# Week 3 - Graphics

Nicholas Kortessis

January 29th, 2025

## Graphics in R

In this module, we will explore some of the graphic capabilities with R's 'base' graphics. There are several R packages focused on graphics, such as 'lattice' and 'ggplot2'. I have used many of these packages and like some of their features. However, most often I go back to using 'base R' so that I have more control. In a follow-up lab we will learn more about ggplot, which is arguably the most popular graphics package in R.

**None of the default graphs that R produces are suitable as an end product. You will have use extra lines of code to manipulate them. In this class, we expect you to produce publication-ready images for your lab projects. These publication-ready figures should follow the principles of data visualization we discussed in the lecture.**

Why should you make plots in R when they require a lot of special code? *BECAUSE*, you can create a single reproducible workflow from raw data to analysis and results all in one program. This makes the scientific process more transparent.

## Outline:

0. A review of importing data, subsetting, and summarizing data.
1. Plotting amounts: barplots
2. Plotting distributions: boxplots, histograms, strip plots, and density plots
3. Plotting 2 characteristics: scatterplots
4. Plotting 3 characteristics simultaneously: Contour plots and heatmaps
5. Output to .pdf for publication

## 0. Review from the first two weeks

Here is a quick review.

### About how R works

Tasks in R are not completed until they are written into the **console** (the blue text). You can always write out code in a **script**, but nothing from the script is processed until moved to the console. R is an object oriented language, which means you assign calculations, data frames, vectors, and many other things to names, which can be recalled. Those objects that have been declared in your R session are available to be seen in the **environment**.

To efficiently move lines of a script to the console, press

- Cmd+ Return on a Mac
- Cntrl+Enter on a PC

## Creating a Reproducible Script

Scripts are most valuable because they contain a complete list of code actions that can be repeated to perform an analysis and produce a data analysis product (e.g., a figure or statistical test output). This is also helpful in providing context for the logic of the analysis and the complete set of directions that go from raw data to inference.

A best step to ensure a 'clean slate' is to begin each script with the code `rm(list = ls())`, which clears the environment of any prior objects you have declared. Once you have cleared and re-run your entire analysis, there is a complete record of your analysis and therefore no 'hidden' steps in the analysis.

## Uploading Data

Data should be entered into a spreadsheet and then converted into either a .csv file (comma separated file) or a plain text file, noting the form of separation between the column entries. Saving data using a .csv file is almost always the easiest approach. Data then can be uploaded using either the function `read.csv()` or `read.table()`. If you want a refresher on how these work, type `?read.table` into the console.

The data required for this week's lab can be found in the Week 3 lab assignment of Canvas. Just like last week, create a folder within your BIO 380 folder on your computer called "Wk 3". Download the data and save it in your Wk 3 folder. To make this workable into the directory where you will store your files and set the working directory using the `setwd()` command.

Here is an example of my working directory.

```
# this is my working directory - set yours according to the correct path on
# your computer
setwd("/Users/nicholaskortessis/Library/CloudStorage/GoogleDrive-kortessn@wfu.edu/My Drive/Import/Wake I
```

Let's use an example with a public health dataset. The CDC regularly provides data on myriad diseases of public health concern. I picked out the most recently updated data on diseases that are caused by waterborne pathogens and enteric pathogens (i.e., microbes that typically live in the gut). These are diseases such as norovirus, *E. coli*, and salmonella infections that we worry about with food and contact with infected water. You can read about the dataset here.

It is labelled as `NORS_20250128.csv` on the Canvas page. Let's load it up and take a look.

```
NORS.df <- read.csv(file = "NORS_20250128.csv")
```

Let's take a look at some of the properties of the data frame. First, let's look at the characteristics it has.

```
str(NORS.df)
```

```
## 'data.frame':    66713 obs. of  19 variables:
##  $ Year                          : int  2023 2023 2023 2023 2023 2023 2023 2023 2023 2023 ...
##  $ Month                         : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ State                         : chr  "Minnesota" "Massachusetts" "North Carolina" "Wisconsin" ...
##  $ Primary.Mode                  : chr  "Food" "Indeterminate/unknown" "Person-to-person" "Person-to-pe
##  $ Etiology                      : chr  "Norovirus Genogroup IX" "Norovirus" "Norovirus unknown" "Norov
##  $ Serotype.or.Genotype          : chr  "GII.P15 GIX.1" "" "" "" ...
##  $ Etiology.Status               : chr  "Confirmed" "Suspected" "Confirmed" "Confirmed;Confirmed" ...
##  $ Setting                       : chr  "Restaurant: Sit-down dining" "Long-term care/nursing home/assi
##  $ Illnesses                     : int  23 7 23 12 9 4 18 7 2 11 ...
##  $ Hospitalizations              : int  0 0 1 0 0 0 0 1 0 NA ...
##  $ Info.On.Hospitalizations      : int  23 0 23 12 9 4 18 7 2 0 ...
##  $ Deaths                        : int  0 0 0 0 0 0 0 0 0 NA ...
##  $ Info.On.Deaths                : int  23 0 23 12 9 4 18 7 2 0 ...
##  $ Food.Vehicle                  : chr  "" "" "" "" ...
##  $ Food.Contaminated.Ingredient: chr  "" "" "" "" ...
```

```
##  $ IFSAC.Category              : chr  "" "" "" "" ...
##  $ Water.Exposure              : chr  "" "" "" "" ...
##  $ Water.Type                  : chr  "" "" "" "" ...
##  $ Animal.Type                 : chr  "" "" "" "" ...
```

We can see that each observation has date and location information (at least the state) and has information about the pathogen (etiology means the cause of a disease). It also has information on illnesses, hospitilizations, and deaths. There's other stuff, but it's not obvious what these measure about the individuals (but the metadata on the CDC website has that information, as it should!).

**Checkpoint 1: Do you have questions about loading data? Looking at the data frame, what is a statistical individual in this case? Stated differently, what is the CDC's unit of sampling?** This is our first BIG dataset. This dataset has 19 characteristics per individual (i.e., 19 columns) and the summary says the dataset has almost 67,000 individuals. Whoa. Good thing we have R, it is well suited for handling all that information.

To see what is going on, let's first get a handle on the time part. Let's see what times are in the dataset.

```
unique(NORS.df$Year)
```

```
##  [1] 2023 2022 2021 2020 2019 2018 2017 2016 2015 2014 2013 2012 2011 2010 2009
## [16] 2008 2007 2006 2005 2004 2003 2002 2001 2000 1999 1998 1997 1996 1995 1994
## [31] 1993 1992 1991 1990 1989 1988 1987 1986 1985 1984 1983 1982 1981 1980 1979
## [46] 1978 1977 1976 1975 1974 1973 1972 1971
```

The function `unique()` gives us all the different kinds of `Year` values in the data set. Turns out this data has reports on enteric and waterborne illnesses every year going back to 1972. That's 53 years of data!

Let's also look to see what `Illnesses` looks like. Above, it says `Illnesses` is an integer. If you haven't figured it out by now, each *individual* in this dataset is an food or water-borne illness. For example, a norovirus outbreak. This outbreak happens at a particular location at a particular time, and causes some number of illness, with some number of deaths. Moreover, the illness can be categorized by the pathogen that caused it. So we could look at the, for example, the size, number, and severity of these outbreaks over time and across different states.

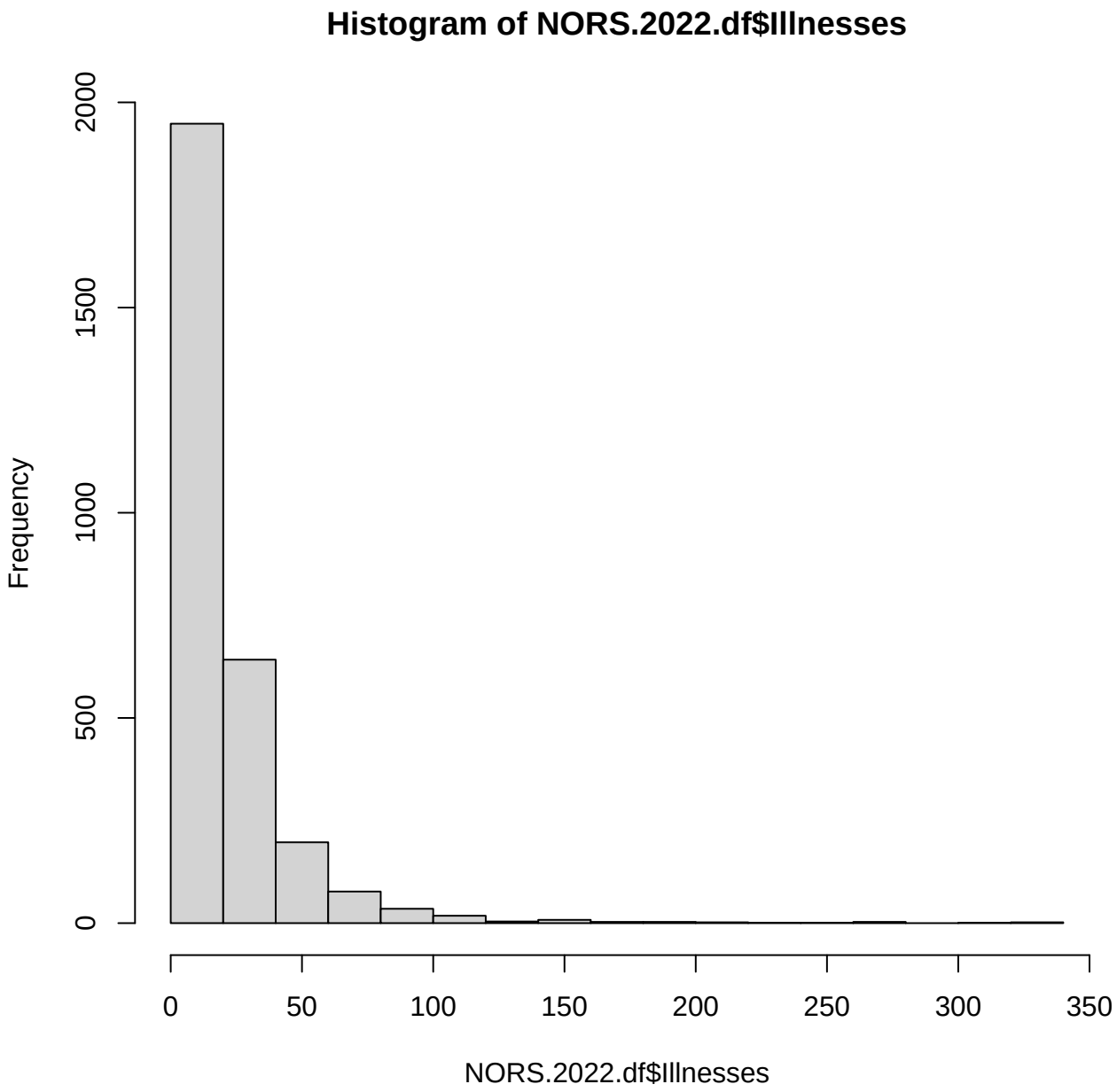
## Subsetting dataframes

To get a handle on this data. Let's start with an example where we want to know how many foodborne and waterborne deaths there were in 2022. To do this, we need to remember how to subset. Here is an example.

```
NORS.2022.df <- subset(NORS.df, subset = (Year == 2022), select = c("Illnesses"))
```

