

Tutorial 4: Histograms

Leah Brooks

February 4, 2020

In today's tutorial, we learn how to make histograms. Histograms are plots that show the distribution of one variable. They are very useful when you want to show details of a variable beyond the mean. For example, if you're interested in showing income inequality, a histogram is one way to visualize income inequality. You can also calculate statistics that summarize inequality; these are more limited descriptions of what you see in a histogram.

In this tutorial, we first work through some histograms with a small annual dataset of hurricanes by year since 1851. After establishing how histograms work, we then turn to a larger dataset of all small neighborhoods in the Washington region, and work on further variations of histograms.

Along the way, we begin to introduce elements of `ggplot`'s `theme()` command for changing the chart background. You also learn how to save graphs via the command line, and how to create a sequence of numbers.

A. Load Packages and Small Data

As you did last week, create an R script for this class. Write all your commands in the R script (recall, a file with R commands ending in `.R`). You can run all of the program at once (code `->` run region `->` run all), or just selected lines.

At the end of the tutorial, you should be able to run your entire R script once and it should run without errors. If this doesn't work, then there is a logical error in the order of your arguments, or some other problem.

A.1. Load packages

Start by loading the `ggplot2`, `scales` and `dplyr` packages with the `library` command. It is good practice to do this at the start of the program.

```
library(ggplot2)
library(scales)
library(dplyr)
```

```
##
## 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
```

A.2. Load and explore data

Now you should download the hurricane [data](#), and load them into R using `read.csv()`. These data come from [this webpage](#); look at this page for complete variable definitions. I copied the online table into Excel

and saved as a csv file.

```
# load hurricane data
hurr <- read.csv("H:/pppa_data_viz/2019/tutorial_data/lecture03/2019-02-02_hurricanes_by_year.csv")
```

Take a quick look at these data. What variables does it have? What do the first five observations look like (head)? And what types of variables does it have (str)?

```
# look at variables and types
names(hurr)
```

```
## [1] "year"          "Named.Storms" "Hurricanes"    "Major"
```

```
head(hurr)
```

```
##   year Named.Storms Hurricanes Major
## 1 1851             6           3     1
## 2 1852             5           5     1
## 3 1853             8           4     2
## 4 1854             5           3     1
## 5 1855             5           4     1
## 6 1856             6           4     2
```

```
str(hurr)
```

```
## 'data.frame':   168 obs. of  4 variables:
## $ year      : int  1851 1852 1853 1854 1855 1856 1857 1858 1859 1860 ...
## $ Named.Storms: int  6 5 8 5 5 6 4 6 8 7 ...
## $ Hurricanes  : int  3 5 4 3 4 4 3 6 7 6 ...
## $ Major       : int  1 1 2 1 1 2 0 0 1 1 ...
```

B. Make a simple histogram

B.1. Check on data

Our first goal is to make a histogram of the number of hurricanes by year. How frequent are years with many hurricanes? Before making a histogram, let's start by making a table shows the data we'll use in the histogram. We do this to check on data quality and to make sure that we are graphing something that can be graphed with a histogram.

We use the command `table` that we've used in the last tutorial.

```
# make sure we know what data will tell us
table(hurr$Major)
```

```
##
##  0  1  2  3  4  5  6  7  8
## 31 48 45 15  9  9  7  2  1
```

So far so good.

B.2. Basic histograms

As a reminder, the three things R needs to make a graph are (i) the dataframe, (ii) the variable you want to graph and (iii) the type of graph you'd like to make.

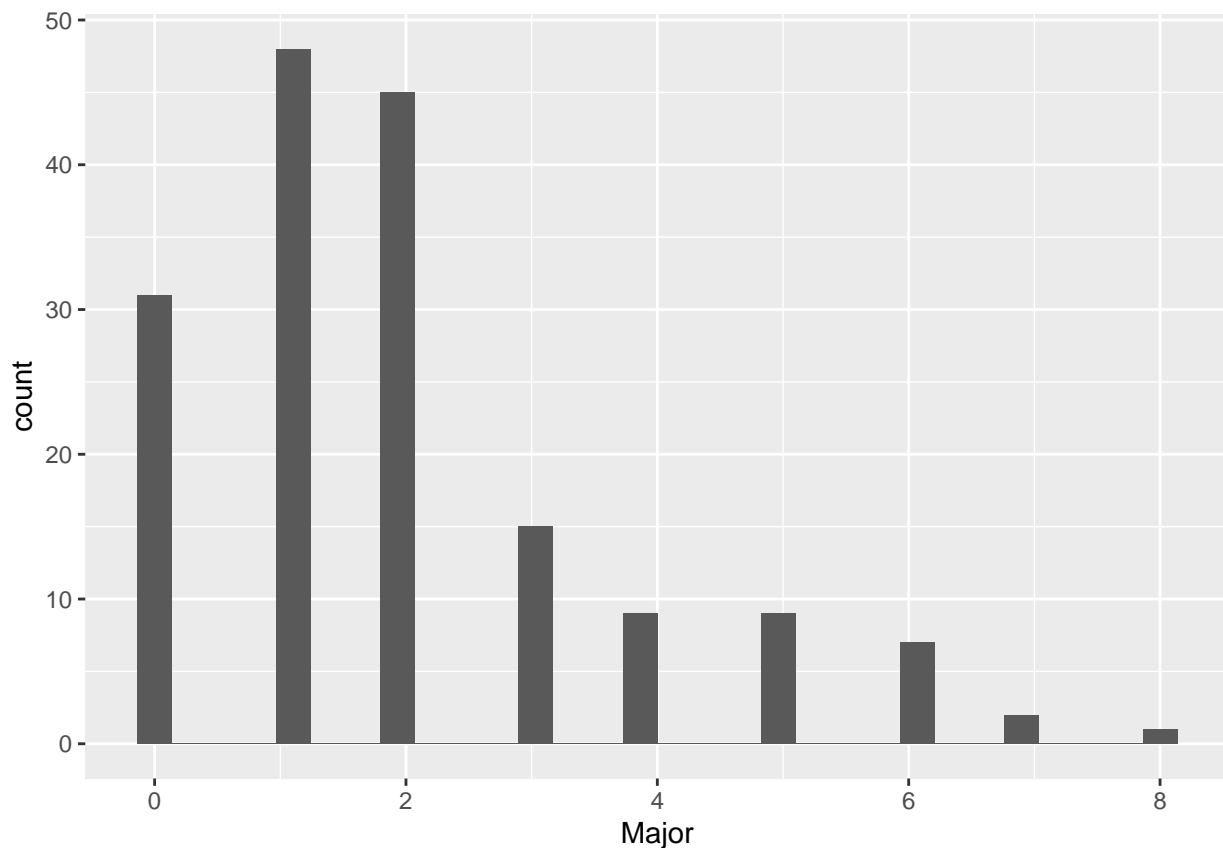
Note that I am creating the chart as an object `c1`, and I can refer to `c1` elsewhere in the program if I like. I can add to it, as we'll see later, and I can output it.

A small programming note: R will fail if you put the `+` on the second line. So make sure that you never start a continuing line of `ggplot` with a plus. See [here](#) for details.

```
# how many hurricanes by year?  
c1 <- ggplot() +  
  geom_histogram(data = hurr,  
                 mapping = aes(x = Major))  
c1
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 1 rows containing non-finite values (stat_bin).
```



This chart looks a bit odd. But before we get to the oddness of the chart, pay attention to the red error message in your console:

```
Warning message:  
Removed 1 rows containing non-finite values (stat_bin).
```

This message translates to “there is one observation for this chart that has a missing value.” What is this one missing? And why didn't we see it in the table command above?

B.3. Finding the missing value

When you have missing values, you should know why. Sometimes an observation (row) has no data – and it has no data when you got it. Frequently, missing data is a result of mistakes in programming.

So we'll now investigate why there is a missing value here. First, I want to check that there really is a missing value, given that the table command above didn't report one. I use the `summary()` command to look at the distribution of a variable:

```
# fix data problem
summary(hurr$Major)
```

```
##   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.   NA's
##  0.000  1.000   2.000   1.964  3.000   8.000     1
```

The summary output tells me that the mean number of hurricanes per year is almost 2 – and that there is one missing value.

Let's look at this problematic observation, by just printing just a subset of the dataframe `hurr` where the variable `Major` takes on a value of missing.

```
# fix data problem
hurr[which(is.na(hurr$Major) == TRUE),]
```

```
##   year Named.Storms Hurricanes Major
## 168  NA           NA           NA  NA
```

I notice that this error occurs in a row 168, which is also without a value for the variable `year`. Putting this together, I conclude that this is an extra row that R imported by mistake.

How do we get rid of this extra row? Use `dim()` to see how many rows the dataframe has

```
dim(hurr)
```

```
## [1] 168  4
```

and we find that it is 168. That means this problematic observation is the last observation.

Let's get rid of this observation. Below I just keep observations 1 to 167 of the dataframe `hurr` (see the first class's tutorial if you don't remember this syntax).

```
hurr <- hurr[1:167,]
```

Once you've figured out where the problem is what's causing it there are alternative ways to get to this same outcome. For example, you could do

```
hurr <- filter(.data = hurr, is.na(Major) == FALSE)
```

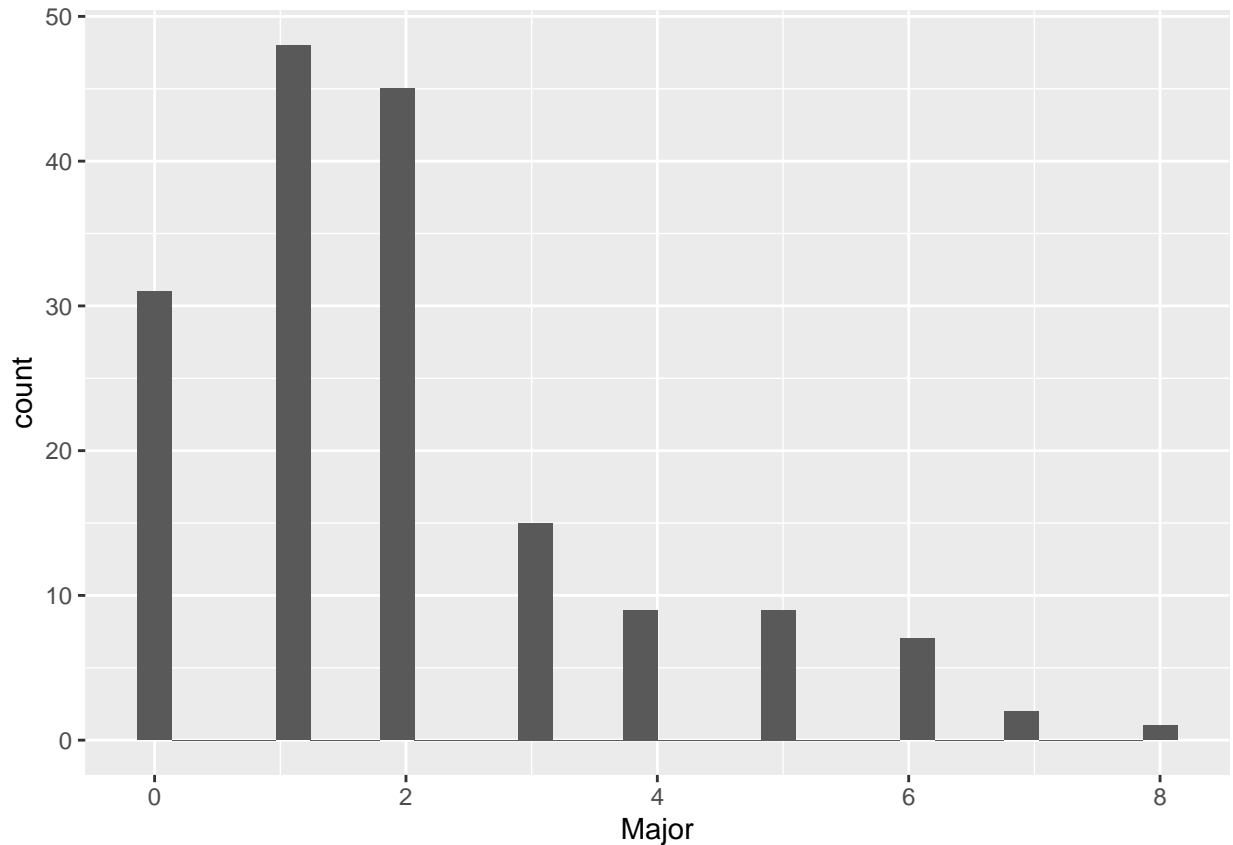
Here we replace dataframe `hurr` with a revised data from where the variable `Major` is not missing (`is.na() == FALSE`).

