# Tutorial 8: Line Charts

*Leah Brooks*

*March 27, 2019*

Today's tutorial focuses on line charts. In addition, I offer you some better code to put a `ggplot()` call into a function. We review summarizing, use of factor variables, making data long (from wide) and summarizing data. In addition, you do some of your own data cleaning to prepare a file to load.

From her review of the tutorial, Jill warns that this one was tougher on RStudio, so be ready to restart as needed.

## A. Load packages

Let's begin by loading packages. The only addition from what we've used before is `scales`, which helps put commas into numbers so they are legible, among other things.

```
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
```

```
library(ggplot2)
```

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

```
library(scales)  # for making numbers with commas
library(tidyr)
```

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

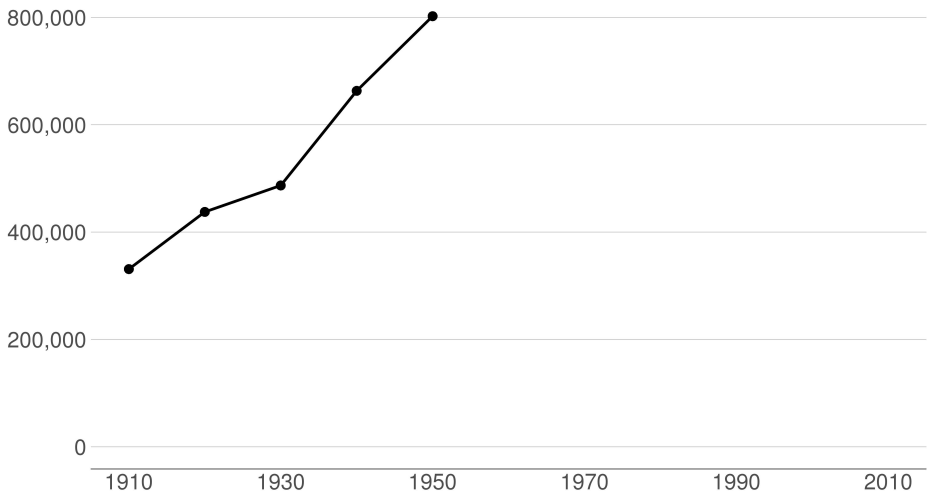
## B. Simple line graph

We are going to begin with a "simple"" line graph. It looks simple, but it took a lot of work to look good!

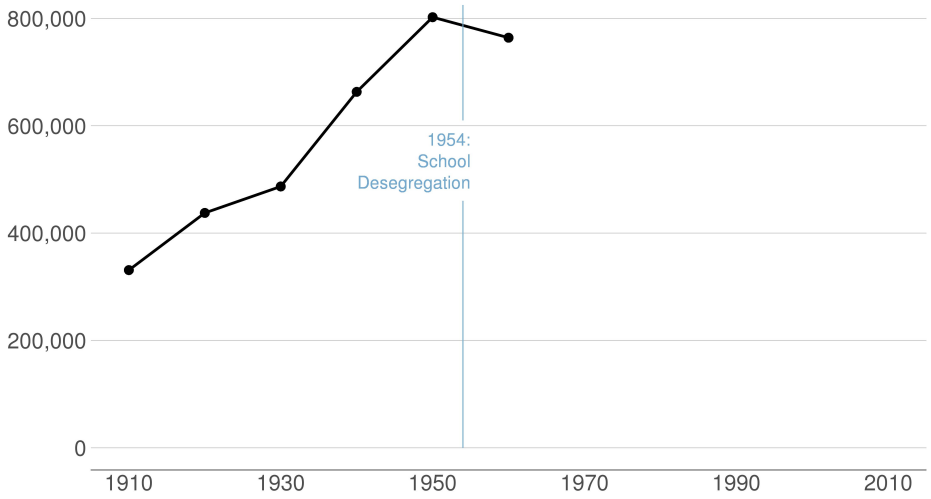
### B.1. Look at the final product

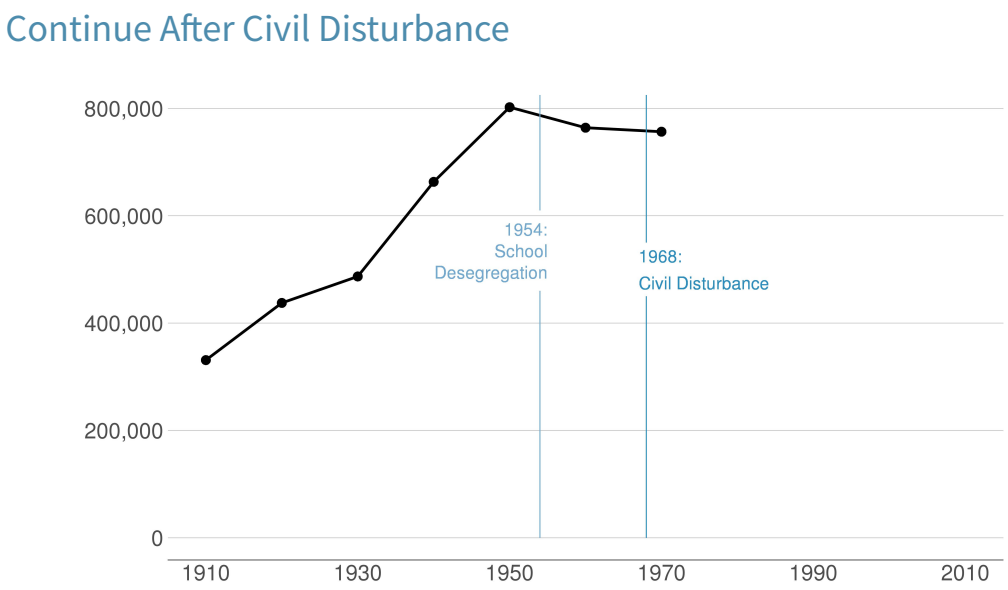
Here are the slides with this graph (sequential) that I used in my presentation.

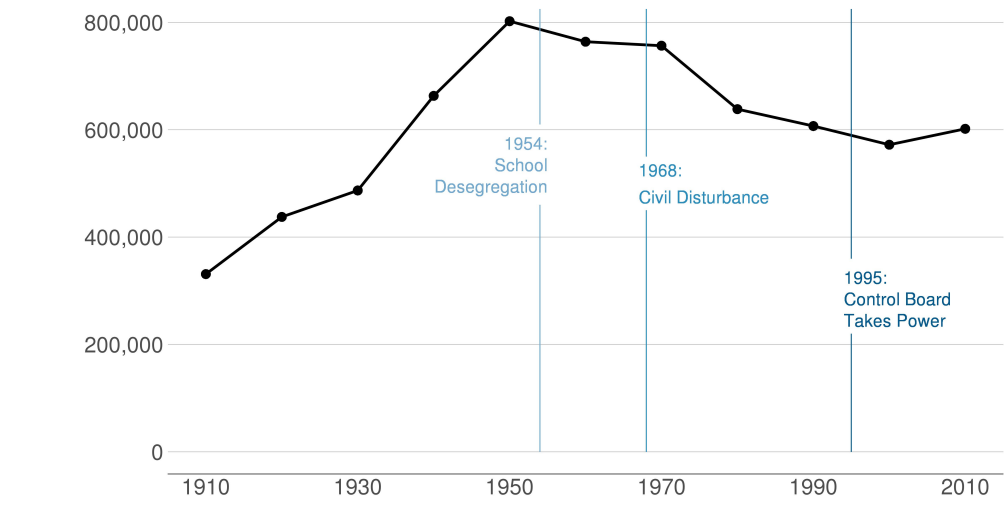
## DC Gains Population Through 1950




## Population Loses Start with Desegregation






## B.2 Load and prepare data

Let's begin by downloading data from here. These are county-level data on population 1910 to 2010 (among other variables).

Use `read.csv` to grab these data as we've done before.

```
# load data
counties <- read.csv("h:/pppa_data_viz/2019/tutorial_data/lecture08/counties_1910to2010_20180116.csv")
```

Now just limit the data to DC. You could do this in the `ggplot` call itself. However, in this case when we are only planning to use DC, this gives us a smaller dataset to work with and that speeds processing. This will also make the coding easier, since we won't have to subset in each graph.

