Tutorial 4: Bar Charts

Leah Brooks

February 18, 2019

Today is dedicated to bar charts: a workhorse data visualization product. Bar charts are great for comparisons across categories, and sometimes for comparisons across groups. Today, in addition to basic bar charts, you'll learn how to make

- grouped bars
- faceted bars
- stacked bars

Like last class, we'll start with a small dataset and then turn to a larger one.

To achieve bar charts that communicate, you'll frequently need to modify the underlying data frame. We also learn some steps to do this. Specifically, we review some R commands we learned previously and introduce new R commands, including

- `as.factor()`
- `levels()`
- `mutate()`
- going from wide data to long data with `gather()`

A. What to start with

As always, we need to load the packages I've listed below. If R tells you you don't have these packages, you can use `install.packages()` to load them.

```r
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.4.4
```

```r
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
```

```r
library(tidyr)
```

```
## Warning: package 'tidyr' was built under R version 3.4.4
```

```r
library(scales)
```

Alternatively, you can use the command `require()` which checks for the package and loads it if it does not exist.

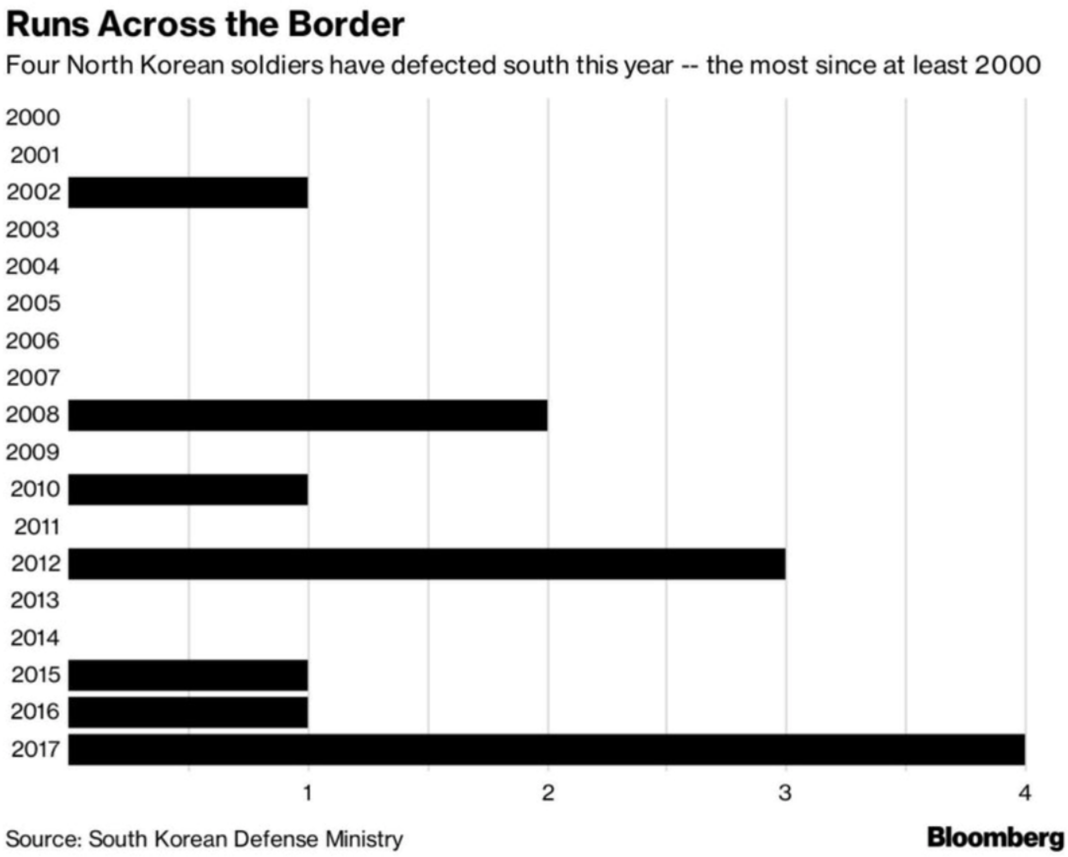


Figure 1: North Korean Defectors

B. Basic Bar Chart

Last year, a student brought Figure 1 to class as her “Good/Bad/Ugly” submission. We use these data to get started with bar chart examples.

B.1. Input data

Figure 1’s bar chart is not the best way to make the point the author was trying to convey: it’s odd to use a bar chart for time, and each year only takes on values 1 to 4, but the chart seems to suggest that you could get 1.5.

We begin by modifying this chart. To modify, we need a dataframe. Below I create a small dataframe based on the information in the chart and print it to make sure it seems ok.

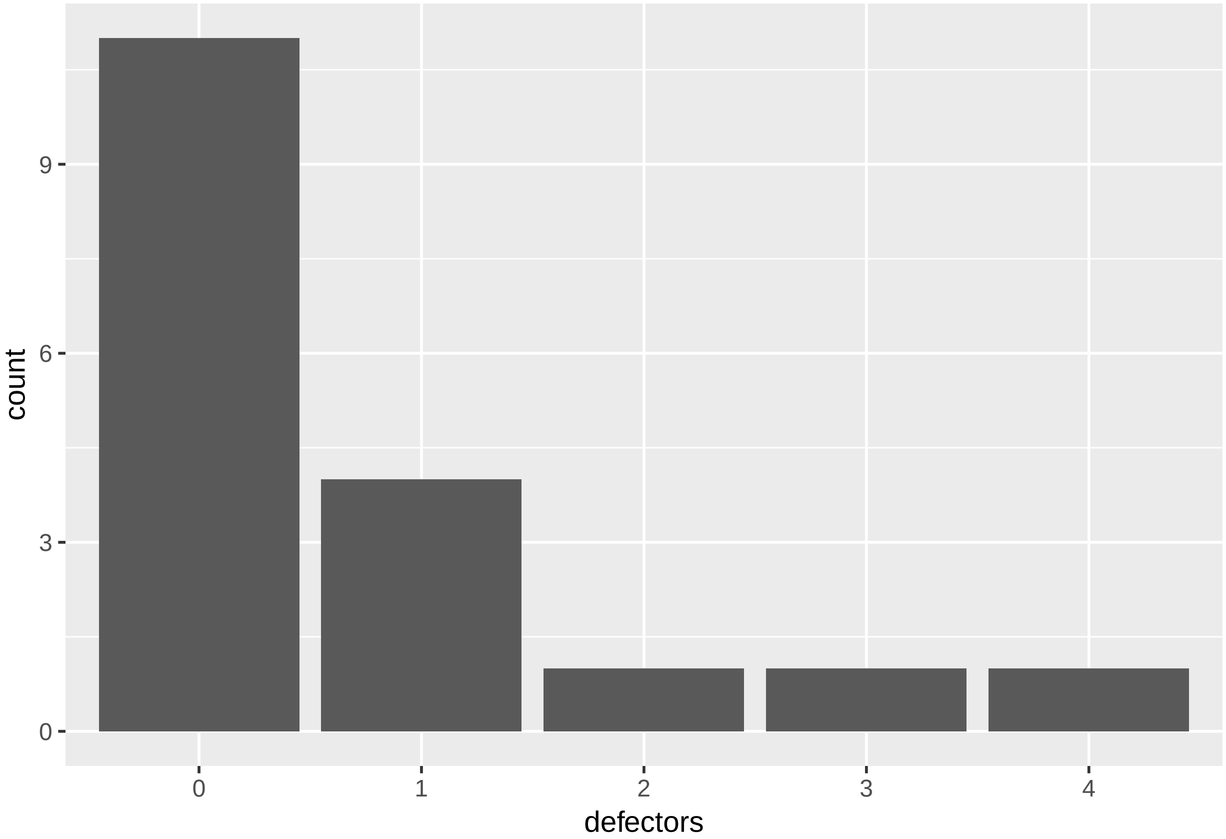
```
# load north korean data
nkd <- data.frame(year = c("2000","2001","2002","2003","2004",
                           "2005","2006","2007","2008","2009",
                           "2010","2011","2012","2013","2014",
                           "2015","2016","2017"),
                  defectors = c("0","0","1","0","0",
                                "0","0","0","2","0",
                                "1","0","3","0","0",
                                "1","1","4"))
nkd
```

```
##    year defectors
## 1  2000         0
## 2  2001         0
## 3  2002         1
## 4  2003         0
## 5  2004         0
## 6  2005         0
## 7  2006         0
## 8  2007         0
## 9  2008         2
## 10 2009         0
## 11 2010         1
## 12 2011         0
## 13 2012         3
## 14 2013         0
## 15 2014         0
## 16 2015         1
## 17 2016         1
## 18 2017         4
```

B.2. Regular bar chart

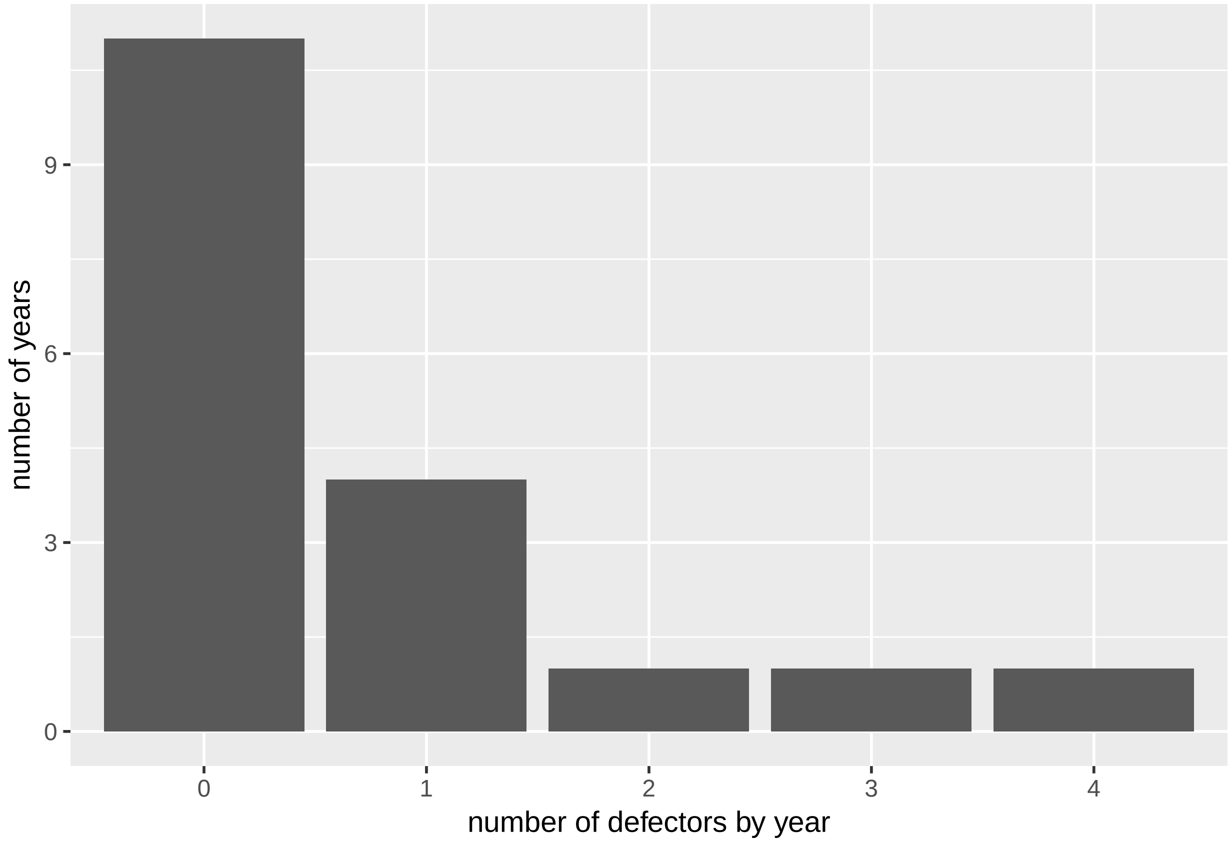
Let's make the most basic possible bar chart, by just adding the `geom_bar()` command. Note that our dataset is at the year level, and that R adds up the number of years for each number of defectors.

```r
# make a bar chart
a2 <- ggplot() +
        geom_bar(data = nkd, aes(x=defectors))
a2
```



This is missing any kind of helpful label to be even vaguely interpretable, so using the `labs()` command we introduced last class, I add labels:

```
a2a <- ggplot() +
        geom_bar(data = nkd, aes(x=defectors)) +
        labs(x = "number of defectors by year", y="number of years")
a2a
```



B.3. Colors within bars

What the above graph is missing is some sense of time. Are the high-defection years recent? To figure this out, I color in the bars that are from the last five years. To "color in", you need to have a variable that indicates whether a year is in the last five.