B.4. Basic histogram, again

Now let's try the graph again. You should have no error message about non-finite values.

```
# how many hurricanes by year?
c1 <- ggplot() +
  geom_histogram(data = hurr,
                mapping = aes(x = Major))
c1
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

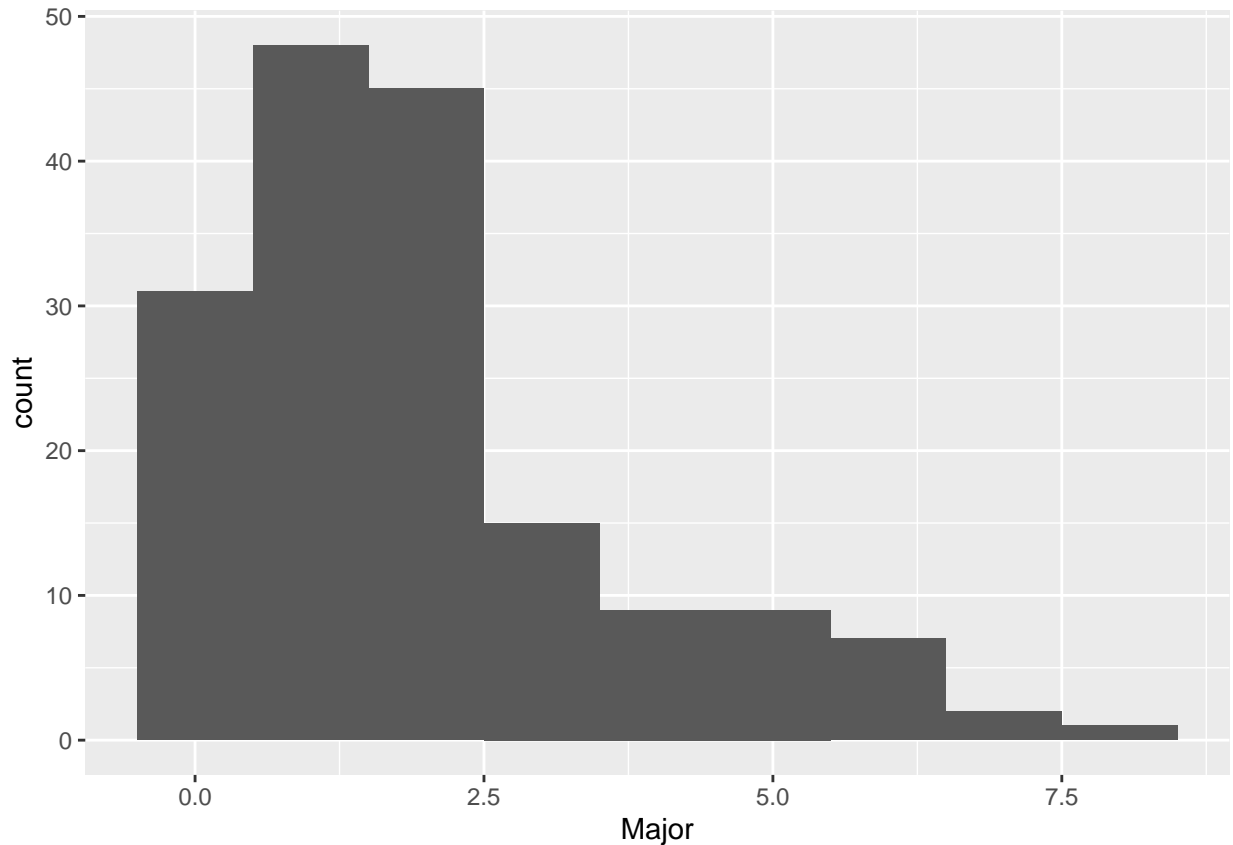


The other error message is about the number of bins. R is putting the data into 30 bins. This is odd, given that we know the maximum number of hurricanes in any year is 8. In other words, R is making 30 bins from the range from 0 to 8. This makes a set of 30 bins like this: $[0-8/30, 8/30-16/30, 16/30-24/30, \dots, 232/30-240/30]$, or in decimals $[0 - 0.26, 0.26 - 0.53, 0.53 - 0.8, \dots, 7.73 - 8]$.

This is crazy! We have only 9 bins, so we need to tell R this explicitly, using the `bins` option inside `geom_histogram`. Note that this option is inside the `geom` call, since it is specific to this type of graph.

```
# we know there are only 9 bins. fix to reflect
c1 <- ggplot() +
  geom_histogram(data = hurr,
    mapping = aes(x = Major),
    bins = 9)
```

c1



This graph could still use some work. For one, the axis labels in the above are entirely unhelpful. We now do two things to make the axes more legible. First, we modify axis labels by using the `labs` command to add in x and y axis labels. You can also use this command to get rid of labels.

Second, to change the numeric labeling on the x axis, I use the `scale_x_continuous(breaks = seq(0,8,1))` to tell R to use numbers 0 to 8 as the labeled breakpoints on the horizontal axis. The command `seq(0,8,1)` means “create a sequence starting at 0, going to 8, by 1.” Alternatively, I could have written `scale_x_continuous(breaks = c(0,1,2,3,4,5,6,7,8))`, but the first one is shorter, cleaner and easier to modify if you want to later make changes.

The sequence framework is easily modifiable. For example, `seq(0,1,0.25)` means “create a sequence starting at 0, going to 1, by 1/4.” Try it and see:

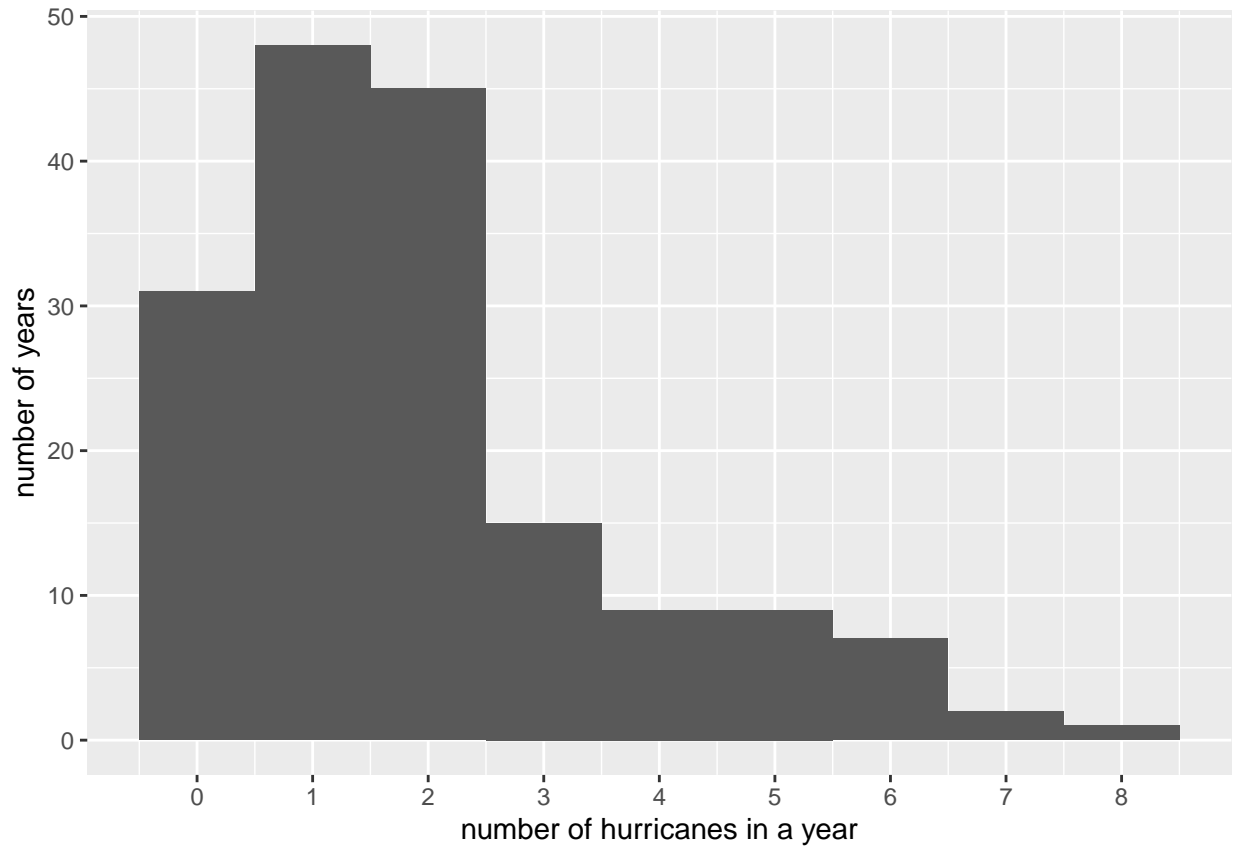
```
my.seq <- seq(0,1,0.25)
my.seq
```

```
## [1] 0.00 0.25 0.50 0.75 1.00
```

Does the sequence look as you expected?

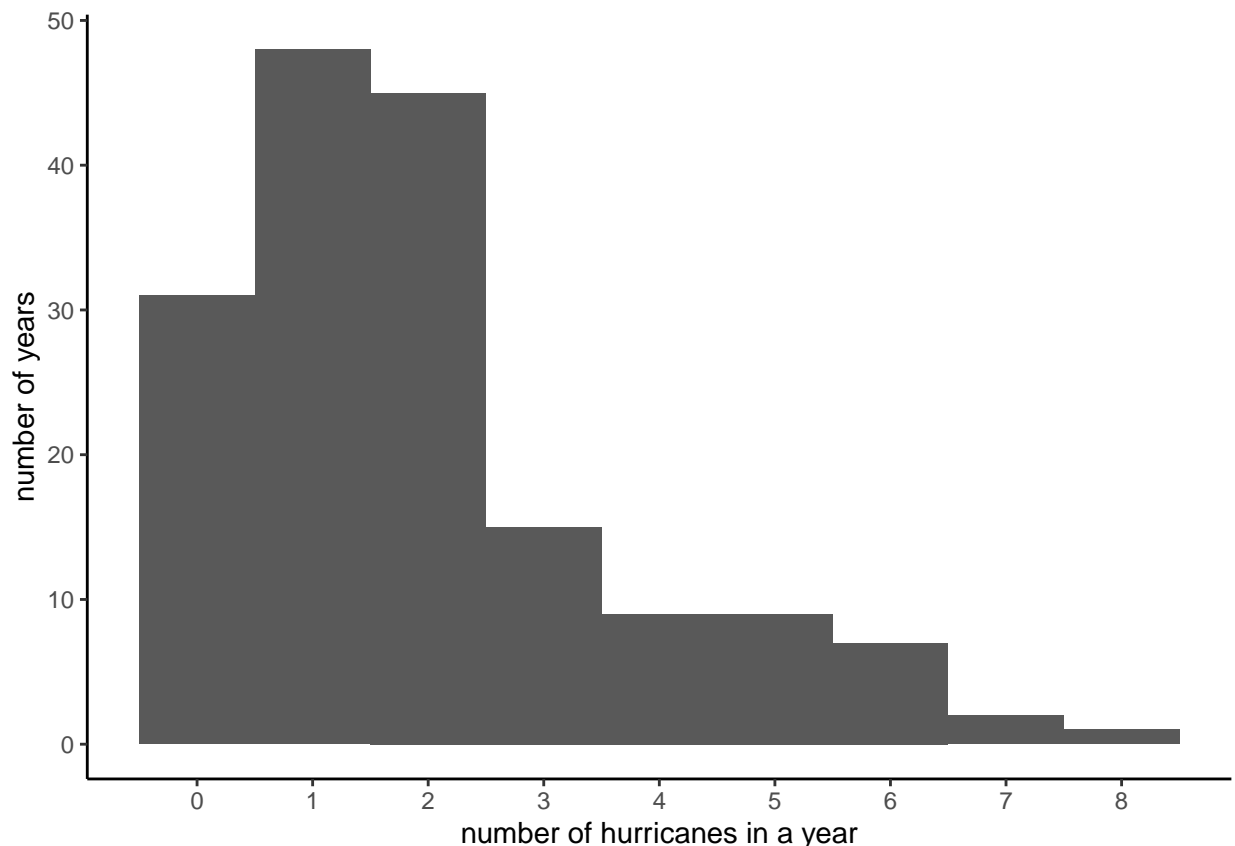
And now for the new graph with both additions:

```
# yes, but nutty labels!  
c1 <- ggplot() +  
  geom_histogram(data = hurr,  
                mapping = aes(x = Major),  
                bins = 9) +  
  scale_x_continuous(breaks = seq(0,8,1)) +  
  labs(x = "number of hurricanes in a year",  
       y = "number of years")  
c1
```



The background in the above chart is distracting and unhelpful. Like almost everything in `ggplot`, the background is also modifiable. The background and the types of axes, and many other things are parts of the “theme” of the graph, and you modify them with the `theme()` command. You can see the zillions of full options for modification [here](#). Right now, we’ll just get rid of the grid and the background color. Here I use `panel.grid`, but you can adjust both the major and minor grids with `panel.grid.major` and `panel.grid.minor`. “Getting rid” means setting to `element_blank()`. I also set the axis lines black.