Take a look at the data after we subset to DC. Does it have the right number of observations?

```
# get just dc
dct <- counties[which(counties$statefips == 11),]
dim(dct)
```

```
## [1] 11 68
```

```
dct[,c("year","statefips","countyfips","cv1")]
```

```
##       year statefips countyfips    cv1
## 285   1910        11          1 331069
## 3244  1920        11          1 437571
## 6314  1930        11          1 486869
## 9418  1940        11          1 663091
## 12520 1950        11          1 802178
## 15626 1960        11          1 763956
## 18764 1970        11          1 756510
## 21899 1980        11          1 638333
## 25039 1990        11          1 606900
## 28182 2000        11          1 572059
## 31326 2010        11          1 601723
```

## B.3. Make the simplest line graph

Now that you know many `ggplot` commands, it will not be a shock to hear that you make a line graph using `geom_line()`. As for all `ggplot` graphs, you should specify a dataframe and x and y variables. Below we make the simplest possible line graph.

```
b3 <- ggplot() +
  geom_line(data = dct, aes(x = year, y = cv1))
b3
```

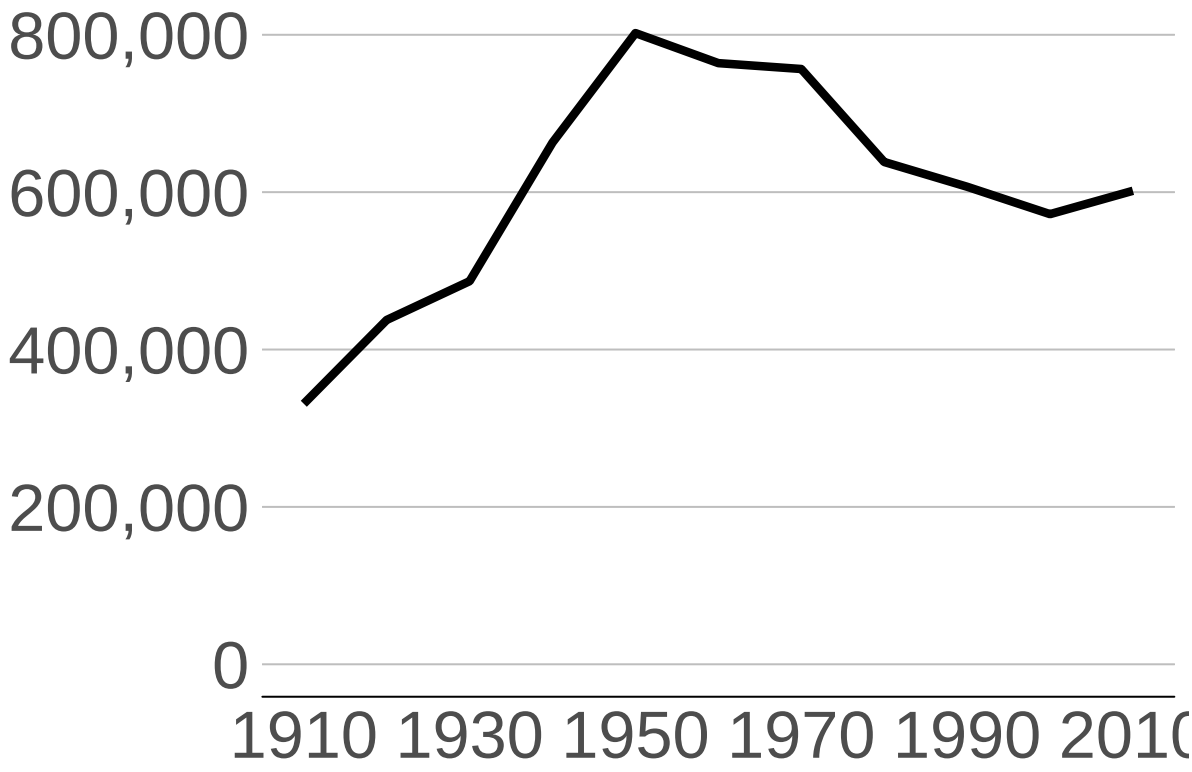

## B.4. Make some improvements

The line graph above is great for getting a sense of the data. It's not so good for communicating. The axis labels don't line up with the years in the data, the vertical axis labels are hard to read, and we don't need the grey background.

We fix the horizontal axis to put 20-year labels with `scale_x_continuous(limits= c(1910, 2010), breaks = c(seq(1910,2010,20)))`. This tells R to start in 1910, and stop in 2010. It also tells R to make breaks on the axis at 1910 and every 20 years until 2010. We fix the vertical axis with `scale_y_continuous(labels = comma, limits = c(0, 825000), breaks = c(seq(0,800000,200000)))`. This tells R to use commas in the numbers, to start at 0 and end at 825,000, and to make value labels every 200,000.

It will also work better if the main line is a little thicker; we adjust this with the `geom_line()` option of `size = 1.5`.

```
done <-
  ggplot() +
  geom_line(data = dct, mapping = aes(x=year, y=cv1), size=1.5) +
  scale_y_continuous(labels = comma,  limits = c(0, 825000), breaks = c(seq(0,800000,200000))) +
  scale_x_continuous(limits= c(1910, 2010), breaks = c(seq(1910,2010,20))) +
  labs(x="", y="") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        panel.grid.major.y = element_line(color="gray"),
        legend.position = "none",
        axis.line.x = element_line(color = "black"),
        axis.ticks.x = element_blank(),
        axis.ticks.y = element_blank(),
        axis.text = element_text(size = 25),
        axis.title = element_text(size = 20),
        plot.title = element_text(size=25))
done
```

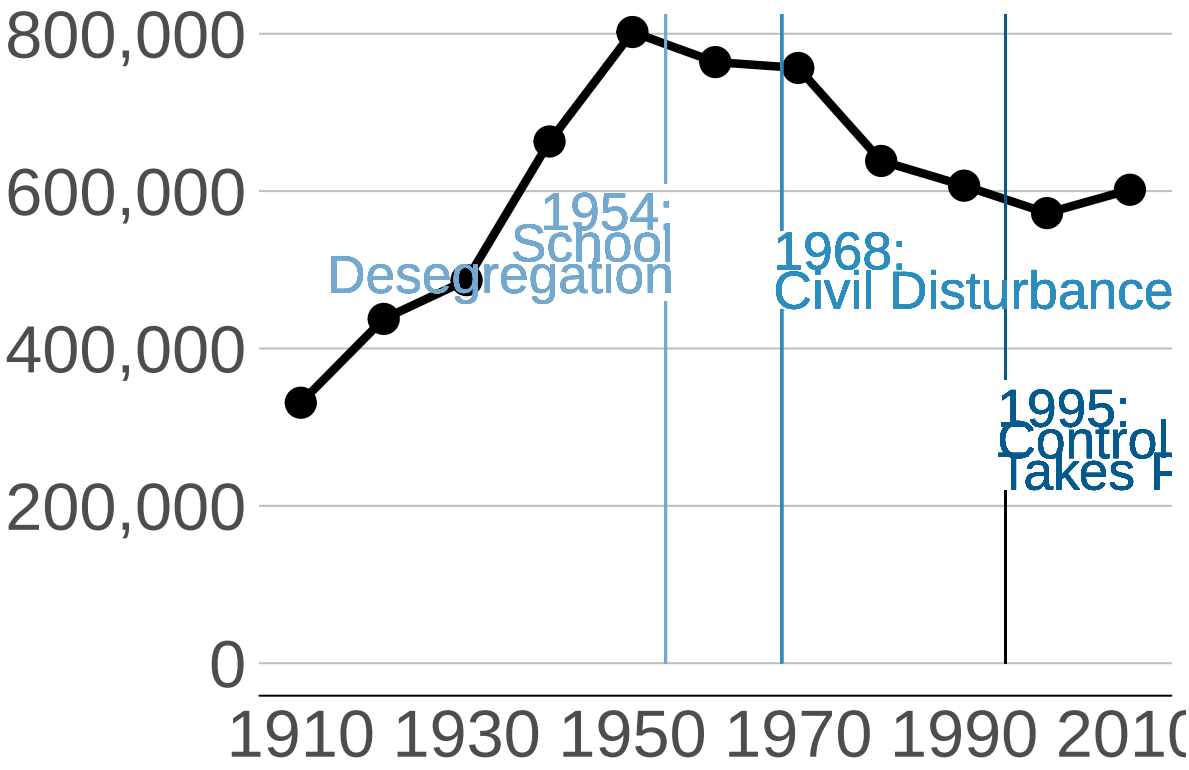



### B.5 A few more improvements

I'd call the above plot functional, but the point of this graph is to point out specific historical moments that explain the shape of the plot To do that we add text and lines to the plot. We add text with `geom_text()`,where we just put in the data directly. We add lines with `geom_segment()`. Getting the right x and y coordinates for these lines took me a very long time.

