA NA NA NA NA ...
```

```
##  $ CDCURR     : logi  NA NA NA NA NA NA ...
##  $ SLDU       : logi  NA NA NA NA NA NA ...
##  $ SLDL       : logi  NA NA NA NA NA NA ...
##  $ ZCTA5      : logi  NA NA NA NA NA NA ...
##  $ PUMA5      : logi  NA NA NA NA NA NA ...
##  $ PUMA1      : logi  NA NA NA NA NA NA ...
##  $ BTTR       : logi  NA NA NA NA NA NA ...
##  $ BTBG       : logi  NA NA NA NA NA NA ...
##  $ FILETYPE   : num  2.01e+08 2.01e+08 2.01e+08 2.01e+08 2.01e+08 ...
##  $ CHARITER   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ SEQUENCE   : int  106 106 106 106 106 106 106 106 106 106 ...
##  $ B00001e1   : int  63 75 80 54 423 101 60 76 204 65 ...
##  $ B00002e1   : int  40 52 33 30 0 46 29 37 44 33 ...
##  $ B02001e1   : int  1296 1322 1430 992 4074 1268 869 976 1863 1209 ...
##  $ B02001e2   : int  1155 1204 1241 924 3079 1131 800 922 1599 1137 ...
##  $ B02001e3   : int  0 27 0 0 323 12 0 0 96 0 ...
##  $ B02001e4   : int  0 0 81 0 0 0 0 0 14 12 ...
##  $ B02001e5   : int  44 46 108 68 500 81 59 7 88 40 ...
##  $ B02001e6   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ B02001e7   : int  97 22 0 0 16 32 0 0 0 20 ...
##  $ B02001e8   : int  0 23 0 0 156 12 10 47 66 0 ...
##  $ B02001e9   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ B02001e10  : int  0 23 0 0 156 12 10 47 66 0 ...
##  $ B02005e1   : logi  NA NA NA NA NA NA ...
##  $ B02005e2   : logi  NA NA NA NA NA NA ...
##  $ B02005e3   : logi  NA NA NA NA NA NA ...
##  $ B02005e4   : logi  NA NA NA NA NA NA ...
##  $ B02005e5   : logi  NA NA NA NA NA NA ...
##  $ B02005e6   : logi  NA NA NA NA NA NA ...
##  $ B02005e7   : logi  NA NA NA NA NA NA ...
##  $ B02005e8   : logi  NA NA NA NA NA NA ...
##  $ B02005e9   : logi  NA NA NA NA NA NA ...
##  $ B02005e10  : logi  NA NA NA NA NA NA ...
##  $ B02005e11  : logi  NA NA NA NA NA NA ...
##  $ B02005e12  : logi  NA NA NA NA NA NA ...
##  $ B02005e13  : logi  NA NA NA NA NA NA ...
##  $ B02005e14  : logi  NA NA NA NA NA NA ...
##  $ B02005e15  : logi  NA NA NA NA NA NA ...
##  $ B02005e16  : logi  NA NA NA NA NA NA ...
##  $ B02005e17  : logi  NA NA NA NA NA NA ...
##  $ B02005e18  : logi  NA NA NA NA NA NA ...
##  $ B02005e19  : logi  NA NA NA NA NA NA ...
##  $ B02005e20  : logi  NA NA NA NA NA NA ...
##  $ B02005e21  : logi  NA NA NA NA NA NA ...
##  $ B02005e22  : logi  NA NA NA NA NA NA ...
##  $ B02005e23  : logi  NA NA NA NA NA NA ...
##  $ B02005e24  : logi  NA NA NA NA NA NA ...
##  $ B02005e25  : logi  NA NA NA NA NA NA ...
##  $ B02005e26  : logi  NA NA NA NA NA NA ...
##  $ B02005e27  : logi  NA NA NA NA NA NA ...
##  $ B02005e28  : logi  NA NA NA NA NA NA ...
##  $ B02005e29  : logi  NA NA NA NA NA NA ...
##  $ B02005e30  : logi  NA NA NA NA NA NA ...
##  $ B02005e31  : logi  NA NA NA NA NA NA ...
```