```
# get rid of crazy background that is confusing
c1 <- ggplot() +
  geom_histogram(data = hurr,
                mapping = aes(x = Major),
                bins = 9) +
  scale_x_continuous(breaks = seq(0,8,1)) +
  labs(x = "number of hurricanes in a year",
       y = "number of years") +
  theme(panel.grid = element_blank(),
        panel.background = element_blank(),
        axis.line = element_line(colour = "black"))
c1
```



C. Modifying the basic histogram

This histogram above is doing one job: telling us the overall distribution of number of major hurricanes by year. Suppose we want to convey some additional information – perhaps we’d like to show that the number of hurricanes by century is on the rise.

Below is an interesting but not-perfect technique for doing this. We color in the bars to reflect the number of observations that come from each century. This is visually clear, but somewhat misleading because each century does not have the same number of observations, so three observations from the 2000s should carry more weight than three observations from the 1900s. We are not going to deal with this concern here, but return to ways to compare distributions later in this tutorial.

To color bars by century, we first need a variable that marks century. I do this with an `ifelse()` command. I then check my work with a `table()` command, making sure that I have roughly the number of observations for each century that I think I should. We also see what type of variable R created – `century` is character variable.

```
# color sections of bars?
summary(hurr$year)

##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      1851   1892   1934   1934   1976   2017

hurr$century <- ifelse(hurr$year < 1900, "1800s",
                      ifelse(hurr$year >= 1900 & hurr$year < 2000, "1900s", "2000s"))
table(hurr$century)

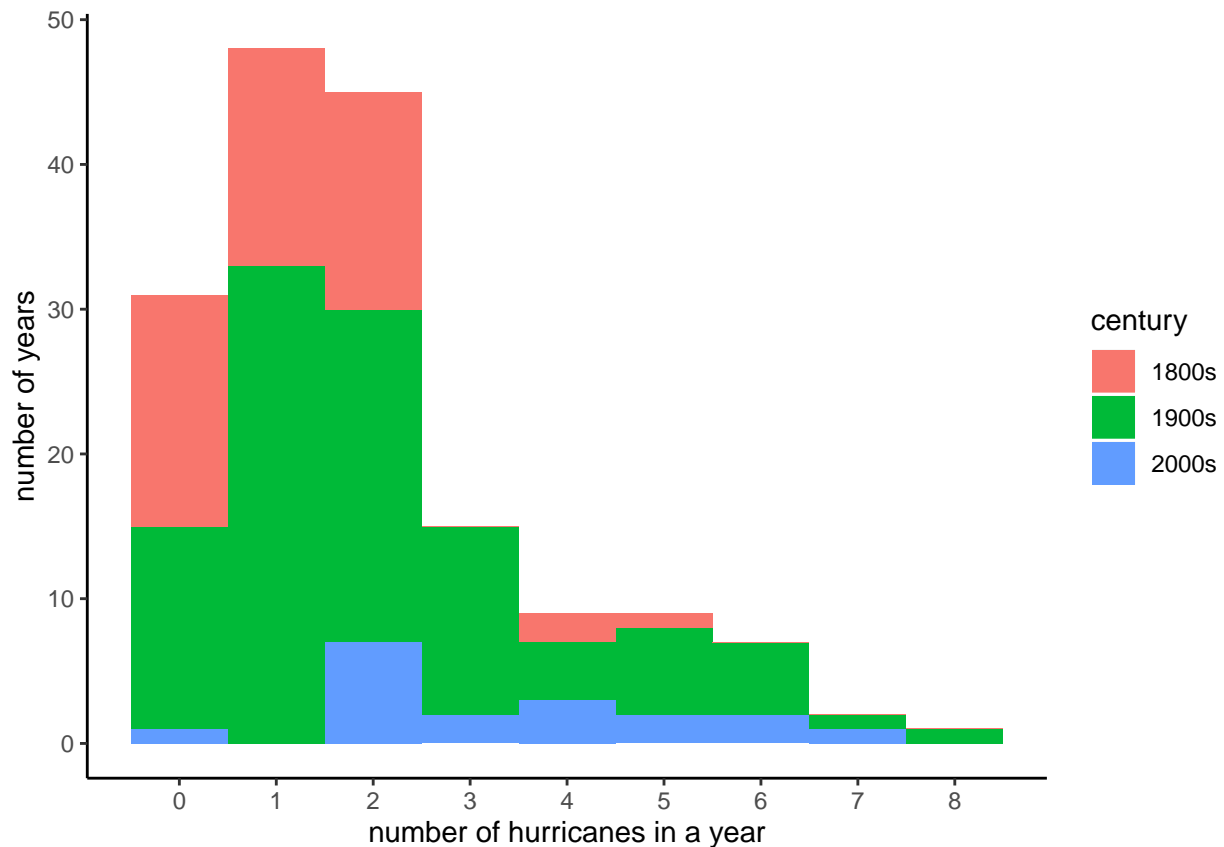
##
## 1800s 1900s 2000s
##    49   100    18

str(hurr$century)

## chr [1:167] "1800s" "1800s" "1800s" "1800s" "1800s" "1800s" "1800s" "1800s" ...
```

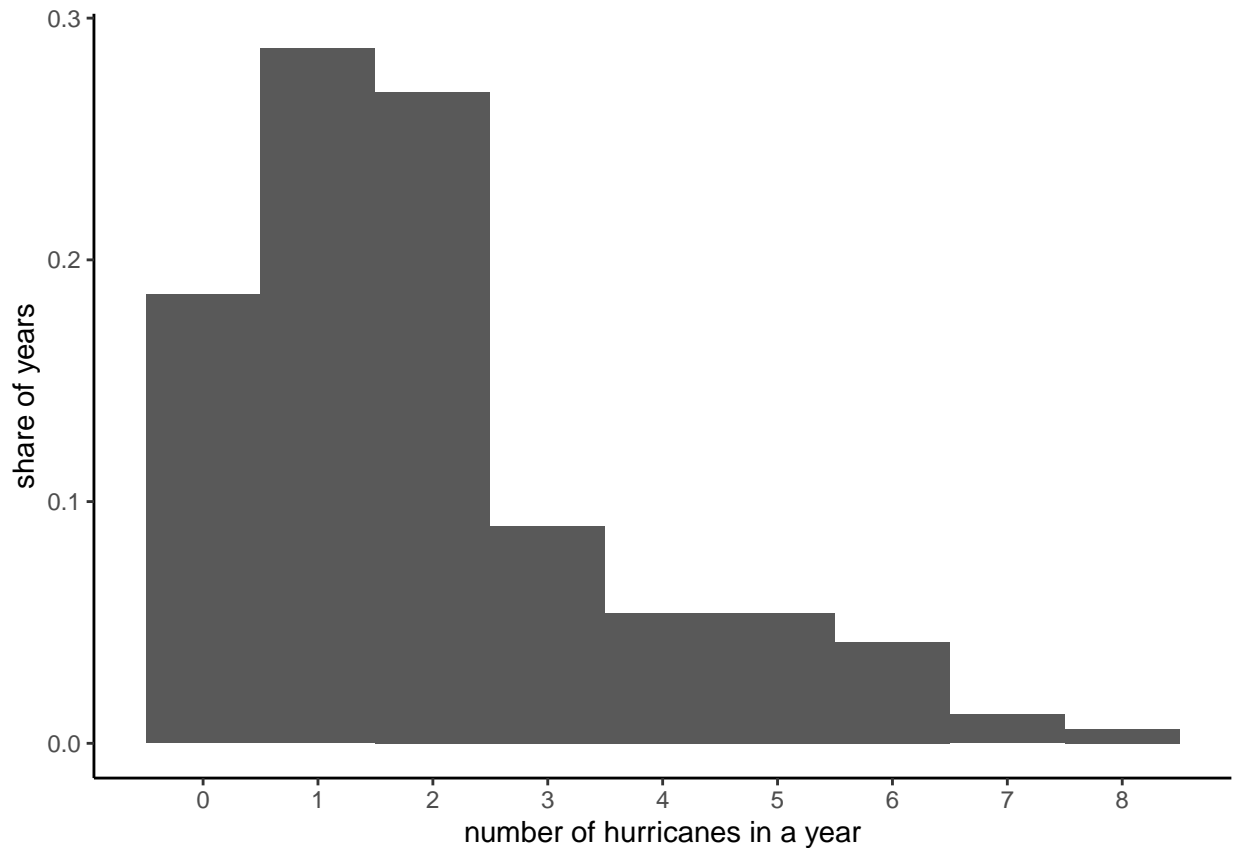
To add color to the graph, I add to the “aesthetics” command for the graph. I tell R to fill in the bars by the century, using `fill = century`. Note that `century` is a character variable; you don’t need a factor variable for the fill (but you do need a categorical variable!).

```
# color bars by century
c1 <- ggplot() +
  geom_histogram(data = hurr,
                mapping = aes(x = Major, fill = century),
                bins = 9) +
  scale_x_continuous(breaks = seq(0,8,1)) +
  labs(x = "number of hurricanes in a year",
       y = "number of years") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        axis.line = element_line(colour = "black"))
c1
```

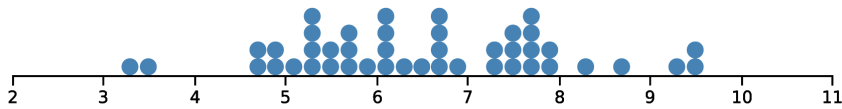


All the graphs we've made till now use the number of observations on the vertical axis. Sometimes it is more useful to show shares rather than numbers. You can do this easily by telling R that the y axis should be a share: `y = ..density..`. This will change the numbering on the y axis, but not the height of the bars (your homework asks why).

```
# show as percentage, rather than number
c1 <- ggplot() +
  geom_histogram(data = hurr,
                mapping = aes(x = Major, y = ..density..),
                bins = 9) +
  scale_x_continuous(breaks = seq(0,8,1)) +
  labs(x = "number of hurricanes in a year",
       y = "share of years") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        axis.line = element_line(colour = "black"))
c1
```



I wanted to also show you how you could make this plot with dots instead of bars, sort of like the example below. However, this requires using a new `geom` command, so we'll save it for later in the semester.



D. Load and examine a new dataset

Now we are going to do more histogram examples with a larger dataset where you cannot see all the values. We do this to work on your skills with larger datasets and to have enough data to make interesting histogram comparisons.

Specifically, we are using data on neighborhoods called block groups. A block group is a neighborhood of typically between 600 to 3,000 people. You can find examples of block groups by looking [here](#).

For each block group, we observe data on people and housing. These data come from surveys conducted by the Census Bureau and are 5-year averages from 2008-2012.

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](#)

Remember where you saved your data, and use that path in the `read.csv()` command below. We also use some of the commands we’ve seen before to explore the data.