We also add dots on the line via `geom_point()`. This alerts readers to the fact (sort of) that the data are only actually at the points. The other parts of the graph are really just filled in.

```
done2 <-
  ggplot(dct) +
  geom_line(dct, mapping = aes(x=year, y=cv1), size=1.5) +
  geom_point(dct, mapping = aes(x=year, y=cv1), size=5) +
  scale_y_continuous(labels = comma,  limits = c(0, 825000), breaks = c(seq(0,800000,200000))) +
  scale_x_continuous(limits= c(1910, 2010), breaks = c(seq(1910,2010,20))) +
  #scale_shape_manual(values = c(16, 21)) +
  labs(x="", y="") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        panel.grid.major.y = element_line(color="gray"),
        legend.position = "none",
        axis.line.x = element_line(color = "black"),
        axis.ticks.x = element_blank(),
        axis.ticks.y = element_blank(),
        axis.text = element_text(size = 25),
        axis.title = element_text(size = 20),
        plot.title = element_text(size=25)) +
  geom_segment(x=1995, y=0, xend=1995, yend=220000, color="black") +
  geom_segment(x=1995, y=360000, xend=1995, yend=825000, color="#045a8d") +
  geom_segment(x=1968, y=0, xend=1968, yend=450000, color = "#2b8cbe") +
  geom_segment(x=1968, y=550000, xend=1968, yend=825000, color = "#2b8cbe") +
  geom_segment(x=1954, y=0, xend=1954, yend=460000, color = "#74a9cf") +
  geom_segment(x=1954, y=610000, xend=1954, yend=825000, color = "#74a9cf") +
  geom_text(x=1955, y=575000, label="1954:", color = "#74a9cf", size=7, hjust=1) +
  geom_text(x=1955, y=535000, label="School", color = "#74a9cf", size=7, hjust=1) +
  geom_text(x=1955, y=495000, label="Desegregation", color = "#74a9cf", size=7, hjust=1) +
  geom_text(x=1967, y=525000, label="1968:", color = "#2b8cbe", size=7, hjust=0) +
  geom_text(x=1967, y=475000, label="Civil Disturbance", color = "#2b8cbe", size=7, hjust=0) +
  geom_text(x=1994, y=325000, label="1995:", color="#045a8d", size=7, hjust=0) +
  geom_text(x=1994, y=285000, label="Control Board", color="#045a8d", size=7, hjust=0) +
  geom_text(x=1994, y=245000, label="Takes Power", color="#045a8d", size=7, hjust=0)
done2
```




## C. Showing multiple counties

The section above graphs just DC. In this section, we graph multiple counties and make them distinguishable.

### C.1 Keep just a few counties

Here we subset the `counties` data to just keep DC (state 11), Maryland's Montgomery and Prince George's counites (state 24, counties 31 and 33), and Virginia's Arlington, Alexandria and Fairfax jurisdictions (state 51, counties 13, 510 and 59).

We also make the year variable numeric for ease of plotting. And then to save space, we get rid of the counties dataframe with `rm(counties)`. You can use `rm()` for any objects that you no longet want.

```
dcm <- counties[which(counties$statefips == 11 |
                      counties$statefips == 24 & counties$countyfips %in% c(31,33) |
                      counties$statefips == 51 & counties$countyfips %in% c(13,510,59)),]
dcm$nyear <- as.numeric(dcm$year)
rm(counties)
```

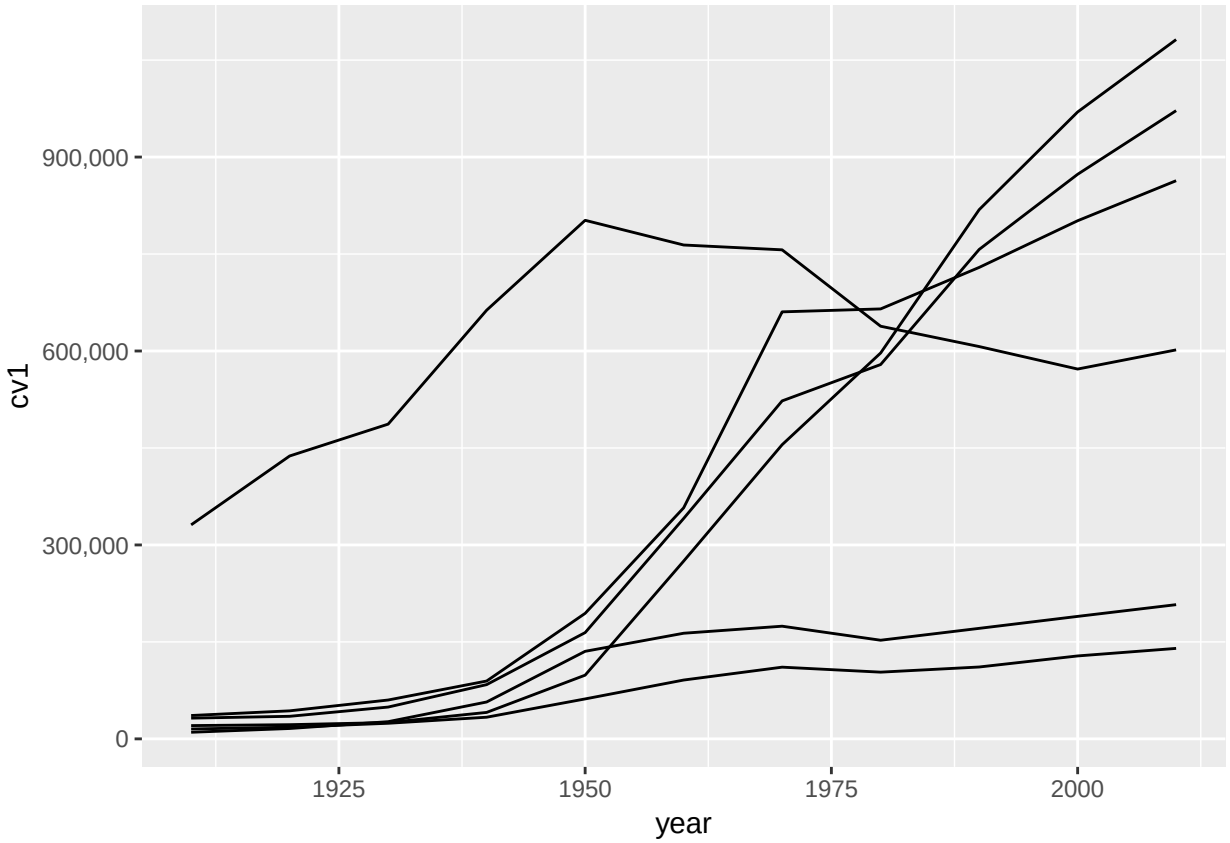
Finally, to identify a county in `ggplot`, we need both the state and county variables together. I use "paste0" to concatenate the state and county variables.

```
# make a state+county variable
dcm$stc <- paste0(dcm$statefips,dcm$countyfips)
```