```
##  $ B02005e32  : logi  NA NA NA NA NA NA ...
##   [list output truncated]
```

Note that this output gets truncated, because there are so many variables that R does not want to print them all.

D. Make some variables and check with simple summary stats

1. Find the average population by block group

```
# find averages population, median income, share white
# population
print("mean population by block groups is")
```

```
## [1] "mean population by block groups is"
```

```
mean(block.groups$B00001e1, trim = 0, na.rm = TRUE)
```

```
## [1] 106.695
```

The `mean()` function tells us the mean of a variable. It can take up to four arguments, and we'll discuss the first three. The first input is the variable the mean of which you want to find. The second input (which you can omit), is to "trim" the data, or omitting very high and very low observations. This is useful when you want your mean not to reflect outliers (but you could also consider using a median!). The final option says to remove missing values when calculating the mean. We will frequently use this option equal to true.

However, we should be concerned if there are many missing values (there are some block groups without people, so a few missings is ok).

To check for block groups without population, use the `is.na` function.

```
# check for number of NA obs
print("number of missing obs is ")
```

```
## [1] "number of missing obs is "
```

```
sum(is.na(block.groups$B00001e1))
```

```
## [1] 20
```

2. Repeat for income

To simplify your life, I'll tell you that the variable name for median household income is B19013e1. You can see what this variable looks like on the two excel sheets and this should help you find other variables.

```
# income
mean(block.groups$B19013e1, na.rm = TRUE)
```

```
## [1] 74523.2
```

```
sum(is.na(block.groups$B19013e1))
```

```
## [1] 108
```

3. Find the share white by block group

Now that the Census allows for reporting of more than one ethnic category, this is not as easy as it used to be! I would generally not recommend you calculate the share of white alone (only identifies as white) for research purposes, but it is fine for practice purposes here.

In the code below, I use the handy `transform()` function to create new variables in the block.groups dataframe. The "transform" function outputs a dataframe, which is to the left of the arrow (I re-write over the original). The inputs of the function are a dataframe, the new variable to be created and how, and what to do with missings.

In the code below, I also look at a few of the values of the total population and the white alone population, as well as my calculated share. My average share white is between 0 and 1, but it must also be true that all values are between 0 and 1, so I use the min and max functions to assure myself that this is true.

This type of data-checking is very important to make sure that you are not causing trouble.

```
# share white alone
# calculate share white
# use B02001
block.groups <- transform(block.groups, shr.white = B02001e2 / B02001e1, na.rm=TRUE )
block.groups$B02001e2[1:10]
```

```
##  [1] 1155 1204 1241  924 3079 1131  800  922 1599 1137
```

```
block.groups$B02001e1[1:10]
```

```
##  [1] 1296 1322 1430  992 4074 1268  869  976 1863 1209
```

```
block.groups$shr.white[1:10]
```

```
##  [1] 0.8912037 0.9107413 0.8678322 0.9314516 0.7557683 0.8919558 0.9205984
##  [8] 0.9446721 0.8582931 0.9404467
```

```
print("average share white by block groups")
```

```
## [1] "average share white by block groups"
```

```
mean(block.groups$shr.white, na.rm = TRUE)
```

```
## [1] 0.6402873
```

```
print("make sure shares are ok.  min is")
```

```
## [1] "make sure shares are ok.  min is"
```

```
min(block.groups$shr.white, na.rm = TRUE)
```

```
## [1] 0
```

```
print("max is ")
```

```
## [1] "max is "
```

```
max(block.groups$shr.white, na.rm = TRUE)
```

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

```
print("how many obs are missing")
```

```
## [1] "how many obs are missing"
```

```
sum(is.na(block.groups$shr.white))
```

```
## [1] 59
```

E. Load the counties data from last class

One of the strengths of R relative to Stata is that it have two datasets (dataframes) in memory at the same time. Without losing the block group data, we'll now load the counties data.

1. Load the data

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

2. Make sure it seems ok – check dimensions