```
# load the data
block.groups <- read.csv("H:/pppa_data_viz/2018/tutorials/lecture02/acs_bgs20082012_dmv_20180123.csv")

# How big is it?
dim(block.groups)

## [1] 9708 3063

# 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 ...
## $ STUSAB      : Factor w/  3 levels "dc","md","va": 1 1 1 1 1 1 1 1 1 ...
## $ SUMLEVEL    : int   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 NA ...
## $ B02005e2 : logi NA NA NA NA NA NA NA ...
## $ B02005e3 : logi NA NA NA NA NA NA NA ...
## $ B02005e4 : logi NA NA NA NA NA NA NA ...
## $ B02005e5 : logi NA NA NA NA NA NA NA ...
## $ B02005e6 : logi NA NA NA NA NA NA NA ...
## $ B02005e7 : logi NA NA NA NA NA NA NA ...
## $ B02005e8 : logi NA NA NA NA NA NA NA ...
## $ B02005e9 : logi NA NA NA NA NA NA NA ...
## $ B02005e10 : logi NA NA NA NA NA NA NA ...
## $ B02005e11 : logi NA NA NA NA NA NA NA ...
## $ B02005e12 : logi NA NA NA NA NA NA NA ...
## $ B02005e13 : logi NA NA NA NA NA NA NA ...
## $ B02005e14 : logi NA NA NA NA NA NA NA ...
## $ B02005e15 : logi NA NA NA NA NA NA NA ...
## $ B02005e16 : logi NA NA NA NA NA NA NA ...
## $ B02005e17 : logi NA NA NA NA NA NA NA ...
## $ B02005e18 : logi NA NA NA NA NA NA NA ...
## $ B02005e19 : logi NA NA NA NA NA NA NA ...
## $ B02005e20 : logi NA NA NA NA NA NA NA ...
## $ B02005e21 : logi NA NA NA NA NA NA NA ...
## $ B02005e22 : logi NA NA NA NA NA NA NA ...
## $ B02005e23 : logi NA NA NA NA NA NA NA ...
## $ B02005e24 : logi NA NA NA NA NA NA NA ...
## $ B02005e25 : logi NA NA NA NA NA NA NA ...
## $ B02005e26 : logi NA NA NA NA NA NA NA ...
## $ B02005e27 : logi NA NA NA NA NA NA NA ...
## $ B02005e28 : logi NA NA NA NA NA NA NA ...
## $ B02005e29 : logi NA NA NA NA NA NA NA ...
## $ B02005e30 : logi NA NA NA NA NA NA NA ...
## $ B02005e31 : logi NA NA NA NA NA NA NA ...
## $ B02005e32 : logi NA NA NA NA NA NA NA ...
## [list output truncated]

```

E. Basic histograms, bigger data

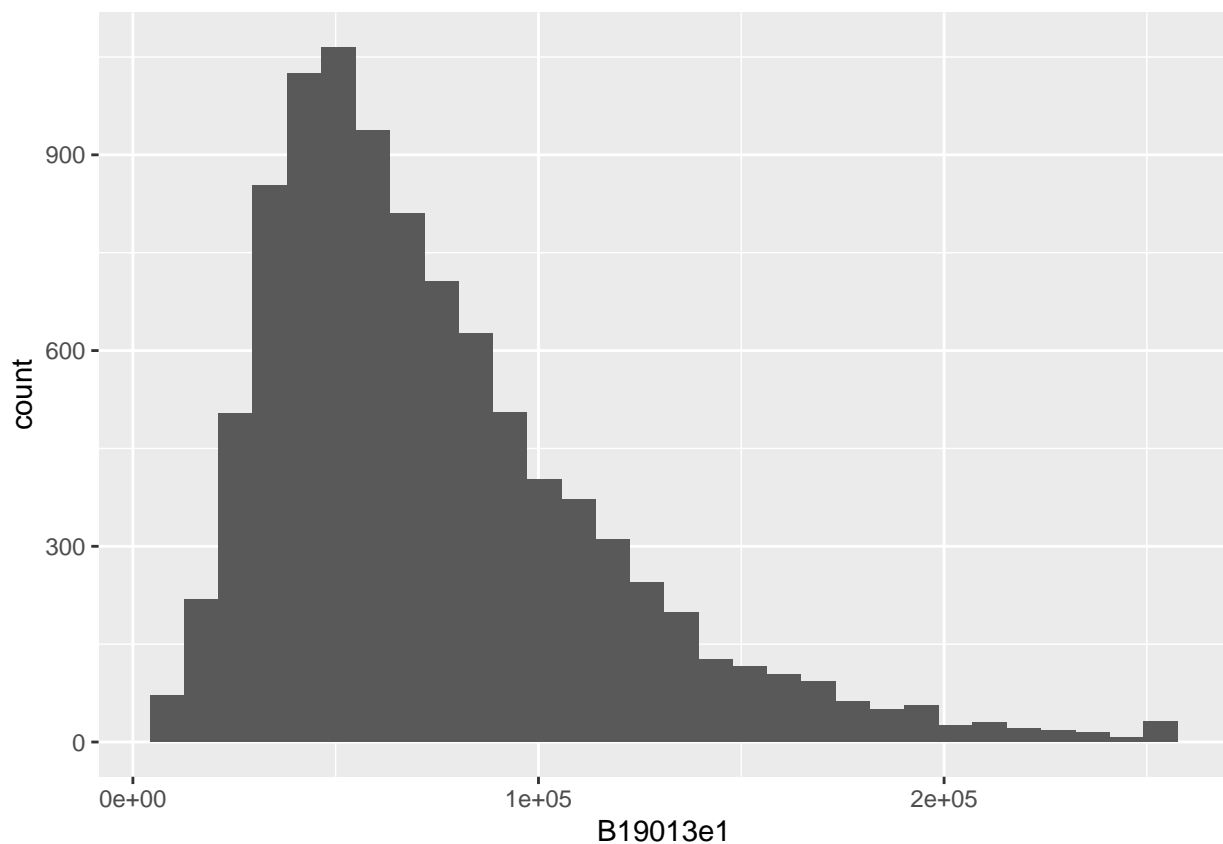
Let's return to histograms and plot the distribution of median household income over the past 12 months, or B19013e1.

Here is what R does with the most basic command:

```
## this makes a basic histogram
e1 <- ggplot() +
  geom_histogram(data = block.groups,
                mapping = aes(x=B19013e1))
e1
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

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



The warning message tells us that there are 108 observations with no income data. This is a small share of the almost 10,000 observations and there are some block groups with very small population for which the census does not report income. We can check if this is the case by looking at the average population (B02001e1) when income is missing (`is.na(B19013e1)`).

```
# overall population
print("all block groups")
```

```
## [1] "all block groups"
```

```
summary(block.groups$B02001e1)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

```
##          0      938      1342      1484      1892      13494
# for block groups with missing income
print("block groups with missing income")

## [1] "block groups with missing income"

mis.inc <- block.groups[which(is.na(block.groups$B19013e1)),]
summary(mis.inc$B02001e1)

##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      0.0      0.0      0.0   668.6   689.2  8469.0
```

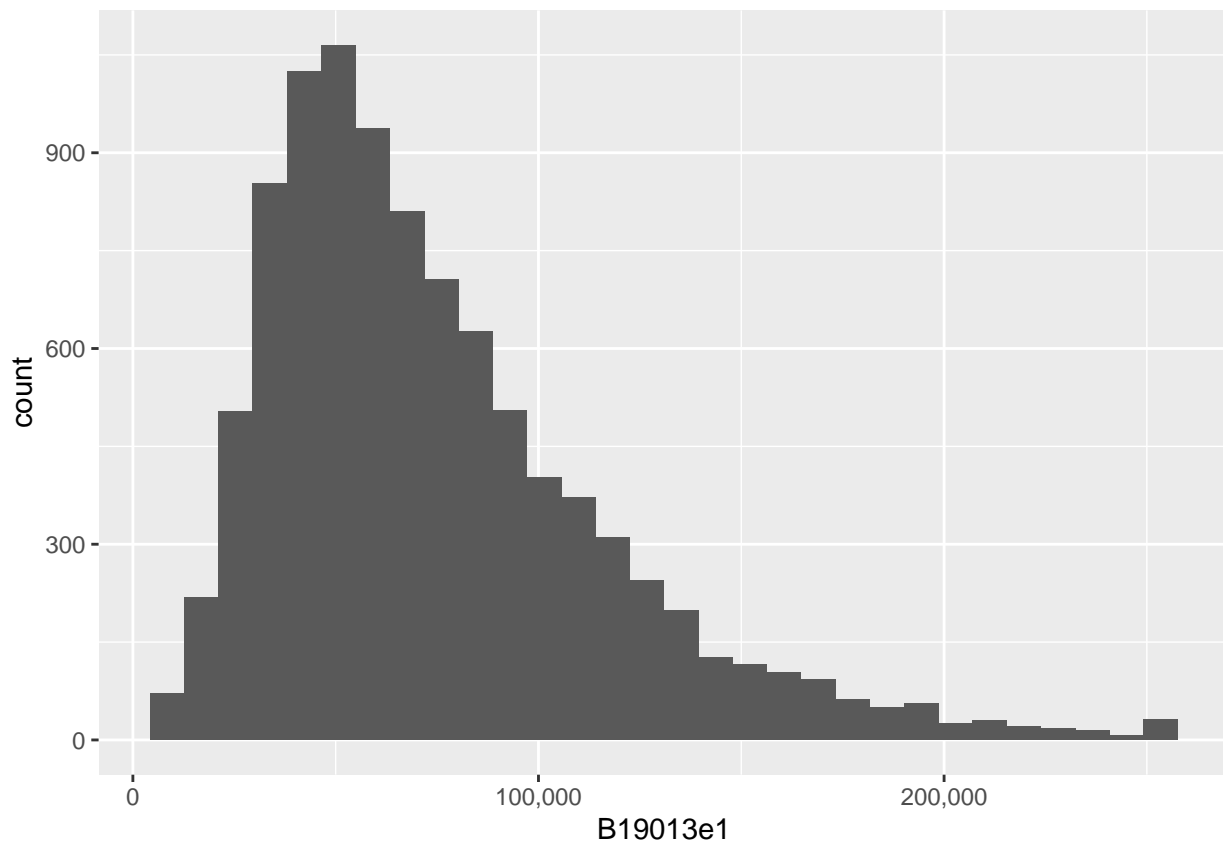
From this, we learn that more than half of the block groups with no income information have no people (good!). So somewhere between 25 and 49 observations have people but no income. We will ignore them for now.

Before we do anything else, let's make the horizontal axis legible by using the `scales` package and the `scale_x_continuous()` option.

```
# fix axis to be legible
e11 <- ggplot() +
  geom_histogram(data = block.groups,
                mapping = aes(x=B19013e1)) +
  scale_x_continuous(labels = comma)
e11
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

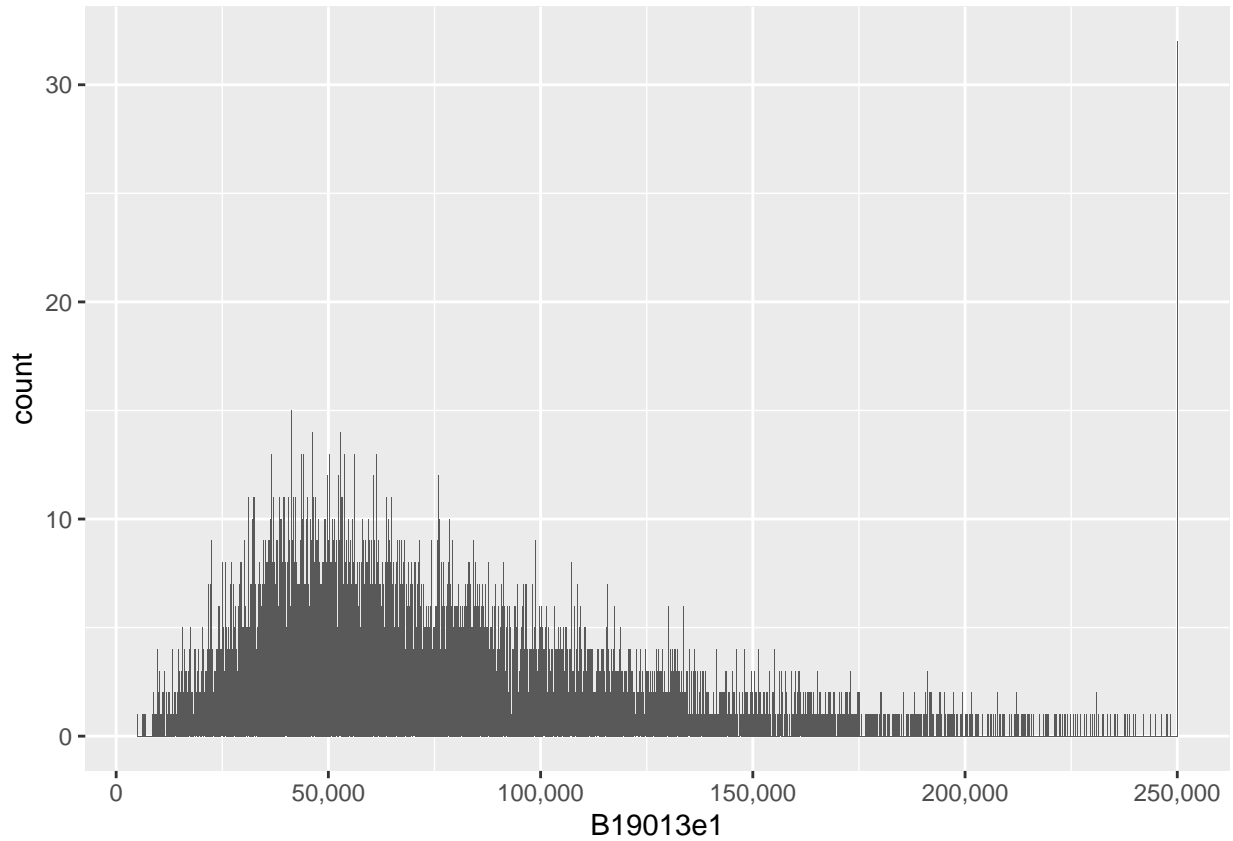
```
## Warning: Removed 108 rows containing non-finite values (stat_bin).
```



Now that we have a wealth of data, we can explore how changing the bin width (the width of the vertical bar, or how many values of income you group together) impacts the look of the graph. In the first, we add up how many block groups are in each \$50 bin. In other words, the number of block groups where the median income is between 0 and \$50, the number of block groups where the median income is between \$50 and \$100, etc.

```
## here we begin changing the width of the histogram bins
e2 <- ggplot() +
  geom_histogram(data = block.groups,
                mapping = aes(x=B19013e1),
                binwidth = 50) +
  scale_x_continuous(labels = comma)
e2
```