To get a visual, let's plot a histogram

```
hist(NORS.2022.df$Illnesses)
```




In 2022, most outbreaks were small, with fewer than 100 people getting sick in any one outbreak. But some were quite large. Some summary statistics of this can be found by using the `summary()` function.

```
summary(NORS.2022.df$Illnesses)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.00    6.00   13.00   20.77   26.00  322.00
```

Here, you can see that half of all outbreaks in 2022 affected 13 people or fewer. And 75% of outbreaks affected 26 people or fewer. The largest outbreak that year affected 322 people. We can ask what the pathogen was that caused the big outbreak in 2022.

```
subset(NORS.df, subset = (Year == 2022 & Illnesses == 322))
```

```
##      Year Month      State   Primary.Mode
## 4471 2022     2 Multistate Animal contact
## 6580 2022    11 Multistate           Food
```

```
##                                                          Etiology
## 4471                                          Salmonella enterica
## 6580 Norovirus Genogroup II;Norovirus unknown;Norovirus Genogroup II
##      Serotype.or.Genotype              Etiology.Status
## 4471          Enteritidis                   Confirmed
## 6580 unknown;GII.P7 GII.7 Confirmed;Confirmed;Suspected
##                                                                                Setting
## 4471 Residence - Single-family home;Agricultural feed store;Residence - Multi-unit housing
## 6580               Restaurant: Sit-down dining;Grocery store/bakery/deli/convenience store
##      Illnesses Hospitalizations Info.On.Hospitalizations Deaths Info.On.Deaths
## 4471       322               52                      179      1            179
## 6580       322                1                      319      0            319
##      Food.Vehicle Food.Contaminated.Ingredient IFSAC.Category Water.Exposure
## 4471
## 6580      oysters                 oysters, raw       Mollusks
##      Water.Type Animal.Type
## 4471               Poultry
## 6580
```

This actually shows two outbreaks with 322 individuals in 2022. Scanning through the data, we can find a few pieces of information.

```
subset(NORS.df, subset = (Year == 2022 & Illnesses == 322), select = c("Deaths",
    "Etiology", "Animal.Type", "Food.Vehicle"))
```

```
##      Deaths                                                  Etiology
## 4471      1                                       Salmonella enterica
## 6580      0 Norovirus Genogroup II;Norovirus unknown;Norovirus Genogroup II
##      Animal.Type Food.Vehicle
## 4471     Poultry
## 6580                 oysters
```

One was caused by Salmonella and the other was caused by norovirus. The Salmonella outbreak was caused by poultry (presumably in raw chicken) and caused one death while the norovirus outbreaks was caused by people eating infected oysters.

**Checkpoint 2: Use the `subset()` function (or any other suitable function to answer the following questions.**

1. How big was the largest outbreak (in terms of number of illnesses) in 2023?
2. What was the median outbreak size in 1977?
3. Where there any outbreaks that were larger than 10000 in the data set? If so, when and where did they occur, and what caused it?

# 1. Plotting amounts

One thing we might want to do is to plot how much there is of something. For example, we could want to plot how much sickness was caused by diseases each year. A typical approach is to use *barplots* to visualize amounts. But before doing that, let's think for a second. Would you ever dream of being able to make a barplot with *this data* in excel? Remember, this dataset has almost *67,000* rows of data!

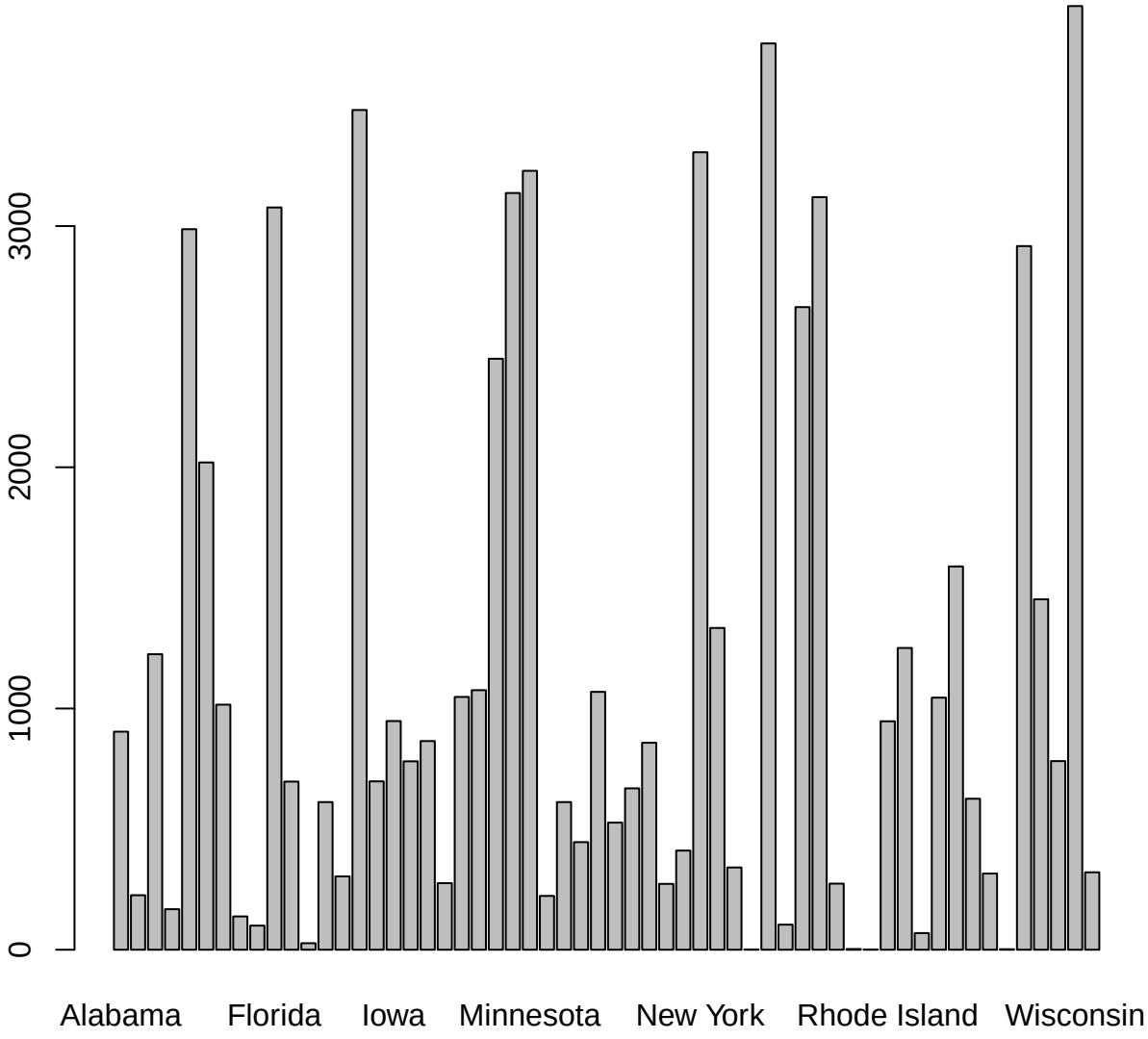
To make a barplot, we simply need the function `barplot()`. Easy enough. Let's begin with something simple. Say we want to plot how many outbreaks there are in each state. We could subset data across all states, but R has an easier way to do this to determine counts of something. The function is `table`. Let's count how many outbreaks there are in each state in the dataset.

```r
(NORS.state.outbreaks <- table(NORS.df$State))
```

```
##
##                          Alabama                           Alaska
##                              904                              226
##                          Arizona                         Arkansas
##                             1225                              168
##                       California                         Colorado
##                             2987                             2020
##                      Connecticut                         Delaware
##                             1016                              138
##             District of Columbia                          Florida
##                              100                             3077
##                          Georgia                             Guam
##                              697                               27
##                           Hawaii                            Idaho
##                              612                              304
##                         Illinois                          Indiana
##                             3481                              698
##                             Iowa                           Kansas
##                              948                              781
##                         Kentucky                        Louisiana
##                              865                              276
##                            Maine                         Maryland
##                             1048                             1076
##                    Massachusetts                         Michigan
##                             2450                             3137
##                        Minnesota                      Mississippi
##                             3229                              223
##                         Missouri                          Montana
##                              612                              446
##                       Multistate                         Nebraska
##                             1069                              527
##                           Nevada                    New Hampshire
##                              669                              858
##                       New Jersey                       New Mexico
##                              273                              411
##                         New York                   North Carolina
##                             3306                             1334
##                     North Dakota          Northern Mariana Islands
##                              341                                1
##                             Ohio                         Oklahoma
##                             3757                              104
##                           Oregon                     Pennsylvania
##                             2664                             3120
##                      Puerto Rico                 Republic of Palau
##                              274                                3
## Republic of the Marshall Islands                     Rhode Island
##                                1                              947
##                     South Carolina                     South Dakota
##                             1251                               69
##                        Tennessee                            Texas
##                             1045                             1589
##                             Utah                          Vermont
```

