

Tutorial 5: Line Charts

Leah Brooks

February 16, 2018

Today we are learning how to make line charts.

A. A very simple dataset: Defectors

To get started, we'll work again with the defectors dataset. This is to get a sense of how these graphs work – and I think this will make it clear why the data graphics people at Bloomberg (authors of the original chart) went with a bar chart, rather than a line one.

First we enter the data, as we did last tutorial.

```
# 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

Now I look at the data types. Different types of data require different commands. For example, factor variables behave differently than numerical variables. R's factor variables are a categorical variable; even when they are numeric (usually integer, but not necessarily), R thinks of them as taking on a limited number of discrete types. For R, factor variables are just levels, so if you have factors 1, 2 and 4, R will space them equally on the graph.

In contrast, if the same variable were numeric, R would put twice as much space between 2 and 4 as it would between 1 and 2.

We can check what kind of variables we have with the `typeof()` command.

```
typeof(nkd$year)
```

```
## [1] "integer"
```

```
typeof(nkd$defectors)
```

```
## [1] "integer"
```

Here we see that the variables are integers. But are they factors? If you don't know you can use the `is.factor()` function to check.

```
is.factor(nkd$year)
```

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

```
is.factor(nkd$defectors)
```

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

Yes – they are both factors.

For some of what we do below, it will be useful to have year as a numeric variable. We create a new variable `nyear` which is the numeric version of `nkd$year`. If you simply type `nkd$nyear <- as.numeric(nkd$year)`, you'll end up with numbers 1 to the total number of unique years. (Try it!) To get numeric year like the original year, use the `levels()` function. I don't actually know why we need the command in square brackets after, but I have had code fail without it – I think it somehow limits the levels to the ones in the current dataframe.

```
# make a numeric year
nkd$nyear <- as.numeric(levels(nkd$year))[nkd$year]
nkd
```

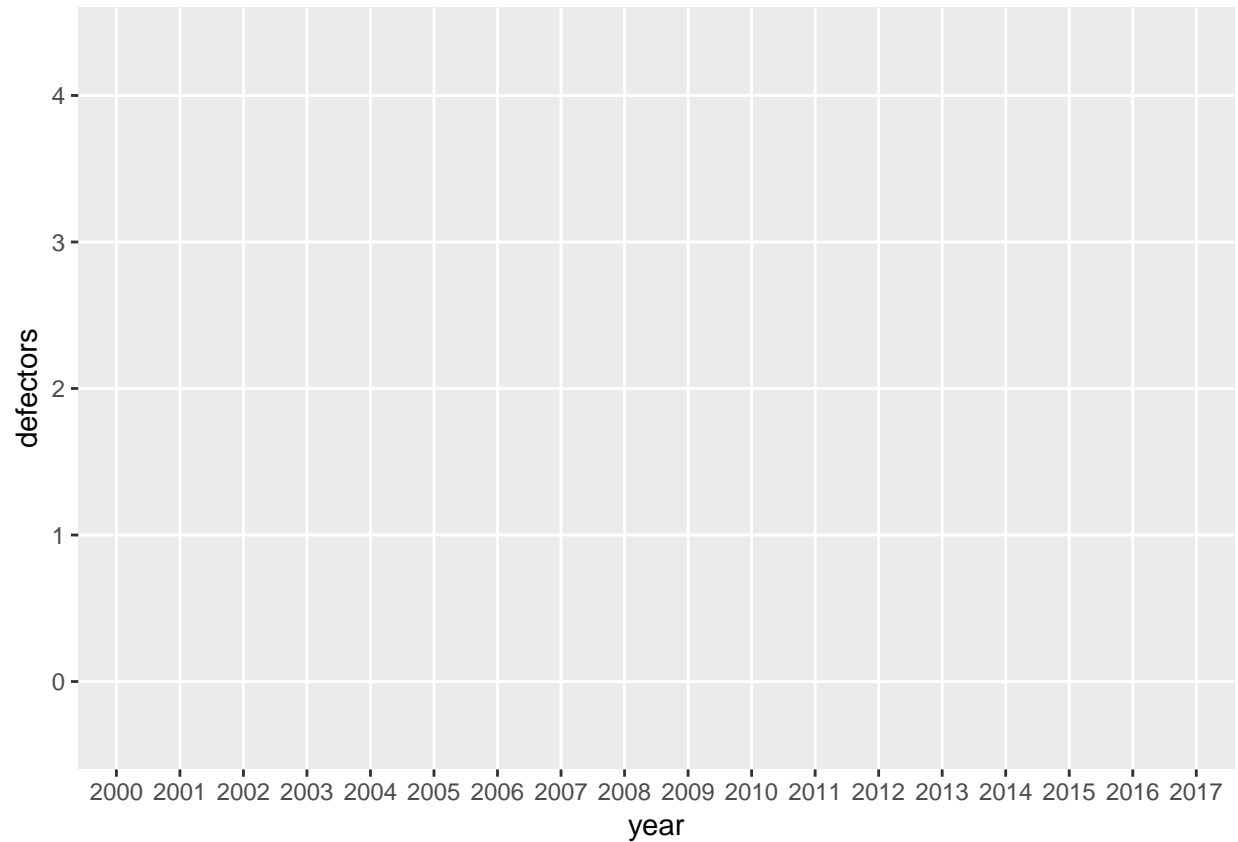
```
##   year defectors nyear
## 1  2000         0  2000
## 2  2001         0  2001
## 3  2002         1  2002
## 4  2003         0  2003
## 5  2004         0  2004
## 6  2005         0  2005
## 7  2006         0  2006
## 8  2007         0  2007
## 9  2008         2  2008
## 10 2009         0  2009
##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

This new variable looks the same, but it performs differently. You'll see examples with `year` and `nyear` below.

Now let's make a first line chart. Of course, load the `ggplot2` library and we'll start with the most basic line plot with `geom_line()`.

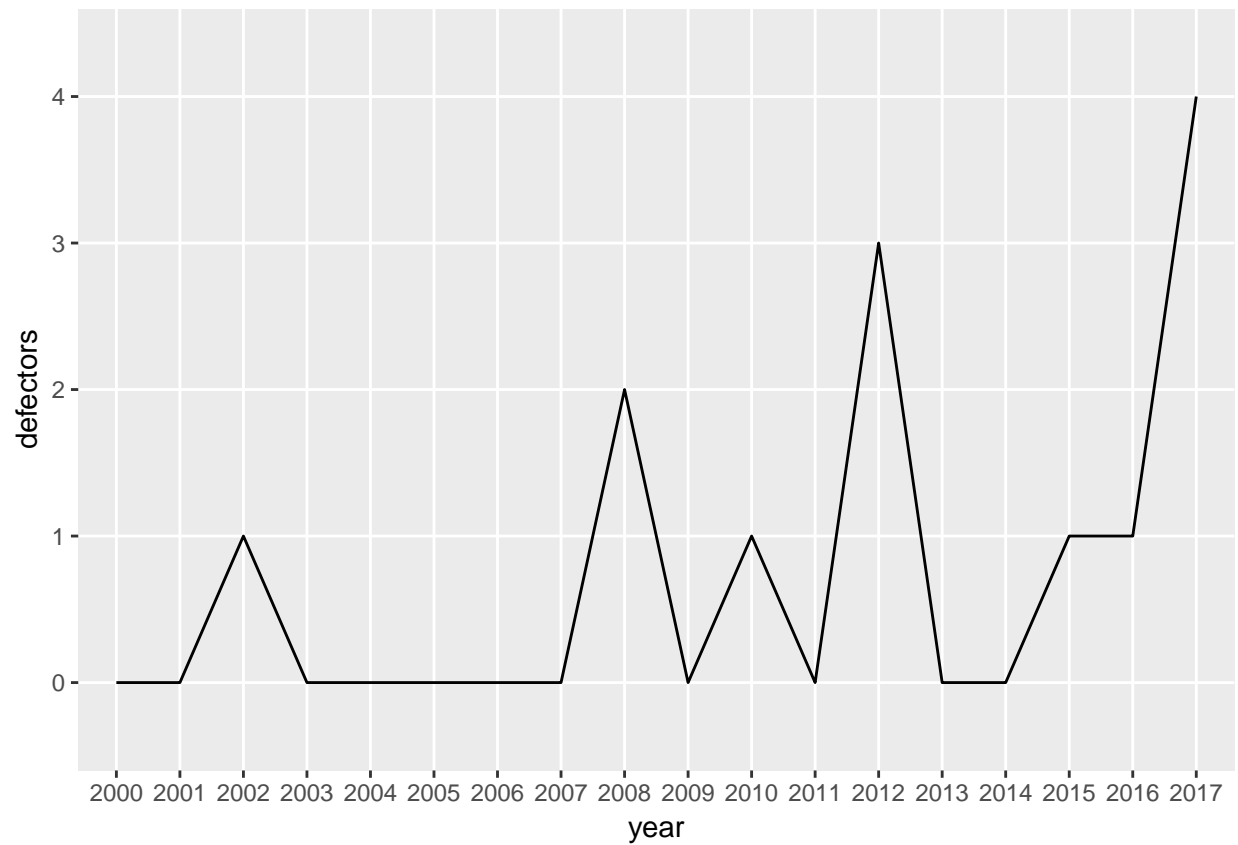
```
# basic line chart
library(ggplot2)
ggplot(nkd, aes(x=year, y=defectors)) + geom_line()
```

```
## geom_path: Each group consists of only one observation. Do you need to
## adjust the group aesthetic?
```