Below we create a new variable called `last.five.yrs` that takes on two values: "after 2012" and "2012 or before," using the `ifselse()` command from the previous class.

We also need to use `as.numeric(as.character())` around the `year` variable. It seems that the `year` variable is a factor, so its actual numeric representation is meaningless. But if we want to use an operator like `>`, we need a number, not a factor or a character. So we first make year a character variable with `as.character()` and then make that character variable numeric with `as.numeric()`.

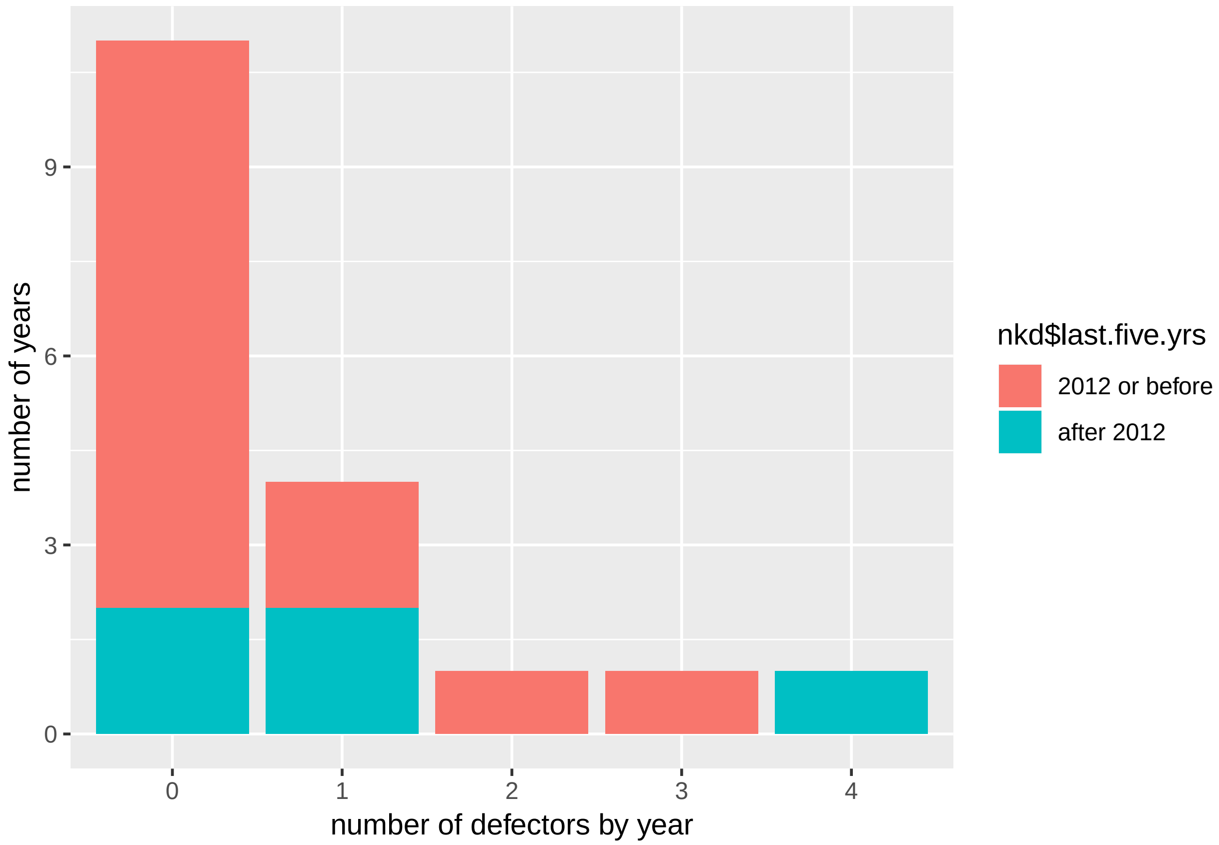
After I create this new variable, I check on it: are there as many "last five years" as we should expect?

```
# create a variable for the last five years
nkd$last.five.yrs <- ifelse(as.numeric(as.character(nkd$year)) > 2012,"after 2012","2012 or before")
table(nkd$last.five.yrs)
```

```
##
## 2012 or before     after 2012
##             13              5
```

With this new variable in hand, we fill in the bars, using the `fill=` command.

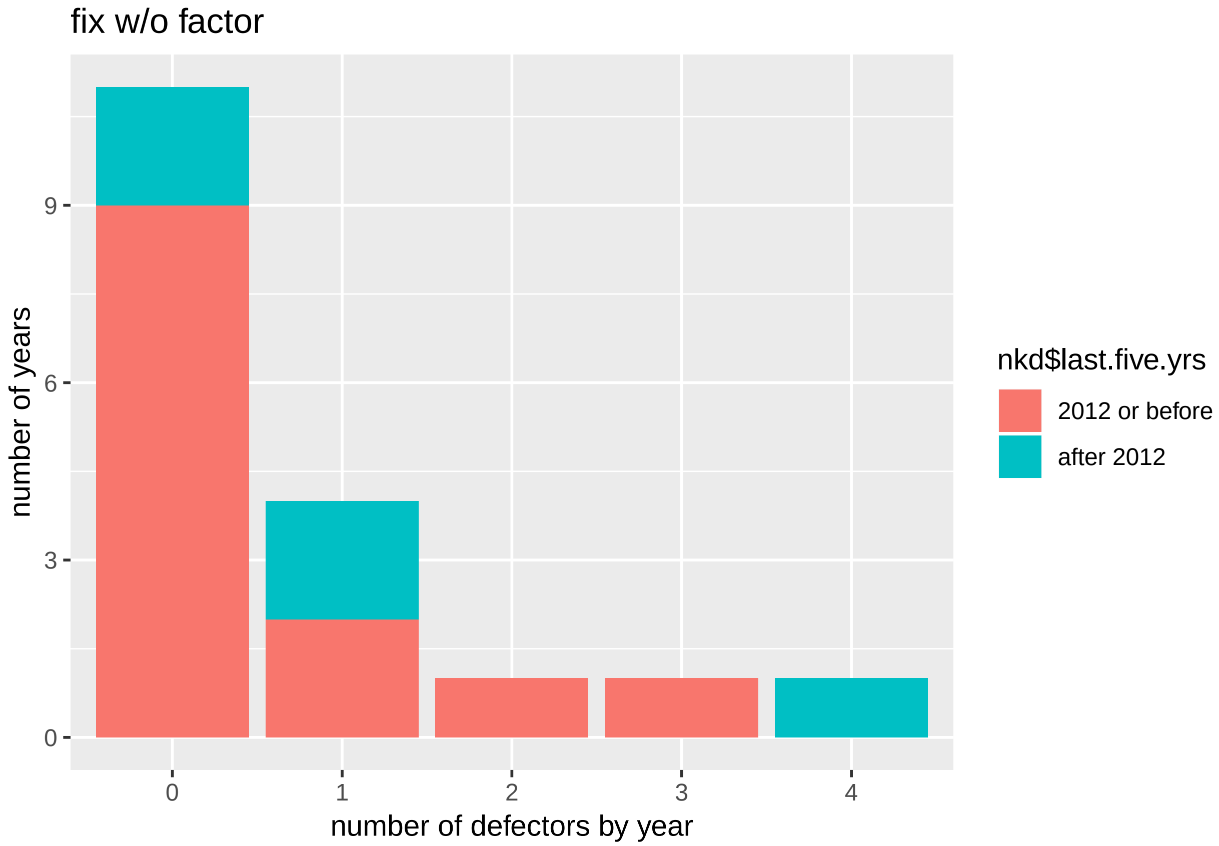
```
a3 <- ggplot() +
        geom_bar(data = nkd, aes(x = defectors, fill = nkd$last.five.yrs)) +
        labs(x = "number of defectors by year", y="number of years")
a3
```



This graph looks a little funny, because the new years are on bottom. Visually, this is disconcerting. There are at least two ways to fix this problem.

The easiest way is to use `position = position_stack(reverse=TRUE)` to flip the order of the bars, as below:

```
# re-order, so that -after- bars are on top
# here, using position_stack()
a3 <- ggplot() +
  geom_bar(data = nkd,
           aes(x = defectors, fill = nkd$last.five.yrs),
           position = position_stack(reverse = TRUE)) +
  labs(x = "number of defectors by year", y="number of years") +
  ggtitle("fix w/o factor")
a3
```



Here we have two categories, so this works fine. But sometimes this fix is insufficient. An alternative method is to make the new variable a factor, and make sure the order of the factor is as we prefer. First, make the factor variable by making a character variable (`last.five.yrsf`) and then making that variable a factor with `as.factor()`. We can check the levels of the factor and its ordering with `levels()`.

```
# make a factor variable, then re-order it
nkd$last.five.yrsff <- as.factor(nkd$last.five.yrs)
str(nkd)
```

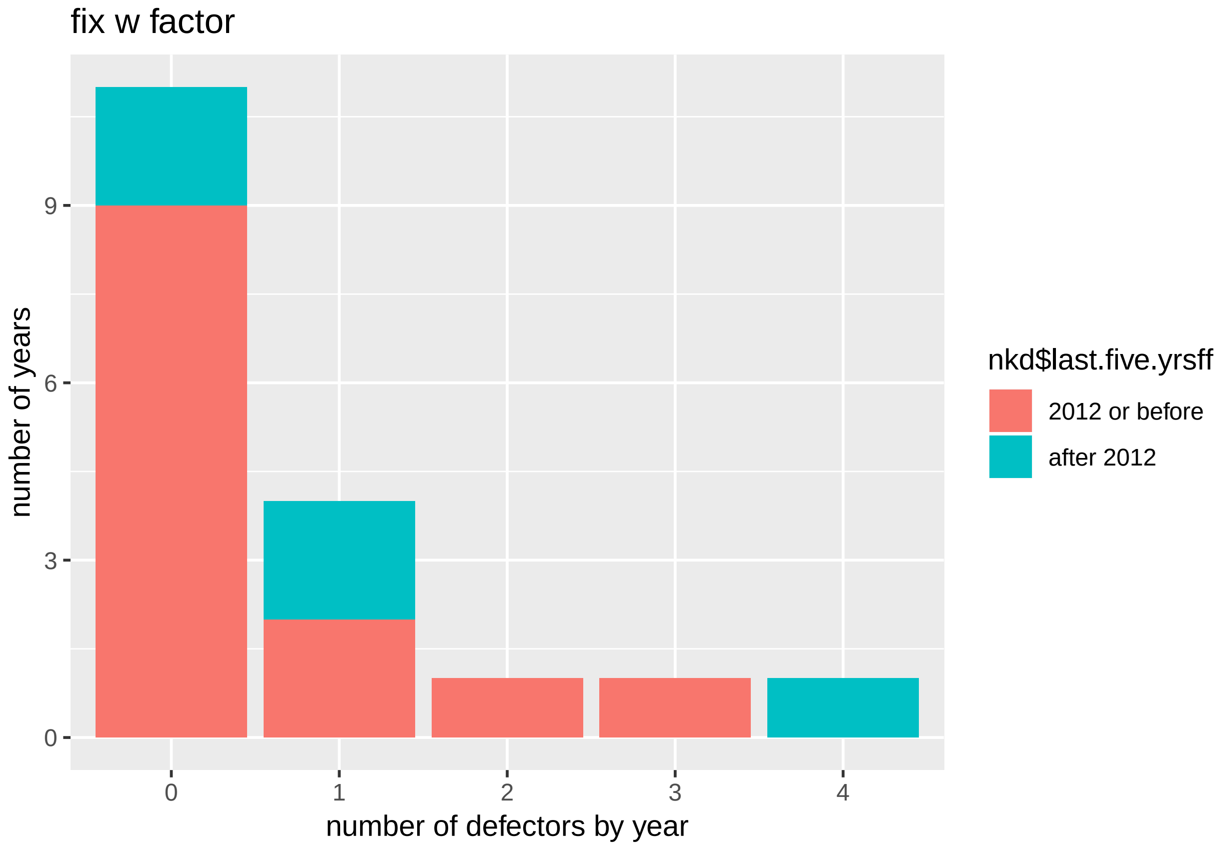
```
## 'data.frame':    18 obs. of  4 variables:
##  $ year           : Factor w/ 18 levels "2000","2001",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ defectors      : Factor w/ 5 levels "0","1","2","3",..: 1 1 2 1 1 1 1 1 3 1 ...
##  $ last.five.yrs  : chr  "2012 or before" "2012 or before" "2012 or before" "2012 or before" ...
##  $ last.five.yrsff: Factor w/ 2 levels "2012 or before",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
## note, if x is not a factor use levels(factor(x))
levels(nkd$last.five.yrsff)
```

```
## [1] "2012 or before" "after 2012"
```