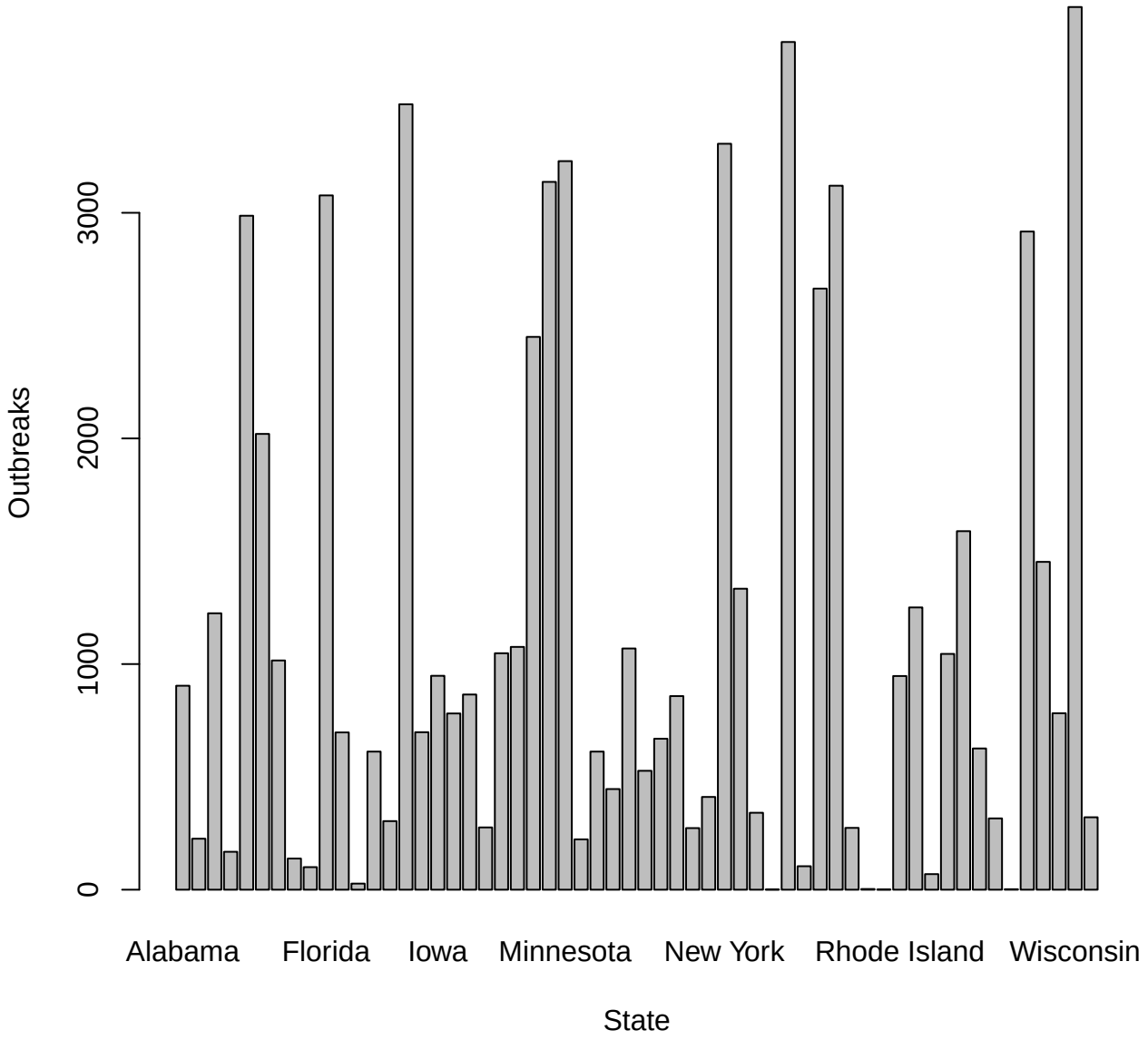
```
##                          626                          316
##                Virgin Islands                     Virginia
##                            2                         2917
##                   Washington                West Virginia
##                         1453                          782
##                    Wisconsin                      Wyoming
##                         3912                          321
```

```
barplot(NORS.state.outbreaks)
```



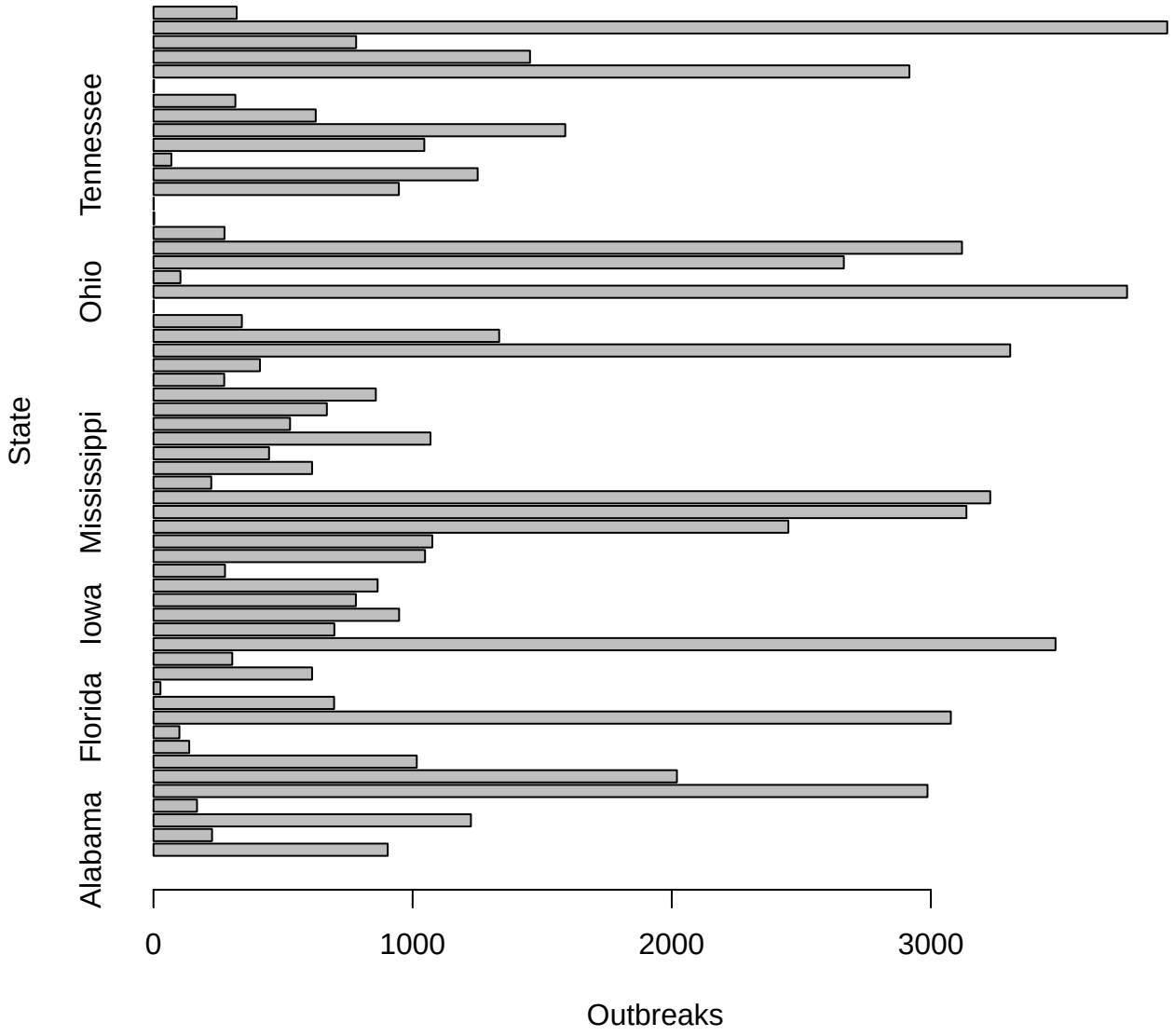

Before going any further, let's think about what this shows and whether there are ways to improve it. The first thing to note is that we need axis labels. To do that, we can use the arguments `xlab` and `ylab` like this

```
barplot(NORS.state.outbreaks, xlab = "State", ylab = "Outbreaks")
```



