STAT 324: Lecture 1 (lecture notes)

Ralph Trane

1/16/2020

This document is intended for lecturers **only**. Most of it is included in the "intro_to_R.pdf", which will be distributed to students for reference **after** class.

Overview of lecture

- Introduce yourself
- Briefly walk through most important parts of the syllabus
- Guided tour of RStudio
  - Top left: your document. This is where you will be working most of the time.
  - Bottom left: the console. This is where R code is actually run. If you need to install packages, or quickly try something simple, this is where you'd do that.
  - Top right: any data sets, variables, etc. that we define in our working session will pop up here. Very neat to keep track of what's going on.
  - Bottom right: any plots or help pages will show up here.
- Intro to R:
  - Install packages in the console
  - Silly code in console (2+2, sqrt(9), 3^2, etc)
- R Markdown essentials:
  - how to start a new document
  - header vs. text vs. R chunks

BEFORE MOVING ON: STRESS THAT THIS IS A LOT, AND THEY ARE NOT EXPECTED TO RETAIN EVERYTHING. THIS IS A DEMONSTRATION.

The idea is to expose them to this as early as possible, as they will catch on sooner.

- "Homework 0"
  - exact replicate of discussion 1 and homework 1
  - download data and questions from Canvas. Talk folders and folder structure!!!
  - Solve homework 0 live (solutions below – this is approximately what I intend to recreate live in class, while explaining the steps. Obviously with much less text.)
- Other cool stuff you can do in R, and interesting questions with data
  - see `cool_stuff.Rmd`

Homework 0 (Discussion 1 + Homework 1)

Before we get started, we load the package `tidyverse`.

```r
library(tidyverse)
```

Import Data

We will use a data set containing information about measles in the US with data from 1928 to 2003. Before we can use the data, we need to read it into R. Since this data is provided as a .csv file, we do so using the `read_csv` function:

```r
measles <- read_csv("measles.csv")
```

```
## Parsed with column specification:
## cols(
##   disease = col_character(),
##   state = col_character(),
##   year = col_double(),
##   weeks_reporting = col_double(),
##   count = col_double(),
##   population = col_double()
## )
```

This creates a new object called `measles`. The arrow `<-` is used to assign things to objects. It takes what's on the right and saves it to what you put on the left, so we can reuse it later.

First look at data

When using the `read_csv` function to read the data, it will only print the first 10 rows by default:

```r
measles
```

```
## # A tibble: 3,876 x 6
##    disease state    year weeks_reporting count population
##    <chr>   <chr>   <dbl>           <dbl> <dbl>      <dbl>
##  1 Measles Alabama  1928              52  8843    2589923
##  2 Measles Alabama  1929              49  2959    2619131
##  3 Measles Alabama  1930              52  4156    2646248
##  4 Measles Alabama  1931              49  8934    2670818
##  5 Measles Alabama  1932              41   270    2693027
##  6 Measles Alabama  1933              51  1735    2713243
##  7 Measles Alabama  1934              52 15849    2731850
##  8 Measles Alabama  1935              49  7214    2749249
##  9 Measles Alabama  1936              40   570    2765853
## 10 Measles Alabama  1937              49   620    2782085
## # ... with 3,866 more rows
```

This is very useful, since there's a total of 3876 rows – way too many to actually show them all!

Click the data on the right to see it all interactively. Talk about the data. What would be an interesting question?

Some summaries

Mean number of cases across time and states:

```
mean(measles$count)
```

```
## [1] 4817.078
```

We use `$` to extract one column of the data.