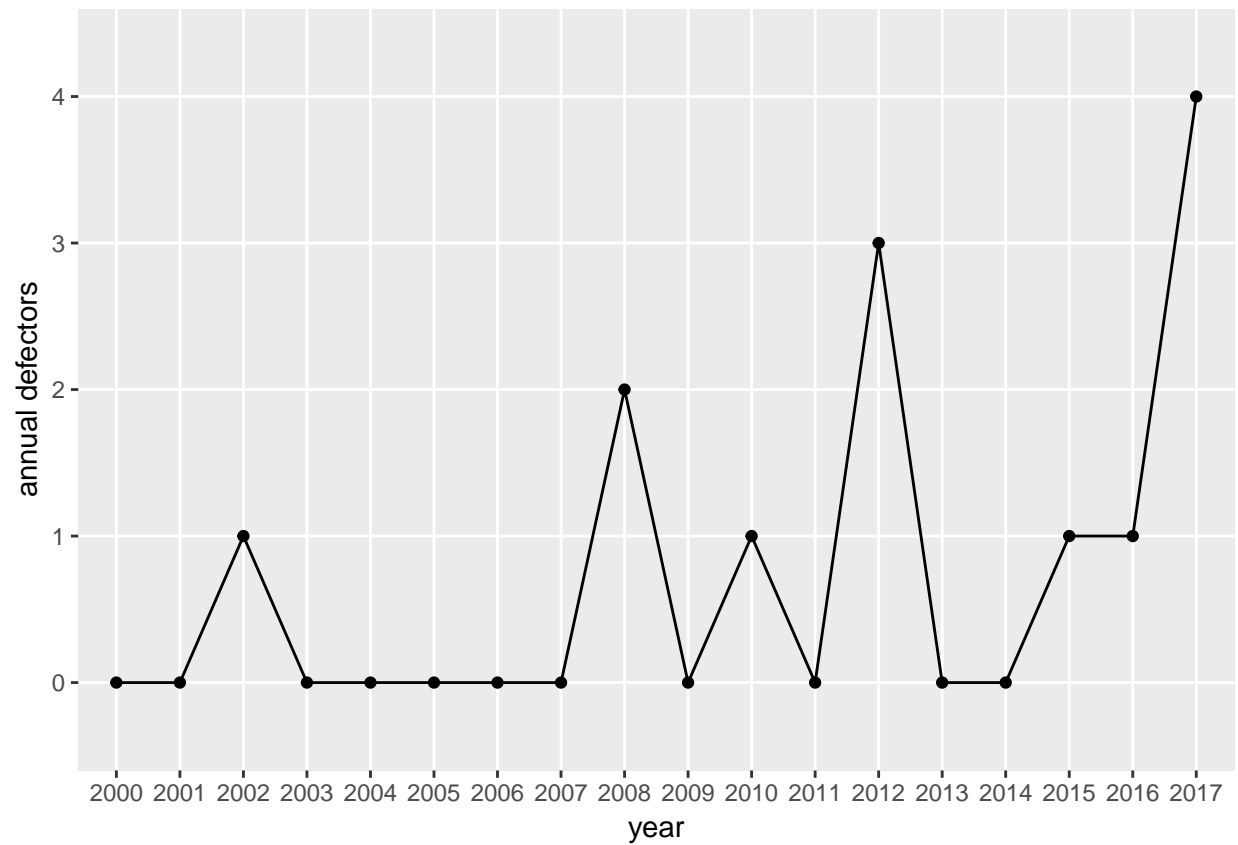
The plot yields nothing! Notice the warning message that R puts out. When you have only one group of lines, you need to tell that to R, and put it inside the `aes()` command as we do below. Setting `group=1` as we do below is just a way of telling R that the data have only one grouping.

```
ggplot(nkd, aes(x=year, y=defectors, group = 1)) + geom_line()
```



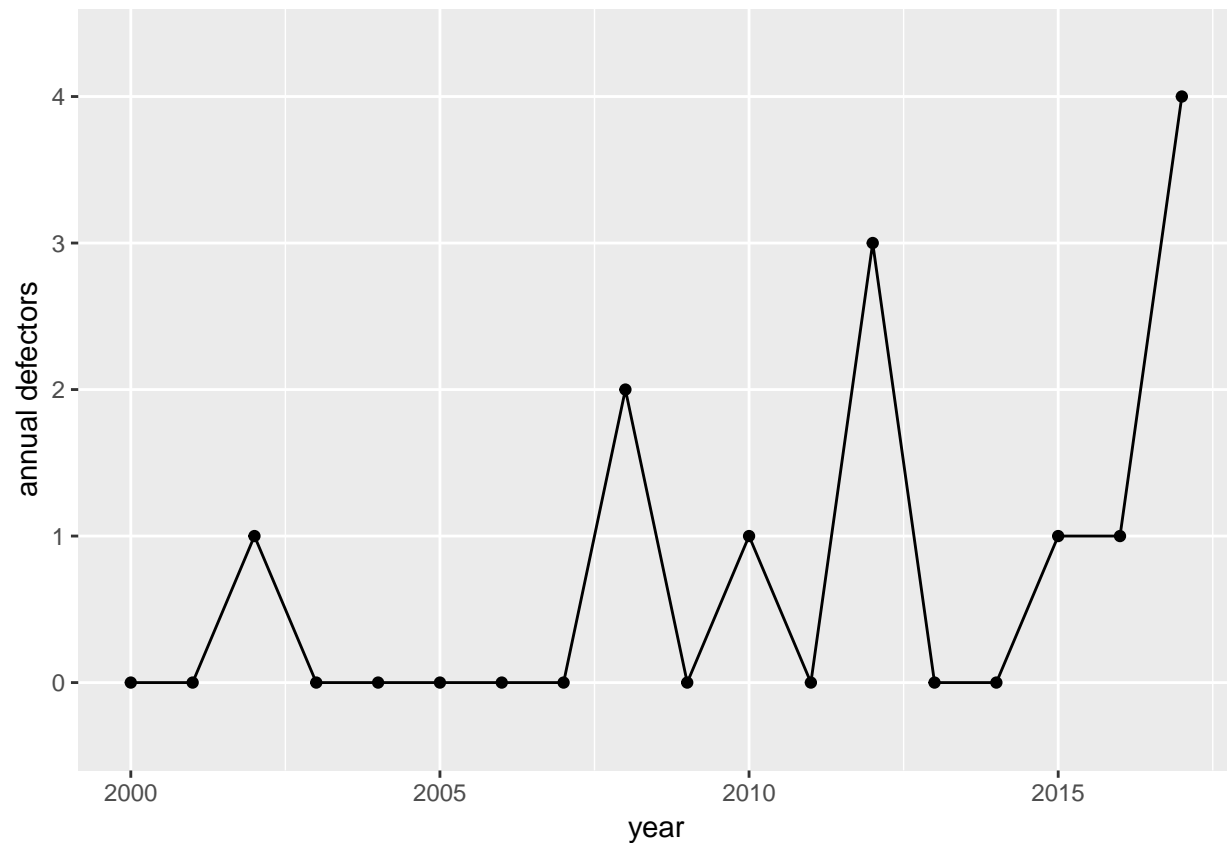
I think it now becomes clear why the Bloomberg data visualization people did not use a line chart for these data. Below we add some points to make clear where the actual data are.

```
ggplot(nkd, aes(x=year, y=defectors, group = 1)) + geom_line() +  
  geom_point() +  
  labs(x = "year", y="annual defectors")
```



Now we'll repeat this with the numeric year, rather than the factor year.

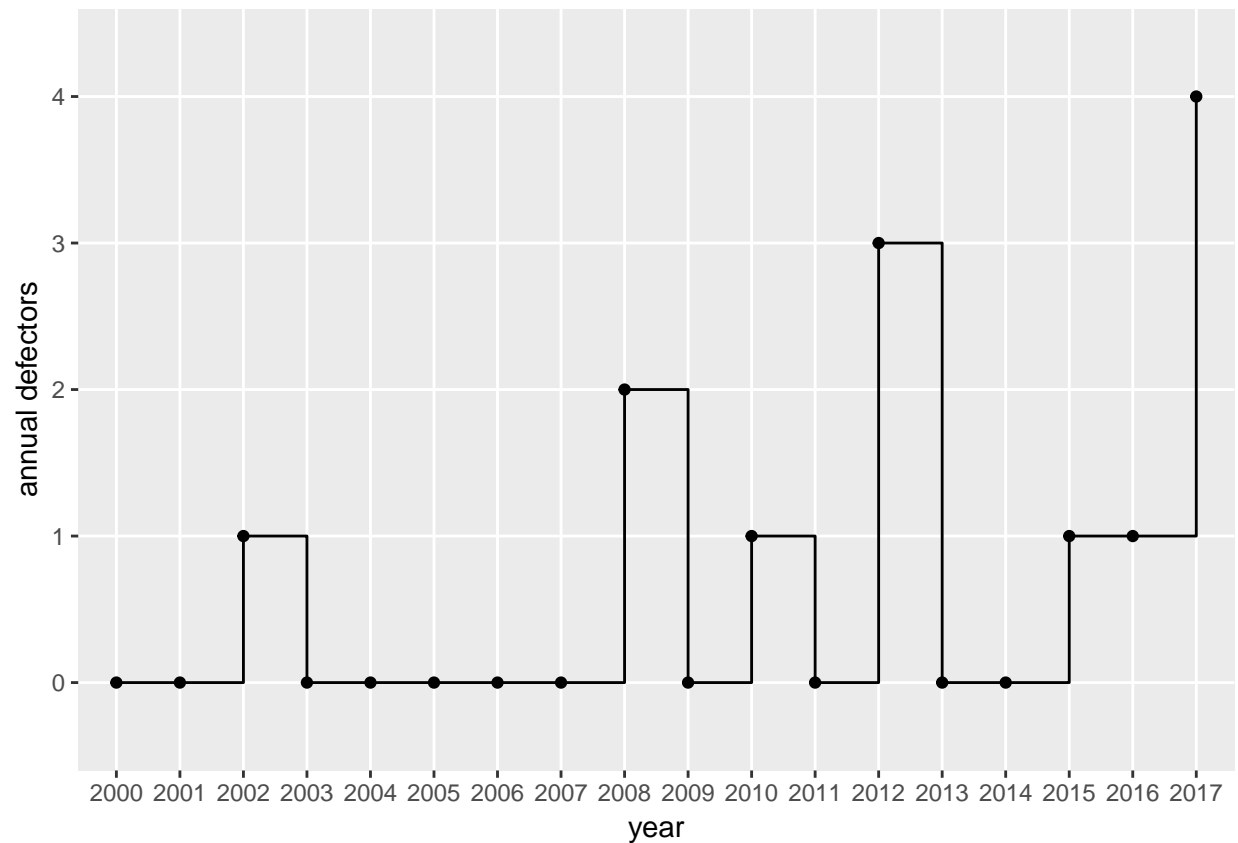
```
ggplot(nkd, aes(x=year, y=defectors, group = 1)) + geom_line() +  
  geom_point() +  
  labs(x = "year", y="annual defectors")
```



This looks better, and you can see that R scales the axis appropriately and R knows how to adjust the numbers.