```
dim(counties)
```

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

F. Merge 2010 counties and block groups together

Merging is one of the most powerful things that statistical software can do for you.

1. Think about the merge

Our goal is to add 2010 county-level info to the block group data.

Think about the structure of the county data – which years does it have?

Think about the structure of the block group data – which years and states does it have?

What variables should we use to combine them?

Any merge undertaken without thinking first is destined to fail.

2. The merge command

The merge command in R gives you lots of options to keep what you'd like from the merge. See the full syntax here.

In short, a command could look like

```
  total <- merge(data.frame.x, data.frame.y, by.x = data.frame.x$var1 , by.y = data.frame.b$var3 )
```

Here, R is taking data frame x and merging with data frame y. We match var1 from data frame x with var3 from data frame y. The `by.x` and `by.y` commands tell R to which dataset the merging variables belong.

3. Implement the merge

Below, I do a number of steps

- create a county dataset with just 2010
- check the size of the block group and county data
- do the merge
- make sure the final product is the right size

The final command (the `merge` command) allows R to merge counties.2010 (the x dataset) and block.groups (the y dataset) using variables that have the same codes in the two datasets but have different variable names. We need to merge by both variables statefips and countyfips because county codes repeat across states.

```
# make a dataset with just 2010
counties.2010 <- subset(counties, counties$year == 2010)

# check size of dataframes before merge
dim(counties.2010)
```

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

```
dim(block.groups)
```

```
## [1] 9708 3065
```

```
# execute merge
county.bg <- merge(x=counties.2010, y=block.groups, by.x = c("statefips", "countyfips"),
                                                    by.y = c("STATE", "COUNTY"))
```

After you execute, check things!

```
# check on # of obs
dim(county.bg)
```

```
## [1] 9708 3131
```

How many observations? Is this reasonable?

G. Calculate block group values relative to county, state and country

Now that we have county and block group variables in the same dataframe, we can calculate a block group's value relative to the corresponding county value.

We do this for three variables

- share with education > high school (repeat from last tutorial)
- share white (as above)
- median household income

Again, we use the transform command to create new variables, and the mean() function to check values.

```
# look at ratios relative to county ratio
# block groups with share with education > HS
county.bg <- transform(county.bg,
                       shr.hged = (B15002e12 + B15002e13 + B15002e14 + B15002e15 +
                                     B15002e16 + B15002e17 + B15002e18 +
                                     B15002e29 + B15002e30 + B15002e31 + B15002e32 +
                                     B15002e33 + B15002e34 + B15002e35)/B15002e1, na.rm = TRUE)
# county share education > HS
county.bg <- transform(county.bg, cnum.hged = cv15 + cv16 + cv17+ cv25 + cv26 + cv27, na.rm = TRUE)
county.bg <- transform(county.bg, cnum.ed = cv8 + cv9 + cv10 + cv11 + cv12 + cv13 + cv14 +
                                    cv15 + cv16 + cv17 + cv18 + cv19 + cv20 + cv21 +
                                    cv22 + cv23 + cv24 + cv25 + cv26 + cv27,
                                  na.rm = TRUE)
county.bg <- transform(county.bg, cshr.hged = cnum.hged / cnum.ed, na.rm = TRUE)
county.bg <- transform(county.bg, rshr.hged = shr.hged/cshr.hged, na.rm = TRUE)

# share white
# for practice purposes, assume share white is total population minus black and other races
county.bg <- transform(county.bg, cshr.white = 1 - (cv3+cv4)/cv1, na.rm = TRUE)
county.bg <- transform(county.bg, rshr.white = shr.white/cshr.white, na.rm = TRUE)
mean(county.bg$cshr.white, na.rm = TRUE)
```