Now when we use the factor variable to fill, things still look bad. The ordering of the bars doesn't match the ordering of the legend.

```
# now use factor variables
a3 <- ggplot() +
  geom_bar(data = nkd,
           aes(x = defectors, fill = nkd$last.five.yrsff),
           position = position_stack(reverse = TRUE)) +
  labs(x = "number of defectors by year", y="number of years") +
  ggtitle("fix w factor")
a3
```



This is because the factor variable is not in the right order. Below we re-order the levels of the factor variable with the `levels` command. The first input into this command is the variable to re-order and the second is the order in which to put them. We define that order as what's currently number two, followed by what's currently number 1.

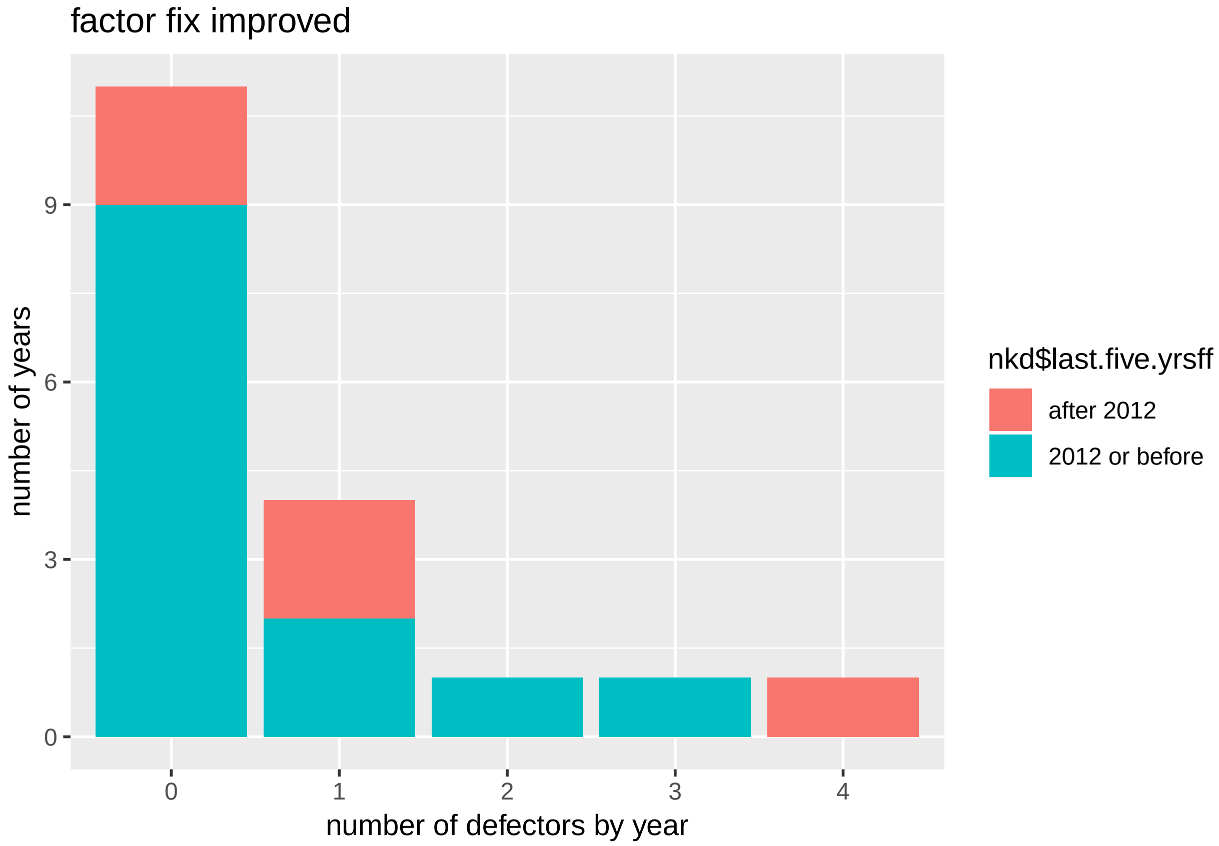
Check the re-ordering with `levels()`.

```
# make the factor levels in a different order
nkd$last.five.yrsff <- factor(nkd$last.five.yrsff,levels(factor(nkd$last.five.yrsff))[c(2,1)])
levels(nkd$last.five.yrsff)
```

```
## [1] "after 2012"     "2012 or before"
```

Now, re-do the graph, remaving the `position_stack` command:

```
# with factors but bad
a4 <- ggplot() +
  geom_bar(data = nkd,
           aes(x = defectors, fill = nkd$last.five.yrsff)) +
  labs(x = "number of defectors by year", y="number of years") +
  ggtitle("factor fix improved")
a4
```



C. Bigger data

It's useful to have small data to get started. However, in general you'll be working with data that are much larger – or at least start out much larger than the defector dataset.

C.1. Load data

So we'll practice bar graphs with a much larger dataset – the one we used last class on DC-area block groups. Find wherever it was you saved it last time, or re-download, following the instructions in tutorial 3.

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

C.2. Set up data

We are going to start with a focus on household income, which is variable B19001. We'll begin with a bar chart for the entire metro area, which means we need a dataframe that has one observation per household income type. We start with `summarize()` to add up the observations and make a new dataframe called metro.

Note that the syntax for this command is `.data =`, not `data =`. This is a `dplyr` syntax requirement. I'm not entirely sure why they do this (though I'm certain there's a reason; if you learn it, let me know!).

```
# lets do housing unit type if it exists
# bars basic: total housing units for metro area
metro <- summarize(.data = block.groups, B19001e2 = sum(B19001e2),
                                         B19001e3 = sum(B19001e3),
                                         B19001e4 = sum(B19001e4),
                   B19001e5 = sum(B19001e5),
                   B19001e6 = sum(B19001e6),
                   B19001e7 = sum(B19001e7),
                   B19001e8 = sum(B19001e8),
                   B19001e9 = sum(B19001e9),
                   B19001e10 = sum(B19001e10),
                   B19001e11 = sum(B19001e11),
                   B19001e12 = sum(B19001e12),
                   B19001e13 = sum(B19001e13),
                   B19001e14 = sum(B19001e14),
                   B19001e15 = sum(B19001e15),
                   B19001e16 = sum(B19001e16),
                   B19001e17 = sum(B19001e17))
metro
```

```
##   B19001e2 B19001e3 B19001e4 B19001e5 B19001e6 B19001e7 B19001e8 B19001e9
## 1   307670   209124   207501   213894   214987   222597   214435   227519
##   B19001e10 B19001e11 B19001e12 B19001e13 B19001e14 B19001e15 B19001e16
## 1    202610    411089    532979    713353    526071    361313    412551
##   B19001e17
## 1    428524
```

This one-observation dataframe is great in terms of the underlying data: it has exactly what we need. But it won't work with `ggplot`, as it's wide instead of long. "Wide" means that the data are in the variable labels; long means that the same data in the label will go into a new variable.

One way to take data from wide to long, is a command called `gather()` from `tidyr`, which does exactly this. It asks for an input dataframe, the name of your new column that you want to put the label into (`key`), the name of the new column into which you want to put the value (`value`), and finally, the variables that you want to made long.

We have a long list of variables to make long, so we use a command to get a list of all of them. The command `grep()` looks for the thing in the first part of the command ("B19001") in the thing in the second part of the command (`names(metro)`) and `checkitout` is the output. It looks odd, but it is the column numbers where any column has the letters "B19001" in its name.

We then use `gather()` to make a new dataframe (`metro.long`) that pushes all household income numbers into one category. And finally we print the final product to make sure it is what we expect.

```
# but this is wide, not long
# make it long to make a bar chart
checkitout <- grep("B19001", names(metro))
checkitout
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
```

```
metro.long <- gather(data = metro, key = invar, value = B19001, grep("B19001", names(metro)) )
metro.long
```

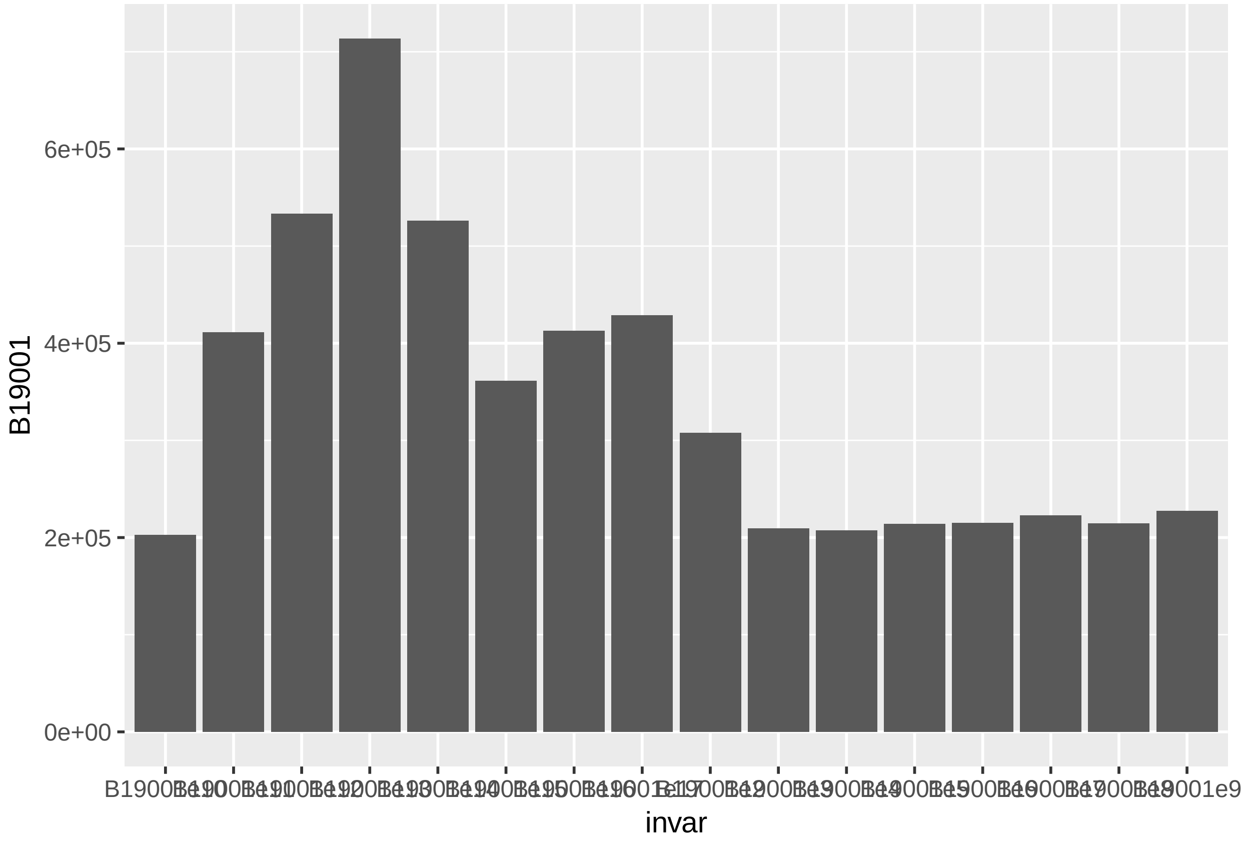
```
##        invar B19001
## 1   B19001e2 307670
## 2   B19001e3 209124
## 3   B19001e4 207501
## 4   B19001e5 213894
## 5   B19001e6 214987
## 6   B19001e7 222597
## 7   B19001e8 214435
## 8   B19001e9 227519
## 9  B19001e10 202610
## 10 B19001e11 411089
## 11 B19001e12 532979
## 12 B19001e13 713353
## 13 B19001e14 526071
## 14 B19001e15 361313
## 15 B19001e16 412551
## 16 B19001e17 428524
```

C.3. Make a graph, finally

We are now ready to make a bar graph! However, because we are just asking R to plot a value in the data already – not to add up observations, we need to make this clear. We can do this in one of two ways.