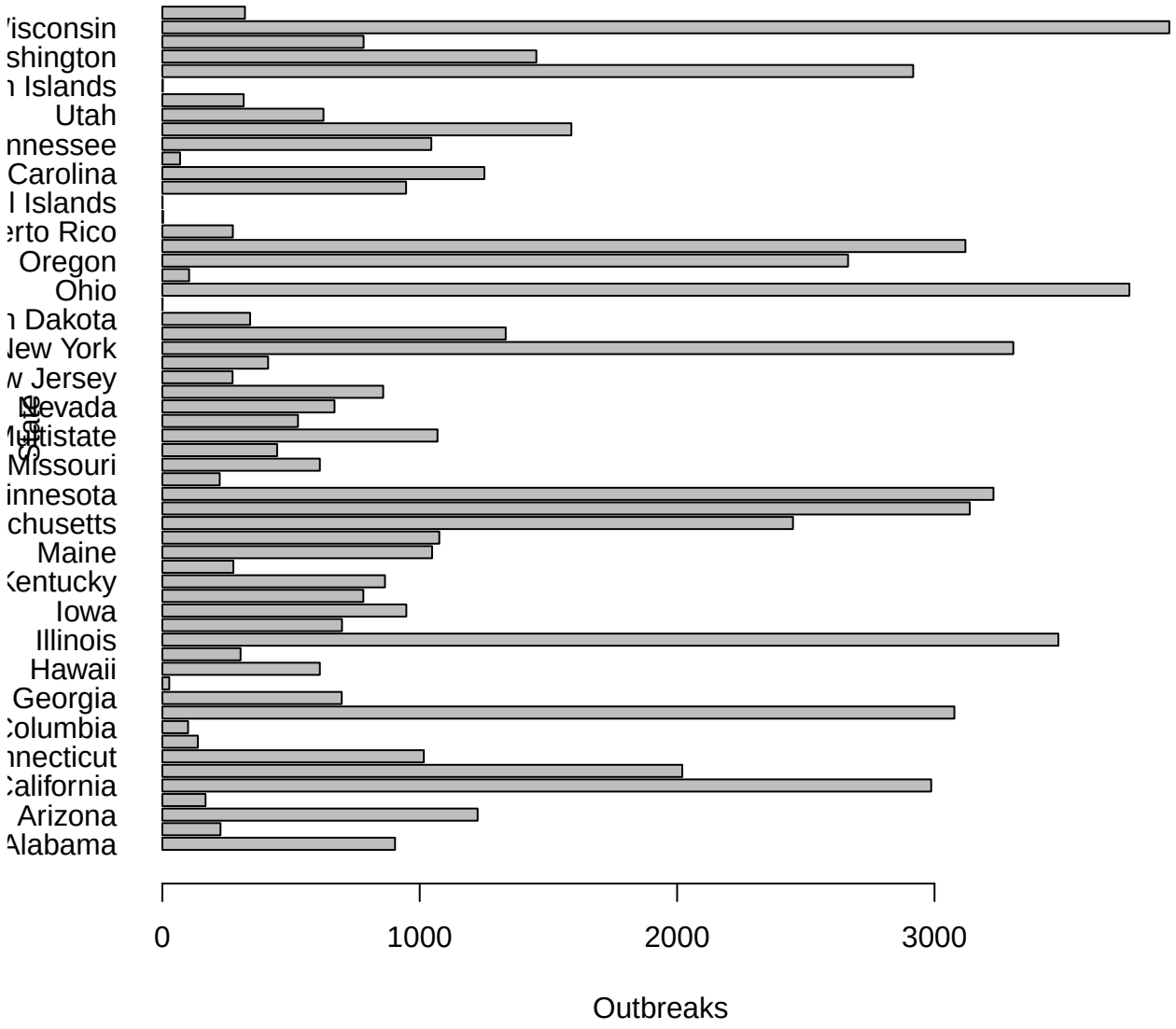

This data has a lot of categories (50 states + some territories. We will never be able to write their names side by side. This is a good time to flip the barplot on its side and make it a *horizontal bar plot*.

```
barplot(NORS.state.outbreaks, horiz = TRUE, xlab = "Outbreaks", ylab = "State")
```



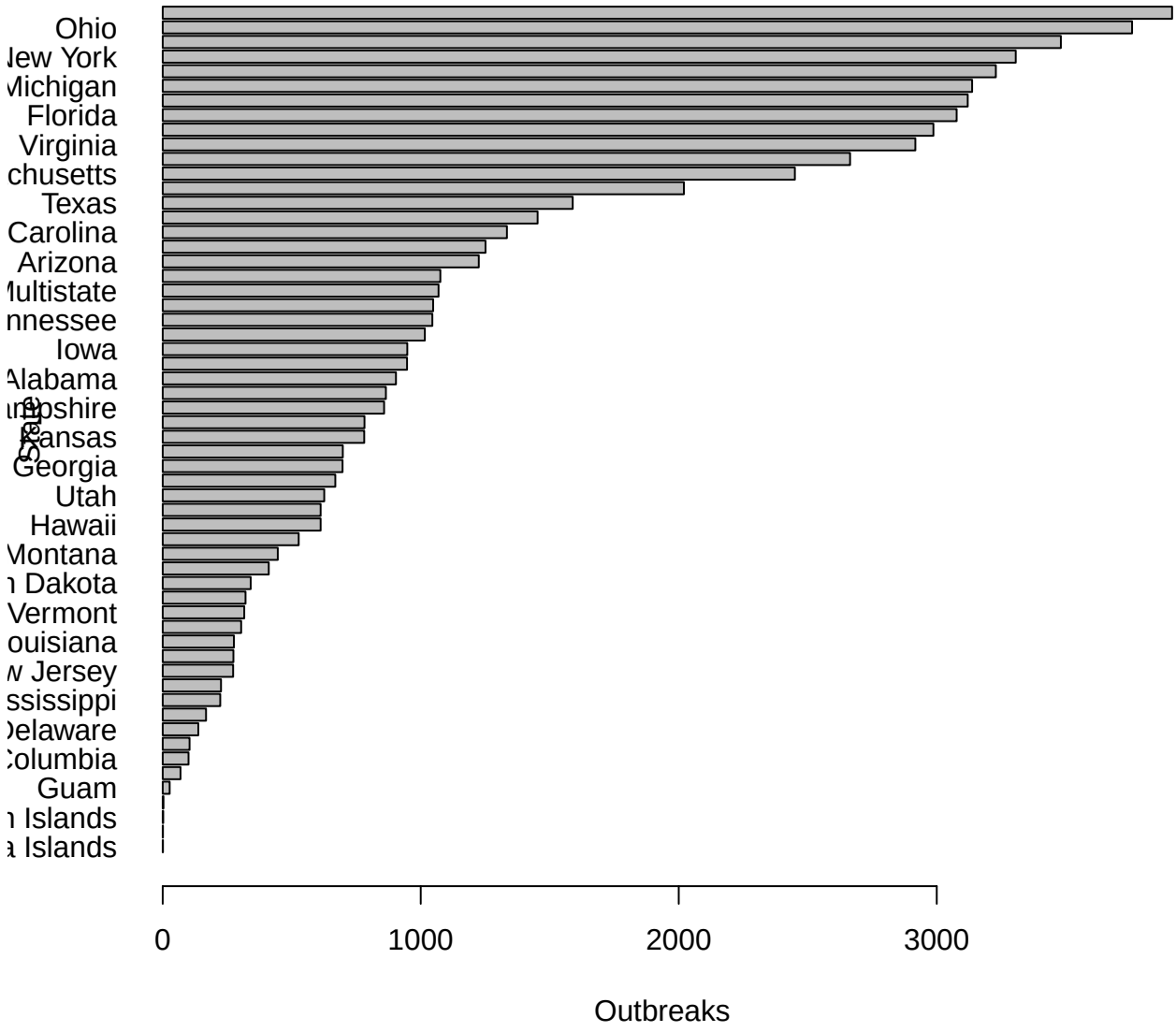

Let's turn the names on their side so they are easier to read.

```
barplot(NORS.state.outbreaks, horiz = TRUE, # make the plot horizontal
        xlab = 'Outbreaks', ylab = 'State', # add axes
        las = 1) # make the labels horizontal
```



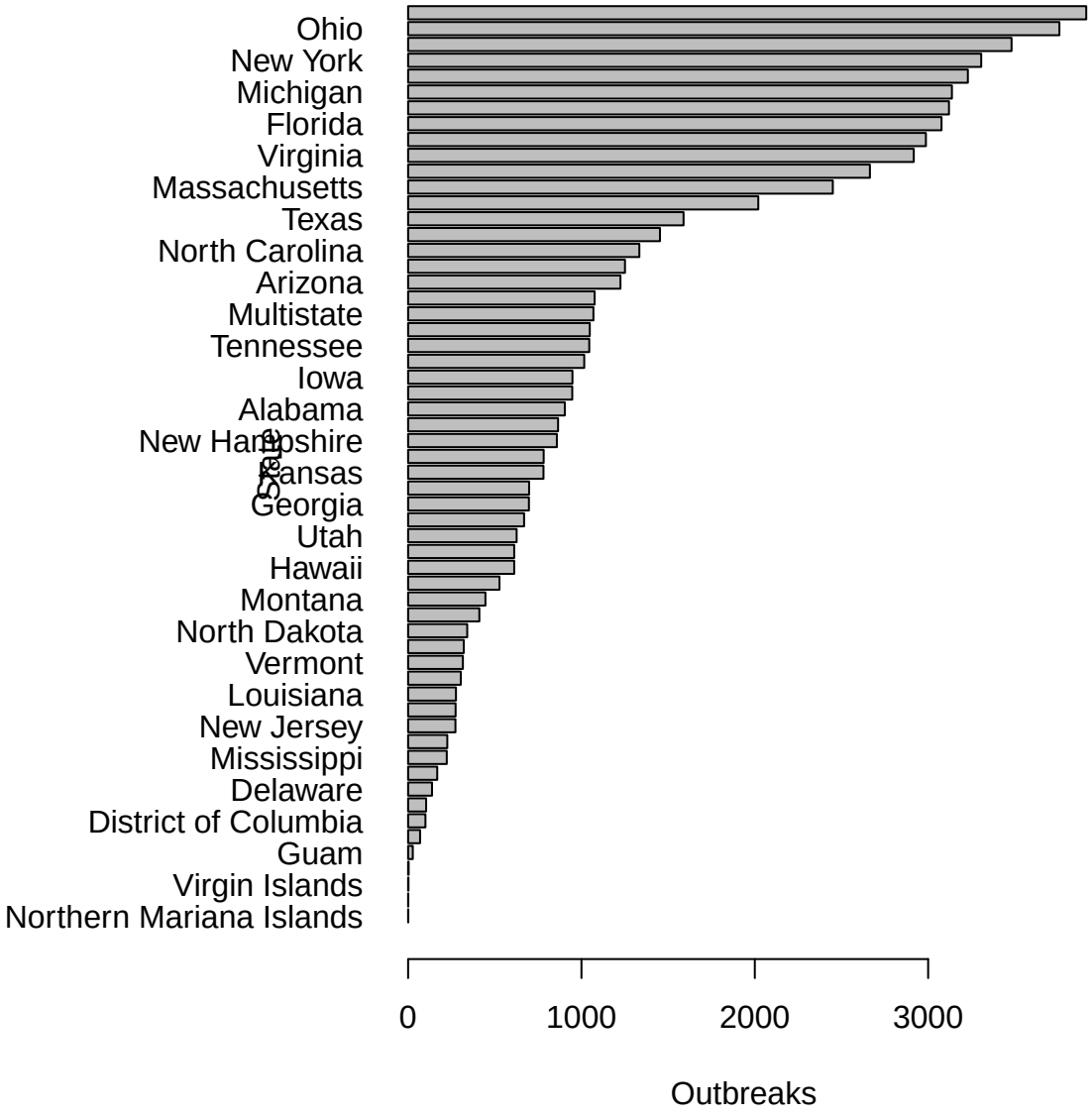

A reasonable reason to make this plot is to see which states have the most outbreaks and which have the fewest. But the categories here (states) are listed in alphabetical order. That's not related to the main question. Instead, let's order them from largest to smallest. To do this, we need to first reorder the dataset.

```
NORS.outbreak.ordered <- NORS.state.outbreaks[order(NORS.state.outbreaks,
                                                    decreasing = FALSE)]

barplot(NORS.outbreak.ordered, horiz = TRUE, #Plot the ordered data as a horizontal barplot
        xlab = 'Outbreaks', ylab = 'State', # Add axis labels
        las = 1) # Rotate the axis labels so they are all horizontal
```