```
## [1] 0.6313426
```

```
mean(county.bg$rshr.white, na.rm = TRUE)
```

```
## [1] 1.013149
```

```
# median hh income
county.bg <- transform(county.bg, rshr.hhmedinc = county.bg$B19013e1 / county.bg$cv92, na.rm = TRUE)
mean(county.bg$cv92)
```

```
## [1] 69113.86
```

```
mean(county.bg$B19013e1, na.rm = TRUE)
```

```
## [1] 74523.2
```

```
mean(county.bg$rshr.hhmedinc, na.rm = TRUE)
```

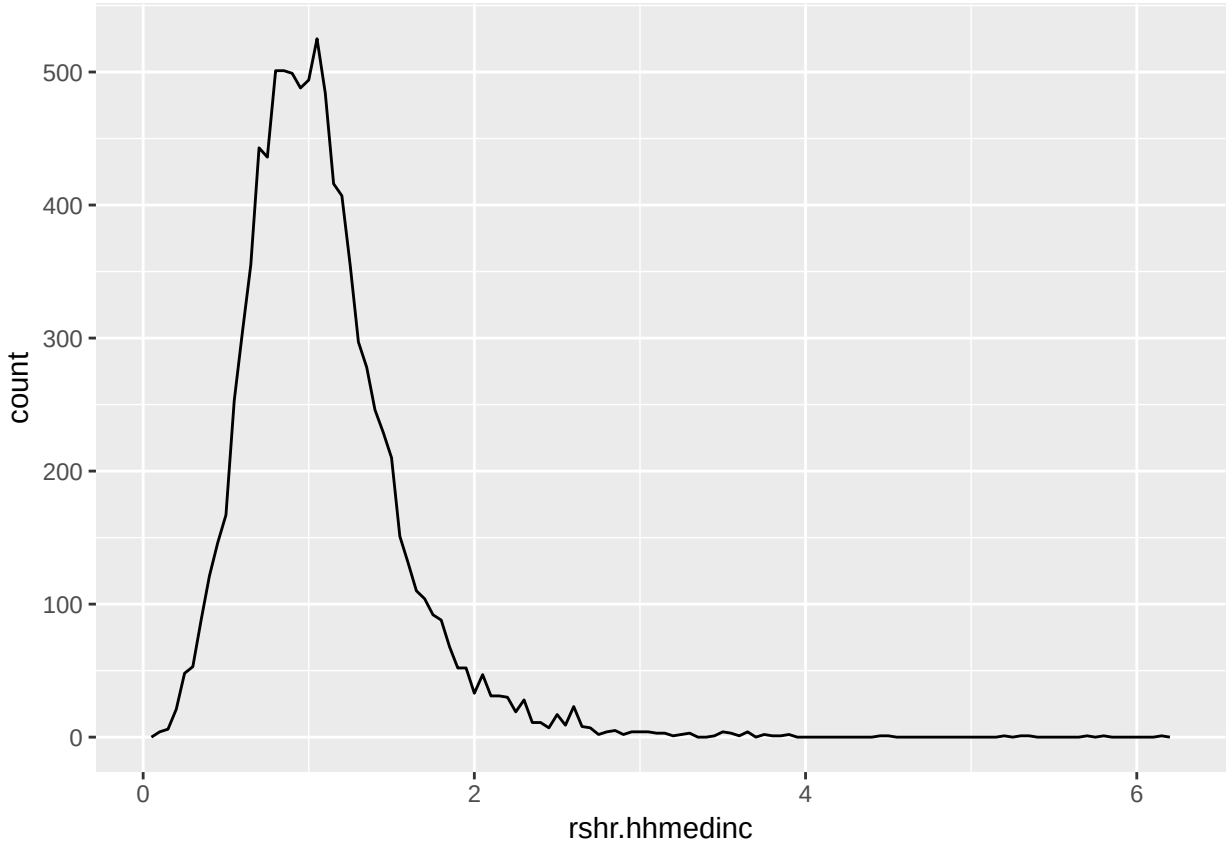
```
## [1] 1.073392
```

Finally, we make a plot one of these ratios. I made a histogram of the relative household income shares; you should make a histogram of whichever you please!

The binwidth tells us the bin over which R makes the histogram. For shares, 0.05 is reasonable. For values, this is probably not a good bin width.