## C.2. Plot all counties

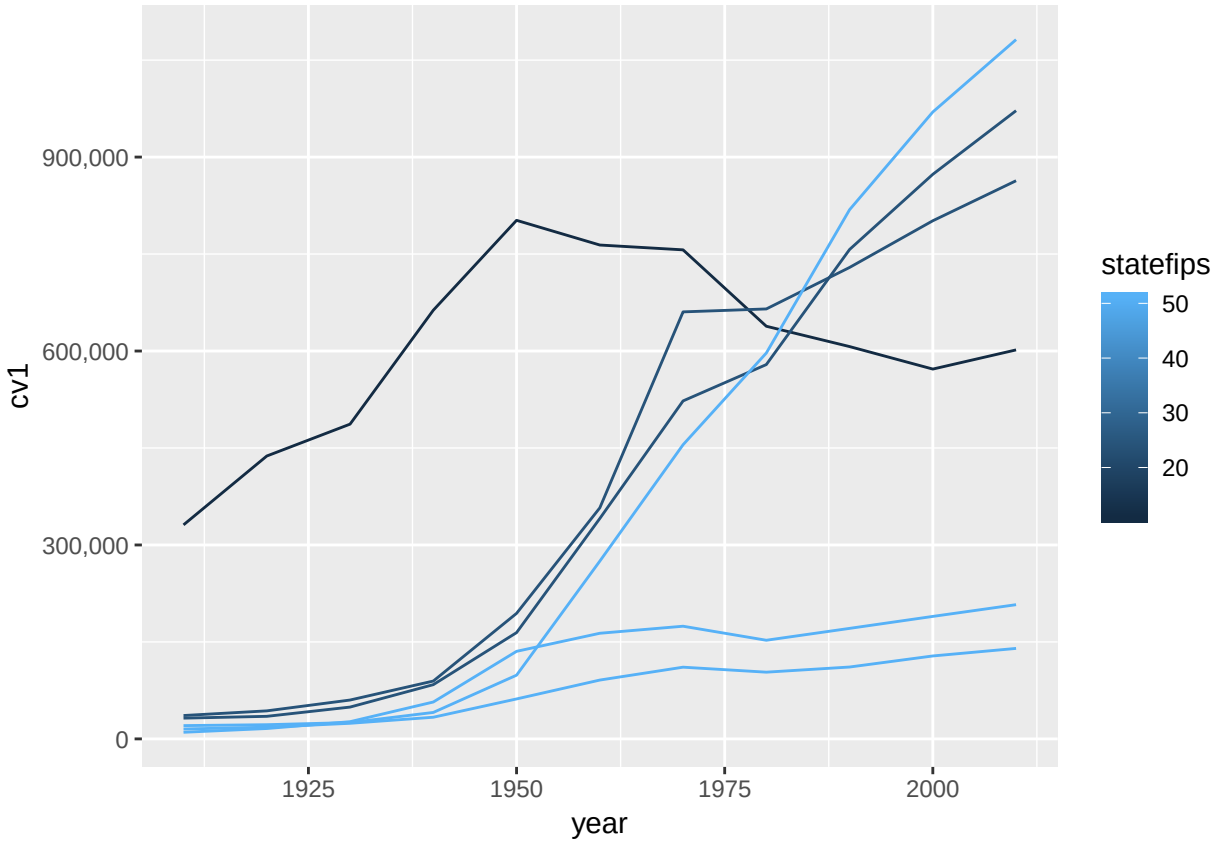
To plot multiple counties at one time, we use the `group()` command and tell R that the groups are by the `stc` variable. For legibility, I add commas in the y-axis values.

```
# all counties
ac <- ggplot() +
  geom_line(data = dcm, aes(x=year, y=cv1, group = stc)) +
  scale_y_continuous(labels = comma)
ac
```



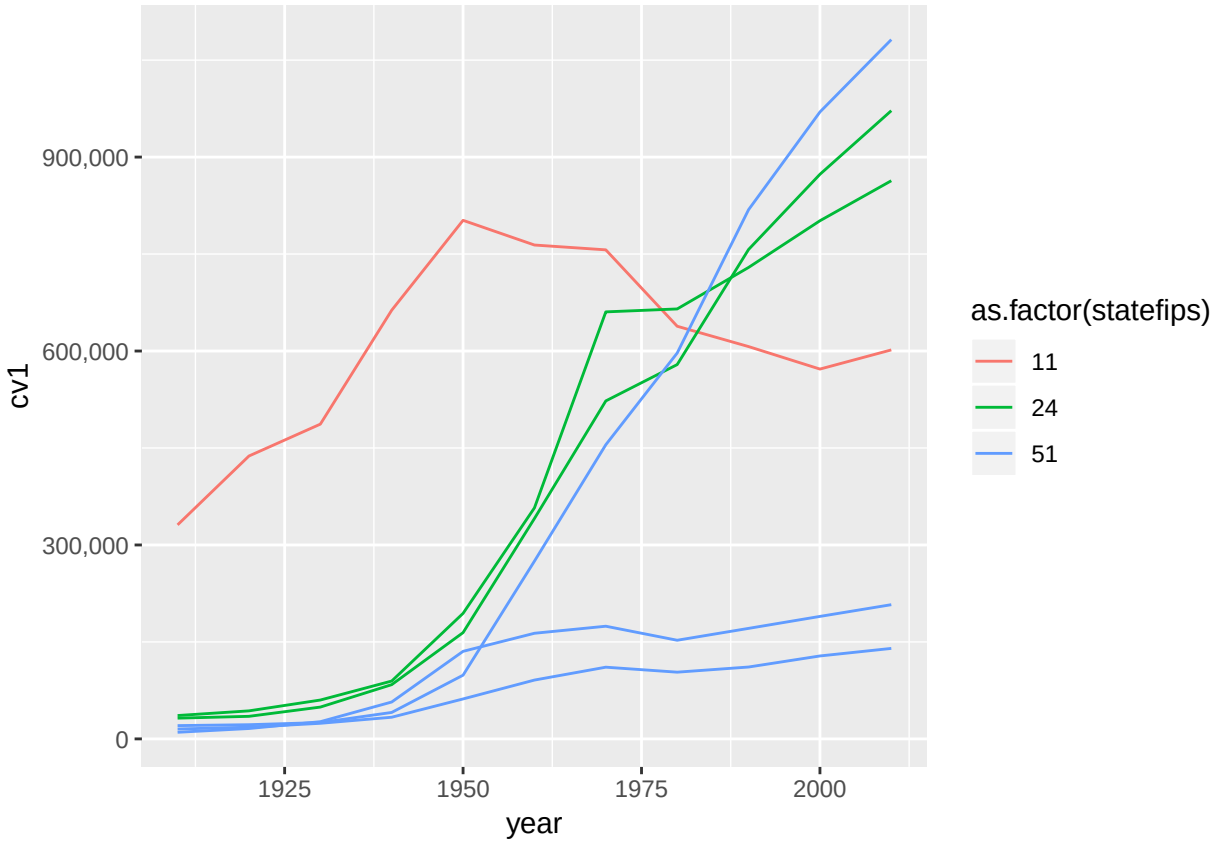
Of course, without a legend or identifying features, this graph is very hard to interpret. We add `color = statefips` so that we can see which states are where on the graph.

```
# color by state
ac <- ggplot() +
  geom_line(data = dcm, aes(x=year, y=cv1, group = stc, color = statefips)) +
  scale_y_continuous(labels = comma)
ac
```



This graph is an improvement, but it has the very unfortunate feature that it presents the state code as if it were a continuous variable. I use `as.factor()` around the state variable to let R kow it is categorical.

```
# color by state, itemized legend
ac <- ggplot() +
  geom_line(data = dcm, aes(x=year, y=cv1, group = as.factor(stc), color = as.factor(statefips))) +
  scale_y_continuous(labels = comma)
ac
```




There are still at least a few issues with this graph. For basic legibility, we should made the state codes numbers. You can do this by making a state factor variable and assigning it a level. We do not do this here, but you can see it in previous tutorial.

## D. Capital Bikeshare data

This section does a little more prep work to get to a line graph. Specifically, we

- practice with functions
- showing quoting problem of last class
- work with date variables
- summarize data

First, download the 2019/01 Capital bikeshare data from here. Use `read.csv()` to load these data.

```
cabi.201901 <- read.csv("H:/pppa_data_viz/2019/tutorial_data/lecture08/201902-capitalbikeshare-tripdata/

head(cabi.201901)
```

```
##   Duration          Start.date            End.date Start.station.number
## 1      206 2019-02-01 00:00:20 2019-02-01 00:03:47                31509
## 2      297 2019-02-01 00:04:40 2019-02-01 00:09:38                31203
## 3      165 2019-02-01 00:06:34 2019-02-01 00:09:20                31303
## 4      176 2019-02-01 00:06:49 2019-02-01 00:09:45                31400
## 5      105 2019-02-01 00:10:41 2019-02-01 00:12:27                31270
## 6      757 2019-02-01 00:12:37 2019-02-01 00:25:14                31503
##                                 Start.station End.station.number
## 1                     New Jersey Ave & R St NW              31636
## 2                   14th & Rhode Island Ave NW              31519
## 3 Tenleytown / Wisconsin Ave & Albemarle St NW              31308
## 4               Georgia & New Hampshire Ave NW              31401
## 5                                 8th & D St NW              31256
## 6                        Florida Ave & R St NW              31126
##                          End.station Bike.number Member.type
## 1 New Jersey Ave & N St NW/Dunbar HS      W21713      Member
## 2                       1st & O St NW      E00013      Member
## 3                  39th & Veazey St NW      W21703      Member
## 4               14th St & Spring Rd NW      W21699      Member
## 5                       10th & E St NW      W21710      Member
## 6                  11th & Girard St NW      W22157      Member
```

# E. Prepare bikeshare data

To get these data ready for a line chart, we need to get the time variable into a useful format. You can store dates as text, but then they don't have the useful properties of numbers – one of which is lining up properly along the x axis.