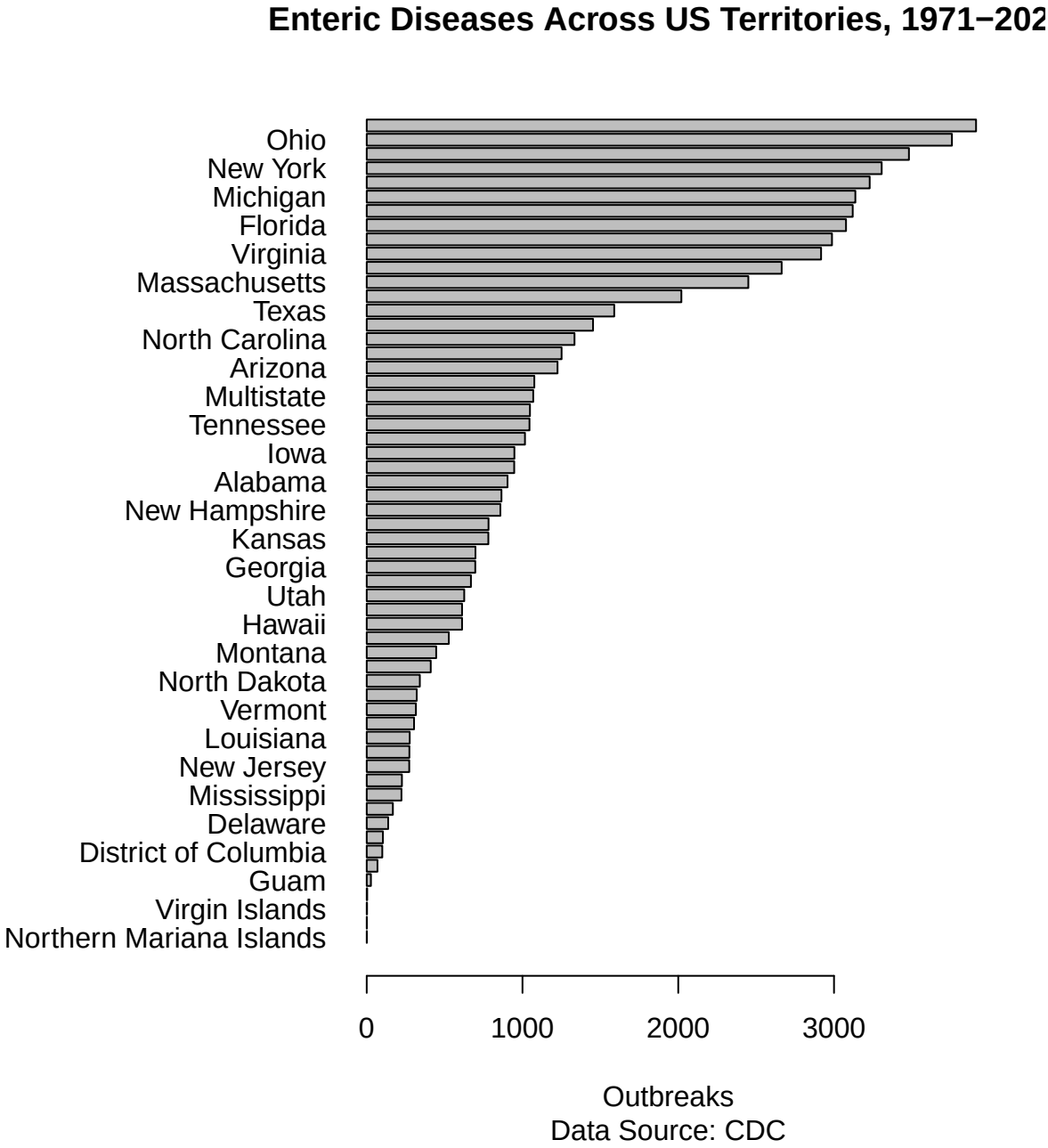

That's much better. But there are still a few issues. For one, the state labels are too big and are running of the figure. We can fix this by making the labels areas bigger or by making the text smaller. Here, we will do a bit of both. To make the plotting areas bigger, we need the `par` function with the argument `mar =` which sets the margins of the figure. The default is `mar = c(5,4,4,2)`. These are the four numbers corresponding to the space on the c('bottom', 'left', 'top', and 'bottom'). We need to make quite a bit more space, so let's make the 'left' a lot bigger. We will change 4 to 15.

```
par(mar = c(5,15,4,2))
barplot(NORS.outbreak.ordered, horiz = TRUE, #Plot the ordered data as a horizontal barplot
        xlab = 'Outbreaks', ylab = 'State', # Add axis labels
        las = 1) # Rotate the axis labels so they are all horizontal
```




**Checkpoint 3: How would the plot change if we wrote `par(mar = c(15,5,4,2)` instead?** Last couple of things. The y-axis is causing us problems. Let's make a title that has the same information. And we will add a note at the bottom saying where the data come from.

```
par(mar = c(5,15,4,2))
barplot(NORS.outbreak.ordered, horiz = TRUE, #Plot the ordered data as a horizontal barplot
        xlab = 'Outbreaks', ylab = '', # Add axis labels
        las = 1, # Rotate the axis labels so they are all horizontal
        main = 'Enteric Diseases Across US Territories, 1971-2023', # add title
        sub = 'Data Source: CDC') # add a sub title at the bottom of the figure
```




There's a lot more we could do here, but that's enough for now. This figure looks quite good and would be acceptable for the public and also for a scientific publication.

**Checkpoint 4: Make a barplot illustrating the number of outbreaks across states in 2023, the most recent year of the data. Make sure your script can reproduce this figure. You will receive credit for the appropriate figure, so make sure it has a title, axis labels where appropriate, and is interpretable by a scientist.**

## 2. Plotting distributions

There are other questions we could ask about this dataset related to multiple observations within a single category. For example, we might be interested in how big outbreaks are. To answer that, we might want to see the distribution of the number of individuals infected in every outbreak in the dataset. This is in essence

making a frequency distribution. We want to see how bad these outbreaks have been in the dataset and how common each is. We can do this using *boxplots*, *histograms*, *strip plots,* and *density plots*. Let's look at each.

To get this question of outbreak size, let's use the number of individuals infected and ill in each outbreak. This data shows up in the "Illnesses" column of the data.

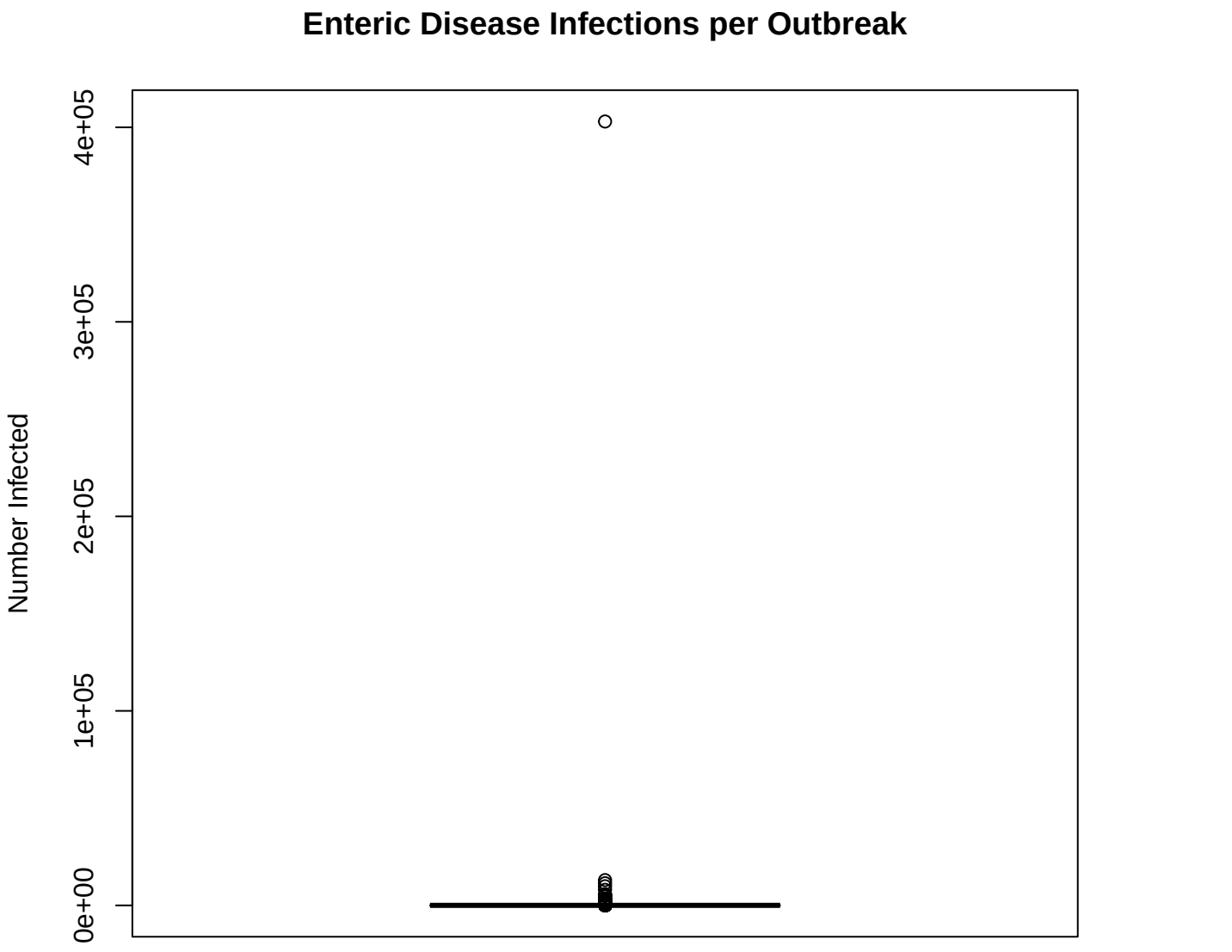
```
head(NORS.df$Illnesses)
```

```
## [1] 23  7 23 12  9  4
```

Woof. There are a bunch (66,713). Visualizations are super helpful in this case.