```
# make histograms of one of these ratios
library(ggplot2)
ggplot(county.bg, aes(rshr.hhmedinc)) + geom_freqpoly(binwidth=0.05)
```

```
## Warning: Removed 108 rows containing non-finite values (stat_bin).
```



H. Statistics for just DC, MD and VA

In this section, we practice a more complicated subsetting, and make a line graph of total population.

To do this, you'll need to understand two important operators: `&` and `|`. In English, these are "and" and "or." In this example, we want to create a subset of the national counties dataset that contains just DC-area counties.

Imagine the DC area just consists of DC and Prince George's County, Maryland. In this case, we would like to keep observations where the county value is equal to the DC value OR the Prince George's value. We will use the `|` symbol below to effect this.

In addition, each county is identified by a state code AND a county code. For example, county `001` could appear in many states. In fact, both Autauga County, Alabama and DC both have code `001`. Therefore, to identify DC you'll need to specify the state code for DC (11) and the county code (`001`); here we'll use the `&` notation.

1. Subset counties data to just DC, MD and VA

```
# subset to DC, MD, VA
# Fairfax City (51/600), Falls Church (51/610), Arlington (51/013),
```

```
# Fairfax County (51/059), and Alexandria City (51/510).
counties.dmv <- subset(counties, ((statefips == 11) & (countyfips == 1)) |
                         ((statefips == 24) & (countyfips == 33)) |
                         ((statefips == 24) & (countyfips == 31)) |
                         ((statefips == 51) & (countyfips == 600)) |
                         ((statefips == 51) & (countyfips == 610)) |
                         ((statefips == 51) & (countyfips == 13)) |
                         ((statefips == 51) & (countyfips == 59)) |
                         ((statefips == 51) & (countyfips == 510)) )
dim(counties.dmv)
```

```
## [1] 78 68
```

```
# make sure you did ok job
table(counties.dmv$statefips)
```

```
##
## 11 24 51
## 11 22 45
```

2. Find total population by year

Now we want to calculate some annual statistics for these counties and return the result as a dataframe – so we can go do other things with it. In particular, this next command will find the average and total population for DC-area counties.

We need both the `plyr` and `dplyr` packages. I think you have these both from last time, but if not you should install both packages. You can check whether you have the packages by looking at the "packages" window.

We will use a command called `ddply`, which automates a more complicated set of commands. The first arguments is `.data`, which is the input data frame. In our case, this is the DC area county data, or `counties.dmv`.

```
output <- ddply(.data, .variables, .fun =, other arguments passed on to the function, other stuff)
```

The second argument is `.variables`, which are the variables by which you'd like to do your analysis. Here we are interested in statistics by year, so we write `c("year)` to let the package know we want to group by the year variable.

The third argument set to `summarise` tells R to create a new dataset. See this example for another option (but don't worry about this now unless you're curious).

The final arguments are telling R what you'd like to calculate. We first calculate the number of counties. The `length` function counts the number of observations in the listed variable. Below, it is calculating the number of counties (because each county has a population number) in each year. This is a useful check, because if the number of counties changes over time, we need to understand why.

The second calculation line finds the mean of population (making sure we allow for calculation in the presence of missing values). The final line finds total (`sum`) population. We could equally well, or additionally, calculate standard deviations, maxima or minima.