Not very informative. What about mean count per year? We create a new object (`measles_grouped`) where the data is grouped by year using the function `group_by`, then use summarize to get the mean count within each group.

```
measles_grouped <- group_by(measles, year)
summarize(measles_grouped, mean_count = mean(count))
```

```
## # A tibble: 76 x 2
##     year mean_count
##    <dbl>      <dbl>
##  1  1928      9477.
##  2  1929      6648.
##  3  1930      7541.
##  4  1931      8597.
##  5  1932      7649.
##  6  1933      7459.
##  7  1934     14257.
##  8  1935     14137.
##  9  1936      5509.
## 10  1937      5762.
## # ... with 66 more rows
```

Gets really messy really quickly. One way to clean up code is using the pipe (`%>%`). This sends whatever comes before it into whatever functino comes after it, and places it at the very front.

Examples:

```
sum(2,1,7,-6,2)
```

```
## [1] 6
```

```
c(2,1,7,-6,2) %>% sum
```

```
## [1] 6
```

```
c(2,1) %>%
  sum(7,-6, 2)
```

```
## [1] 6
```

```r
sqrt(7)
```

```
## [1] 2.645751
```

We can combine multiple functions at once:

```r
7 %>% sqrt
```

```
## [1] 2.645751
```

```r
3 %>%
  sum(4) %>%
  sqrt
```

```
## [1] 2.645751
```

Now use this to clean up the code:

```r
measles %>%
  group_by(year) %>%
  summarize(mean_count = mean(count))
```

```
## # A tibble: 76 x 2
##     year mean_count
##    <dbl>      <dbl>
##  1  1928      9477.
##  2  1929      6648.
##  3  1930      7541.
##  4  1931      8597.
##  5  1932      7649.
##  6  1933      7459.
##  7  1934     14257.
##  8  1935     14137.
##  9  1936      5509.
## 10  1937      5762.
## # ... with 66 more rows
```

Easily do the same thing by state instead of by year:

```r
measles %>%
  group_by(state) %>%
  summarize(mean_count = mean(count))
```

```
## # A tibble: 51 x 2
##    state                mean_count
##    <chr>                     <dbl>
##  1 Alabama                   2758.
##  2 Alaska                     222.
##  3 Arizona                   2116.
##  4 Arkansas                  1766.
##  5 California               18116.
##  6 Colorado                  3232.
##  7 Connecticut               4901.
##  8 Delaware                   450.
##  9 District Of Columbia       830.
## 10 Florida                   2171.
## # ... with 41 more rows
```

Comparing these doesn't make much sense – obviously states with larger populations will have a higher count. So, instead of using the counts directly, we turn them into rates.

To create new variables (i.e. new columns), use `mutate`:

```
measles %>%
  mutate(rate = count / population) %>%
  group_by(state) %>%
  summarize(mean_rate = mean(rate))
```

```
## # A tibble: 51 x 2
##    state                mean_rate
##    <chr>                    <dbl>
##  1 Alabama               0.000927
##  2 Alaska               NA
##  3 Arizona               0.00242
##  4 Arkansas              0.000937
##  5 California            0.00192
##  6 Colorado              0.00244
##  7 Connecticut           0.00246
##  8 Delaware              0.00147
##  9 District Of Columbia  0.00126
## 10 Florida               0.000766
## # ... with 41 more rows
```

Alaska is `NA`. Why?

```
measles %>%
  filter(state == "Alaska")
```

```
## # A tibble: 76 x 6
##    disease state   year weeks_reporting count population
##    <chr>   <chr>  <dbl>           <dbl> <dbl>      <dbl>
##  1 Measles Alaska  1928               0     0         NA
##  2 Measles Alaska  1929               0     0         NA
##  3 Measles Alaska  1930               0     0         NA
##  4 Measles Alaska  1931               0     0         NA
##  5 Measles Alaska  1932               0     0         NA
##  6 Measles Alaska  1933               0     0         NA
##  7 Measles Alaska  1934               0     0         NA
##  8 Measles Alaska  1935               0     0         NA
##  9 Measles Alaska  1936               0     0         NA
## 10 Measles Alaska  1937               0     0         NA
## # ... with 66 more rows
```