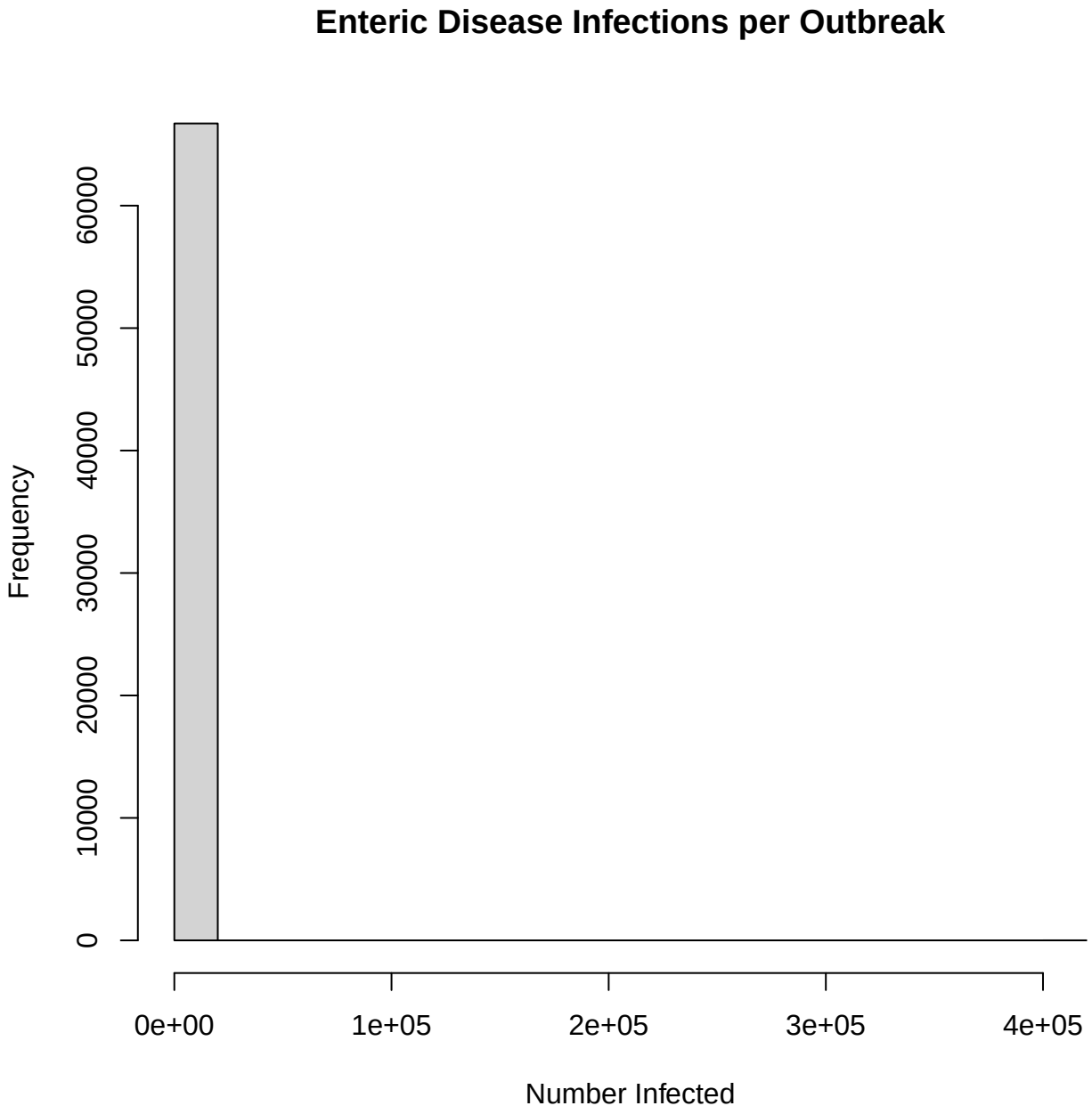
First, a boxplot.

```
boxplot(NORS.df$Illnesses, ylab = "Number Infected", main = "Enteric Disease Infections per Outbreak")
```



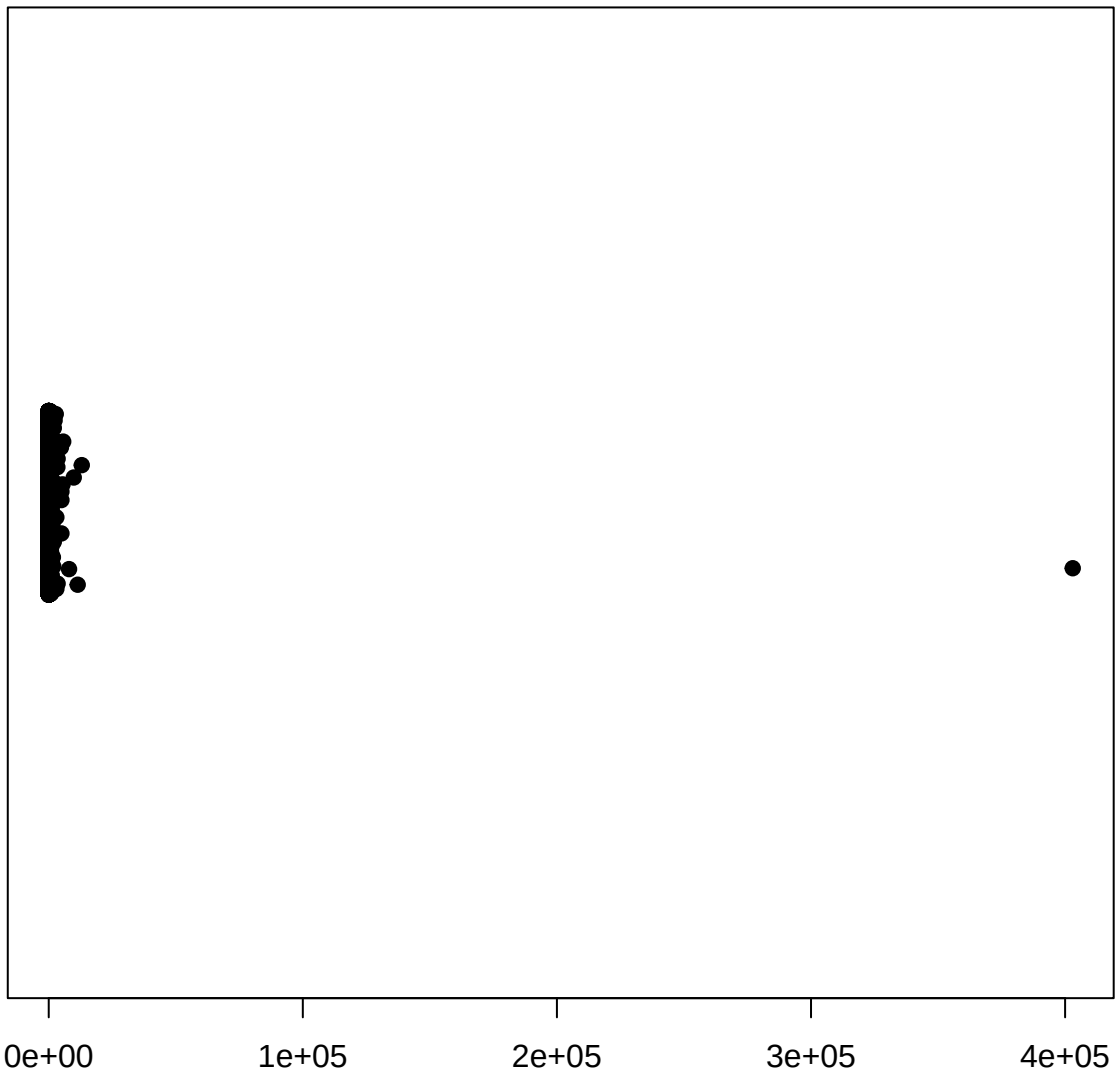

Well. That's a boxplot, but it doesn't look to helpful. Let's try the something else. How about a histogram?

```r
hist(NORS.df$Illnesses, xlab = "Number Infected", main = "Enteric Disease Infections per Outbreak")
```