We also need to shrink the size of these data through summary statistics since 150,000 is too many to plot.

## E.1. Make date variables

Let's begin with calculating date variables. Date variables are a special kind of variable. They store time and date as the number of days since January 1, 1970. To convert the string variables `Start.date` and `End.date`, we use the `as.POSIXct` command.

You tell R what parts of the string correspond to which parts of the date.

```
cabi.201901$time.start <- as.POSIXct(strptime(cabi.201901$Start.date, "%Y-%m-%d %H:%M:%S"))
cabi.201901$time.stop  <- as.POSIXct(strptime(cabi.201901$End.date, "%Y-%m-%d %H:%M:%S"))
```

## E.2. Use date variables for duration calculation

Now that we have two date variables, we can make our own measure of duration and check the bikeshare's measure.

```
# my duration calculation
cabi.201901$my.duration <- cabi.201901$time.stop - cabi.201901$time.start

# comparing my results to built-in results
summary(cabi.201901$my.duration)
```

```
##   Length    Class     Mode
##   158130 difftime  numeric
```

```
summary(cabi.201901$Duration)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    60.0   349.0   577.0   895.9   957.0 86100.0
```

```
summary(as.numeric(cabi.201901$my.duration))
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##    1.000    5.817    9.633   14.939   15.967 1435.000
```

```
summary(as.numeric(cabi.201901$Duration))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    60.0   349.0   577.0   895.9   957.0 86100.0
```

This looks similar, but our calculated duration measure is in minutes and the bikeshare's measure is in seconds. I could divide `cabi$Duration` by 60 to see if they are the same. I can also look at the correlation between the two measures using `cor()`. I find that the correlation between the two measures is 1. This makes we suspect that the bikeshare people calcualted the duration in the exact way we did.

```
# look at the correlation -- looks like 1
cor(x = as.numeric(cabi.201901$my.duration),
    y = as.numeric(cabi.201901$Duration),
    method = c("pearson"))
```

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

# F. Plotting bikeshare data

The trick to successfully plotting these data is to reduce their dimensionality. The dataframe has about 150,000 observations – way too many to show on any plot.

## F.1. Make hour level summary stats

Our first pass at reducing the dimensionality is find hourly averages. First I extract the hour component from the date variable using the date notation. We then check the output using both `summary()` and `table()`.

```
# get the hour out of the date variable
cabi.201901$start.hour <- as.numeric(format(cabi.201901$time.start, "%H"))
summary(cabi.201901$start.hour)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00    9.00   14.00   13.63   17.00   23.00
```

```
table(cabi.201901$start.hour)
```

```
##
##     0     1     2     3     4     5     6     7     8     9    10    11
##   949   539   383   162   284  1369  3931  9314 15908  9409  6123  6818
##    12    13    14    15    16    17    18    19    20    21    22    23
##  8025  8304  8311  9728 12994 18785 13846  8570  5713  4082  2904  1679
```

## F.2. Find total number of rides and average duration by hour

We use `group_by()` and `summarize()` to find hourly traveling information, calculating both the number of rides (`no_rides`) and the average duration of those rides (`mean_dur`).

```
# summarize to hourly data
cabi.201901 <- group_by(cabi.201901, start.hour)
cabisum <- summarize(.data = cabi.201901, no_rides = n(), mean_dur = mean(Duration))
dim(cabisum)
```

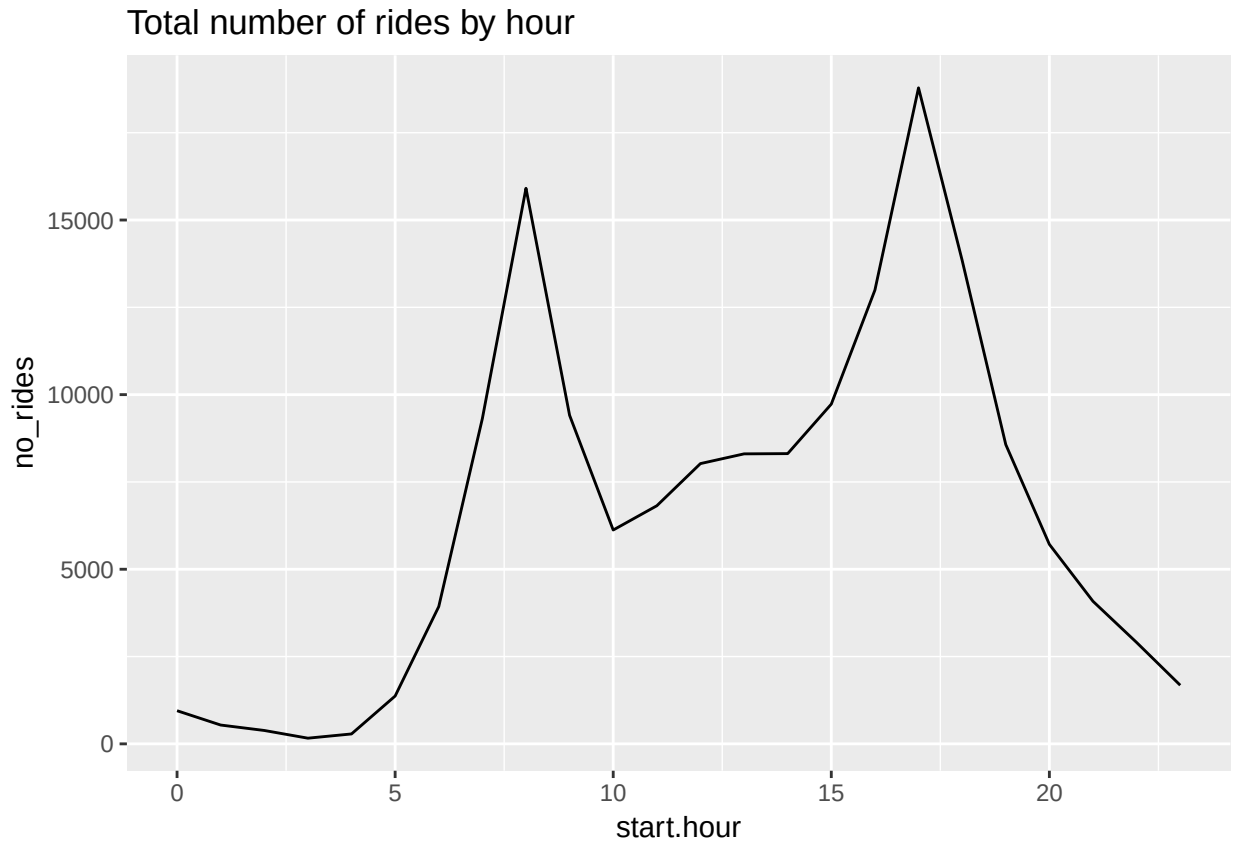
```
## [1] 24  3
```

## F.3. Plot of number of rides and mean duration

Now we'll plot the results. First, let's make sure we can get one variable to plot. Then we'll work on getting both variables wto plot using a function.