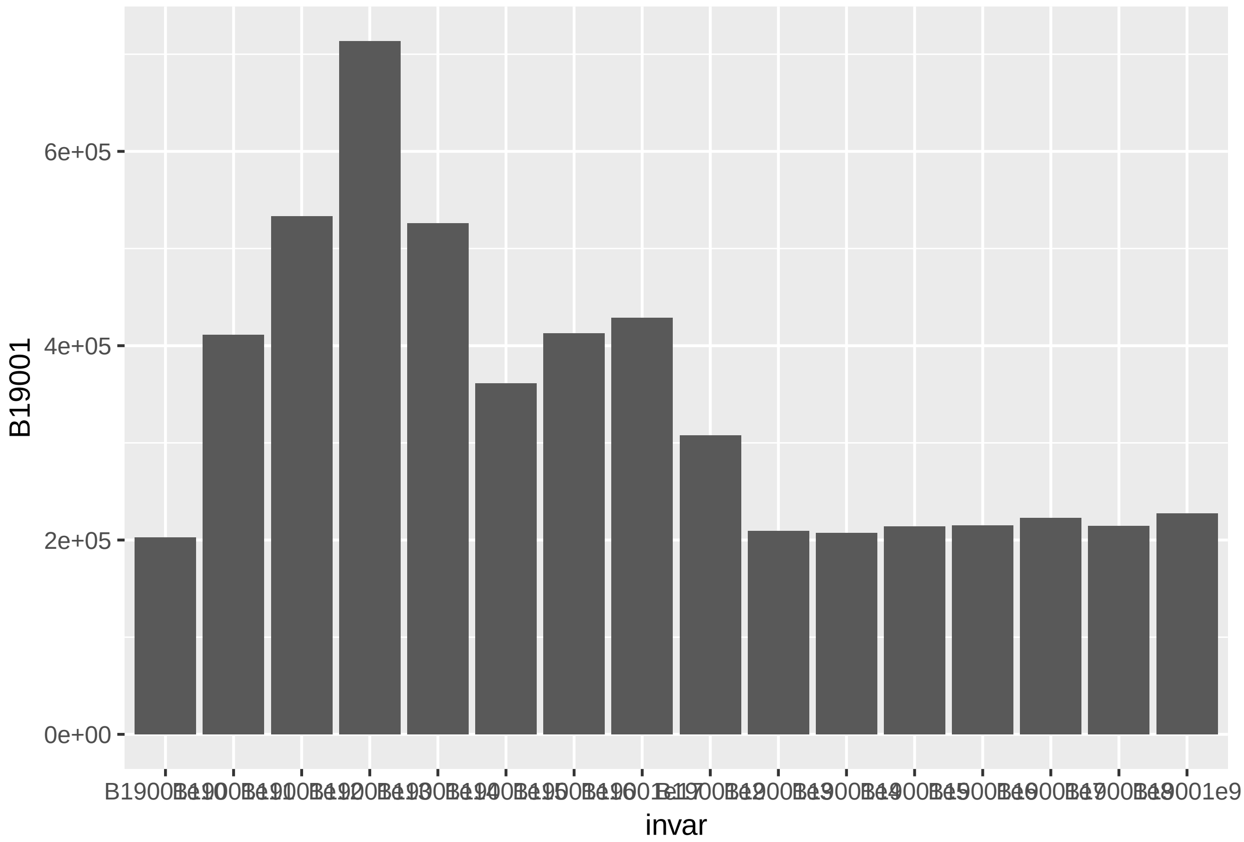
The first way is to use `geom_col()` which knows that it should put the data in the y variable, and not count the observations.

```
# since we are using values in the data (not asking r to sum up something), we use geom_col()
b3.1 <- ggplot() +
        geom_col(data = metro.long, aes(x = invar, y = B19001))
b3.1
```



Alternatively, we can still use `geom_bar()`, but let R know that we want to use the actual data with a command inside `geom_bar()`: `stat = "identity"`.

```
# can also do this with geom_bar
b3.2 <- ggplot() +
        geom_bar(data = metro.long, aes(x = invar, y = B19001), stat = "identity")
b3.2
```



C.4. Fixing legend

The labels above look terrible! R uses the values of `invar` since it doesn't know what these values mean. We can fix this by making a factor variable with decent labels. We do this by combining `ifelse()` commands with the `as.factor()` command as below.

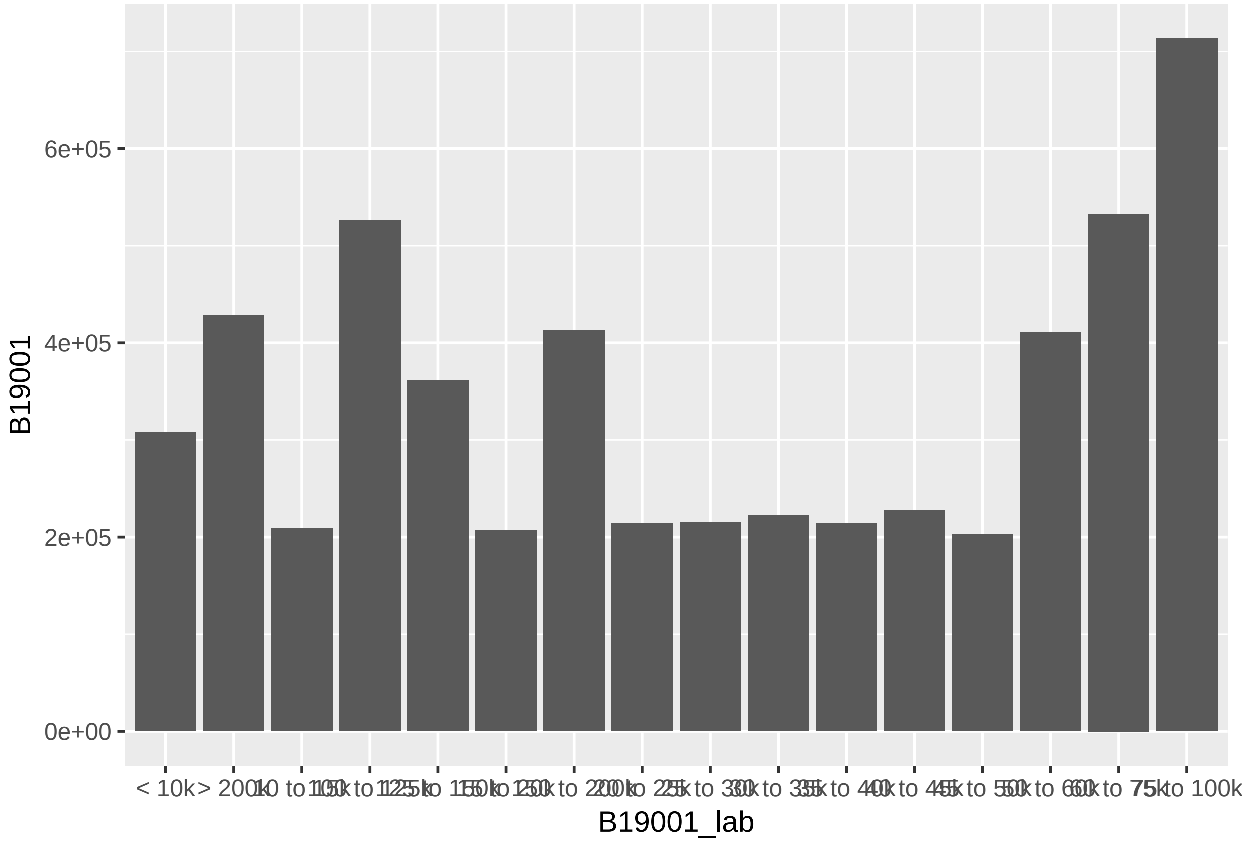
```
metro.long$B19001_lab <- as.factor(
                        ifelse(metro.long$invar == "B19001e2", "< 10k",
                          ifelse(metro.long$invar == "B19001e3", "10 to 15k",
                           ifelse(metro.long$invar == "B19001e4", "15 to 20k",
                            ifelse(metro.long$invar == "B19001e5", "20 to 25k",
                             ifelse(metro.long$invar == "B19001e6", "25 to 30k",
                              ifelse(metro.long$invar == "B19001e7", "30 to 35k",
                               ifelse(metro.long$invar == "B19001e8", "35 to 40k",
                                ifelse(metro.long$invar == "B19001e9", "40 to 45k",
                                 ifelse(metro.long$invar == "B19001e10", "45 to 50k",
                                  ifelse(metro.long$invar == "B19001e11", "50 to 60k",
                                   ifelse(metro.long$invar == "B19001e12", "60 to 75k",
                                    ifelse(metro.long$invar == "B19001e13", "75 to 100k",
                                     ifelse(metro.long$invar == "B19001e14", "100 to 125k",
                                      ifelse(metro.long$invar == "B19001e15", "125 to 150k",
                                       ifelse(metro.long$invar == "B19001e16", "150 to 200k",
                                        ifelse(metro.long$invar == "B19001e17", "> 200k","bad")))))))))

metro.long
```

```
##        invar B19001  B19001_lab
## 1   B19001e2 307670       < 10k
## 2   B19001e3 209124   10 to 15k
## 3   B19001e4 207501   15 to 20k
## 4   B19001e5 213894   20 to 25k
## 5   B19001e6 214987   25 to 30k
## 6   B19001e7 222597   30 to 35k
## 7   B19001e8 214435   35 to 40k
## 8   B19001e9 227519   40 to 45k
## 9  B19001e10 202610   45 to 50k
## 10 B19001e11 411089   50 to 60k
## 11 B19001e12 532979   60 to 75k
## 12 B19001e13 713353  75 to 100k
## 13 B19001e14 526071 100 to 125k
## 14 B19001e15 361313 125 to 150k
## 15 B19001e16 412551 150 to 200k
## 16 B19001e17 428524      > 200k
```