We see that there are years where we simply do not know the population of Alaska! The same is actually the case for Hawaii. To not have to deal with this, we exclude these two states, and save the results in a new object called `new_measles`. We also calculate and include the rate in this object:

```
new_measles <- measles %>%
  filter(state != "Alaska", state != "Hawaii") %>%
  mutate(rate = count / population)

new_measles
```

```
## # A tibble: 3,724 x 7
##    disease state    year weeks_reporting count population     rate
##    <chr>   <chr>   <dbl>           <dbl> <dbl>      <dbl>    <dbl>
##  1 Measles Alabama  1928              52  8843    2589923 0.00341
##  2 Measles Alabama  1929              49  2959    2619131 0.00113
##  3 Measles Alabama  1930              52  4156    2646248 0.00157
##  4 Measles Alabama  1931              49  8934    2670818 0.00335
##  5 Measles Alabama  1932              41   270    2693027 0.000100
##  6 Measles Alabama  1933              51  1735    2713243 0.000639
##  7 Measles Alabama  1934              52 15849    2731850 0.00580
##  8 Measles Alabama  1935              49  7214    2749249 0.00262
##  9 Measles Alabama  1936              40   570    2765853 0.000206
## 10 Measles Alabama  1937              49   620    2782085 0.000223
## # ... with 3,714 more rows
```

What the difference between `measles` and `new_measles`?

```
new_measles %>%
  filter(state == "Wisconsin", year == 1993)
```

```
## # A tibble: 1 x 7
##   disease state      year weeks_reporting count population  rate
##   <chr>   <chr>     <dbl>           <dbl> <dbl>      <dbl> <dbl>
## 1 Measles Wisconsin  1993               9     0    5013015     0
```

Now we have the rate for each state for each year. Which state has had the highest average rate of measles over the years? We can use `arrange` to basically sort the data by `mean_rate`:

```
new_measles %>%
  group_by(state) %>%
  summarize(mean_rate = mean(rate)) %>%
  arrange(mean_rate)
```

```
## # A tibble: 49 x 2
##    state       mean_rate
##    <chr>           <dbl>
##  1 Mississippi  0.000249
##  2 Louisiana    0.000341
##  3 Georgia      0.000621
##  4 Oklahoma     0.000651
##  5 Missouri     0.000738
##  6 Nevada       0.000750
##  7 Florida      0.000766
##  8 Nebraska     0.000869
##  9 Alabama      0.000927
## 10 Arkansas     0.000937
## # ... with 39 more rows
```

By default, it is sorted in ascending order. Luckily it is easy to change it to descending:

```
new_measles %>%
  group_by(state) %>%
  summarize(mean_rate = mean(rate)) %>%
  arrange(desc(mean_rate))
```

```
## # A tibble: 49 x 2
##    state         mean_rate
##    <chr>             <dbl>
##  1 Wisconsin       0.00444
##  2 Vermont         0.00379
##  3 Utah            0.00343
##  4 Montana         0.00310
##  5 New Jersey      0.00261
##  6 Connecticut     0.00246
##  7 Colorado        0.00244
##  8 Arizona         0.00242
##  9 Michigan        0.00236
## 10 Massachusetts   0.00234
## # ... with 39 more rows
```

One thing that could be interesting to look at is how the rate of measles has changed over the years. To do so, we calculate the mean rate per year:

```
new_measles %>%
  group_by(year) %>%
  summarize(mean_rate = mean(rate))
```

```
## # A tibble: 76 x 2
##     year mean_rate
##    <dbl>     <dbl>
##  1  1928   0.00348
##  2  1929   0.00247
##  3  1930   0.00298
##  4  1931   0.00315
##  5  1932   0.00294
##  6  1933   0.00239
##  7  1934   0.00622
##  8  1935   0.00511
##  9  1936   0.00255
## 10  1937   0.00228
## # ... with 66 more rows
```