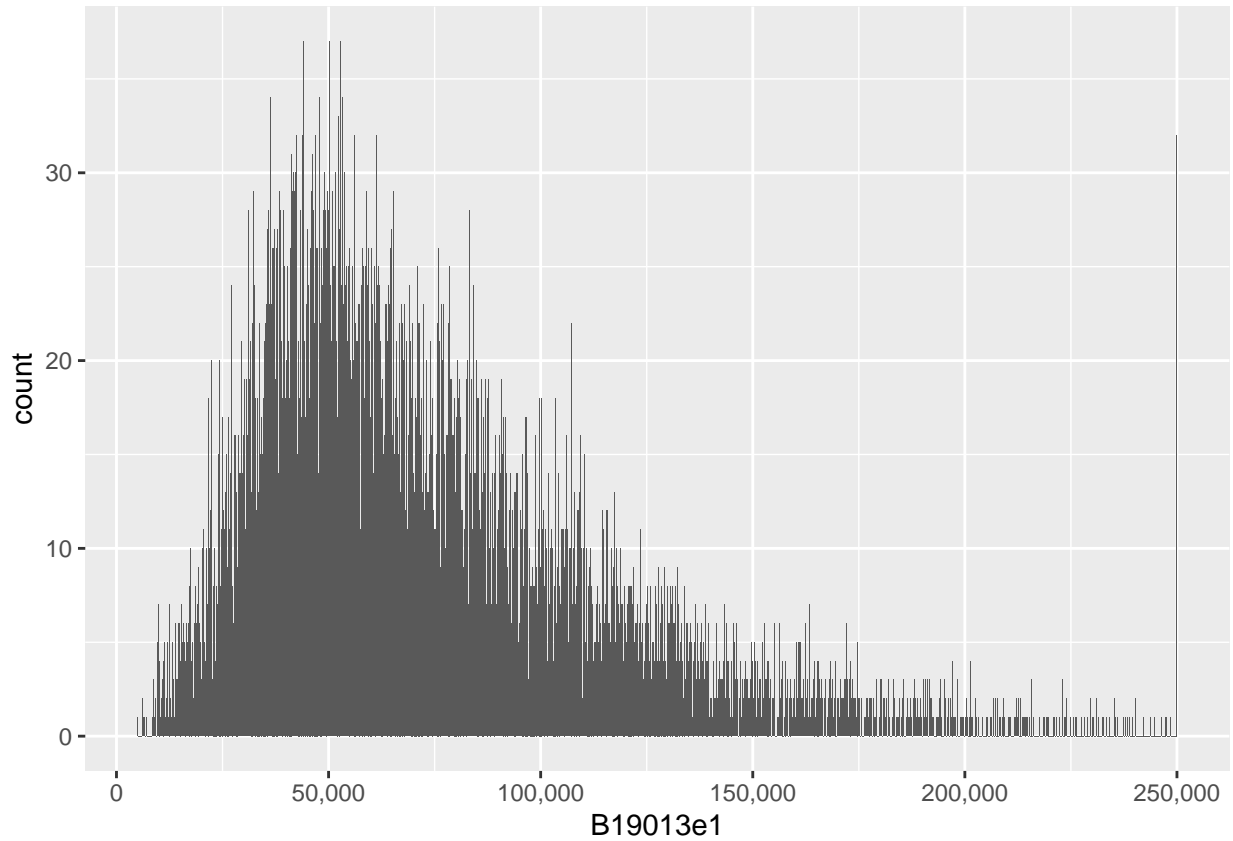
```
## Warning: Removed 108 rows containing non-finite values (stat_bin).
```



This looks crazy! Bins are so small that the distribution looks very jagged. What if we make the bins 4 times wider?

```
e3 <- ggplot() +  
  geom_histogram(data = block.groups,  
                mapping = aes(x=B19013e1),  
                binwidth = 200) +  
  scale_x_continuous(labels = comma)  
e3
```

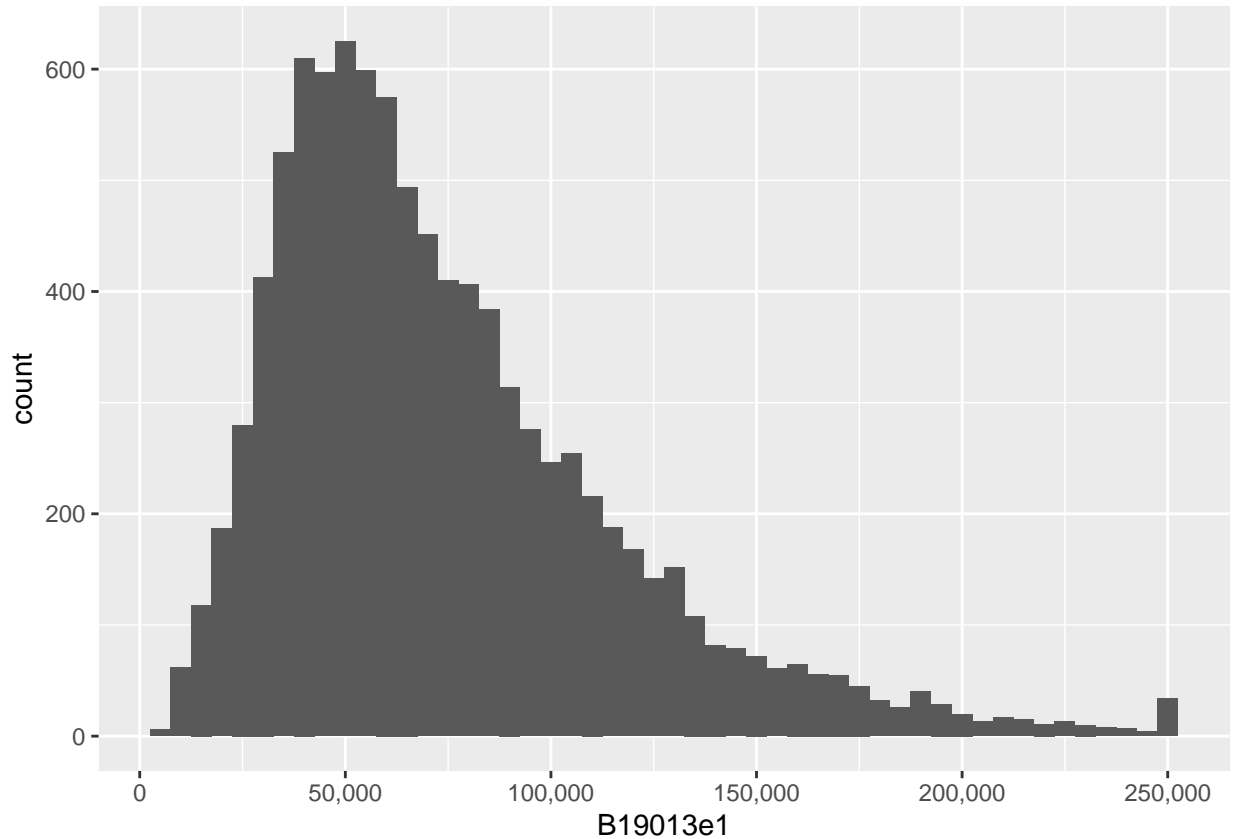
```
## Warning: Removed 108 rows containing non-finite values (stat_bin).
```



Still a little odd. Let's make them 25 times bigger, so a width of \$5000.

```
e4 <- ggplot() +  
  geom_histogram(data = block.groups,  
                mapping = aes(x=B19013e1),  
                binwidth = 5000) +  
  scale_x_continuous(labels = comma)  
e4
```

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

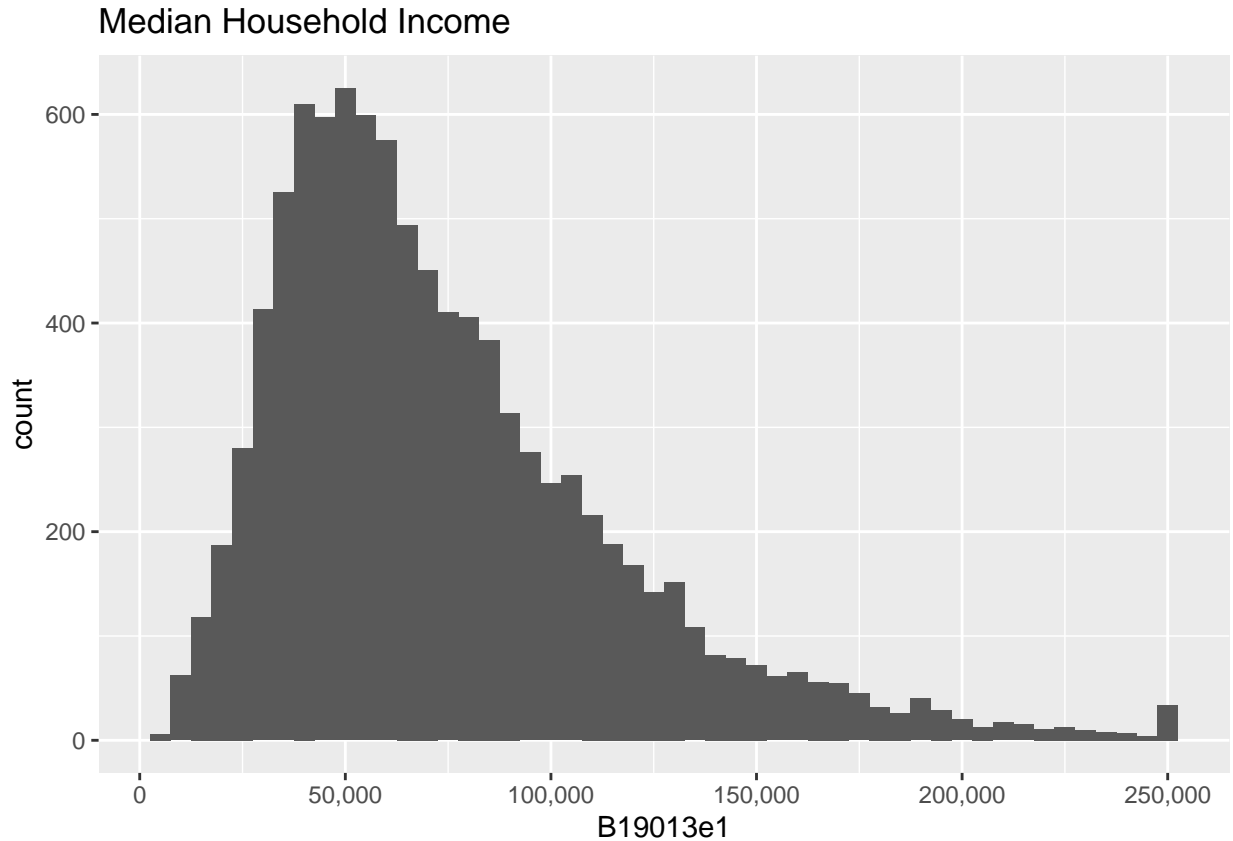


Note that the choice of bin is very important to the final look of the graph. Also notice that very small bins make the top-coded final category (for all block groups with a median income greater than 250,000, the Census reports 250,000) look big.

Even with the small number of plots we just made, we can get lost without titles. (Though frequently I end up omitting the title in the very final product because I put the plot in a presentation slide and put the title on in the slide itself.) Here again we use `labs` for title and axis labels.

```
## make things legible: titles and axes
f1 <- ggplot() +
  geom_histogram(data = block.groups,
                mapping = aes(x=B19013e1),
                binwidth = 5000) +
  labs(title = "Median Household Income") +
  scale_x_continuous(labels = comma)
f1
```

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



We will talk more later about the power of titles and axis labels. You should consider them key to any decent final product.

F. Other ggplot options for types of histograms

Here we explore density curves and subsets.