```
# average population by year
# requires package plyr
library(plyr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:plyr':
```

```
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
totpop.dmv <- ddply(counties.dmv, c("year"), summarise,
                    counties.n = length(cv1),
                    pop.mn = mean(cv1, na.rm=FALSE),
                    pop.tot = sum(cv1, na.rm=FALSE)
)
totpop.dmv
```

```
##    year counties.n    pop.mn pop.tot
## 1  1910          6  74233.50  445401
## 2  1920          6  95313.67  571882
## 3  1930          6 112033.00  672198
## 4  1940          6 161330.83  967985
## 5  1950          7 209155.57 1464089
## 6  1960          7 285985.29 2001897
## 7  1970          8 339108.88 2712871
## 8  1980          8 345509.88 2764079
## 9  1990          8 402887.25 3223098
## 10 2000          8 445784.38 3566275
## 11 2010          8 487642.00 3901136
```

Does the output seem reasonable?

Now, for purposes of graphing, the big population numbers are difficult to read. So I make population in thousands (that means divide the population number by 1,000). Of course, it's not that straightforward. You can't write

```
totpop.dmv <- transform(totpop.dmv, popthou = pop.tot/1000)
```

because R thinks you're trying to divide a vector – total population in each each – by one integer (the number 1,000) and it doesn't like it. To fix this, make a new variable in the dataframe where each observation is equal to 1,000. There are probably other ways than the method I use below (but the below one works!) that are equally as good or better.

To generate a new variable in the totpov.dmv dataframe that is always equal to 1000, I repeat (`rep`) the number 1000 by the length of the totpov.dmv datafram (`dim(totpop.dmv)[1]`). I look at the result – is it ok? I then divide by this new variable and check the result again.

```
# make population in 1000s
totpop.dmv$thousand <- rep(1000, dim(totpop.dmv)[1])
totpop.dmv$thousand
```

```
##  [1] 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
```

```
totpop.dmv <- transform(totpop.dmv, pop.thous = pop.tot / thousand)
totpop.dmv
```

```
##    year counties.n    pop.mn pop.tot thousand pop.thous
```

```
## 1  1910          6  74233.50  445401     1000   445.401
## 2  1920          6  95313.67  571882     1000   571.882
## 3  1930          6 112033.00  672198     1000   672.198
## 4  1940          6 161330.83  967985     1000   967.985
## 5  1950          7 209155.57 1464089     1000  1464.089
## 6  1960          7 285985.29 2001897     1000  2001.897
## 7  1970          8 339108.88 2712871     1000  2712.871
## 8  1980          8 345509.88 2764079     1000  2764.079
## 9  1990          8 402887.25 3223098     1000  3223.098
## 10 2000          8 445784.38 3566275     1000  3566.275
## 11 2010          8 487642.00 3901136     1000  3901.136
```

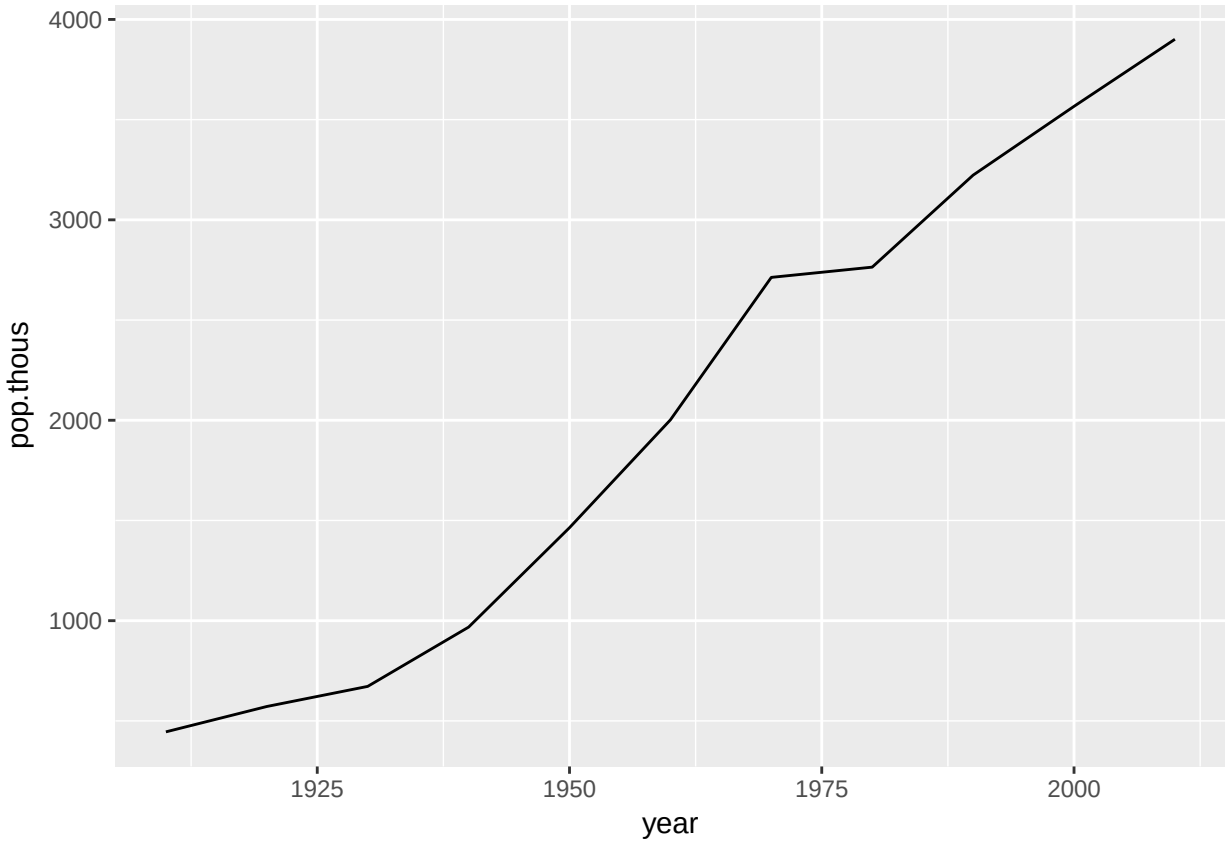
3. Make a line graph of total population over time