First we plot the number of rides by hour. Our first one goes smoothly:

```
# get it to work outside of a function
c3 <- ggplot() +
  geom_line(data = cabisum, mapping = aes(x = start.hour, y = no_rides)) +
  labs(title = "Total number of rides by hour")
c3
```

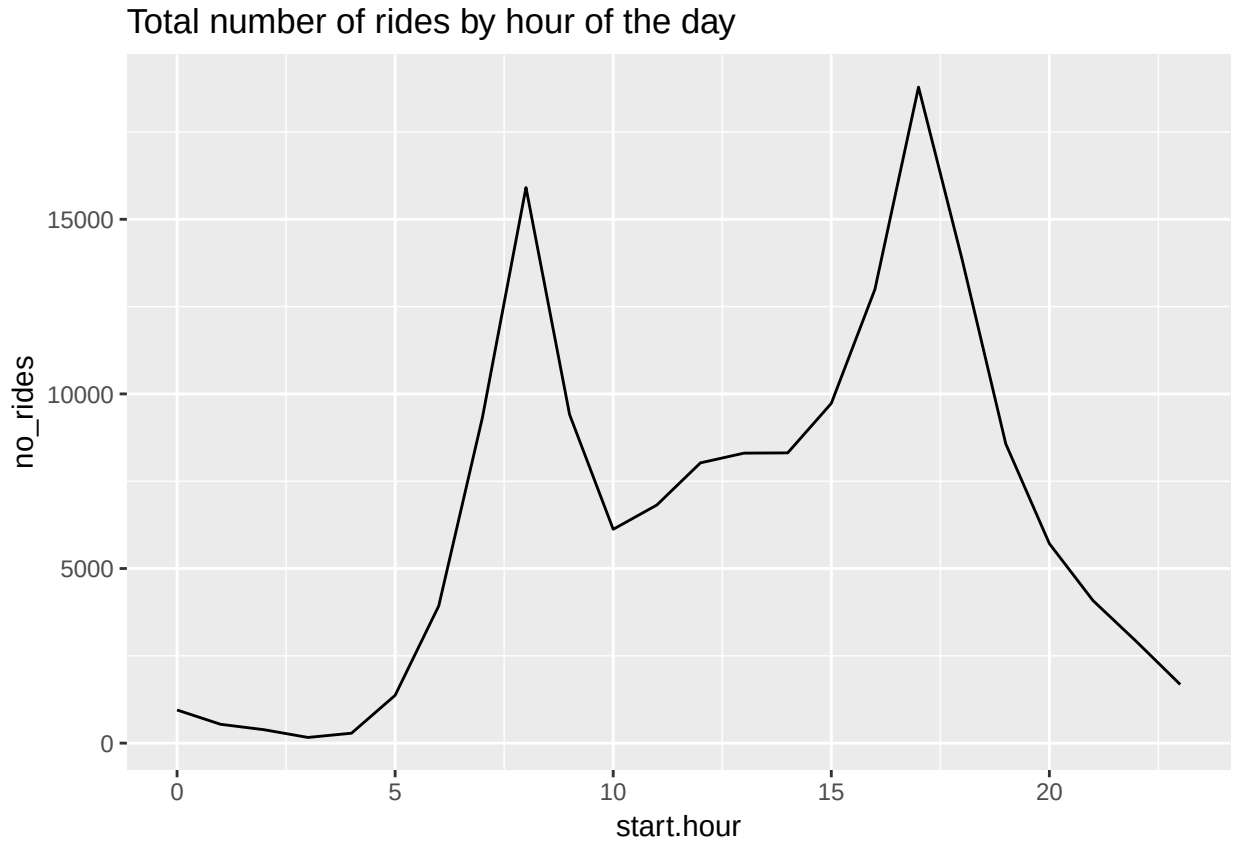



We'd prefer not to copy and paste this code multiple times for reasons we discussed in class. Instead, make a function. Many of you asked how to iterate through a variable with `ggplot()` in a function. The below shows you how. This document explains the "why."

The bottom line is that `ggplot()` uses non-standard evaluation, so it takes in text in a different way than base R commands. Thus, to replace text in a `ggplot` command, you need to input it with `quo()` and get rid of the `quo()` with `!!` before the new text.

```
# write a simple function for two variables
# see examples here
# https://dplyr.tidyverse.org/articles/programming.html
c3func <- function(varin,vardescp){
  c3 <- ggplot() +
    geom_line(data = cabisum, mapping = aes(x = start.hour, y = !! varin)) +
    labs(title = vardescp)
  print(c3)
}

c3a <- c3func(varin = quo(no_rides), "Total number of rides by hour of the day")
```




```
c3a <- c3func(varin = quo(mean_dur), "Average duration of ride by hour of the day")
```


## G. Stacked lines, useful for money

The final type of line graph we're trying today is stacked lines, which can sometimes be very helpful to convey change over time along with the relative importance of categories.

### G.1. Load data

Since this is a policy class, we'll use budget data for this topic. We are introducing a new dataset: US federal budget statistics. You can find the data from the Office of Management and Budget here. Download the zip file from the top of the page and unzip it.

I am not prepping these data for you, since I want to make sure you learn how to put raw data into R. You will find that there are many small issues that cause trouble – this is not atypical, and I want to be sure you know how to handle them.

Unzip the file you downloaded, and you'll see a bunch of files in this new folder. They follow the naming convention on the page from which you downloaded. Open up Tables 1.3 (hist01z3.xls; for homework) and 2.3 (hist02z3.xls; for now) in Excel.

From Table 2.3, we want the year and columns B, C, D, G, H and I. Create a new excel document with just this information, and make one row at top with names that you'll understand. Keep just through 2017, and make sure that you don't have any junk at the bottom of the table. Save this file as csv (file, save as, choose "csv" option for file type). If there are numeric variables that take the value `*'', make them`.", which is code for missing.

Load the csv file you just saved.

```
### makeup of receipts ####
hist02z3 <- read.csv("H:/pppa_data_viz/2018/tutorials/lecture05/omb_data/hist02z3.csv")
names(hist02z3)
```

```
## [1] "year"         "income.taxes" "corp.taxes"   "social.ins"
## [5] "excise"       "other"        "total"
```

Begin by making sure that what you've imported into R is what you expect.

We'll start with the `year` variable, using `tables()`.

```
# make sure year is always ok
table(hist02z3$year)
```

```
##
##      1934 1935 1936 1937 1938 1939 1940 1941 1942 1943 1944 1945 1946 1947
##    6    1    1    1    1    1    1    1    1    1    1    1    1    1    1
## 1948 1949 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 1960 1961 1962
##    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
## 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977
##    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
## 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992
##    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
## 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007
##    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
## 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017   TQ
##    1    1    1    1    1    1    1    1    1    1    1
```

Notice that there are some odd things here. A few observations with no year at all, and one observation where year is "TQ". Let's fix this.

We use subset the data to get rid of the strange years and then create a numeric variable using the same method as in part B. We use `summary()` to make sure this cleaned up variable takes on reasonable values.

```
hist02z3 <- hist02z3[which(hist02z3$year != ""),]
hist02z3 <- hist02z3[which(hist02z3$year != "TQ"),]
hist02z3$nyear <- as.numeric(levels(hist02z3$year))[hist02z3$year]
```

```
## Warning: NAs introduced by coercion
```

```
summary(hist02z3$nyear)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1934    1955    1976    1976    1996    2017
```

Good – the year variable now seems to have just numeric years.

## G.2. Make data long

To make a stacked line (or really, any multiple line), the data needs to be long, not wide. As a refresher, wide data look like this, with one observation per unit:

```
wide <- data.frame(state = c("6","36","48"), female_pop = c("10","12","14"), male_pop = c("11","13","12
wide
```

```
##   state female_pop male_pop
## 1     6         10       11
## 2    36         12       13
```

```
## 3    48         14       12
```

Long data look like this, with one observation per unit and type:

```
long <- data.frame(state = c("6","36","48","6","36","48"),
                   pop = c("10","12","14","11","13","12"),
                   sex = c("female","female","female","male","male","male"))
long
```

```
##   state pop    sex
## 1     6  10 female
## 2    36  12 female
## 3    48  14 female
## 4     6  11   male
## 5    36  13   male
## 6    48  12   male
```

Note how this dataset requires a variable that tells you which type of population the row contains.

Neither data format is "right." If you were doing a regression and wanted to control for male and female population, you'd need the wide format. However, to make a line graph with multiple lines in R, you need a long dataset. We begin by re-naming our variables of interest to make the shift to a long dataframe easier.

```
## first rename stuff so the reshape command works
hist02z3$r1 <- as.numeric(hist02z3$income.taxes)
hist02z3$r2 <- as.numeric(hist02z3$corp.taxes)
hist02z3$r3 <- as.numeric(levels(hist02z3$social.ins))[hist02z3$social.ins]
```

```
## Warning: NAs introduced by coercion
```

```
hist02z3$r4 <- as.numeric(hist02z3$excise)
hist02z3$r5 <- as.numeric(hist02z3$other)
hist02z3$r6 <- as.numeric(hist02z3$total)
```

Let's be sure the values of these variables are all numeric. We do this with a loop. Loops are generally not preferred in R because whatever you do in a loop is usually better done with a command like `lapply()`, which we'll review in a future class.