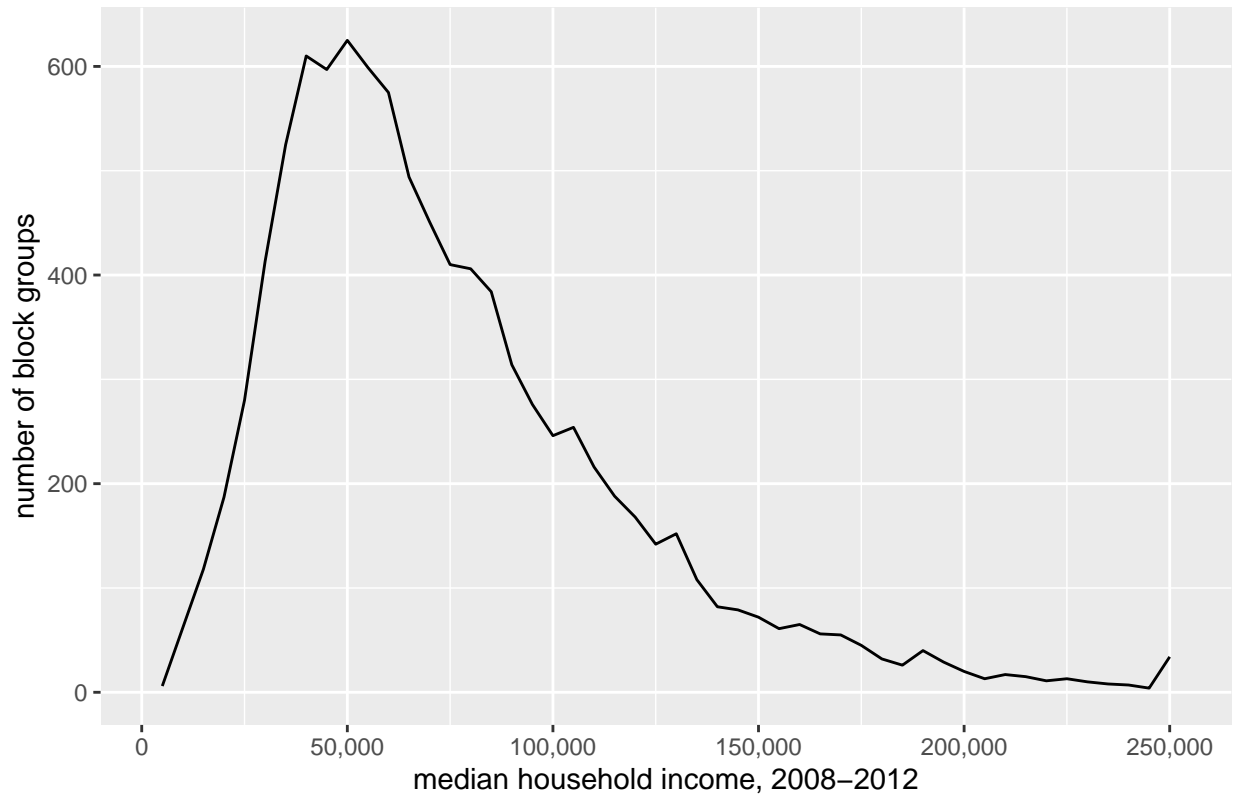
F.1. Density curves

We begin by using curves. One way to get a curve is to use `stat_bin()`, which calculates statistics by bin.

```
# density curve instead of bars
f3 <- ggplot() +
  stat_bin(data = block.groups,
           mapping = aes(x=B19013e1),
           binwidth = 5000,
           geom = "line") +
  labs(title = "Median Household Income",
       x="median household income, 2008-2012",
       y="number of block groups") +
  scale_x_continuous(labels = comma)
f3
```

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

Median Household Income

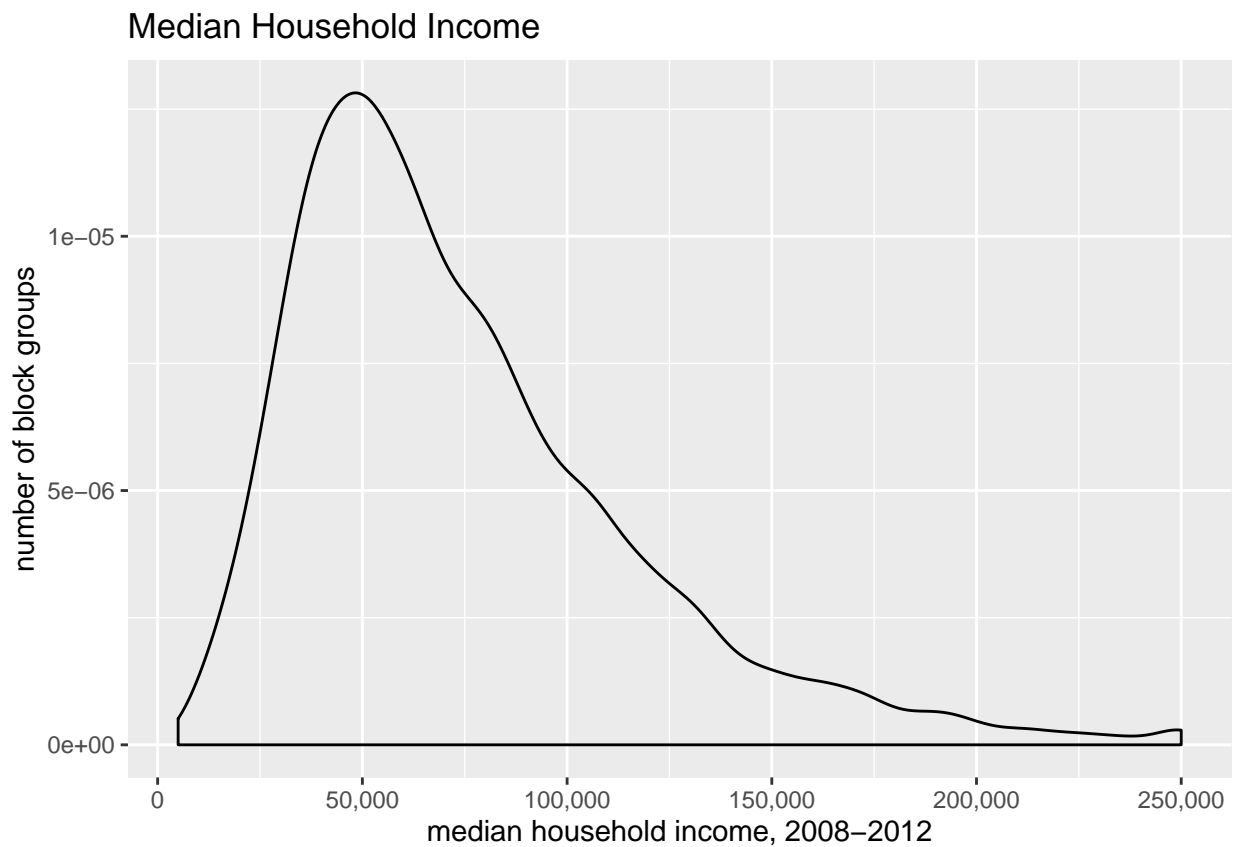


Alternatively, you can use `geom_density()`, which makes a smoothed density curve (very close to a continuous histogram). Whether or not smoothing is appropriate depends on your data and goals.

```
# with geom_density
ggplot() +
  geom_density(data = block.groups,
              mapping = aes(x=B19013e1),
              binwidth = 5000) +
  stat_bin(geom = "line") +
  labs(title = "Median Household Income",
       x="median household income, 2008-2012",
       y="number of block groups") +
  scale_x_continuous(labels = comma)
```

```
## Warning: Ignoring unknown parameters: binwidth
```

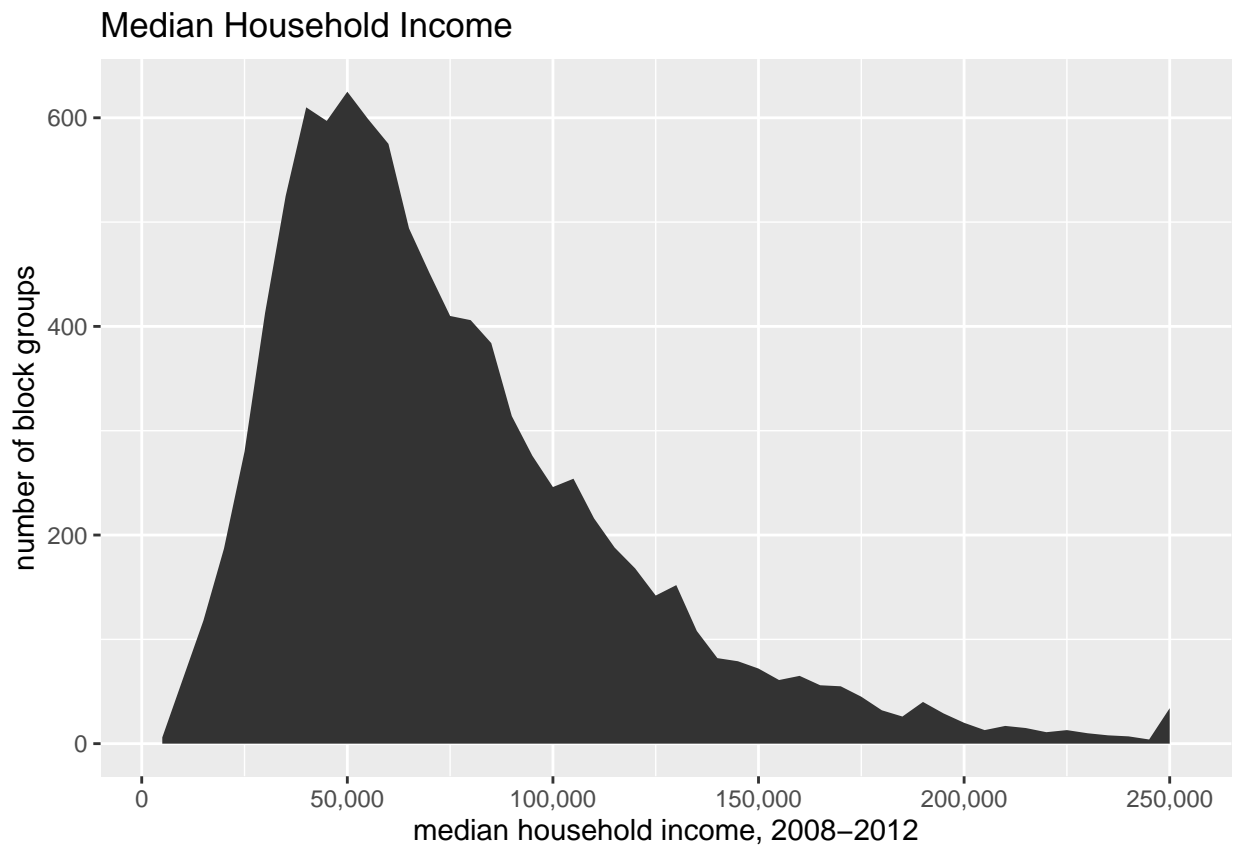
```
## Warning: Removed 108 rows containing non-finite values (stat_density).
```



It is sometimes useful to color in the area under the line, which you can do by changing `geom="line"` to `geom="area"`:

```
ggplot() +  
  stat_bin(data = block.groups,  
           mapping = aes(x=B19013e1),  
           binwidth = 5000,  
           geom = "area") +  
  labs(title = "Median Household Income",  
        x="median household income, 2008-2012",  
        y="number of block groups") +  
  scale_x_continuous(labels = comma)
```

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



I don't view these as superior to the previous final histogram because to me they seem more difficult to understand without any other benefit.