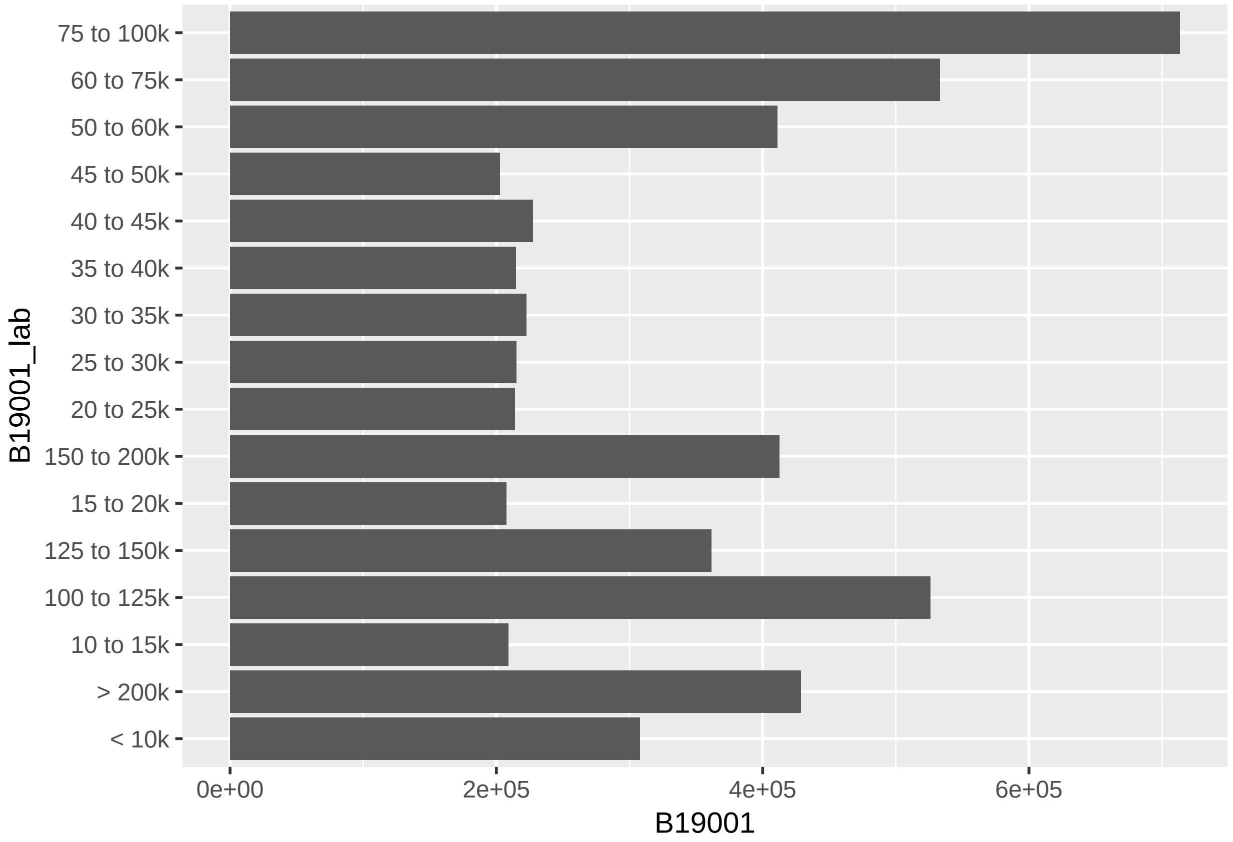
Here's the graph with these new labels. Note the change in x variable.

```
# looking at the new labels
b4.1 <- ggplot() +
  geom_col(data = metro.long, aes(x = B19001_lab, y = B19001))
b4.1
```



But it's pretty hard to read this way. Make the bars horizontal using `coord_flip()` and they are much easier to read:

```
# flip the axes
b4.2 <- ggplot() +
  geom_col(data = metro.long, aes(x = B19001_lab, y = B19001)) +
  coord_flip()
b4.2
```



However, this chart is still kind of nutty, because the bars are not in the right order. It's "right" in some sense, but unbearable to read. So we use the `levels()` command to re-order the factor.

```
# wait! bars in wack-a-doodle order
is.factor(metro.long$B19001_lab)
```

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

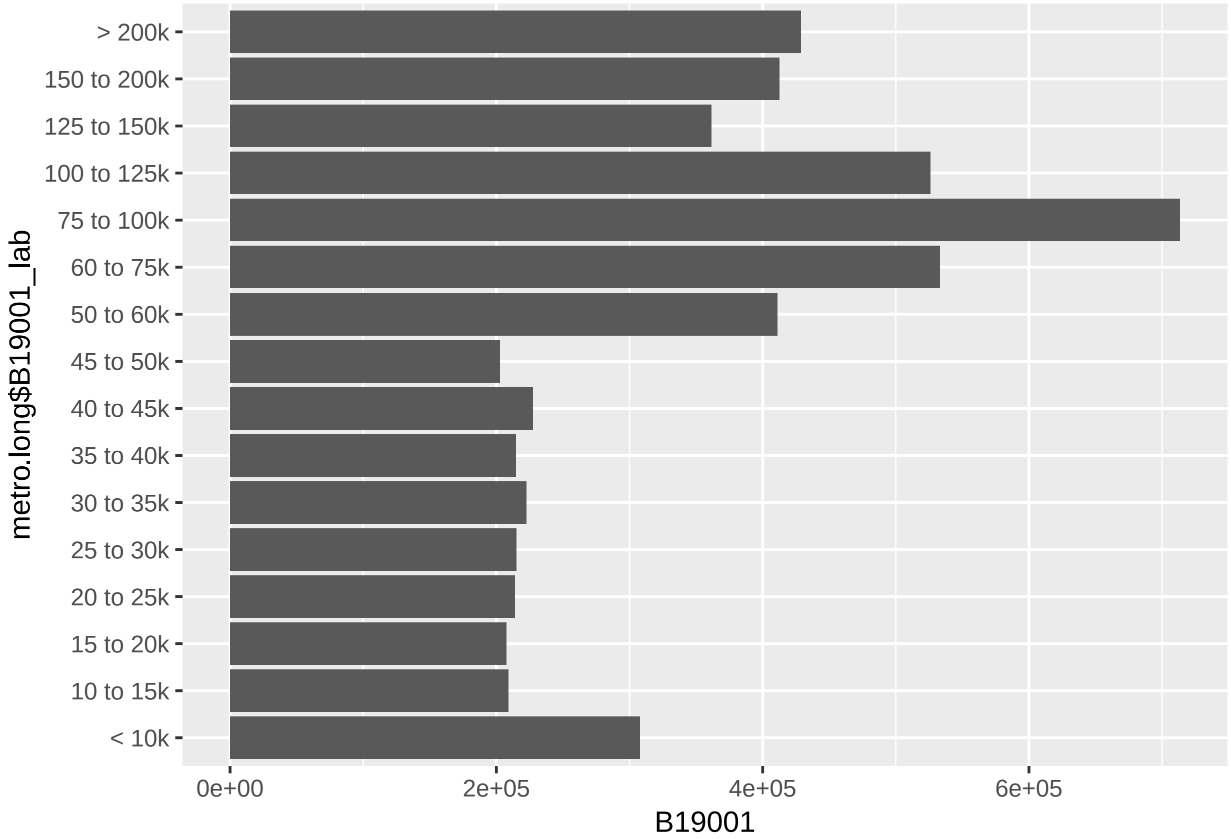
```
levels(metro.long$B19001_lab)
```

```
##  [1] "< 10k"       "> 200k"      "10 to 15k"   "100 to 125k" "125 to 150k"
##  [6] "15 to 20k"   "150 to 200k" "20 to 25k"   "25 to 30k"   "30 to 35k"
## [11] "35 to 40k"   "40 to 45k"   "45 to 50k"   "50 to 60k"   "60 to 75k"
## [16] "75 to 100k"
```

```
metro.long$B19001_lab <- factor(metro.long$B19001_lab, levels = c("< 10k",
                                "10 to 15k",
                                "15 to 20k",
                                "20 to 25k",
                                "25 to 30k",
                                "30 to 35k",
                                "35 to 40k",
                                "40 to 45k",
                                "45 to 50k",
                                "50 to 60k",
                                "60 to 75k",
                                "75 to 100k",
                                "100 to 125k",
                                "125 to 150k",
                                "150 to 200k",
                                "> 200k"))
```

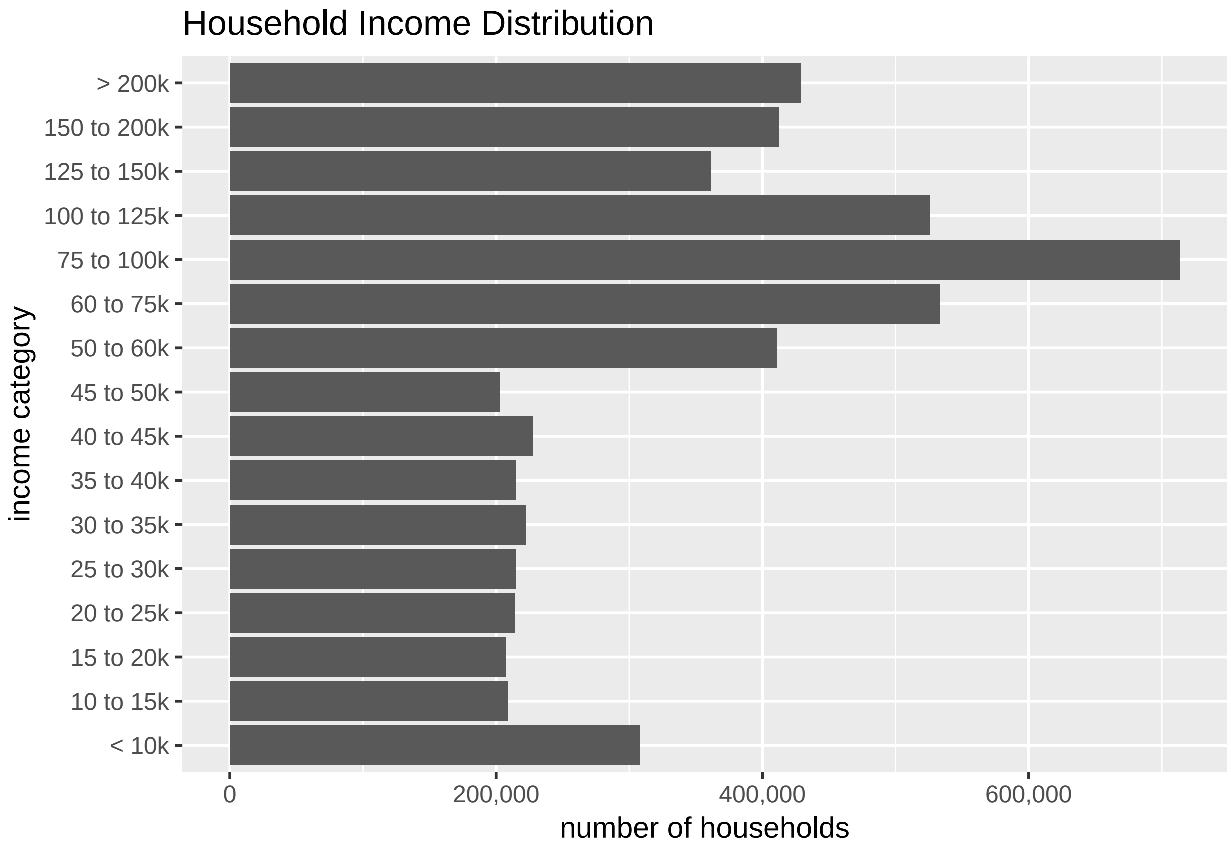
Making the graph with the fixed order, things look a lot more pleasant:

```
# fixed order
b4.3 <- ggplot() +
  geom_col(data = metro.long, aes(x = metro.long$B19001_lab, y = B19001)) +
  coord_flip()
b4.3
```



Of course, the graph still has scientific notation, which is entirely illegible. Using the scales library, we can add scale_y_continuous(labels = comma) to fix this. You might be thinking that you want to fix the x axis, not the y one – why do we name the y axis? Because the y axis is the "true" axis for these data, or the one with which the variable is linked in the aes() command.

```
# get rid of scientific notation
require(scales)
b4.4 <- ggplot() +
  geom_col(data = metro.long, aes(x = B19001_lab, y = B19001)) +
  coord_flip() +
  scale_y_continuous(labels = comma) +
  labs(title = "Household Income Distribution",
       y = "number of households",
       x = "income category")
b4.4
```



It is sometimes also of interest to graph shares, rather than levels as we've done above. When you are looking at just one jurisdiction or unit, the shares and levels are visually equivalent. However, when you want to make cross-unit comparisons, it is frequently the share that is of interest.