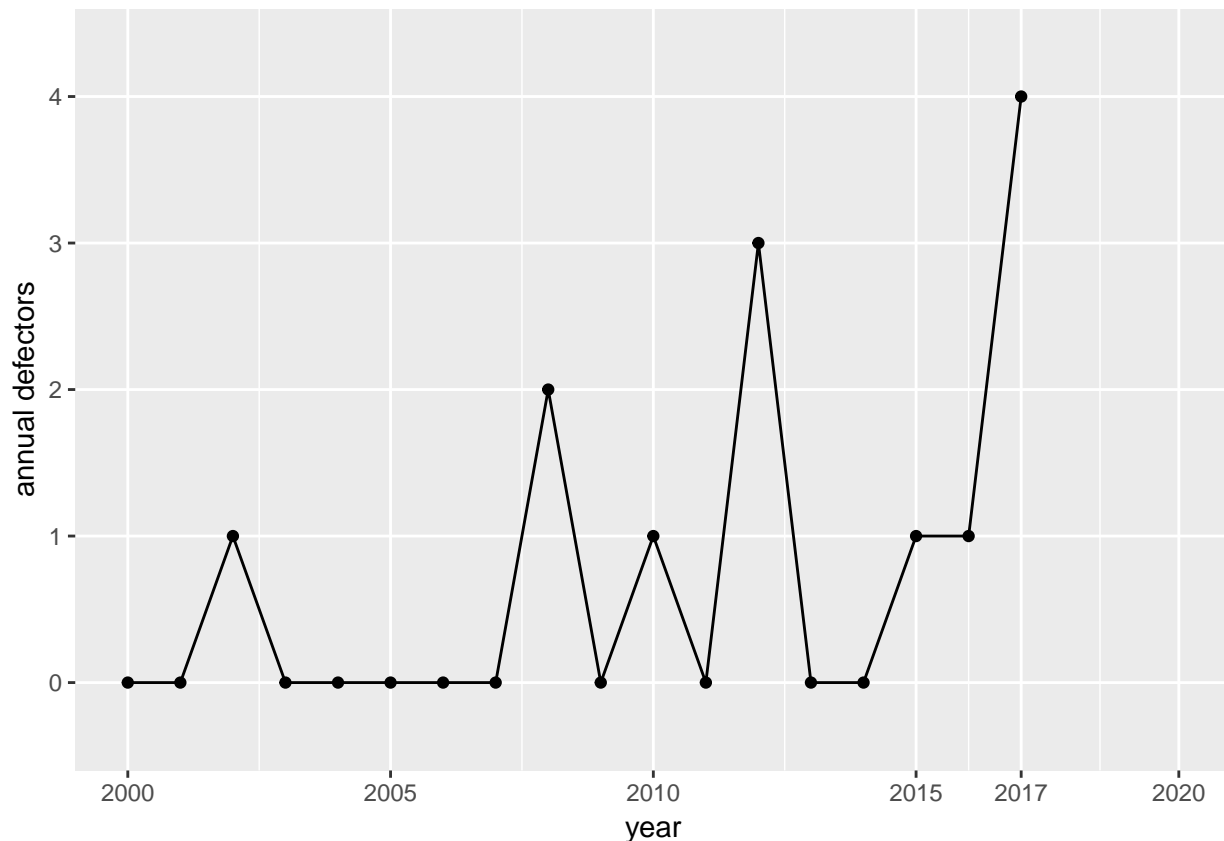
Just so you can see, we also try out the `geom_step()` rather than `geom_line()` command to avoid the cone like look. This is maybe better, but this does not seem like a good way to represent these data.

```
ggplot(nkd, aes(x=year, y=defectors, group = 1)) + geom_step() +  
  geom_point() +  
  labs(x = "year", y="annual defectors")
```



Finally, we'll clean up the horizontal axis. Below I use `scale_x_continuous()` to change the axis labels. The `limits = c()` option sets the beginning and the end, where you put the limits into `c()`. The `breaks = c()` sets axis breaks, again inside the `c()` vector.

```
# make the x-axis legible
ggplot(nkd, aes(x=year, y=defectors, group = 1)) + geom_path() +
  geom_point() +
  scale_x_continuous(limits= c(2000, 2020), breaks = c(2000, 2005, 2010, 2015, 2017, 2020)) +
  labs(x = "year", y="annual defectors")
```



B. Multiple Lines

Here we return to the county data to make line charts. Recall that the county data include information about counties from 1910 to 2010, so we can use them to make line charts.

Bring in the county data.

```
# load the data
counties <- read.csv("h:/pppa_data_viz/2018/tutorials/lecture01/counties_1910to2010_20180115.csv")
```

Now we are going to find total population by state and year. To do so, we'll use the `ddply` command that we've used in past tutorials. In brief, this command takes the county dataset and summarizes (shrinks) it so that it is by state and year (`c("statefips", "year")`), and reports the total population (`cv1 = sum(cv1)`). For a more detailed explanation, see tutorials from the previous classes.

```
# calculate total population by state over time
library(plyr)
states <- ddply(counties, c("statefips", "year"), summarize, cv1 = sum(cv1))
states[1:10,]
```

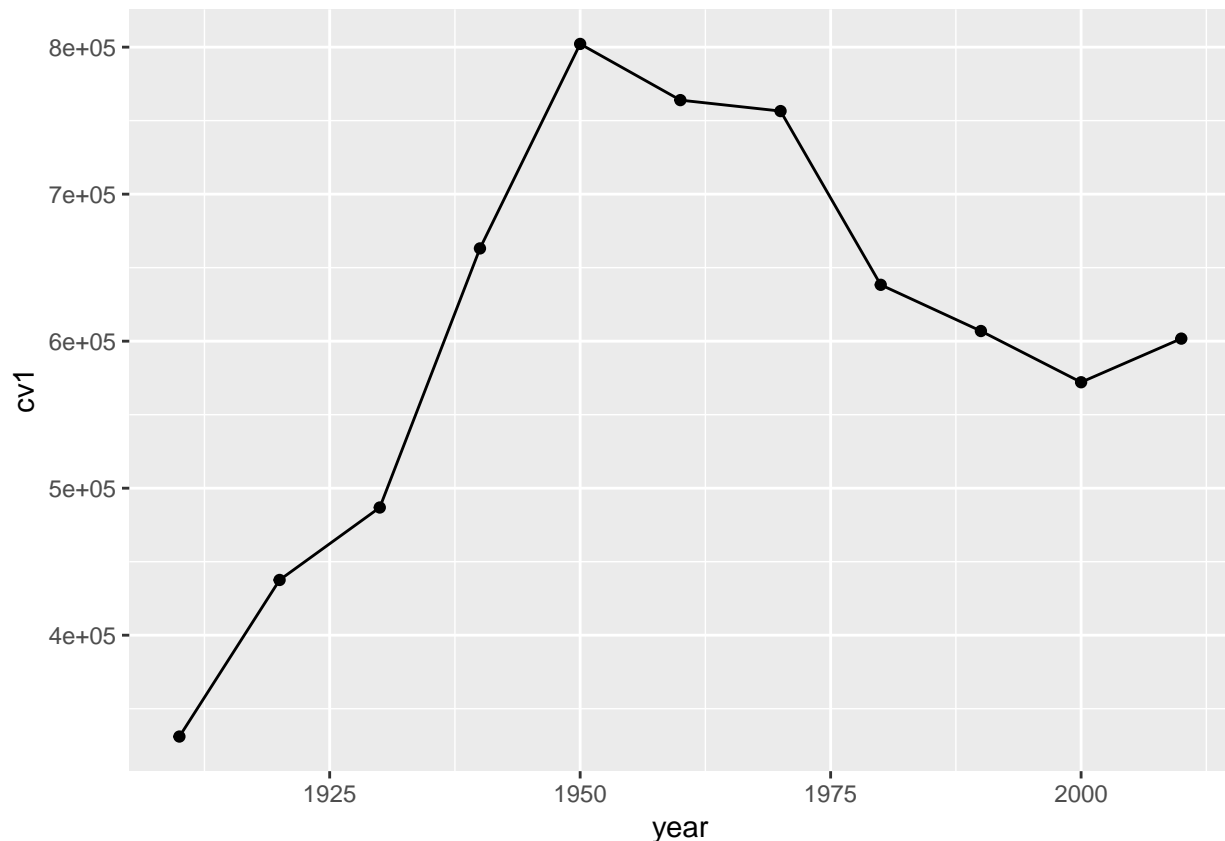
```
##   statefips year    cv1
## 1         1 1910 2138093
## 2         1 1920 2348174
## 3         1 1930 2646248
## 4         1 1940 2832961
## 5         1 1950 3061743
## 6         1 1960 3266740
## 7         1 1970 3444165
## 8         1 1980 3893888
```



```
## 9          1 1990 4040587
## 10         1 2000 4447100
```

With this state/year level dataset in hand, we'll make some graphs. Let's start by making a graph of DC's population over time. Rather than put just the name of the new dataframe (`states`), we'll tell R that we want only a subset of these data by writing `subset(states, statefips %in% c("11"))`. This means take a subset of the dataframe `states` where `statefips` is 11, which is the state code for DC.

```
# just one state
ggplot(subset(states, statefips %in% c("11")), aes(x=year, y=cv1), group = 1) +
  geom_line() +
  geom_point()
```