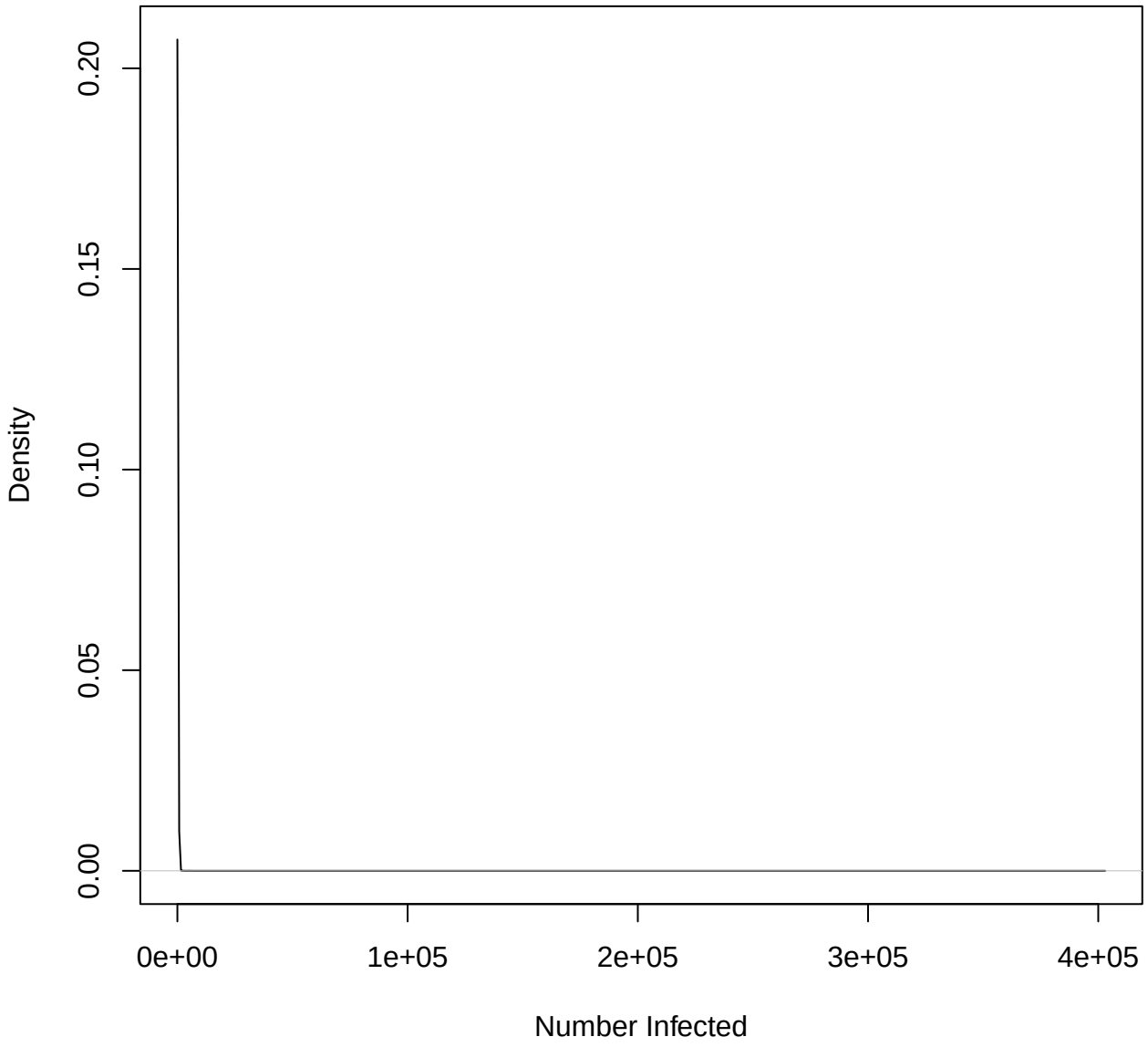

Hmm. Also not super helpful. Let's try a strip plot. We will 'jitter' the points, which is to say that we move them a little bit so they are all not right on top of each other. Clearly we have a bunch of zeros. We don't want all those zeros in the same space, so we move them a bit to see each one.

```r
stripchart(NORS.df$Illnesses, # strip chart of illnesses
           method = 'jitter', # move the points a little bit
           xlab = 'Number Infected', # make x label
           main= 'Enteric Disease Infections per Outbreak', # add title
           pch = 19) # make filled circles
```




Hmm. That's better, but again, not super helpful. A bunch of data points are more or less obscured because they are all in the same space. Let's try the density plot approach.

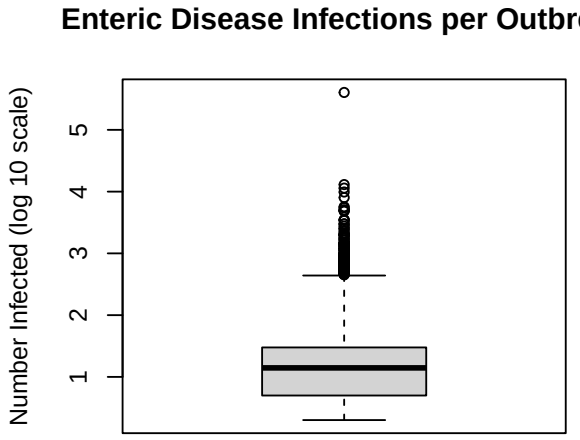
```
hosp.density <- density(NORS.df$Illnesses) # Estimate the density curve
plot(hosp.density,                         # And make a plot of it
     xlab = 'Number Infected',
     main = 'Distribution of Outbreak Size')
```

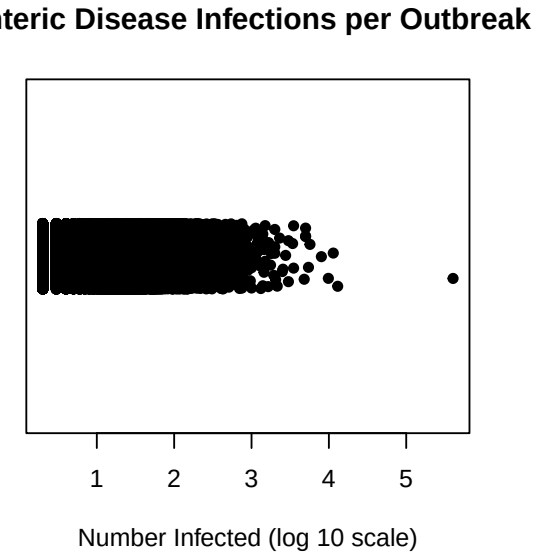


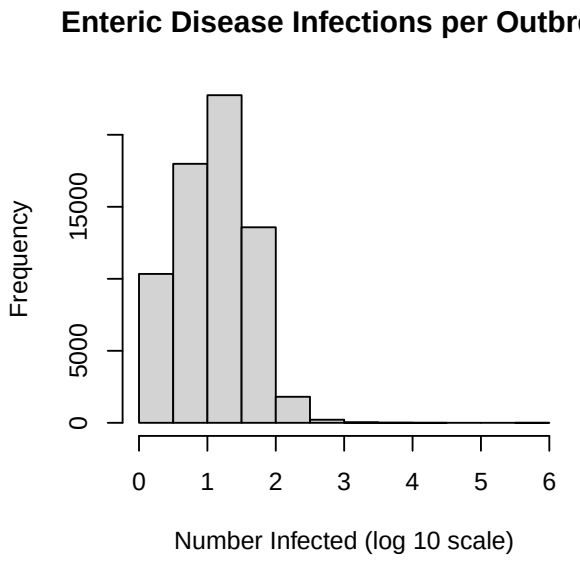

This one isn't helpful either. The problem here is that there are many many many small outbreaks and a very small number of absolutely gargantuan outbreaks. One way to deal with this kind of issue is to put the data on a log scale. Let's try that.

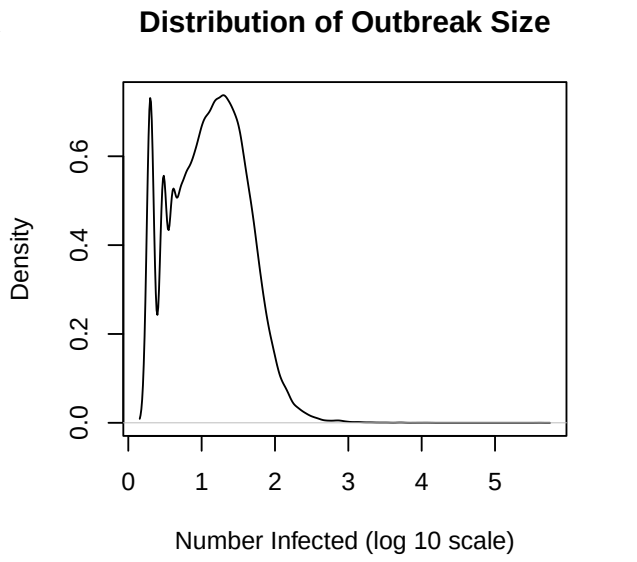
```
log.illnesses <- log10(NORS.df$Illnesses)
par(mfcol = c(2,2)) # This makes a four panel figure with 2 columns and 2 rows

boxplot(log.illnesses, # make boxplot
        ylab = 'Number Infected (log 10 scale)', #add label
        main= 'Enteric Disease Infections per Outbreak') # add title

hist(log.illnesses, # make histogram
        xlab = 'Number Infected (log 10 scale)', #x label
        main= 'Enteric Disease Infections per Outbreak') #title
```

```r
stripchart(log.illnesses, # make stipchart
           method = 'jitter', # add jitter
           xlab = 'Number Infected (log 10 scale)', # xlabel
           main= 'Enteric Disease Infections per Outbreak', #title
           pch = 19) # make filled circles

log.illness.density <- density(log.illnesses)

plot(log.illness.density, #Plot density
     xlab = 'Number Infected (log 10 scale)', #x label
     main = 'Distribution of Outbreak Size') # title
```




This helped quite a bit. Now we see that most epidemics are less than $10^2 = 100$ (i.e., the base 10 logarithm of 100 is 2). There are only a few that are out near 4,5,6 (=1000; 10,000; 100,000) illnesses. Each plot shows us something a little bit different. Which one is appropriate might depend the exact features of the distribution that you want to show.