To graph shares, first calculate them. We begin by finding the denominator: how many households are there in the metro area? We use the mutate() command, which adds a row to a dataframe based on within-dataframe calculations. Here we tell mutate to make a new variable B19001_tot, which is the sum of the column B19001. As always, we print the dataframe to check this calculation.

```
# find total number of households
metro.long <- mutate(.data = metro.long, B19001_tot = sum(B19001))
metro.long
```

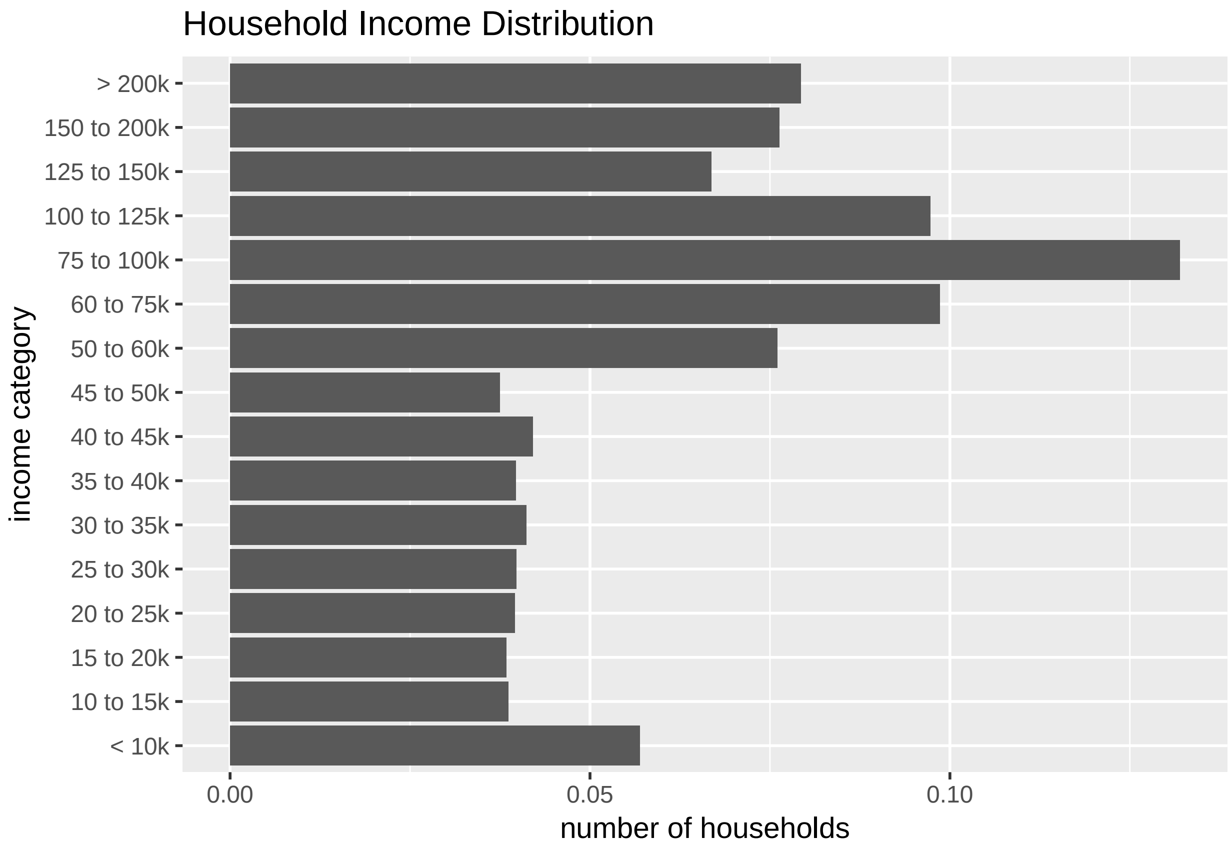
```
##        invar B19001  B19001_lab B19001_tot
## 1   B19001e2 307670       < 10k    5406217
## 2   B19001e3 209124   10 to 15k    5406217
## 3   B19001e4 207501   15 to 20k    5406217
## 4   B19001e5 213894   20 to 25k    5406217
## 5   B19001e6 214987   25 to 30k    5406217
## 6   B19001e7 222597   30 to 35k    5406217
## 7   B19001e8 214435   35 to 40k    5406217
## 8   B19001e9 227519   40 to 45k    5406217
## 9  B19001e10 202610   45 to 50k    5406217
## 10 B19001e11 411089   50 to 60k    5406217
## 11 B19001e12 532979   60 to 75k    5406217
## 12 B19001e13 713353  75 to 100k    5406217
## 13 B19001e14 526071 100 to 125k    5406217
## 14 B19001e15 361313 125 to 150k    5406217
## 15 B19001e16 412551 150 to 200k    5406217
## 16 B19001e17 428524      > 200k    5406217
```

Now we have everything we need to calculate the ratio: divide number of households in this income group by the total number of households:

```
# calculate share
metro.long$B19001_shr <- metro.long$B19001 / metro.long$B19001_tot
```

With this share in hand, we re-do the chart

```
# make the chart
b4.5 <- ggplot() +
  geom_col(data = metro.long, aes(x = B19001_lab, y = B19001_shr)) +
  coord_flip() +
  labs(title = "Household Income Distribution",
       y = "number of households",
       x = "income category")
b4.5
```



C.5. Make data long for grouped bars

Now we are going to make grouped bar charts, which are bar charts that replicate the same set of bars across units. Our “units” are states. We’ll focus on the education variable B15002. To make a grouped bar chart, we need a long dataframe at the state level. So instead of one observation for each education level, as we would have had above, now we’ll have three: one for DC, one for MD and one for VA.

This coding is very similar to what we did above, except that we need to group the dataframe below we summarize. I use the same `grep` command to get a list of all variables that begin with B15002 into the list `edvars`. We also use the `summarize_at()` command to summarize a bunch of similarly-named varaibles – so we don’t have to name them all like above. Here we tell R the tbl we are using (a variant on a dataframe, and we use the grouped dataframe), the variables we want to summarize (the education ones in `edvars`) and the function we want to use (`sum`).

The resulting dataframe should have three observations. We check the output by printing `metro.ed` to the screen.

```r
# grouped bars by state
edvars <- grep("B15002", names(block.groups))
edvars
```

```
##  [1] 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979
## [18] 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996
## [35] 997
```

```r
block.groups.g <- group_by(block.groups, block.groups$STATE)
```

```
## Warning: `as_dictionary()` is soft-deprecated as of rlang 0.3.0.
## Please use `as_data_pronoun()` instead
## This warning is displayed once per session.
```

```
## Warning: `new_overscope()` is soft-deprecated as of rlang 0.2.0.
## Please use `new_data_mask()` instead
## This warning is displayed once per session.
```

```
## Warning: The `parent` argument of `new_data_mask()` is deprecated.
## The parent of the data mask is determined from either:
##
##   * The `env` argument of `eval_tidy()`
##   * Quosure environments when applicable
## This warning is displayed once per session.
```

```
## Warning: `overscope_clean()` is soft-deprecated as of rlang 0.2.0.
## This warning is displayed once per session.
```

```r
metro.ed <- summarize_at(.tbl = block.groups.g, .vars = edvars, .funs = sum)
metro.ed
```

```
## # A tibble: 3 x 36
##   `block.groups$STA~ B15002e1 B15002e2 B15002e3 B15002e4 B15002e5 B15002e6
##                <int>    <int>    <int>    <int>    <int>    <int>    <int>
## 1                 11   417432   195790     2612     1816     2423     3419
## 2                 24  3875282  1823873    17692    12847    25671    32255
## 3                 51  5356571  2572117    29698    17867    35196    66512
## # ... with 29 more variables: B15002e7 <int>, B15002e8 <int>,
## #   B15002e9 <int>, B15002e10 <int>, B15002e11 <int>, B15002e12 <int>,
## #   B15002e13 <int>, B15002e14 <int>, B15002e15 <int>, B15002e16 <int>,
## #   B15002e17 <int>, B15002e18 <int>, B15002e19 <int>, B15002e20 <int>,
## #   B15002e21 <int>, B15002e22 <int>, B15002e23 <int>, B15002e24 <int>,
## #   B15002e25 <int>, B15002e26 <int>, B15002e27 <int>, B15002e28 <int>,
## #   B15002e29 <int>, B15002e30 <int>, B15002e31 <int>, B15002e32 <int>,
## #   B15002e33 <int>, B15002e34 <int>, B15002e35 <int>
```

There are way too many education categories to graph, so we combine them into three types: less than high school, more than high school, but less than a BA, and a BA or more. I use a R function called `rowSums()` to add up multiple rows. For this command, you name the columns you want to add in `c()`. Note that we need to add values for men plus values for women.

```r
# make education categories
metro.ed$ed.ls.hs <- rowSums(metro.ed[,c("B15002e3","B15002e4",
                                         "B15002e5","B15002e6","B15002e7",
                                         "B15002e8","B15002e9","B15002e10",
                                         "B15002e11",
                                         "B15002e20","B15002e21","B15002e22",
                                         "B15002e23","B15002e24","B15002e25",
                                         "B15002e26","B15002e27","B15002e28")])
metro.ed$ed.lt.col <- rowSums(metro.ed[,c("B15002e12","B15002e13","B15002e14",
                                          "B15002e29","B15002e30","B15002e31")])
metro.ed$ed.gt.col <- rowSums(metro.ed[,c("B15002e15","B15002e16","B15002e17","B15002e18",
                                          "B15002e32","B15002e33","B15002e34","B15002e35")])
```

Now we have a wide dataframe with the three variables of interest and the state number, as well as a lot of other variables we don't need. You can check on this by seeing what's in `metro.ed` with the `names()`command.

```
names(metro.ed)
```

```
##  [1] "block.groups$STATE" "B15002e1"           "B15002e2"
##  [4] "B15002e3"           "B15002e4"           "B15002e5"
##  [7] "B15002e6"           "B15002e7"           "B15002e8"
## [10] "B15002e9"           "B15002e10"          "B15002e11"
## [13] "B15002e12"          "B15002e13"          "B15002e14"
## [16] "B15002e15"          "B15002e16"          "B15002e17"
## [19] "B15002e18"          "B15002e19"          "B15002e20"
## [22] "B15002e21"          "B15002e22"          "B15002e23"
## [25] "B15002e24"          "B15002e25"          "B15002e26"
## [28] "B15002e27"          "B15002e28"          "B15002e29"
## [31] "B15002e30"          "B15002e31"          "B15002e32"
## [34] "B15002e33"          "B15002e34"          "B15002e35"
## [37] "ed.ls.hs"           "ed.lt.col"          "ed.gt.col"
```

To make this wide dataframe, first we re-name one variable that has a strange name that will cause trouble (first line below). We then keep only the state variable and the three education variables to make things easier when we use `gather()`. (If you don't drop variables you're not interested in, they are repeated in all long observations, which seems likely to cause confusion.)

Finally, we use gather to make a long dataset. Compare the long and wide formats so you understand what's going on.

```
# now make it long
names(metro.ed)[names(metro.ed) == "block.groups$STATE"] <- "state"
names(metro.ed)
```

```
##  [1] "state"     "B15002e1"  "B15002e2"  "B15002e3"  "B15002e4"
##  [6] "B15002e5"  "B15002e6"  "B15002e7"  "B15002e8"  "B15002e9"
## [11] "B15002e10" "B15002e11" "B15002e12" "B15002e13" "B15002e14"
## [16] "B15002e15" "B15002e16" "B15002e17" "B15002e18" "B15002e19"
## [21] "B15002e20" "B15002e21" "B15002e22" "B15002e23" "B15002e24"
## [26] "B15002e25" "B15002e26" "B15002e27" "B15002e28" "B15002e29"
## [31] "B15002e30" "B15002e31" "B15002e32" "B15002e33" "B15002e34"
## [36] "B15002e35" "ed.ls.hs"  "ed.lt.col" "ed.gt.col"
```

```
metro.ed <- metro.ed[,c("state","ed.ls.hs","ed.lt.col","ed.gt.col")]
names(metro.ed)
```

```
## [1] "state"     "ed.ls.hs"  "ed.lt.col" "ed.gt.col"
```