However, for completeness, here's a loop example. The basic idea of a loop is that you repeat a set of code multiple times, modifying the code by the loop index each time.

Here's a loop that just prints something to the screen.

```
for(i in seq(1,3,1)){
  print(paste0("Hi Mr. Number ",i))
}
```

```
## [1] "Hi Mr. Number 1"
## [1] "Hi Mr. Number 2"
## [1] "Hi Mr. Number 3"
```

Let's make a little more complicated loop that checks if the variables are numeric.

```
# check numeric
nums <- seq(1,6,1)
nums
```

```
## [1] 1 2 3 4 5 6
```

```
for(i in nums){
  cv <- paste("hist02z3$r",i,sep="")
  ok <- is.numeric(cv)
```

```
  print(paste0("variable ",cv, " is numeric: ", ok))
}
```

```
## [1] "variable hist02z3$r1 is numeric: FALSE"
## [1] "variable hist02z3$r2 is numeric: FALSE"
## [1] "variable hist02z3$r3 is numeric: FALSE"
## [1] "variable hist02z3$r4 is numeric: FALSE"
## [1] "variable hist02z3$r5 is numeric: FALSE"
## [1] "variable hist02z3$r6 is numeric: FALSE"
```

Now we use tidyr's `gather()` as we have already done once before, to make a long dataframe.

```
## make this wide dataset long
r.long <- gather(data = hist02z3,
                 key = rtype,
                 value = revenue,
                 r1:r6)
r.long[1:15,]
```

```
##    year income.taxes corp.taxes social.ins excise other total nyear rtype
## 1  1934          0.7        0.6          .    2.2   1.3   4.8  1934    r1
## 2  1935          0.7        0.8          .    2.0   1.5   5.1  1935    r1
## 3  1936          0.8        0.9        0.1    2.0   1.1   4.9  1936    r1
## 4  1937          1.2        1.2        0.7    2.1   0.9   6.1  1937    r1
## 5  1938          1.4        1.4        1.7    2.1   0.9   7.5  1938    r1
## 6  1939          1.1        1.2        1.8    2.1   0.7   7.0  1939    r1
## 7  1940          0.9        1.2        1.8    2.0   0.7   6.7  1940    r1
## 8  1941          1.1        1.8        1.7    2.2   0.7   7.5  1941    r1
## 9  1942          2.2        3.2        1.7    2.3   0.5   9.9  1942    r1
## 10 1943          3.5        5.2        1.6    2.2   0.4  13.0  1943    r1
## 11 1944          9.2        6.9        1.6    2.2   0.5  20.5  1944    r1
## 12 1945          8.1        7.1        1.5    2.8   0.5  19.9  1945    r1
## 13 1946          7.1        5.2        1.4    3.1   0.5  17.2  1946    r1
## 14 1947          7.5        3.6        1.4    3.0   0.6  16.1  1947    r1
## 15 1948          7.4        3.7        1.4    2.8   0.6  15.8  1948    r1
##    revenue
## 1      0.7
## 2      0.7
## 3      0.8
## 4      1.2
## 5      1.4
## 6      1.1
## 7      0.9
## 8      1.1
## 9      2.2
## 10     3.5
## 11     9.2
## 12     8.1
## 13     7.1
## 14     7.5
## 15     7.4
```

Notice that `gather` has kept all the original variables. This may be confusing because the same data repeats across observations, making it unclear what uniquely identifies the data. So let's modify the dataframe to keep only the time-varying data that we just made long as as not to get confused.

```
## keep only relevant columns so we dont get confused
r.long <- r.long[,c("nyear","rtype","revenue")]
```

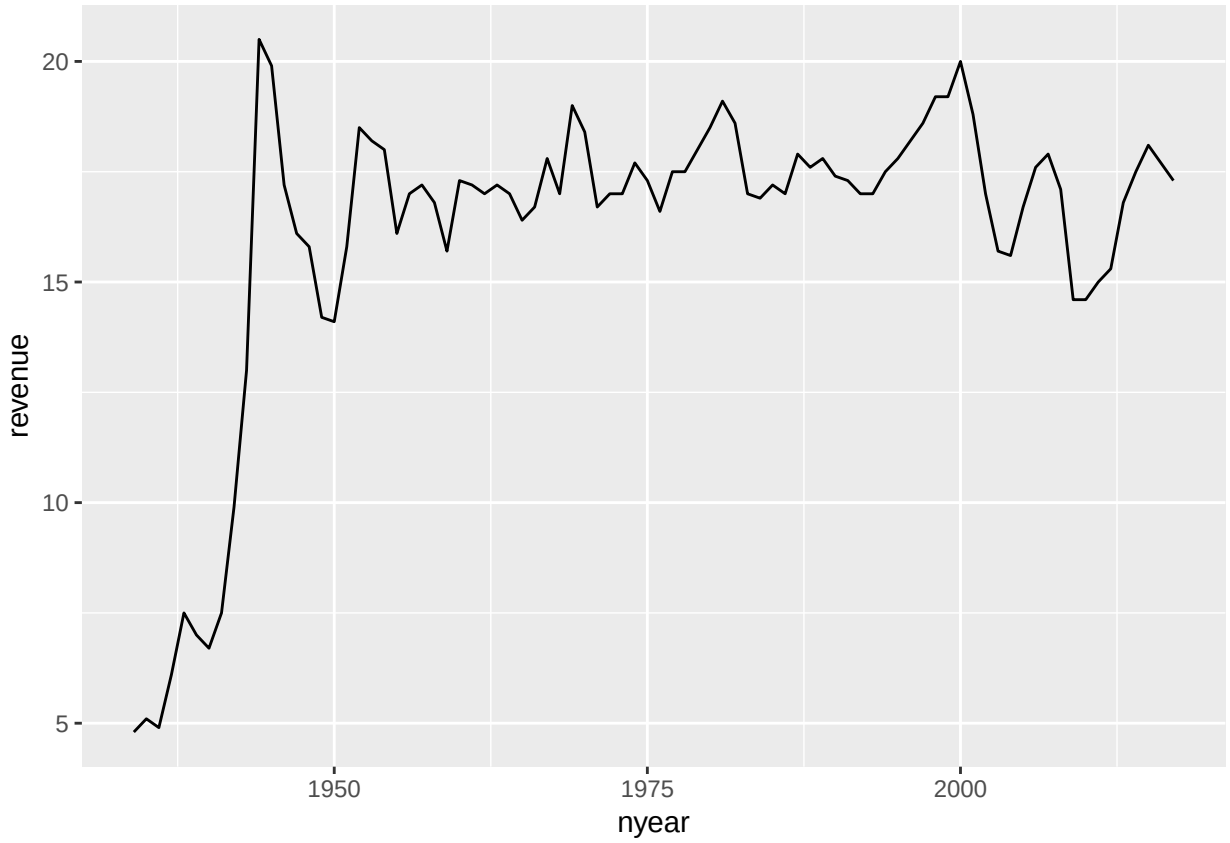
Now we create a character variable to record the type of tax revenue for each variable. This not required, but we do it so we can follow what's going on and use the names of the tax later on. Here we use the `ifelse()` command described in previous tutorials.

```
# make a type of receipts variable
r.long$rname <- ifelse(r.long$rtype == "r1","income taxes",
                       ifelse(r.long$rtype == "r2", "corp. taxes",
                              ifelse(r.long$rtype == "r3", "social ins.",
                                     ifelse(r.long$rtype == "r4", "excise",
                                            ifelse(r.long$rtype == "r5", "other",
                                                   ifelse(r.long$rtype == "r6", "total","."))))))
```

## G.3. Graphs

Let's start with total tax revenue over time. As in the previous section, we need to note `group=1`, and recall that total is `r.long$rtype == 6`.