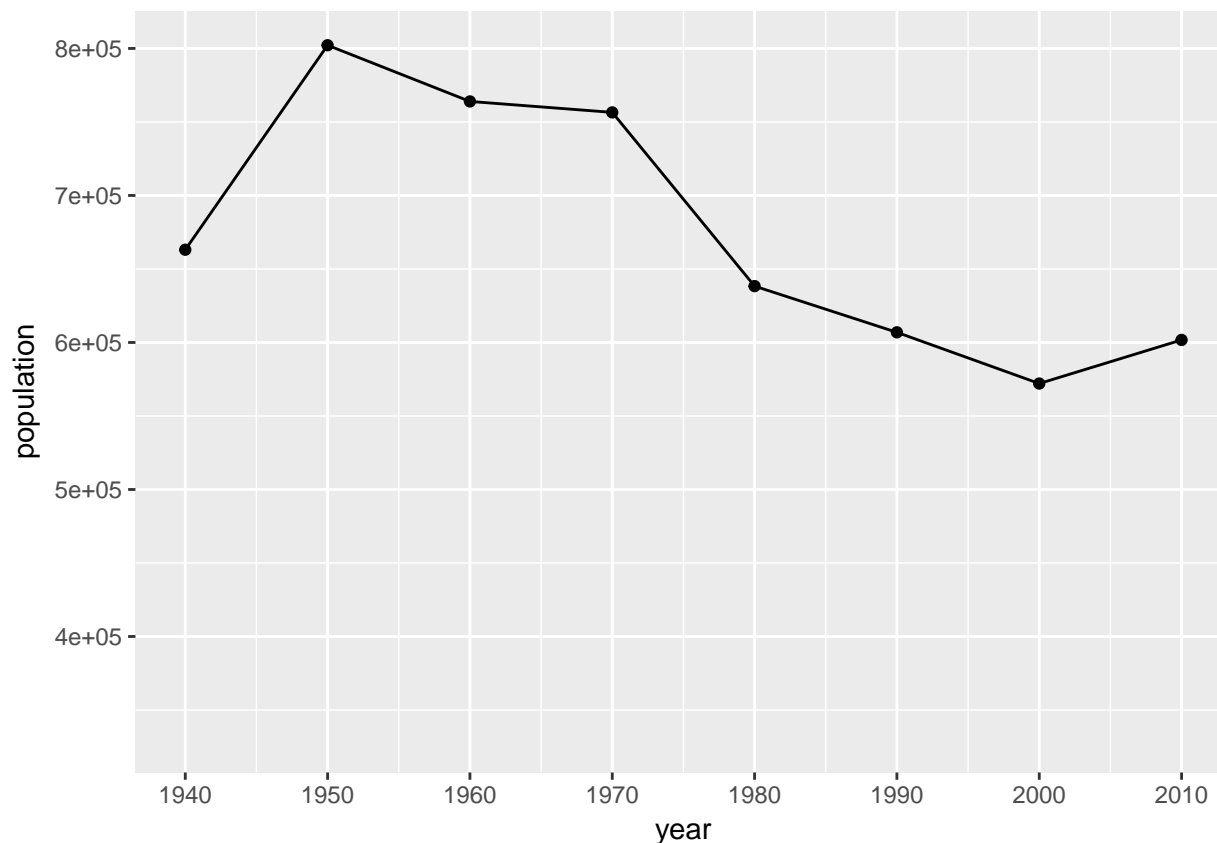


This yields a very basic looking chart. Let's change the limits of the horizontal axis and add some labels.

```
# modifying the x axis
ggplot(subset(states, statefips %in% c("11")), aes(x=year, y=cv1), group = 1) +
  geom_line() +
  geom_point() +
  scale_x_continuous(limits= c(1940, 2010), breaks = c(seq(1940,2010,10))) +
  labs(x = "year", y="population")
```

```
## Warning: Removed 3 rows containing missing values (geom_path).
```

```
## Warning: Removed 3 rows containing missing values (geom_point).
```

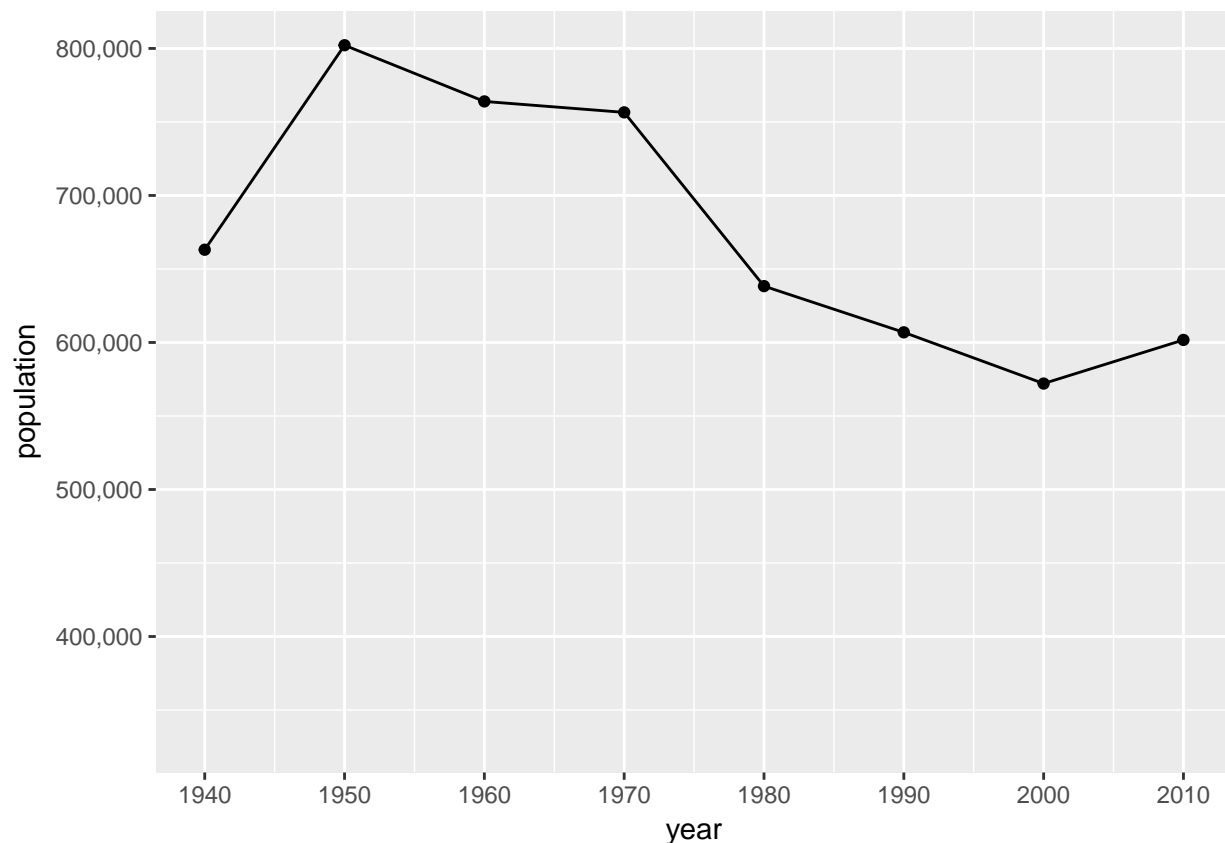


The vertical axis is hard to read with so many zeros. There are two options to fix it. You could create a new variable that is in hundreds of thousands (create a new variable that is equal to the original one divided by 1,000), or put commas in the labels. Here we do the latter. Note that you need the library `scales` to do this. (Alternatively, as Rosa points out, you could put the code `options(scipen = 999)` at the beginning of your program to avoid scientific notation entirely.)

```
# modifying the y labels
library(scales)
ggplot(subset(states, statefips %in% c("11")), aes(x=year, y=cv1), group = 1) +
  geom_line() +
  geom_point() +
  scale_y_continuous(labels = comma) +
  scale_x_continuous(limits= c(1940, 2010), breaks = c(seq(1940,2010,10))) +
  labs(x = "year", y="population")
```

```
## Warning: Removed 3 rows containing missing values (geom_path).
```

```
## Warning: Removed 3 rows containing missing values (geom_point).
```

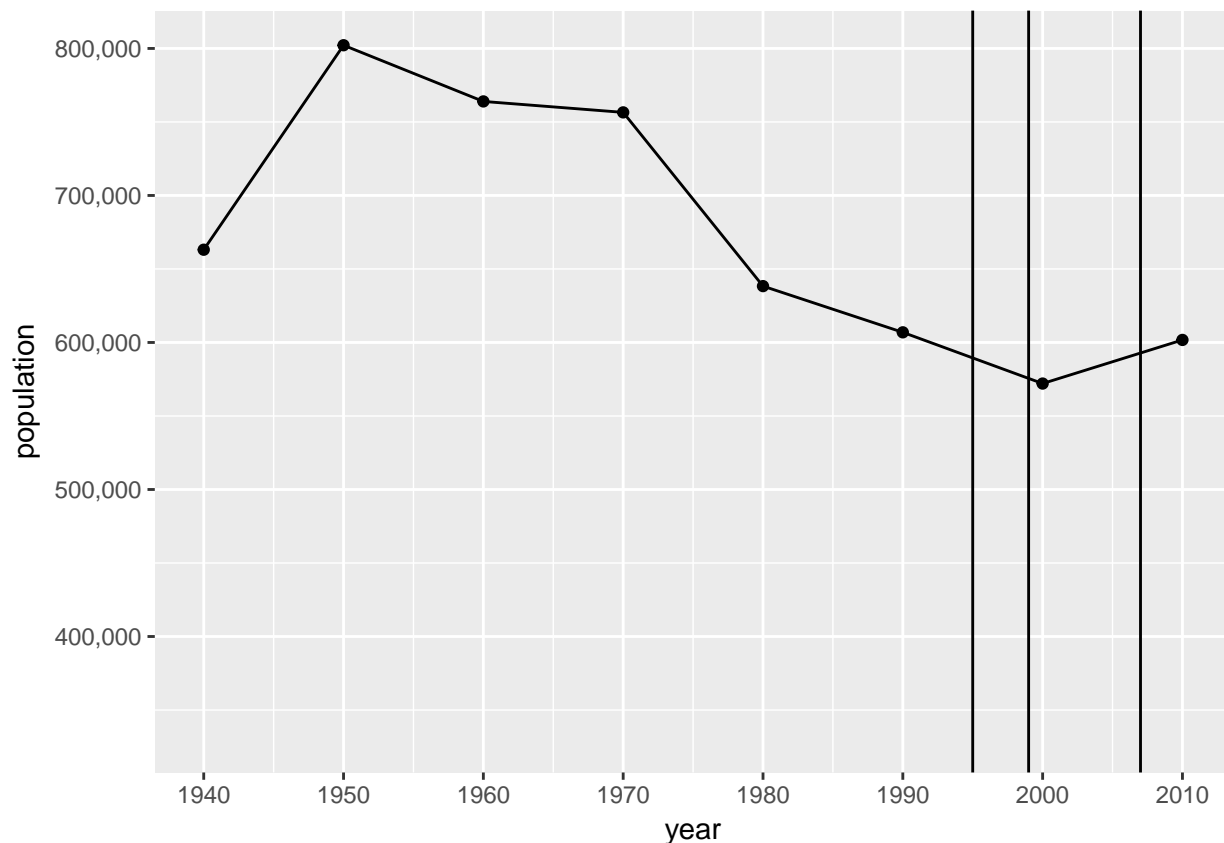


Now I'd like to use this chart at an event where I want to point out three things that happen near the end of this period. I highlight these three points with lines using `geom_vline()`, where you note the value on the x axis where the vertical line should start.

```
# adding a vertical line at key points
ggplot(subset(states, statefips %in% c("11")), aes(x=year, y=cv1), group = 1) +
  geom_line() +
  geom_point() +
  scale_y_continuous(labels = comma) +
  scale_x_continuous(limits= c(1940, 2010), breaks = c(seq(1940,2010,10))) +
  labs(x = "year", y="population") +
  geom_vline(xintercept = 1995) +
  geom_vline(xintercept = 1999) +
  geom_vline(xintercept = 2007)
```

```
## Warning: Removed 3 rows containing missing values (geom_path).
```

```
## Warning: Removed 3 rows containing missing values (geom_point).
```