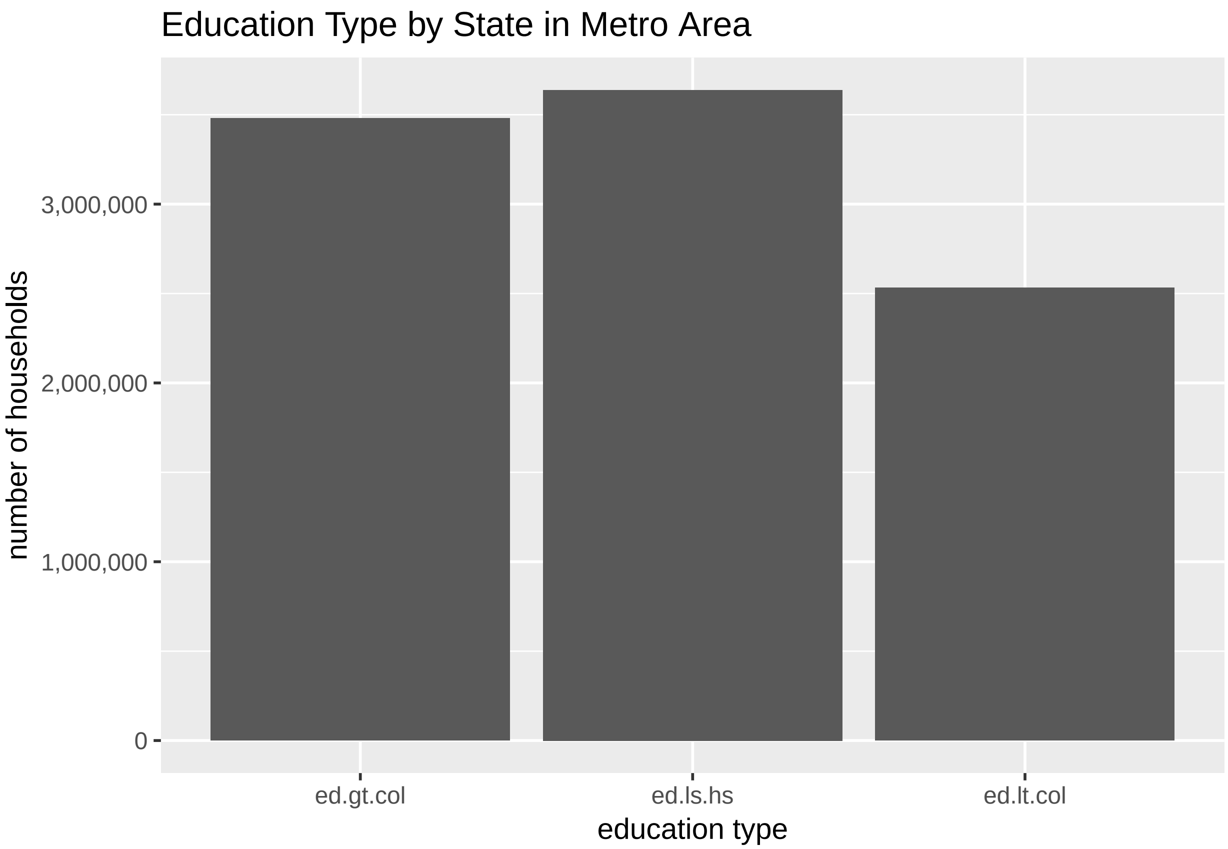
```
metro.long <- gather(data = metro.ed,
                     key = "ed.type",
                     value = "ed.num.ppl",
                     c("ed.ls.hs","ed.lt.col","ed.gt.col"))
metro.long
```

```
## # A tibble: 9 x 3
##   state ed.type   ed.num.ppl
##   <int> <chr>          <dbl>
## 1    11 ed.ls.hs     131610.
## 2    24 ed.ls.hs    1454263.
## 3    51 ed.ls.hs    2051403.
## 4    11 ed.lt.col     72073.
## 5    24 ed.lt.col   1012698.
## 6    51 ed.lt.col   1447591.
## 7    11 ed.gt.col    213749.
## 8    24 ed.gt.col   1408321.
## 9    51 ed.gt.col   1857577.
```

C.6. Grouped bars!

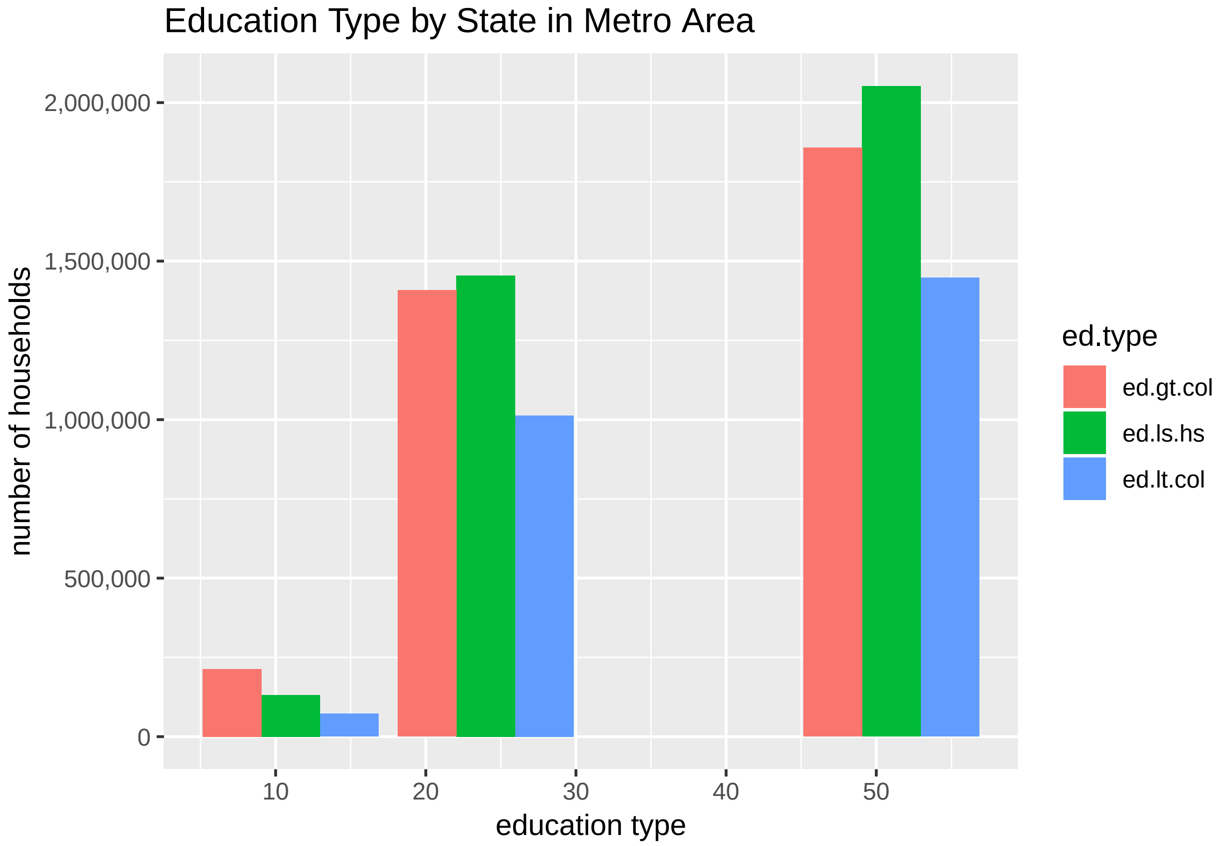
We are finally ready to make grouped bars. To understand what the grouping does, let's first use these data without any grouping. This graph shows the total number of people in all states in each education grouping.

```r
# grouped bars by education -- but w/o grouping
b6.1 <- ggplot() +
  geom_col(data = metro.long, aes(x = ed.type, y = ed.num.ppl)) +
  scale_y_continuous(labels = comma) +
  labs(title = "Education Type by State in Metro Area",
       y = "number of households",
       x = "education type")
b6.1
```



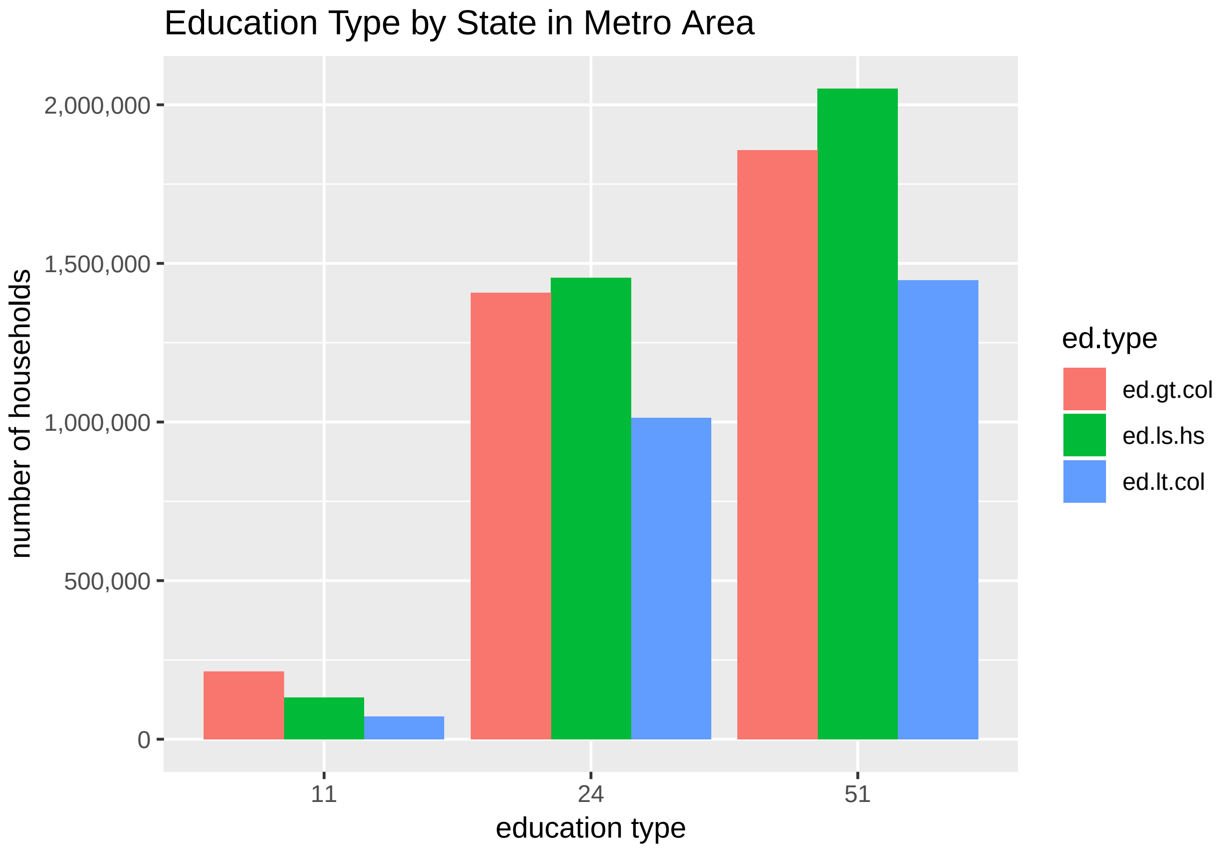
To make these into grouped bars, we start by changing the x variable to `state`. To have grouped, rather than stacked, bars, we use `position = position_dodge()`. We'll keep refining this picture for a bit.

```
# grouped bars by education shares and state
b6.2 <- ggplot() +
  geom_bar(data = metro.long,
           aes(x = state, y = ed.num.ppl, fill = ed.type),
           position = position_dodge(),
           stat = "identity") +
  scale_y_continuous(labels = comma) +
  labs(title = "Education Type by State in Metro Area",
       y = "number of households",
       x = "education type")
b6.2
```