This long table of numbers is kind of boring… Let's create a plot! First, we will need to save the mean rates in a new object:

```r
mean_rates_per_year <- new_measles %>%
  group_by(year) %>%
  summarize(mean_rate = mean(rate))
```

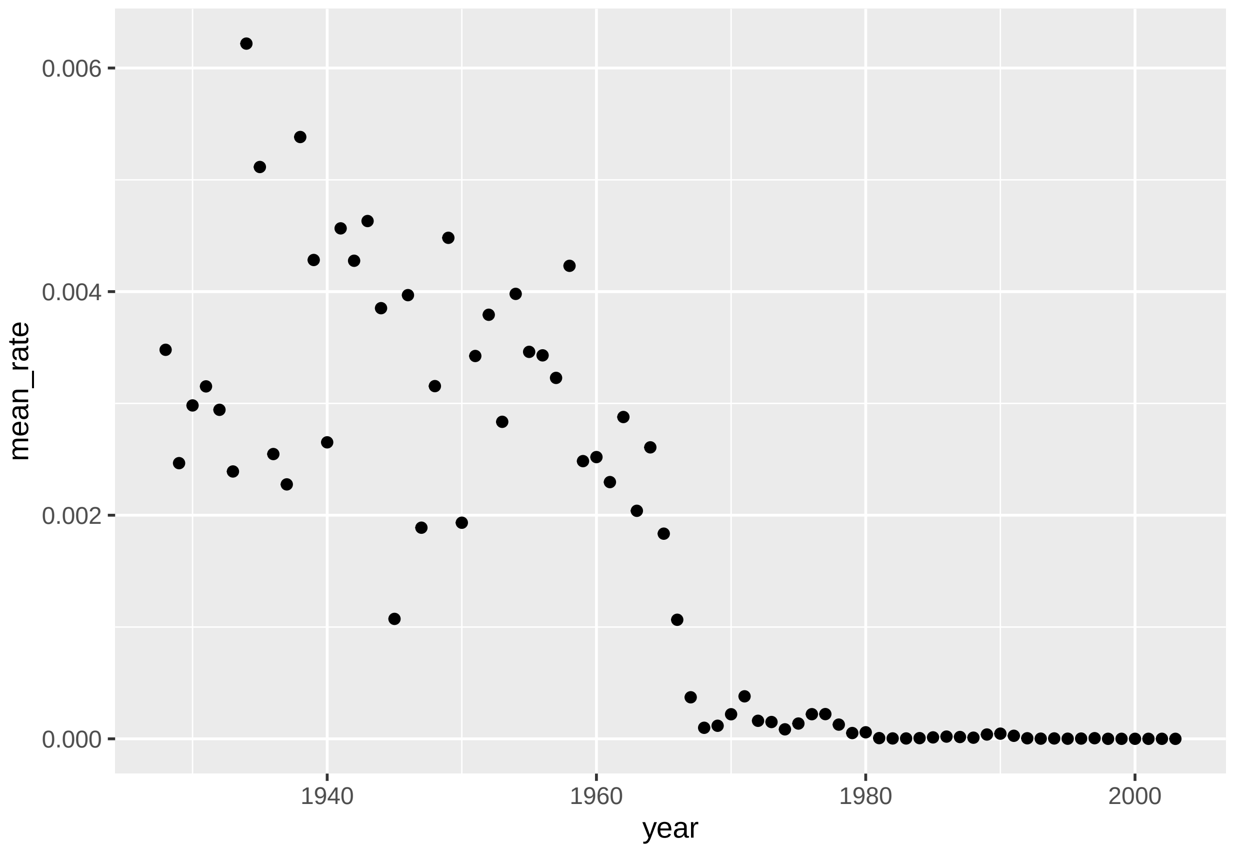
Some Plots

We then use the function `ggplot` to create a plot. We need to tell this function a few things:

1. What data set to use
2. What aesthetics to use (`aes`)
3. What kind of plot we want

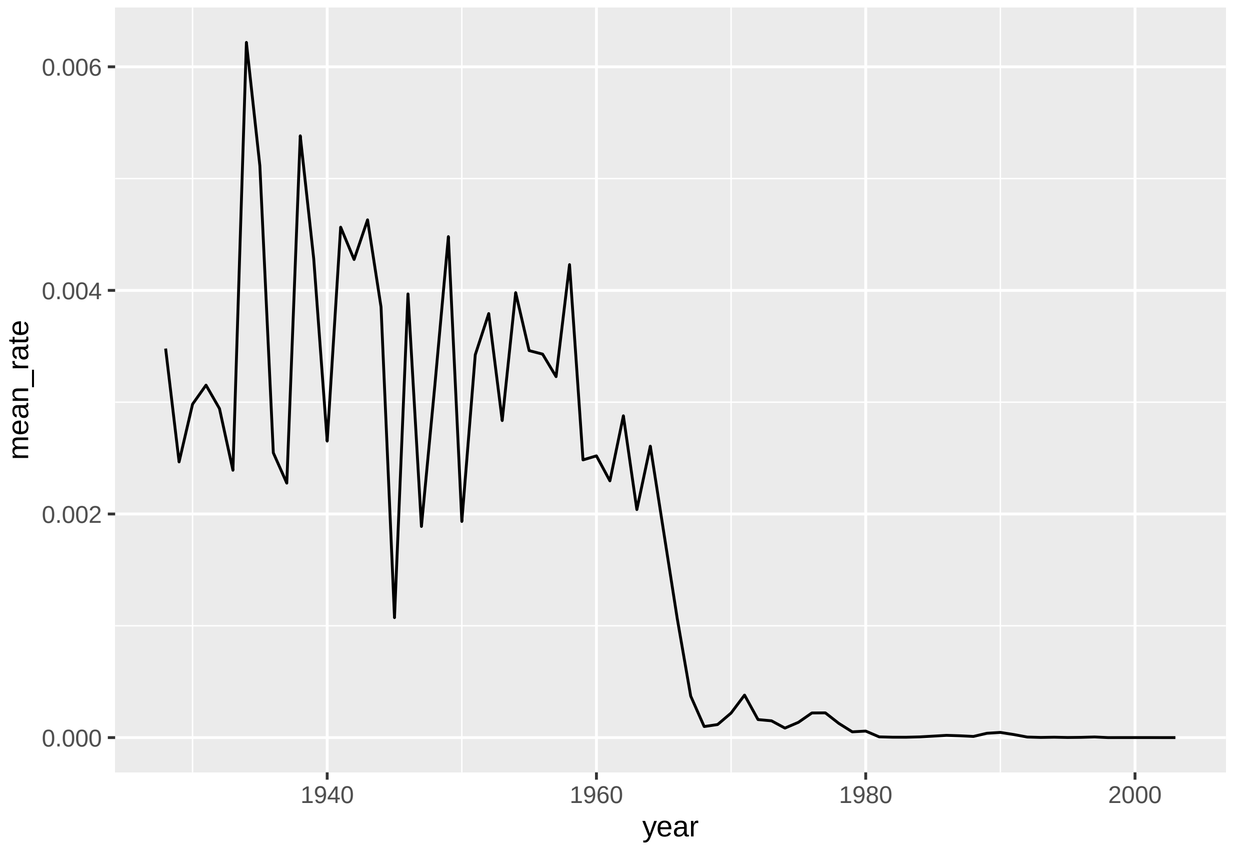
This is done in the following way:

```r
ggplot(data = mean_rates_per_year,
       aes(x = year, y = mean_rate)) +
  geom_point()
```