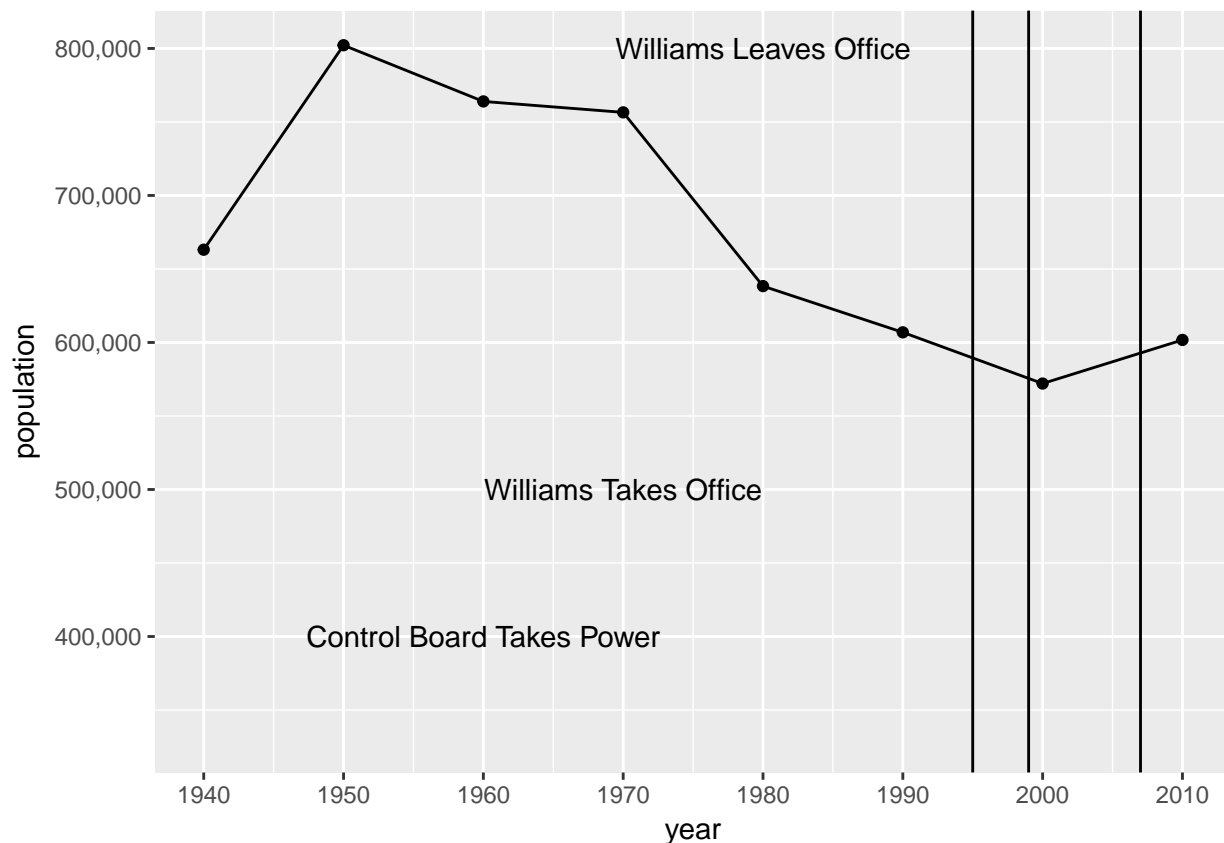


Of course, this chart doesn't clarify what's happening at these three points. Here I add text to explain what happens at each of these points, using the `annotate()` command. The `x` and `y` points in this command at the location at which the text in `label=` will be centered. Note that the text isn't at the same exact location as the line, since that would render it unreadable.

```
# adding labels onto the chart
ggplot(subset(states, statefips %in% c("11")), aes(x=year, y=cv1), group = 1) +
  geom_line() +
  geom_point() +
  scale_y_continuous(labels = comma) +
  scale_x_continuous(limits= c(1940, 2010), breaks = c(seq(1940,2010,10))) +
  labs(x = "year", y="population") +
  geom_vline(xintercept = 1995) +
  geom_vline(xintercept = 1999) +
  geom_vline(xintercept = 2007) +
  annotate("text", x=1960, y=400000, label="Control Board Takes Power") +
  annotate("text", x=1970, y=500000, label="Williams Takes Office") +
  annotate("text", x=1980, y=800000, label="Williams Leaves Office")
```

```
## Warning: Removed 3 rows containing missing values (geom_path).
```

```
## Warning: Removed 3 rows containing missing values (geom_point).
```



This plot looks ok, but I'd still like to improve the final version. I do a number of things in this final command.

- I make the plot an object by doing `done <- ggplot(...)`
- an object is something I can call later
- or export as a file, which is what I do below
- I clear most of the background with a variety of `theme()` commands
- I change the color of the lines to match the annotated text
- I drop the `year` label on the horizontal axis, since I think that's obvious

Because I am creating an object, this doesn't put anything into the plot window. I need to use the command `plot` to make it show up.

```
# making something decent looking
done <-
ggplot(subset(states, statefips %in% c("11")), aes(x=year, y=cv1), group = 1) +
  geom_line() +
  geom_point() +
  scale_y_continuous(labels = comma) +
  scale_x_continuous(limits= c(1940, 2010), breaks = c(seq(1940,2010,10))) +
  labs(x="", y="population") +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
        panel.background = element_blank(), panel.grid.major.y = element_line(color="gray"),
        axis.ticks.x = element_blank(), axis.ticks.y = element_blank()) +
  geom_vline(xintercept = 1995, color = "red4") +
  geom_vline(xintercept = 1999, color = "steelblue3") +
  geom_vline(xintercept = 2007, color = "royalblue4") +
  annotate("text", x=1960, y=575000, label="Control Board Takes Power", color = "red4") +
  annotate("text", x=1970, y=525000, label="Williams Takes Office", color = "steelblue3") +
```

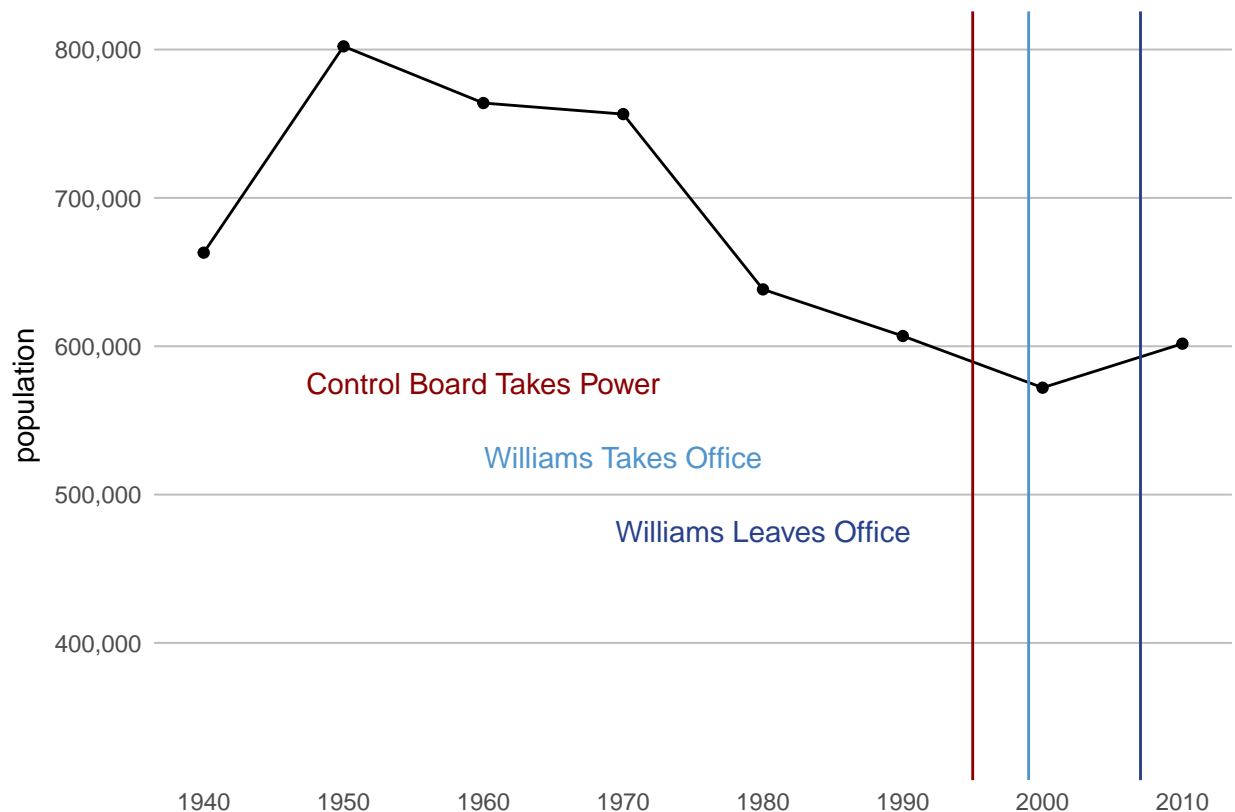
```

  annotate("text", x=1980, y=475000, label="Williams Leaves Office",color = "royalblue4" )
plot(done)

```

```
## Warning: Removed 3 rows containing missing values (geom_path).
```

```
## Warning: Removed 3 rows containing missing values (geom_point).
```



I'm still not entirely thrilled with this chart, since I think the text should be closer to the line. I am contemplating how to fix!