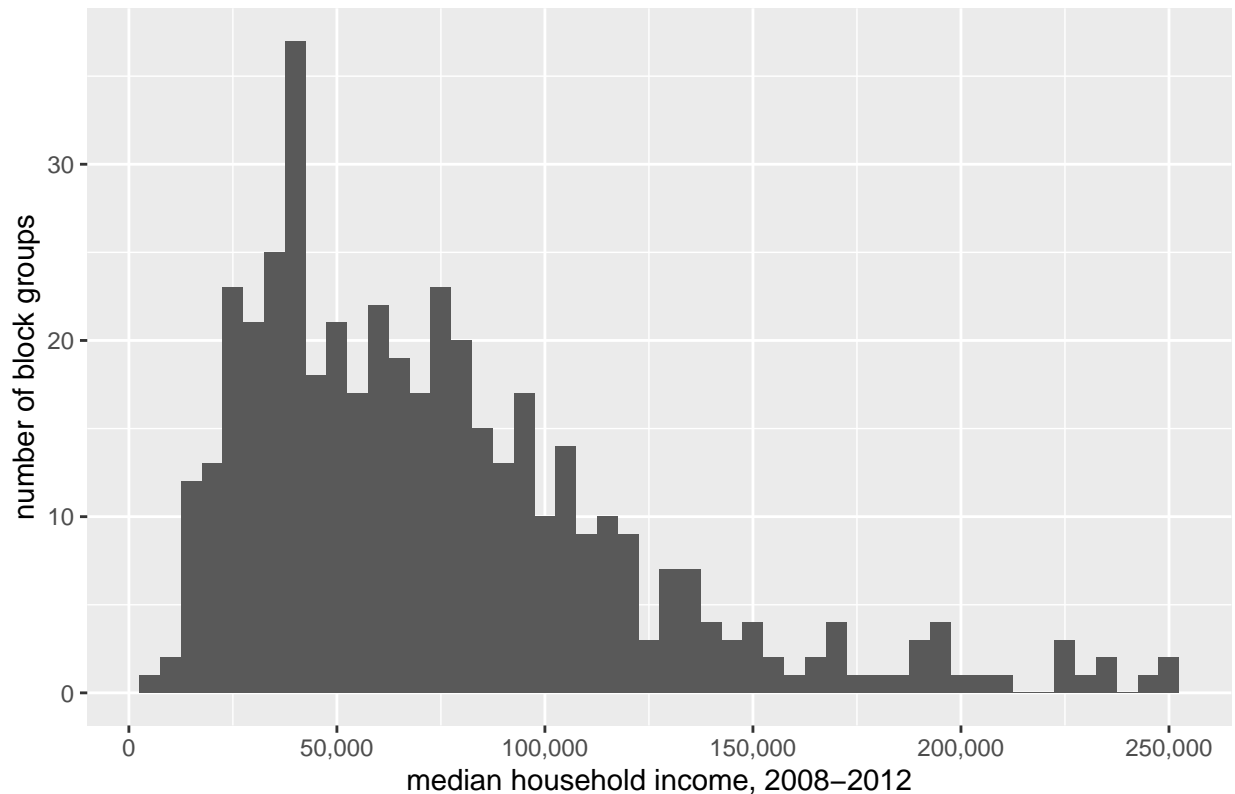
F.2. Subgroups

We can limit the analysis only to a subgroup – this can frequently be useful and enlightening. Note that we use the double equals sign for evaluating (here we are evaluating, not assigning).

```
## subsetting to only certain data
# one way
ggplot() +
  geom_histogram(data = block.groups[which(block.groups$STATE == 11),],
                mapping = aes(x=B19013e1),
                binwidth = 5000) +
  stat_bin(geom = "area") +
  labs(title = "Median Household Income",
       x="median household income, 2008-2012",
       y="number of block groups") +
  scale_x_continuous(labels = comma)
```

```
## Warning: Removed 3 rows containing non-finite values (stat_bin).
```

Median Household Income



There are many ways to subset data in R. An alternative, that I show but do not run, is to use the filter command. The method above and the method below are entirely equivalent.

```
# alternative
dc.only <- filter(.data = block.groups, STATE == 11)
dc.only <- block.groups[which(block.groups$STATE == 11),]
ggplot() +
  geom_histogram(dc.only,
                mapping = aes(x=B19013e1, na.rm = TRUE),
```

```
      binwidth = 5000) +
stat_bin(geom = "area") +
labs(title = "Median Household Income: DC Only",
      x="median household income, 2008-2012",
      y="number of block groups") +
scale_x_continuous(labels = comma)
```

Compare this distribution to the previous to see how DC's relative distribution. Of course, looking across two graphs is not an ideal comparison method. We'll work on this as we go along.

G. Comparing the three jurisdictions

Now we want to show the block group income distribution in DC, MD and VA together on one chart.

G.1. Showing by group

If you have used other programming languages, you might think about using a loop. However, loops are not very R-like. Almost everything that a loop does is better done with a matrix or list in R. You sometimes have to do heroic coding to make a loop work like you'd like, and heroics are not needed.

Rather than a loop, use `ggplot`'s built-in tools to make graphs by group.

The important missing ingredient before we do this is to recall our discussion of R's "factor" variables. In essence, a factor variable is a variable that takes on a limited number of values. This may also be known as a categorical variable.

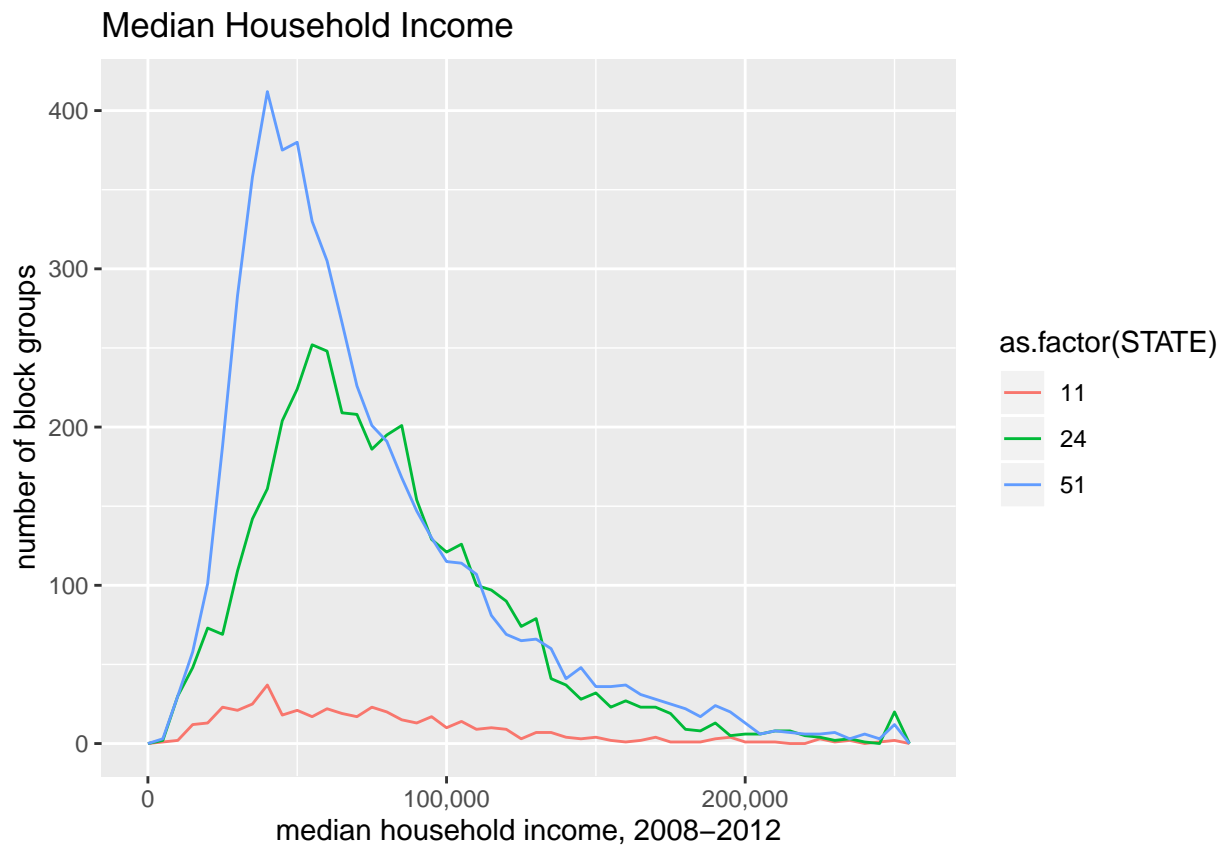
We care about factors here because we can only make graphs by factor variables. If something is not already a factor variable, but it takes on a limited set of values, we can pretend it is by putting `as.factor()` around it.

Here we use yet another potential command for creating histograms: `geom_freqpoly()`. This command also generates a density curve, but does not smooth like `geom_density()`.

The code below shows the number of block groups by state for median income:

```
h1 <- ggplot() +  
  geom_freqpoly(data = block.groups,  
               mapping = aes(B19013e1, color = as.factor(STATE)),  
               binwidth=5000) +  
  labs(title = "Median Household Income",  
       x="median household income, 2008-2012",  
       y="number of block groups") +  
  scale_x_continuous(labels = comma)  
h1
```

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

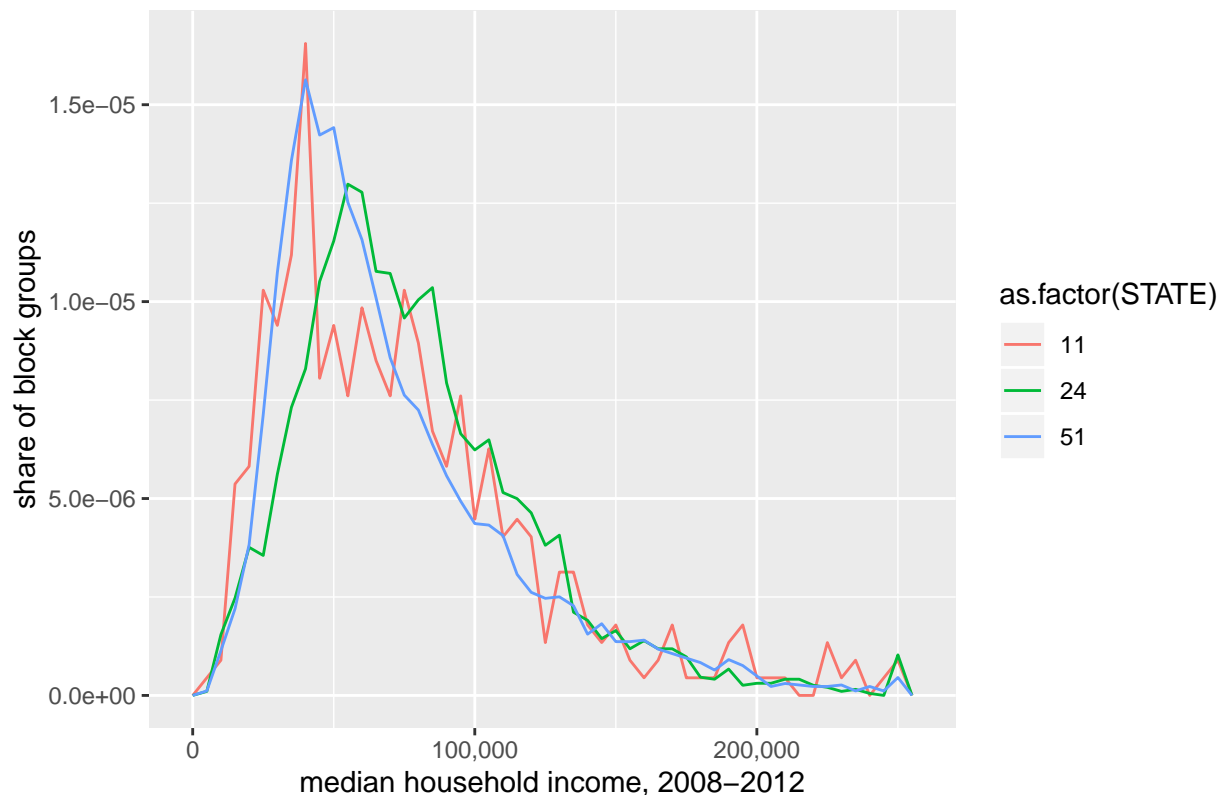


One might prefer to show the share of block groups, rather than the number when making this cross-state comparison. To do so, tell R that the y variable is `..density..`.

```
h1 <- ggplot() +
  geom_freqpoly(data = block.groups,
               mapping = aes(x = B19013e1, y = ..density.., color = as.factor(STATE)),
               binwidth=5000) +
  labs(title = "Share of Block Groups by Median Household Income by State in the DMV",
       x="median household income, 2008-2012",
       y="share of block groups") +
  scale_x_continuous(labels = comma)
h1
```