```
### total population over time

# check data
totpop.dmv
```

```
##    year counties.n    pop.mn pop.tot thousand pop.thous
## 1  1910          6  74233.50  445401     1000   445.401
## 2  1920          6  95313.67  571882     1000   571.882
## 3  1930          6 112033.00  672198     1000   672.198
## 4  1940          6 161330.83  967985     1000   967.985
## 5  1950          7 209155.57 1464089     1000  1464.089
## 6  1960          7 285985.29 2001897     1000  2001.897
## 7  1970          8 339108.88 2712871     1000  2712.871
## 8  1980          8 345509.88 2764079     1000  2764.079
## 9  1990          8 402887.25 3223098     1000  3223.098
## 10 2000          8 445784.38 3566275     1000  3566.275
## 11 2010          8 487642.00 3901136     1000  3901.136
```

```
# make picture
library(ggplot2)
ggplot(totpop.dmv, aes(x=year, y=pop.thous)) + geom_line()
```




```
# save it
ggsave("h:/pppa_data_viz/2018/assignments/lecture01/dmv_line_overtime.jpg", plot = last_plot(), device=
```

```
## Saving 6.5 x 4.5 in image
```

What if you forget to put in the library? (While we've already installed the ggplot2 package, we still need to tell R that we need it now, which is what calling the library does.)

```
> # make picture
> ggplot(totpop.dmv, aes(x=year, y=pop.thous)) + geom_line()
Error in ggplot(totpop.dmv, aes(x = year, y = pop.thous)) :
  could not find function "ggplot"
In addition: Warning message:
In mean.default(county.bg$rshr.white, na.rm = TRUE) :
  argument is not numeric or logical: returning NA
```

I know that this is an error about the library because it tells me that it can't find the function. The remainder of the error message is a bit of a red herring.

I. Homework

1. Pick a few block group level variables of interest and take averages by county. Explain your results a bit
2. Make some function (average, standard deviation, etc) of three other county variables by year for the entire US.