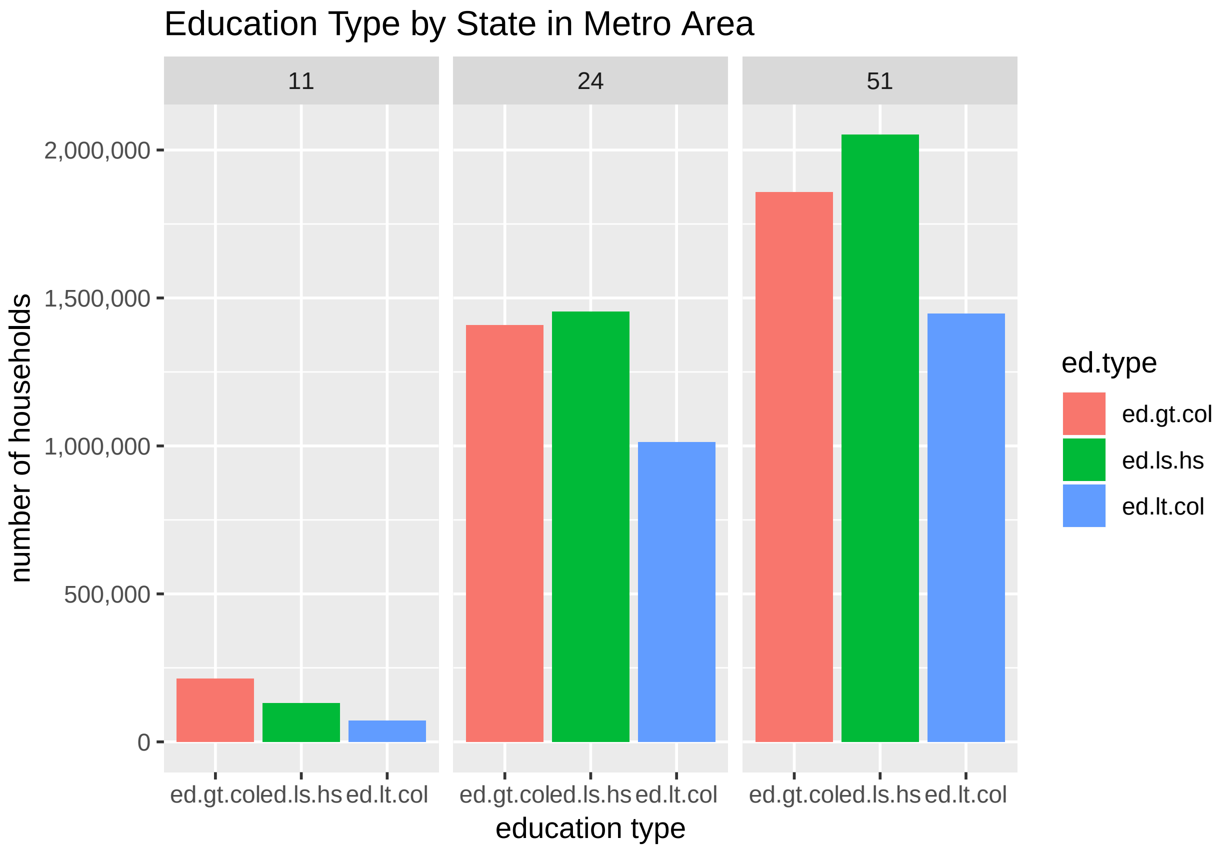
First, the x axis is uninterpretable and uneven. To fix, we make state a factor:

```r
# grouped bars by education shares and state -- state as factor
b6.3 <- ggplot() +
  geom_bar(data = metro.long,
           aes(x = as.factor(state), y = ed.num.ppl, fill = ed.type),
           position = position_dodge(),
           stat = "identity") +
  scale_y_continuous(labels = comma) +
  labs(title = "Education Type by State in Metro Area",
       y = "number of households",
       x = "education type")
b6.3
```



It is sometimes preferable to do such a graph as "small multiples," which we can do with `facet_wrap()` as below:

```
# small multiples
b6.4 <- ggplot() +
  geom_bar(data = metro.long,
           aes(x = ed.type, y = ed.num.ppl, fill = ed.type),
           stat = "identity") +
  facet_wrap(vars(state))+
  scale_y_continuous(labels = comma) +
  labs(title = "Education Type by State in Metro Area",
       y = "number of households",
       x = "education type")
b6.4
```



In your homeowrk, you'll undertake a set of fixes to make this graph look decent:

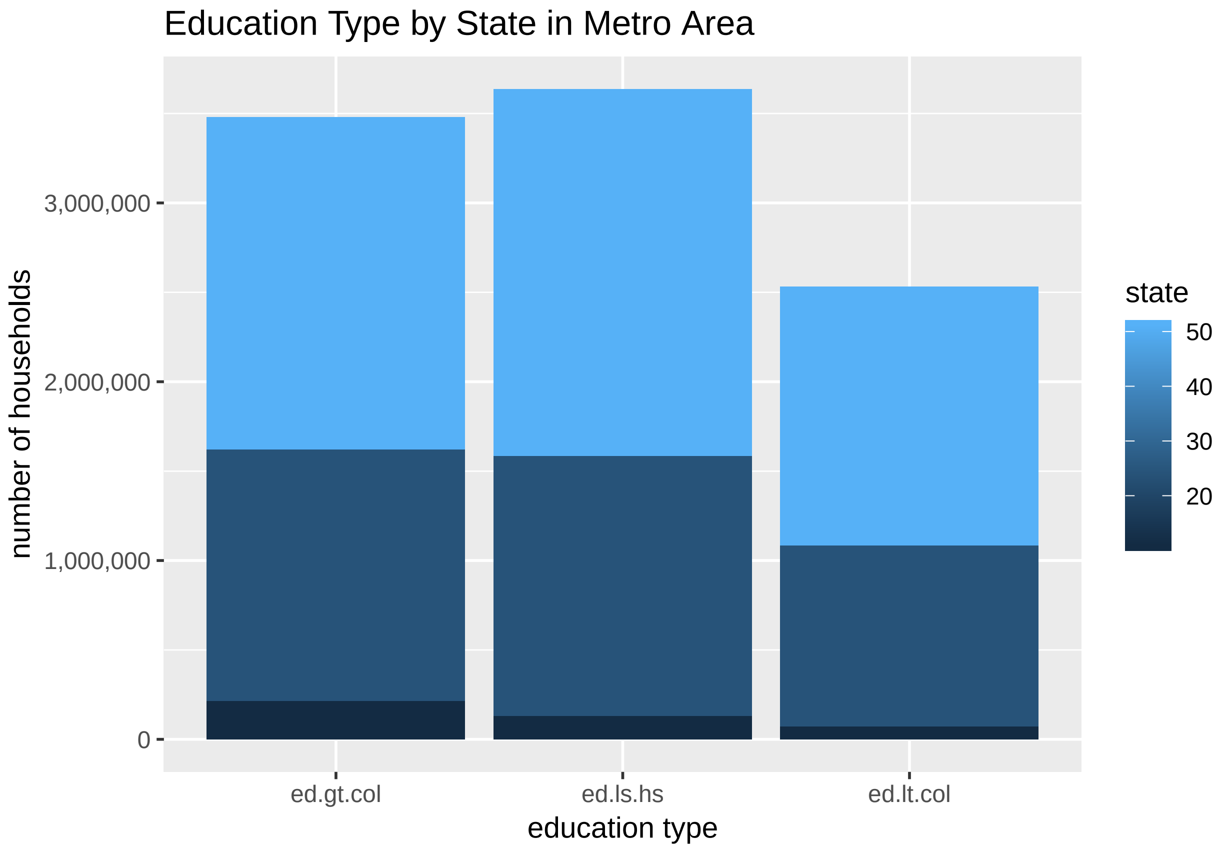
- make shares to compare across states
- label states
- clear up education labeling
- get rid of useless background

C.7. Stacked bars

Finally, we consider stacked bars. These should be used with caution, but can be very useful in the appropriate circumstances.

Here is a basic stacked bar by education type, filled in by state.

```
# stacked by education type
b7.1 <- ggplot() +
  geom_col(data = metro.long, aes(x = ed.type, y = ed.num.ppl, fill = state)) +
  scale_y_continuous(labels = comma) +
  labs(title = "Education Type by State in Metro Area",
       y = "number of households",
       x = "education type")
b7.1
```

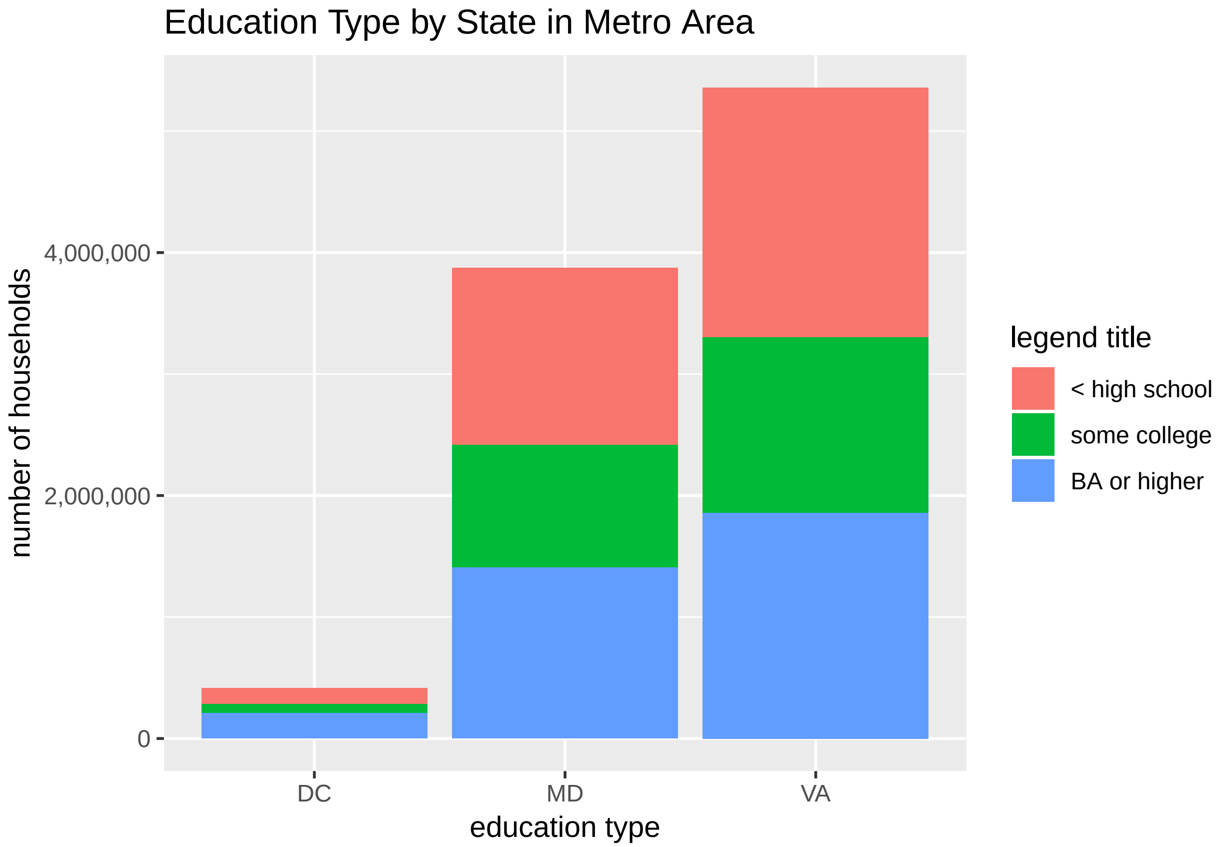


This isn't a very natural way to look at the data – usually we want to make cross-state comparisons and we can do that by making the state factor variable the x variable (with labels) and filling by education type (which we also make a factor with labels).

```r
# make state a factor with names
metro.long$statef <- factor(metro.long$state,
                               levels = c(11, 24, 51),
                               labels = c("DC", "MD", "VA"))

# make an education level factor with names
metro.long$ed.type.f <- factor(metro.long$ed.type,
                               levels = c("ed.ls.hs","ed.lt.col","ed.gt.col"),
                               labels = c("< high school", "some college", "BA or higher"))

# stacked by state by education shares and state
b7.2 <- ggplot() +
  geom_bar(data = metro.long, aes(x = statef, y = ed.num.ppl, fill = ed.type.f), stat = "identity") +
  scale_y_continuous(labels = comma) +
  labs(title = "Education Type by State in Metro Area",
       y = "number of households",
       x = "education type",
       fill = "legend title")
b7.2
```



D. Homework

1. Do the graph clean-up listed in C.6.
2. Make a simple bar chart (no grouping) without a legend box (put labels on the graph if needed) from a new dataset (not one we used in class already).
3. Re-do the creation of a state-level long dataset from C.6. with another variable from the `block.groups` dataframe and make a grouped bar chart to show it.