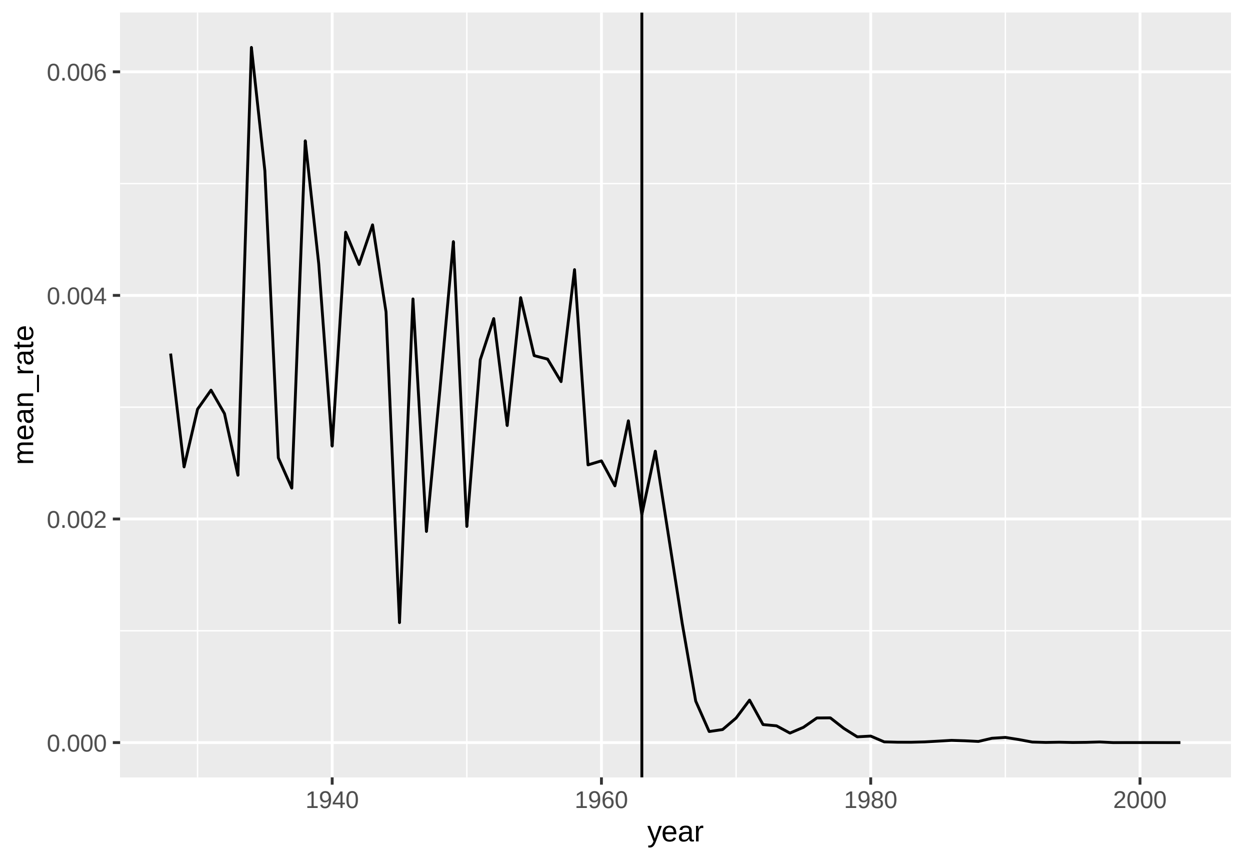
We could also have done this using a line instead of points:

```r
ggplot(mean_rates_per_year,
       aes(x = year, y = mean_rate)) +
  geom_line()
```



Something definitely happens around 1955. Turns out the polio vaccine became widely used in 1954:

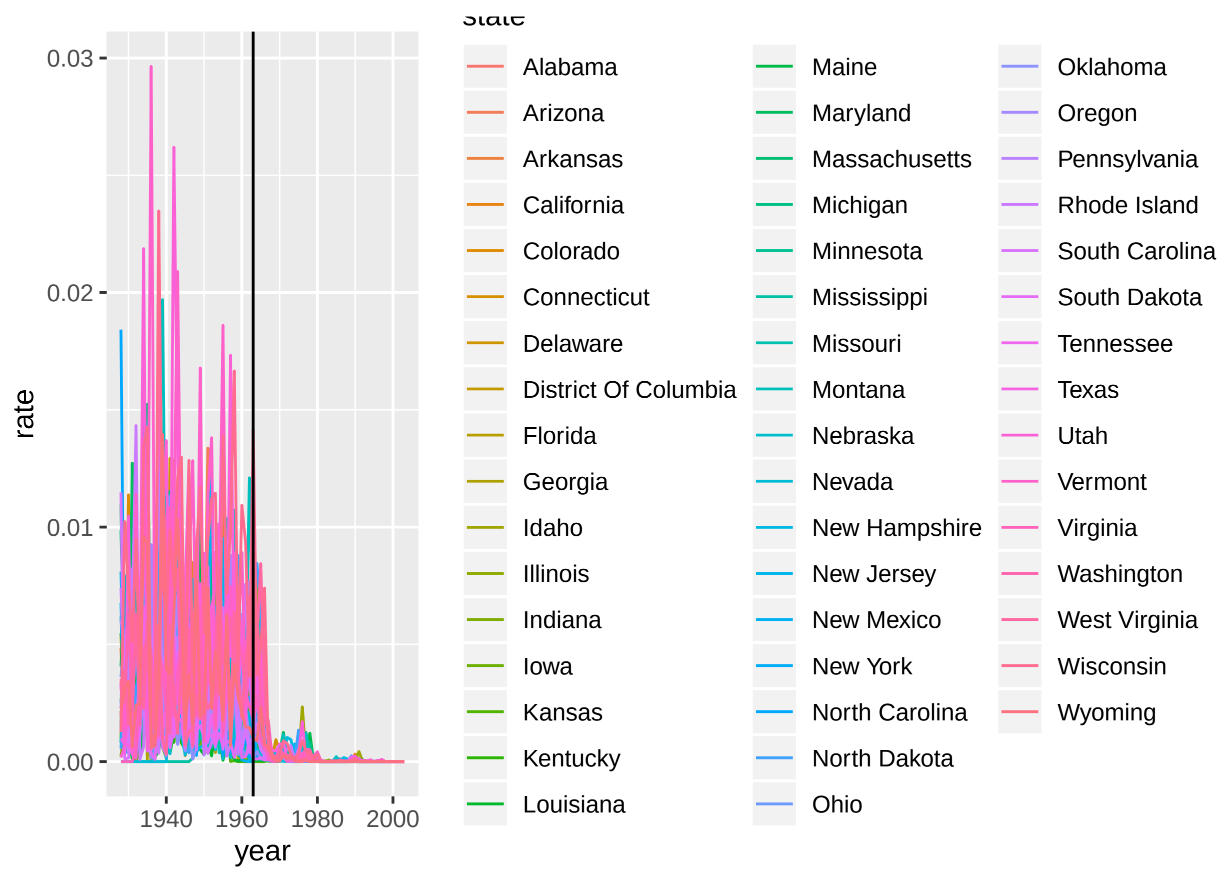
```r
ggplot(data = mean_rates_per_year,
       aes(x = year, y = mean_rate)) +
  geom_line() +
  geom_vline(xintercept = 1963)
```