Finally, if you'd like to put this chart somewhere else, you'll need to actually save this file. You can do this at the plot window, but you can also do it at the command line. Here we tell R that we want to save to file 20180226_dc_pop_over_tiem.jpg the plot done, as a jpg file, in the location specified by path=. The new object should be 11 units wide, 8.5 units high, and the relevant unit is inches. Finally, we tell R that we'd like the final product to have a resolution of 300 dpi (dots per inch). People see to agree that R's default output of 72 dpi is ok for web but not good for printed products.

```

# save it as a jpg with ggsave
ggsave("20180226_dc_pop_over_time.jpg", plot=done,
  device = "jpg",
  path = "H:/center_for_washington_area_studies/events/williams_development_legacy",
  width = 11,
  height = 8.5,
  units = c("in"),
  dpi = 300)

```

```
## Warning: Removed 3 rows containing missing values (geom_path).
```

```
## Warning: Removed 3 rows containing missing values (geom_point).
```

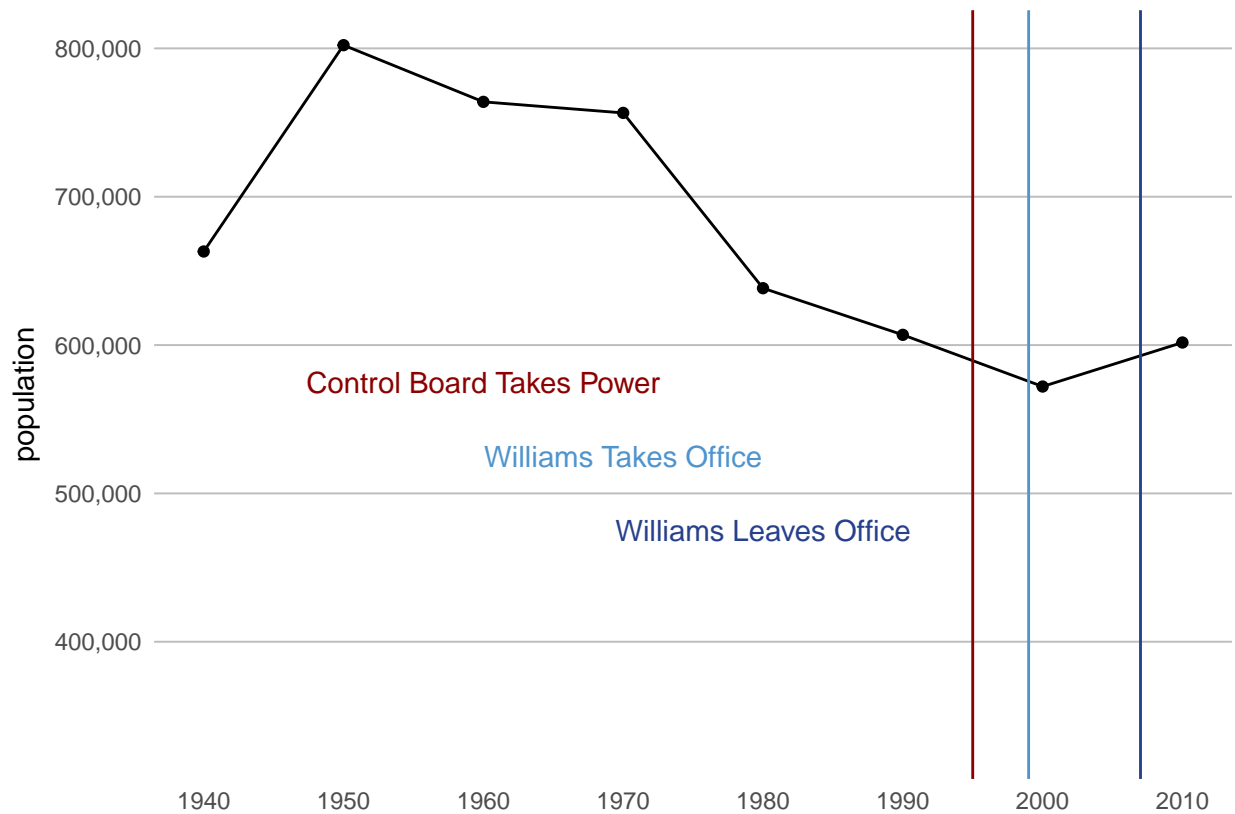
Open the jpg and make sure it seems like you think it should be.

I print my plot below so you can see it for comparison.

```
plot(done)
```

```
## Warning: Removed 3 rows containing missing values (geom_path).
```

```
## Warning: Removed 3 rows containing missing values (geom_point).
```



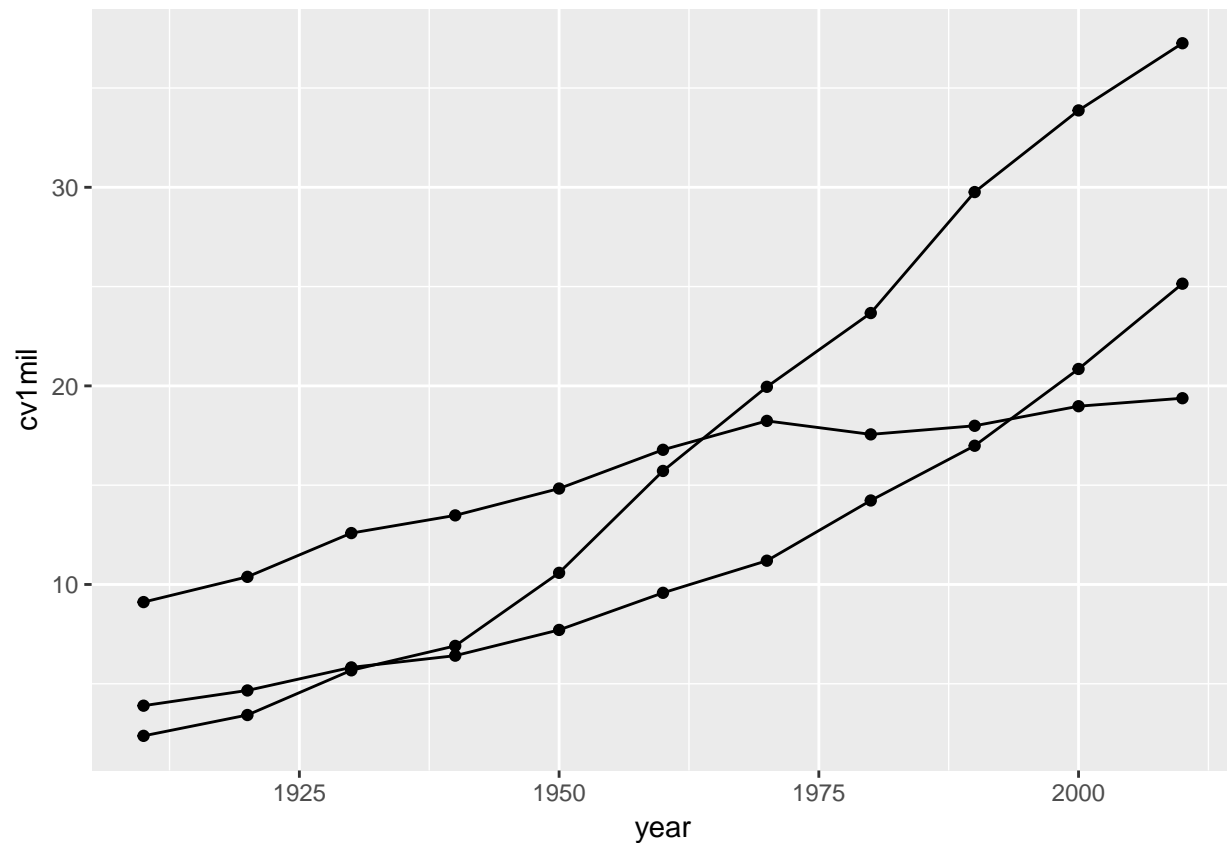
What if we want three states in one chart? Below, I make a variable that makes population in millions and then make a chart with population in California (6), New York (36) and Texas (48) over time.

```
# make a population in millions
```

```
states$cv1mil <- states$cv1 / 1000000
```

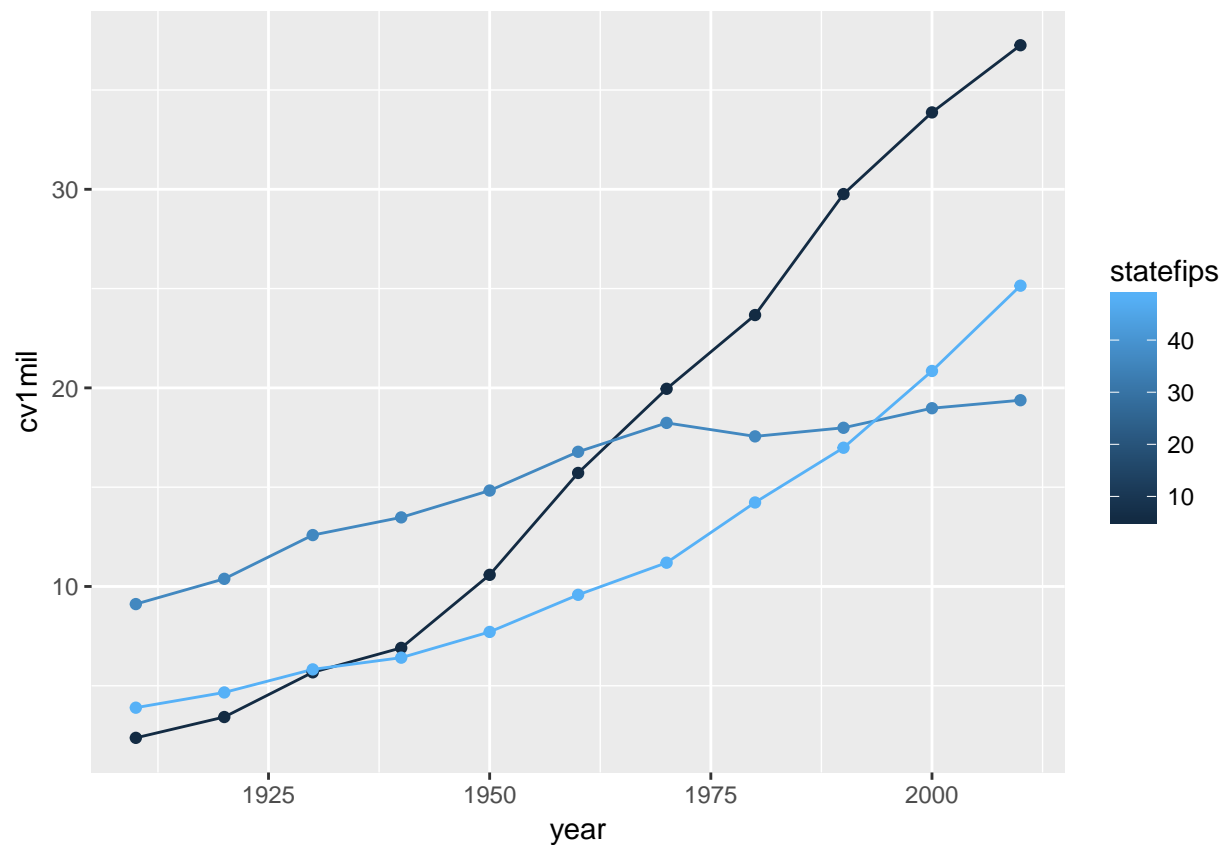
```
# three states
```

```
ggplot(subset(states, statefips %in% c("6", "36", "48")), aes(x=year, y=cv1mil, group = statefips)) +  
  geom_line() +  
  geom_point()
```