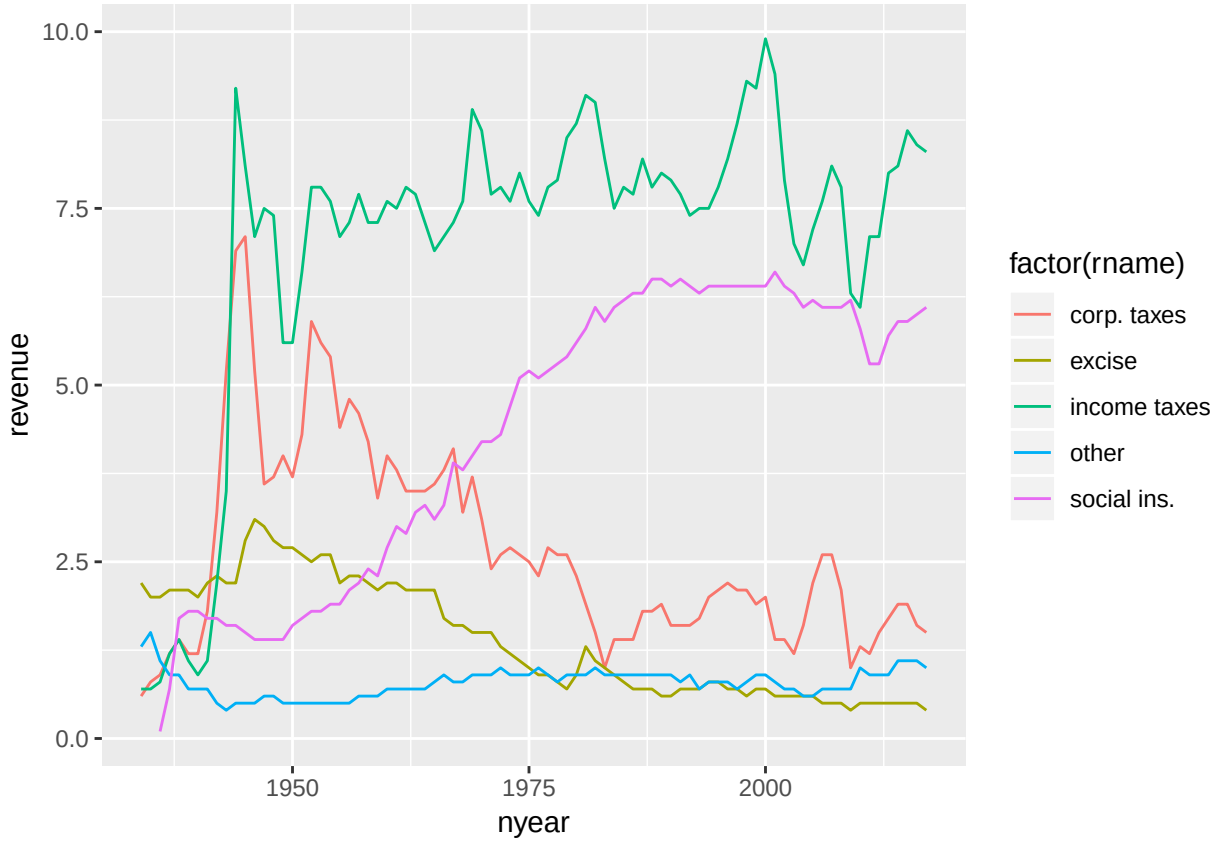
```
#### line chart of total receipts
g4.1 <-
  ggplot() +
  geom_line(r.long[which(r.long$rtype=="r6"),],
            mapping = aes(x=nyear, y=revenue, group=1))
g4.1
```



Now we'll modify the chart to have all the categories but the total. I do this by subetting `r.long` into all record types that are not 6 (`rtype != 6`). In addition, I tell R that the group by which we want to make the graph is a variable called `rname`, which R should treat as a factor. You could equally well use `rtype`, except that the labels on the chart would be numbers rather than names. We also tell r to color the lines by `rname` (`color=factor(rname)`).

```
#### line chart of total receipts by type ###
g4.2 <-
  ggplot() +
  geom_line(r.long[which(r.long$rtype != "r6"),],
            mapping = aes(x=nyear, y=revenue, group=factor(rname), color=factor(rname)))
g4.2
```

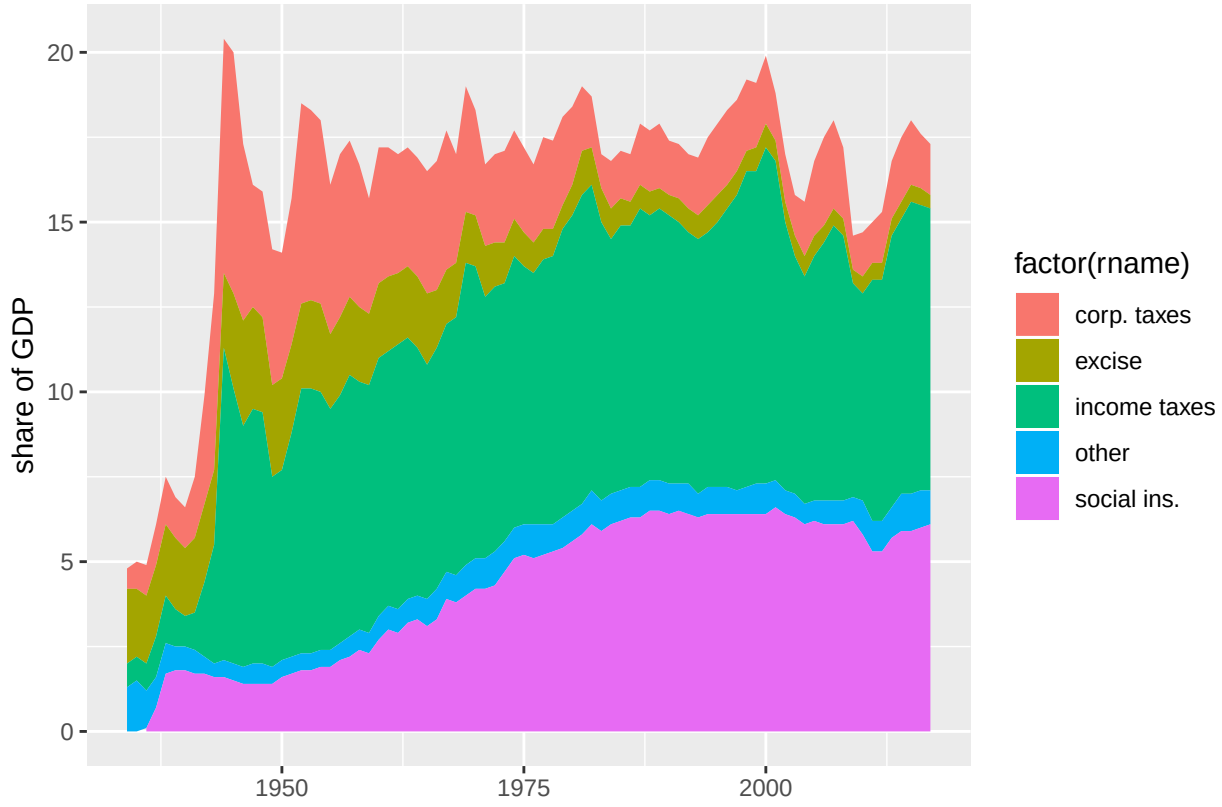
```
## Warning: Removed 2 rows containing missing values (geom_path).
```



This graph is very hard to read. Too many lines, and we don't get a sense of the total, which may be a key point. An alternative is a stacked line. Stacked likes highlight the total amount, and give readers some sense of the relative share of different categories.

```
#### stacked chart of total receipts by type ###
## without factor() this doesnt work
g4.3 <- ggplot() +
  geom_area(r.long[which(r.long$rtype != "r6"),],
            mapping = aes(x=nyear, y=revenue, group=factor(rname), fill=factor(rname)),
            position="stack") +
  labs(x="", y="share of GDP")
g4.3
```

```
## Warning: Removed 2 rows containing missing values (position_stack).
```




## H. Homework

1. Modify your code to make the three step-by-step versions of my chart of DC population over time.
2. Using the bikeshare data, re-do the by-hour pictures to be minute-by-minute. Add some annotations to your graph to point out salient features.
3. More stacked areas

Now you try to load your own budget data!

Use Table 1.3 (his01z3.xls), from which we want the year and columns E, F, G and columns I, J and K.

Create a new excel document with just this information, and make one row at top with names that you'll understand. Keep just through 2017, and make sure that you don't have any junk at the bottom of the table. Save this file as csv (file, save as, choose "csv" option for file type).

Load it into R and make a stacked area graph of receipts, outlays and deficits over time.

Having done this myself, here are a few suggestions

- make long data, as we did above
- make year numeric, as we did for the social insurance revenue above
- get rid of commas in the data. My command to do this, for one variable, is

```
hist01z3$b1 <- as.numeric(gsub(",", "", hist01z3$cd.receipts, fixed = TRUE))
```