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

Share of Block Groups by Median Household Income by State in the DM

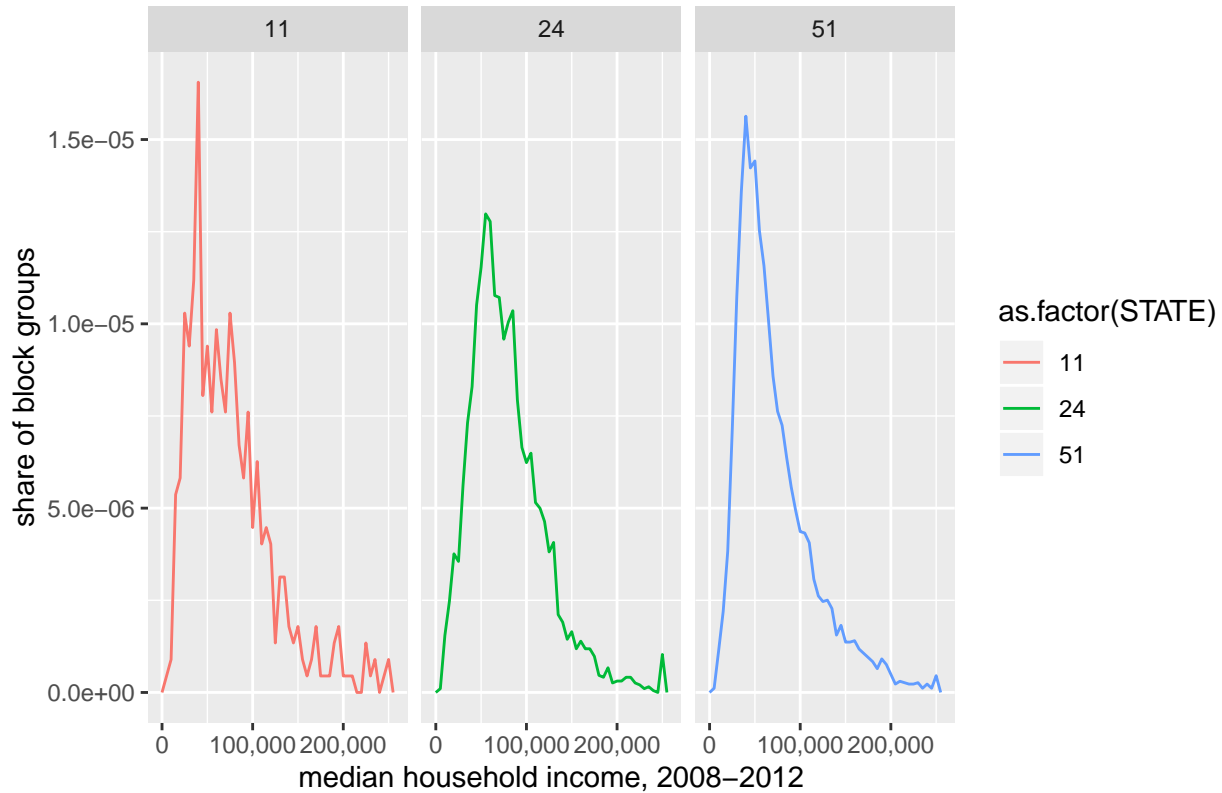


Alternatively, you can show multiple small versions of the same chart using `facet_wrap()`:

```
h1 <- ggplot() +
  geom_freqpoly(data = block.groups,
               mapping = aes(x = B19013e1, y = ..density.., color = as.factor(STATE)),
               binwidth=5000) +
  facet_wrap(block.groups$STATE) +
  labs(title = "Share of Block Groups by Median Household Income by State in the DMV",
       x="median household income, 2008-2012",
       y="share of block groups") +
  scale_x_continuous(labels = comma)
h1
```

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

Share of Block Groups by Median Household Income by State in the DM



G.2. Cleaning up the graph

There are still plenty of unpleasant things about these graphs. In this list, I would include

- probably bad smoothing making the DC line more jerky than the others
- density in illegible units
- hard to read legend
- unhelpful color scheme
- potentially useless grey background
- poor axis labels
- lines too jerky to convey desired idea
- state ids should be abbreviations or names

We'll now fix some of these issues and leave the remainder for future tutorials.

To make the states legible, we need to re-name the factor levels. Below we first make a factor state variable called `state.factor` using the `as.factor()` command. We then look at the levels of this factor, which are the fips state codes.

```
# fix state factor to have names  
block.groups$state.factor <- as.factor(block.groups$STATE)  
levels(block.groups$state.factor)
```

```
## [1] "11" "24" "51"
```

These are great for data processing and merging, but not good for conveying to people not deeply familiar with federal information processing codes. So we tell R to reassign the levels of the factor as below. We then check using `levels()` that we did what we thought we did – which is have abbreviations instead of numbers.

```
levels(block.groups$state.factor) <- list("DC" = "11", "MD" = "24", "VA" = "51")
levels(block.groups$state.factor)
```

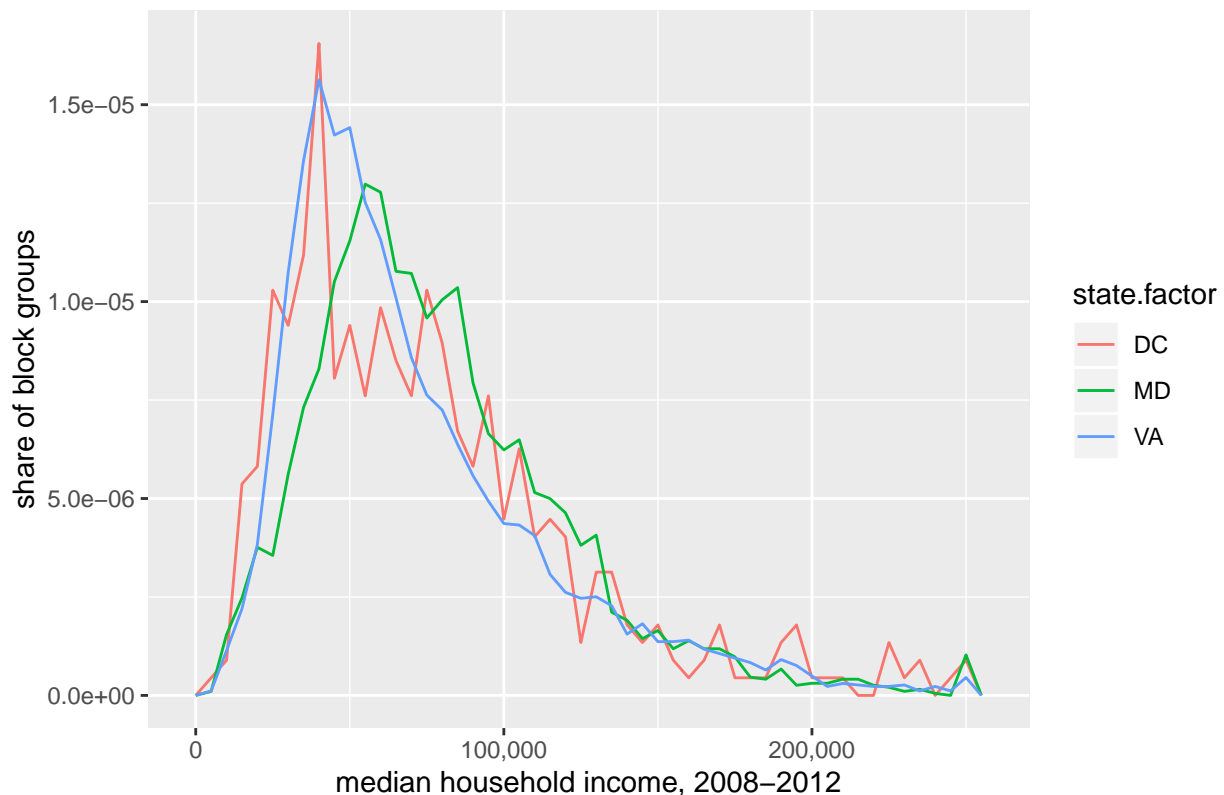
```
## [1] "DC" "MD" "VA"
```

Now that we've fixed this, we re-do the chart with the new `state.factor` variable.

```
# with state as a named factor
h1 <- ggplot() +
  geom_freqpoly(data = block.groups,
               mapping = aes(x = B19013e1, y = ..density.., color = state.factor),
               binwidth=5000) +
  labs(title = "Share of Block Groups by Median Household Income by State in the DMV",
       x="median household income, 2008–2012",
       y="share of block groups") +
  scale_x_continuous(labels = comma)
h1
```

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

Share of Block Groups by Median Household Income by State in the DM



But now we have an uninterpretable y axis. We fix this by telling R to make a sequence of numbers and to not use scientific notation (`labels = commas`).

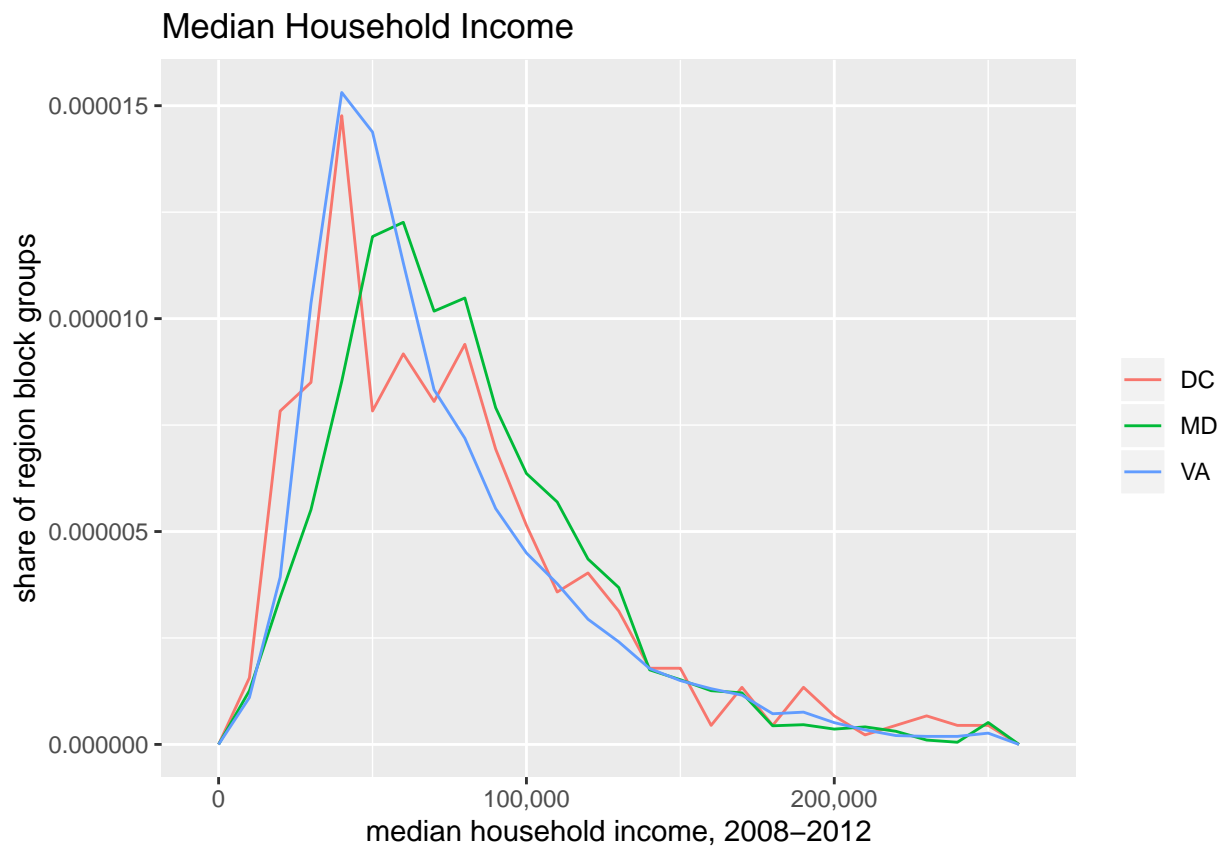
```
# F.5.2. fix bins for everyone, and fix y axis to legible
f2 <- ggplot() +
```

```

geom_freqpoly(data = block.groups,
              mapping = aes(B19013e1, ..density.., color = state.factor),
              binwidth=10000) +
labs(title = "Median Household Income",
     x="median household income, 2008-2012",
     y="share of region block groups") +
scale_y_continuous(breaks = seq(0,0.000015,0.000005),
                  labels = comma) +
scale_x_continuous(labels = comma) +
theme(legend.title = element_blank())
f2

```

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



H. Saving your output

Finally, you will sometimes need to save your graph, not just see it in the plot window. You can use the export button in the plot window.. but then you realize you've made a mistake in your graph and you export .. and you find another mistake...

Just write some code and you only need to re-run this code. I have an example below that does this. There are many many more options in the [full documentation](#) than the ones I use here.

Below I tell R where I want to save the final product (`filename =`) and I name the full path with the extension for the file type I want to use (`".jpeg"`). R supports a variety of file types if you prefer another one.

I tell R which plot I'd like to save, which is plot `f2` that I created above. We also need to tell R what kind of file we want using the `driver=` command. Also tell R how big you want the final product to be with `width=` and `height=` and the units that you're using ("in", "cm", "mm").

The final component below is the number of dots per inch, or dpi. 300 dpi is sufficient for digital display.

```
ggsave(filename = "H:/pppa_data_viz/2020/tutorials/tutorial_04/20200204_median_income_freq.jpg",
        plot = f2,
        device = "jpeg",
        width = 11,
        height = 8,
        units = "in",
        dpi = 300)
```

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

I. Homework: Try it yourself

1. When we change from a histogram that reports the number of years by number of hurricanes to share of years by number of hurricanes, the numbering on the y axis changes, but the height of the bars is constant. Why?
2. Why does the income histogram look so odd when bins are very narrow? Explain in your own words.
3. We created the histogram with income distribution for all three states. In one of those we have the number of block groups on the vertical axis; in the other the share. Explain (a) what the share is of (number of block groups from what total number) and (b) why the share makes a more reasonable comparison than the levels.
4. Create a sequence of odd numbers from 1 to 11, inclusive. (Not a graph, just a list of numbers.)
5. Choose a new variable for which to make a histogram from the block group data (not income!). Plot it by the three states, grouped or faceted as you please.
6. Make two histograms from a dataset (one dataset is sufficient) that we have not used in this class. A good place to start if you have no favorite data is Open Data DC [here](#).
 - use whatever variables you please
 - write 2 to 3 sentences describing what you find