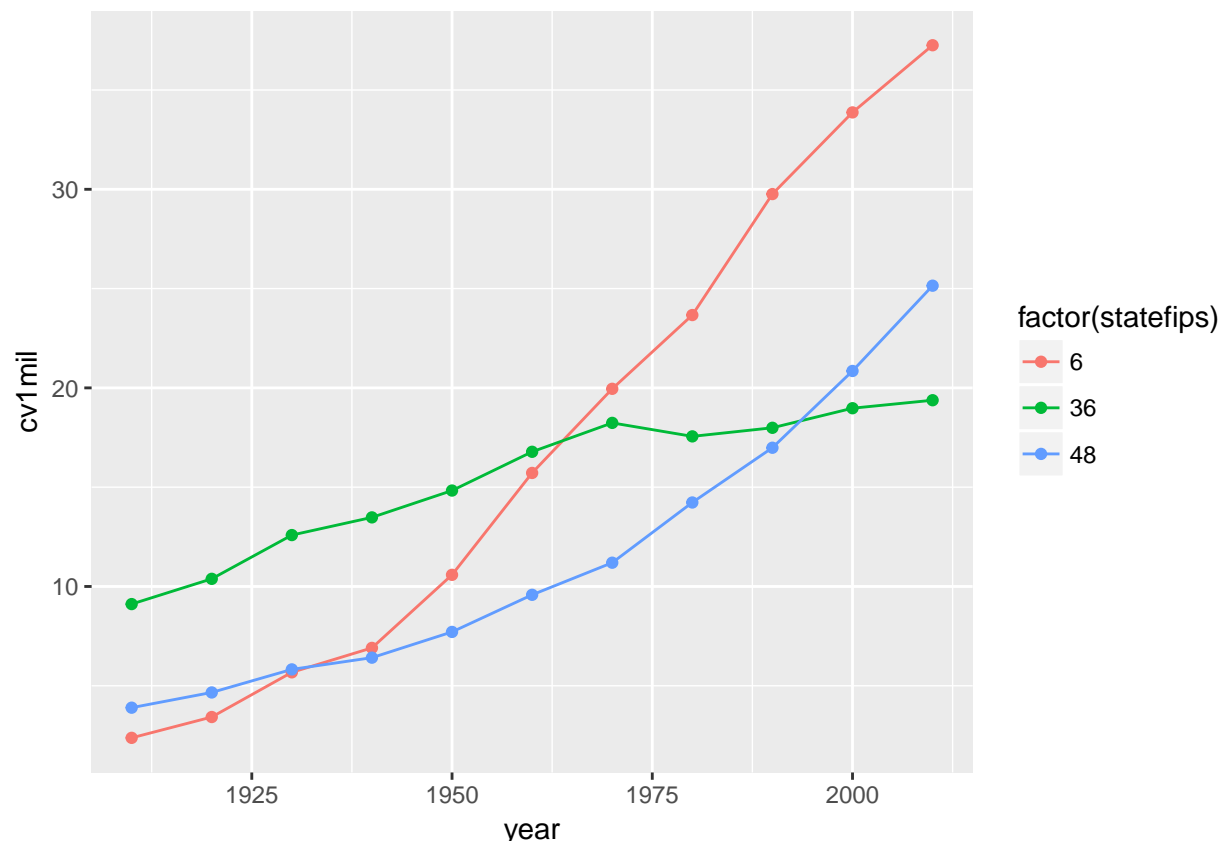
This plot gets the basics right, but we can't tell which state is which. Below, we ask R to color in each state differently with `color=statefips`.

```
# three states, make them legible
ggplot(subset(states, statefips %in% c("6", "36", "48")),
  aes(x=year, y=cv1mil, group = statefips, color = statefips)) +
  geom_line() +
  geom_point()
```

This plot is an improvement, but there is this odd color gradient for the state variable. These numbers are categorical; they have no underlying meaning. We fix this by telling R that `statefips` is a factor variable, and R adjusts the legend accordingly.

```
# three states, itemized legend
ggplot(subset(states, statefips %in% c("6", "36", "48")),
  aes(x=year, y=cv1mil, group = factor(statefips), color = factor(statefips))) +
  geom_line() +
  geom_point()
```



There are other things we could fix, but this gets the basic outlines correct.

C. Try stacked lines

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.

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"
```

Alternatively, Rosa called the `readxl` library for a command called `read_excel` which will read in an Excel file, and even lets you choose a particular sheet (here, Rosa's sheet is 2) and note any special characters that are NAs.

```
library(readxl)
hist02z3.rosa <- read_excel("hist02z3.xlsx", sheet = 2, na = "*")
```

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
## 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
## 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
## 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
## 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
## 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()` 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 <- subset(hist02z3, year != "")
hist02z3 <- subset(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.

To make a stacked line (or really, any multiple line), the data needs to be long, not wide. What does this mean?

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"))
```

```
##   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.

To make a long dataset from a wide dataset in R, you need to have all the variables you’d like to make long have a consistent name, such as `var1`, `var2`, `var3`. Anything that ends with consecutive numbers is ok. In the steps below, I rename variables so that they follow this requirement.

In addition, I make non-numeric variables numeric. The social insurance variable is a factor and I convert it into a numeric variable using the code below.

```
## 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)
```

Now we use R’s reshape command to go from wide to long (you can also go from long to wide by setting `direction = "wide"`). The variables in the command below are

- the dataframe, `hist02z3`
- the variables that are wide that we would like to make long: `c("r1","r2","r3","r4","r5","r6")`
- the new variable we create that will have the type number from the wide name: `rtype`
- the variable that identifies unique observations in the wide dataset: `nyear`
- the type of dataset we’d like to create – wide or long
- and whether there is a separator between the variable name and the type number. For example, if the variables we want to make long are `t_1`, `t_2`, etc., then the separator is `_`

```
## make this wide dataset long
r.long <- reshape(hist02z3, varying = c("r1","r2","r3","r4","r5","r6"),
                  timevar="rtype",
                  idvar = "nyear", direction = "long",
                  sep="")

r.long[1:15,]
```

```
##      year income.taxes corp.taxes social.ins excise other total nyear
```

```
## 1934.1 1934      0.7      0.6      .      2.2      1.3      4.8 1934
## 1935.1 1935      0.7      0.8      .      2.0      1.5      5.1 1935
## 1936.1 1936      0.8      0.9      0.1      2.0      1.1      4.9 1936
## 1937.1 1937      1.2      1.2      0.7      2.1      0.9      6.1 1937
## 1938.1 1938      1.4      1.4      1.7      2.1      0.9      7.5 1938
## 1939.1 1939      1.1      1.2      1.8      2.1      0.7      7.0 1939
## 1940.1 1940      0.9      1.2      1.8      2.0      0.7      6.7 1940
## 1941.1 1941      1.1      1.8      1.7      2.2      0.7      7.5 1941
## 1942.1 1942      2.2      3.2      1.7      2.3      0.5      9.9 1942
## 1943.1 1943      3.5      5.2      1.6      2.2      0.4     13.0 1943
## 1944.1 1944      9.2      6.9      1.6      2.2      0.5     20.5 1944
## 1945.1 1945      8.1      7.1      1.5      2.8      0.5     19.9 1945
## 1946.1 1946      7.1      5.2      1.4      3.1      0.5     17.2 1946
## 1947.1 1947      7.5      3.6      1.4      3.0      0.6     16.1 1947
## 1948.1 1948      7.4      3.7      1.4      2.8      0.6     15.8 1948
##      rtype      r
## 1934.1      1 0.7
## 1935.1      1 0.7
## 1936.1      1 0.8
## 1937.1      1 1.2
## 1938.1      1 1.4
## 1939.1      1 1.1
## 1940.1      1 0.9
## 1941.1      1 1.1
## 1942.1      1 2.2
## 1943.1      1 3.5
## 1944.1      1 9.2
## 1945.1      1 8.1
## 1946.1      1 7.1
## 1947.1      1 7.5
## 1948.1      1 7.4
```

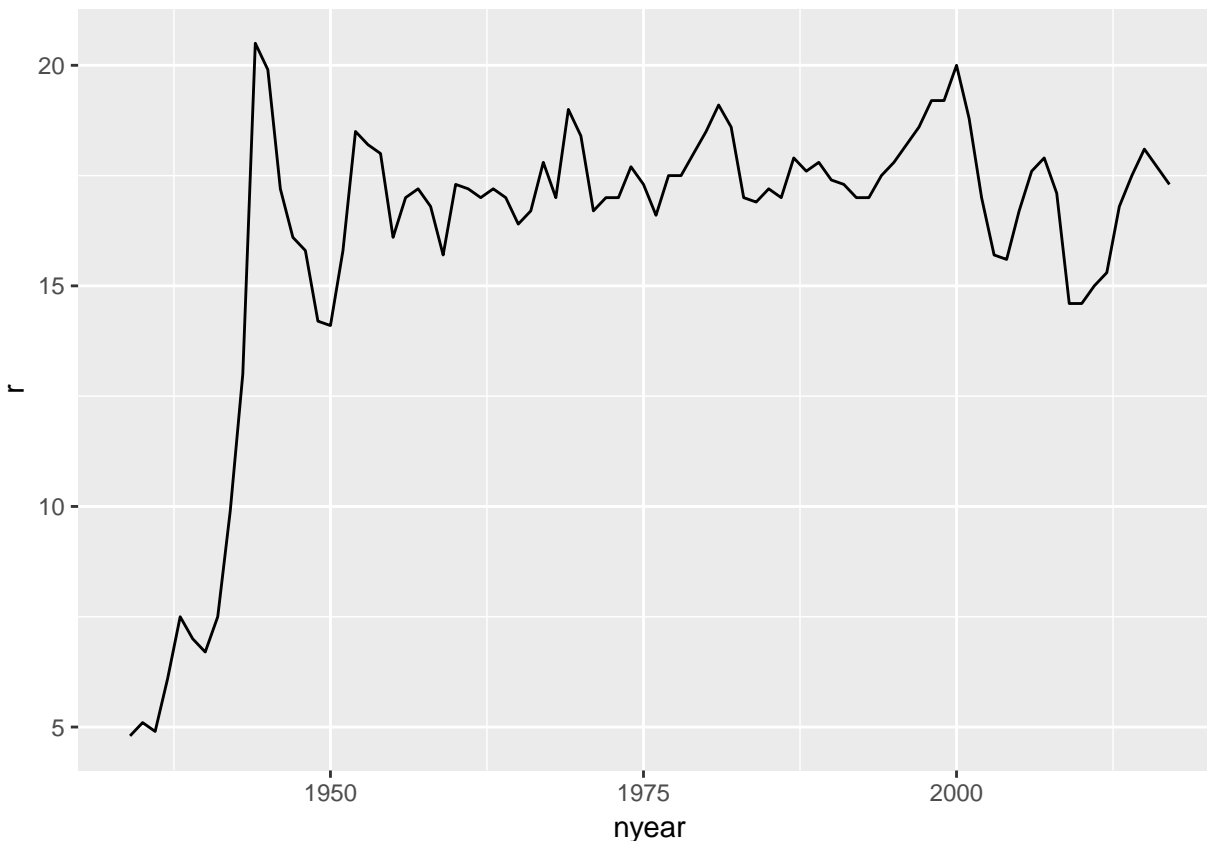
This new dataset keeps all the old variables, but the key new variables are `r` and `rtype`. The variable `r` takes on the values of each revenue category, and the variable `rtype` tells you which revenue category it is.

Now I create a new variable that has the name of the tax that defined by the number. This not required, but I do it so I can follow what's going on and use the names of the tax later on. Here we use the `ifelse()` command that I have described in a previous tutorial.