THIS IS THE END OF WHAT WILL BE IN DISCUSSION 1 AND ON HOMEWORK 1

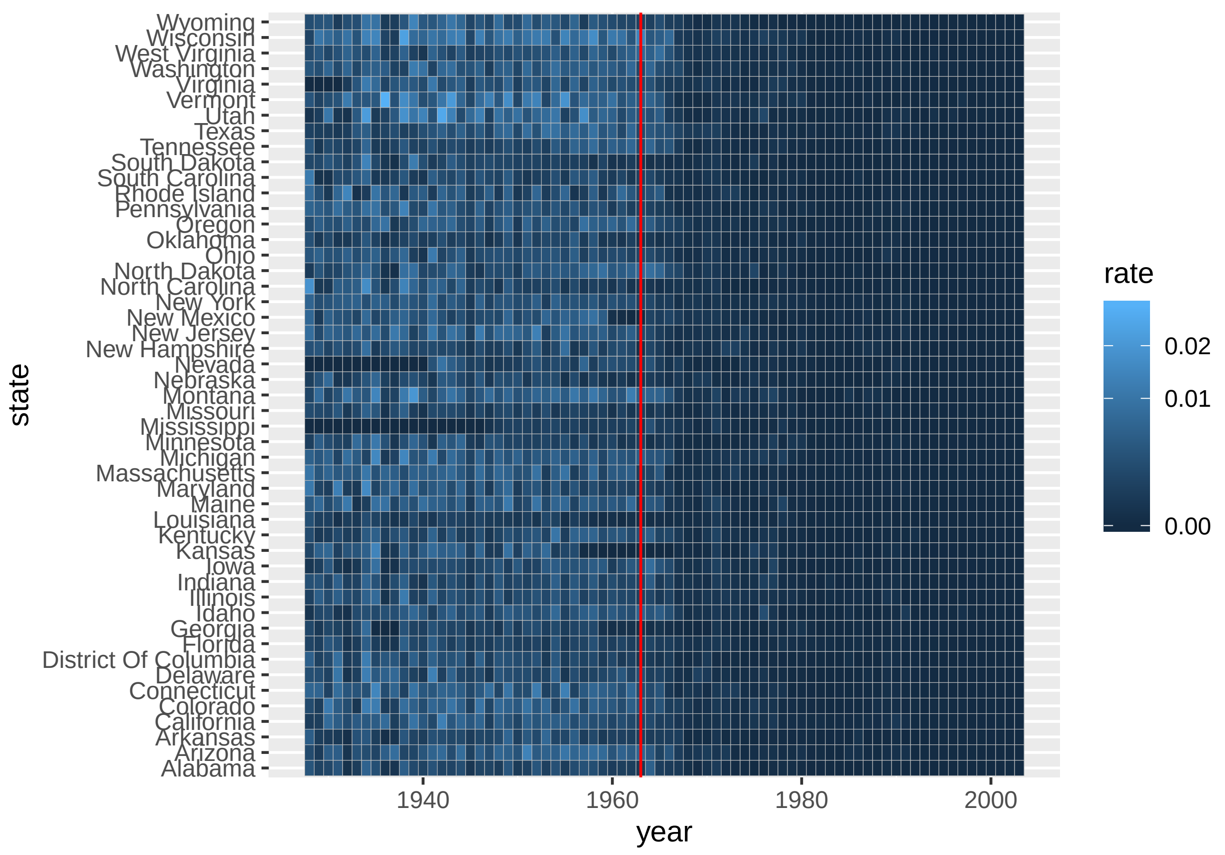
If we use the original data set, we can create the same plot, but with one line for each state:

```
ggplot(data = new_measles,
       aes(x = year, y = rate, color = state)) +
  geom_line() +
  geom_vline(xintercept = 1963)
```



Not very useful. A much better version:

```
ggplot(new_measles,
       aes(x = year, y = state, fill = rate)) +
  geom_tile(color = "grey") +
  scale_fill_continuous(trans = "sqrt") +
  geom_vline(xintercept = 1963, color = "red")
```



Other Examples

This section is simply to peak interest in using `R` – hence the showcase of some cool things we can do. See `cool_stuff.Rmd`.

Take-aways

- import data using 'read_csv"
- extract single column using `$`
- create new variable using `mutate` and simple (or complicated, if you'd like) math
- calculate values per state/year using `group_by` and `summarize`
- filter data using `filter`
- create plots using `ggplot`
  - need to specify data, aesthetics, and type of plot (`geom_*`)