```
# make a type of receipts variable
r.long$rname <- ifelse(r.long$rtype == "1", "income taxes",
                      ifelse(r.long$rtype == "2", "corp. taxes",
                              ifelse(r.long$rtype == "3", "social ins.",
                                      ifelse(r.long$rtype == "4", "excise",
                                              ifelse(r.long$rtype == "5", "other",
                                                      ifelse(r.long$rtype == "6", "total", "."))))))))
```

Before heading into stacked lines, 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`.

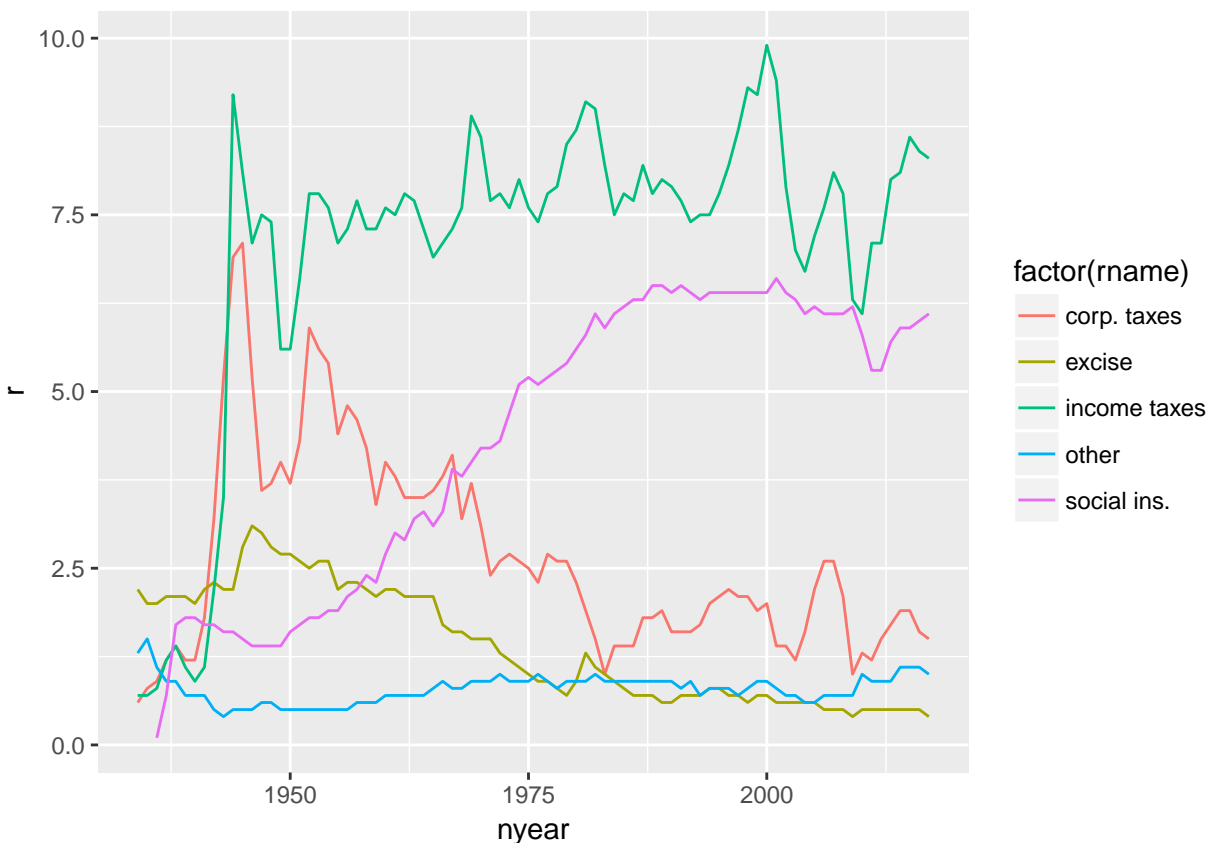
```
#### line chart of total receipts
ggplot(subset(r.long, rtype==6), aes(x=nyear, y=r, group=1)) + geom_line()
```



Now we'll modify the chart to have all the categories but the total. I do this by subsetting `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 ###
ggplot(subset(r.long, rtype != 6), aes(x=nyear, y=r, group=factor(rname), color=factor(rname))) +
  geom_line()
```

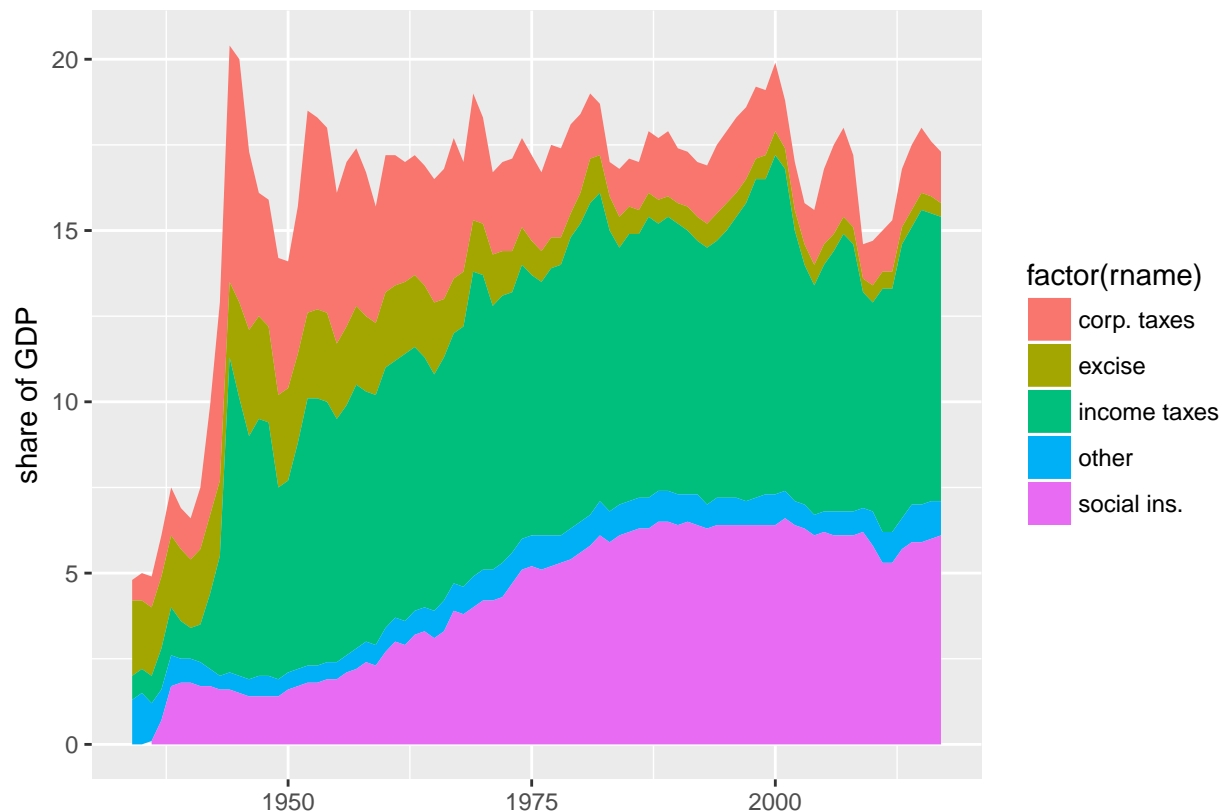
```
## Warning: Removed 2 rows containing missing values (geom_path).
```



But these types of receipts all add up to a total, and we care about how that total evolves over time, which is a case for a stacked line chart. We do this by changing `geom_line()` to `geom_area()`, and by telling R to fill by `rname`, which it should treat as a factor (`fill=factor(rname)`).

```
#### stacked chart of total receipts by type ###
## without factor() this doesnt work
ggplot(subset(r.long, rtype != 6), aes(x=nyear, y=r, fill=factor(rname))) +
  geom_area(position="stack") +
  labs(x="", y="share of GDP")
```

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



There are more things we could do to make this chart more beautiful, and that is your homework!

D. Homework

1. Improve the final chart from section C.

I am open to any types of improvement – and different improvements will tell different stories. Make one superior chart and write a sentence or two explaining what it does and why it's better.

2. 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))
```