**Checkpoint 5: Which plot (or combination of plots) would you use to show that most epidemics are very small, but there are a few very, very big ones? Provide some justification for your answer.**

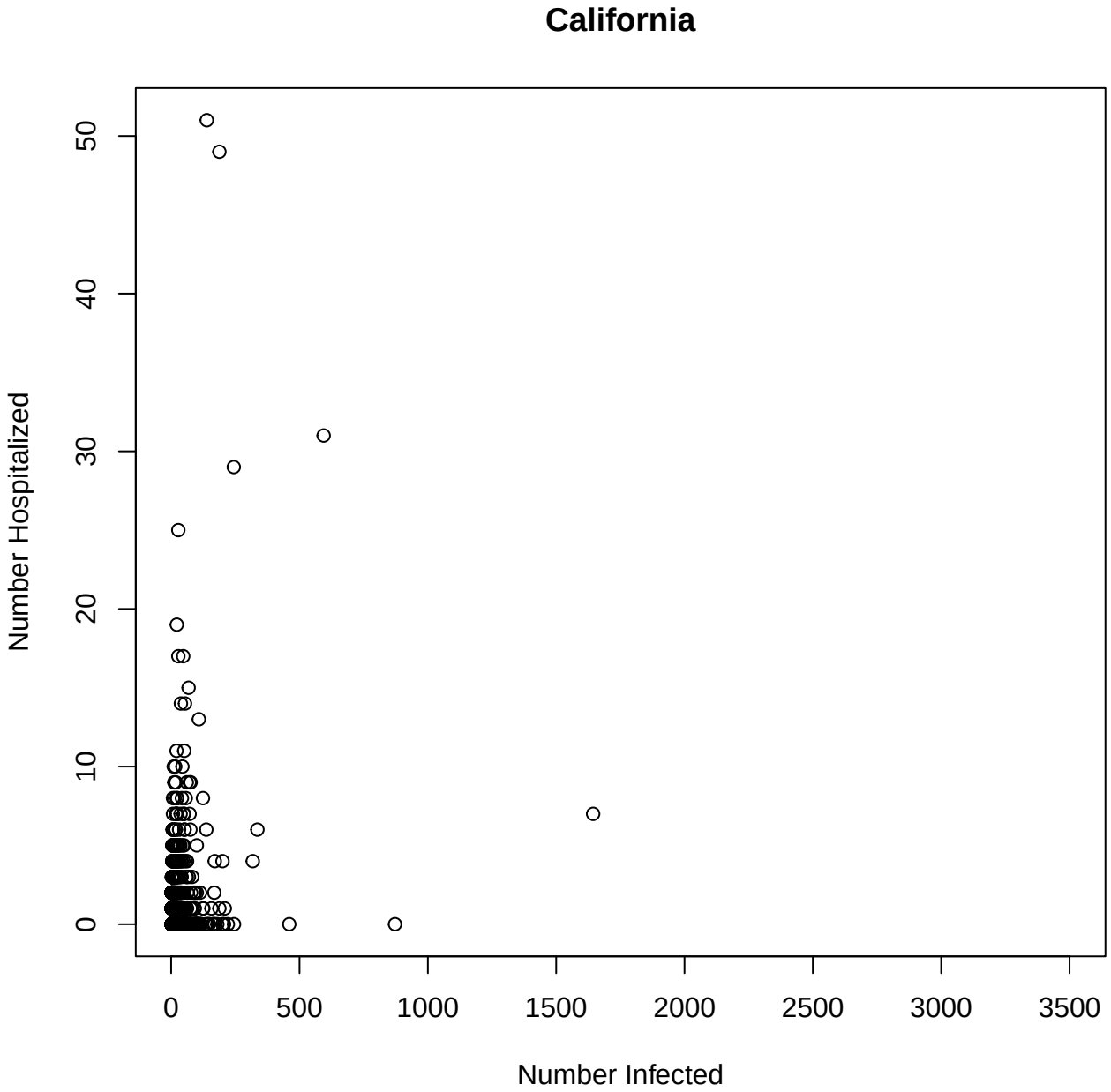
**Checkpoint 6: Write code to create a figure of the distribution of outbreak sizes in the state of New York.**

## 3. Plotting scatterplots

Scatterplots, with or without lines are used for visualizing relationships between multiple numeric characters. Lines are helpful when there is a third feature that connects points together. This is often time. Scatterplots without lines are the basic plotting function in R, and so they are relatively easy to use and can be done with the function `plot`. You simply give the `plot` function a set of x values and a set paired y values and you are good to go.

Let's give this a try by looking at the relationship between the size of an outbreak and the number of individuals that were hospitalized during the outbreak as part of their illness in the state of California.

```
CA.NORS.df <- subset(NORS.df, subset = State == 'California')
plot(CA.NORS.df$Illnesses, # x-values
     CA.NORS.df$Hospitalizations, # y-values
     xlab = 'Number Infected', ylab = 'Number Hospitalized',
     main = 'California')
```




Each point in this figure represents a statistical individual – an outbreak. Each outbreak has multiple characteristics. In this case, we know that each outbreak was located in California, and it' position along the x-axis tells us how many individuals were infected during the course of that outbreak, and the y-axis tells us how many individuals were admitted to the hospital during the course of that outbreak.

**Checkpoint 7: What does this graph tell you about the relationship between outbreak size and outbreak severity?** Another kind of x-y scatterplot is one where the data points are ordered, such as when data points are linked over time. Such a graph is called a *time series* graph. These are very common in epidemiology. Let's take a look.

Say we want to see how the size of outbreaks each year. To do this, we need to sum over the number of illnesses each year. This can be a challenge to do by hand. Instead, we can use a function called `aggregate`.

The function `aggregate` works similarly to `tapply` that we learned last week, except that it has a more natural language to it. You write the variable you are interested in *as a function of other characteristics.* For our question of the total number of illnesses each year, we want to sum all the illnesses in each outbreak each

year. We can do this with `aggregate` as follows.

```
(yearly.illnesses <- aggregate(Illnesses ~ Year, data = NORS.df, FUN = sum))
```

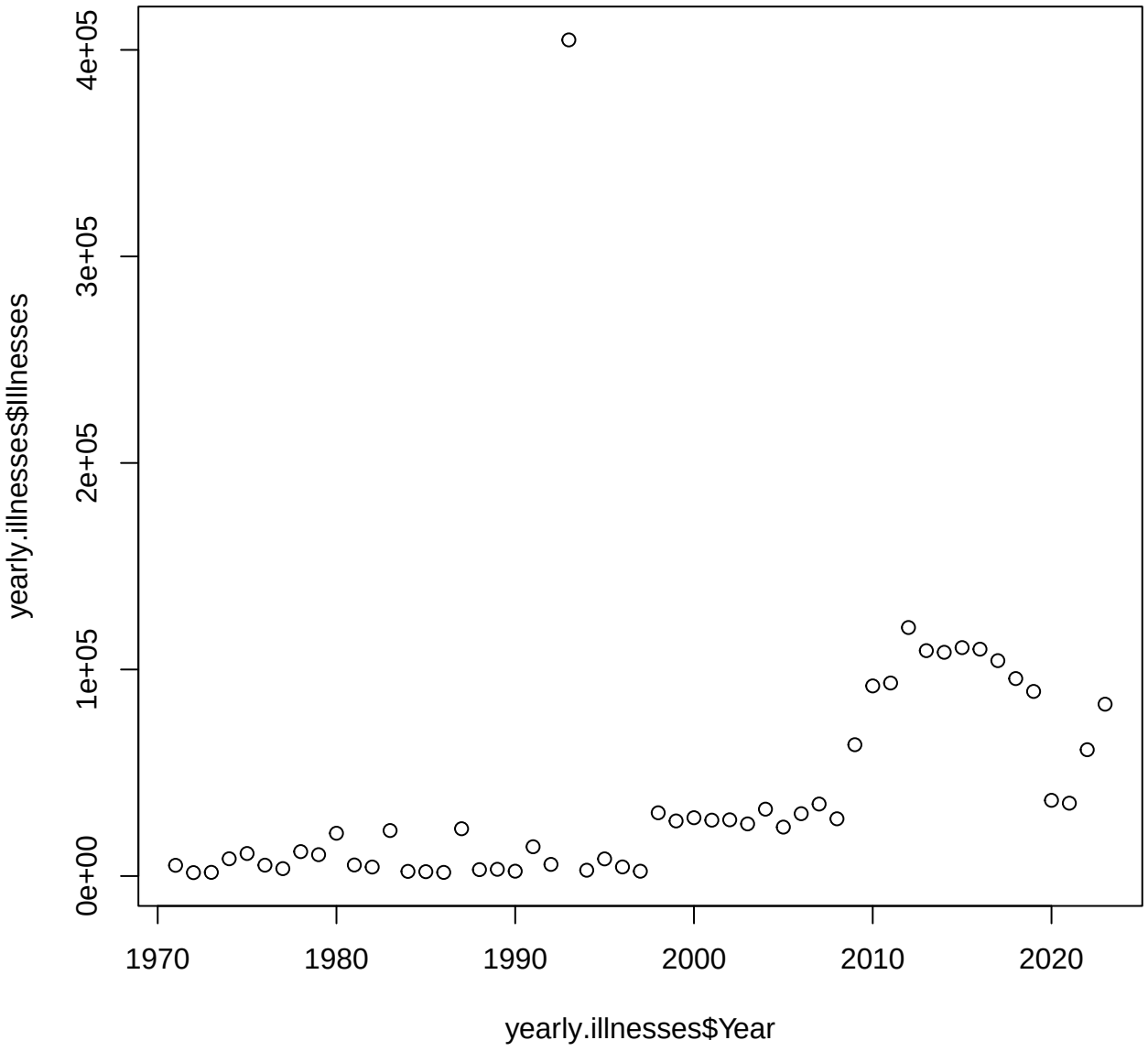
```
##    Year Illnesses
## 1  1971      5184
## 2  1972      1650
## 3  1973      1784
## 4  1974      8354
## 5  1975     10912
## 6  1976      5263
## 7  1977      3586
## 8  1978     11822
## 9  1979     10333
## 10 1980     20722
## 11 1981      5358
## 12 1982      4361
## 13 1983     22028
## 14 1984      2246
## 15 1985      2109
## 16 1986      1760
## 17 1987     22904
## 18 1988      3094
## 19 1989      3255
## 20 1990      2345
## 21 1991     14184
## 22 1992      5631
## 23 1993    404844
## 24 1994      2813
## 25 1995      8300
## 26 1996      4400
## 27 1997      2326
## 28 1998     30624
## 29 1999     26671
## 30 2000     28232
## 31 2001     27064
## 32 2002     27220
## 33 2003     25247
## 34 2004     32384
## 35 2005     23688
## 36 2006     30188
## 37 2007     34857
## 38 2008     27722
## 39 2009     63582
## 40 2010     92043
## 41 2011     93453
## 42 2012    120323
## 43 2013    109134
## 44 2014    108317
## 45 2015    110600
## 46 2016    109848
## 47 2017    104286
## 48 2018     95605
## 49 2019     89378
## 50 2020     36659
```

```
## 51 2021      35267
## 52 2022      61153
## 53 2023      83205
```

The way to interpret this code is that we read `Illnesses~Year` as "let's look at how illnesses changes as a function of year" where the `~` symbol (typically found just to the left of the number 1 on your keyboard) reads "as a function of".

Let's see how it looks when we plot it.

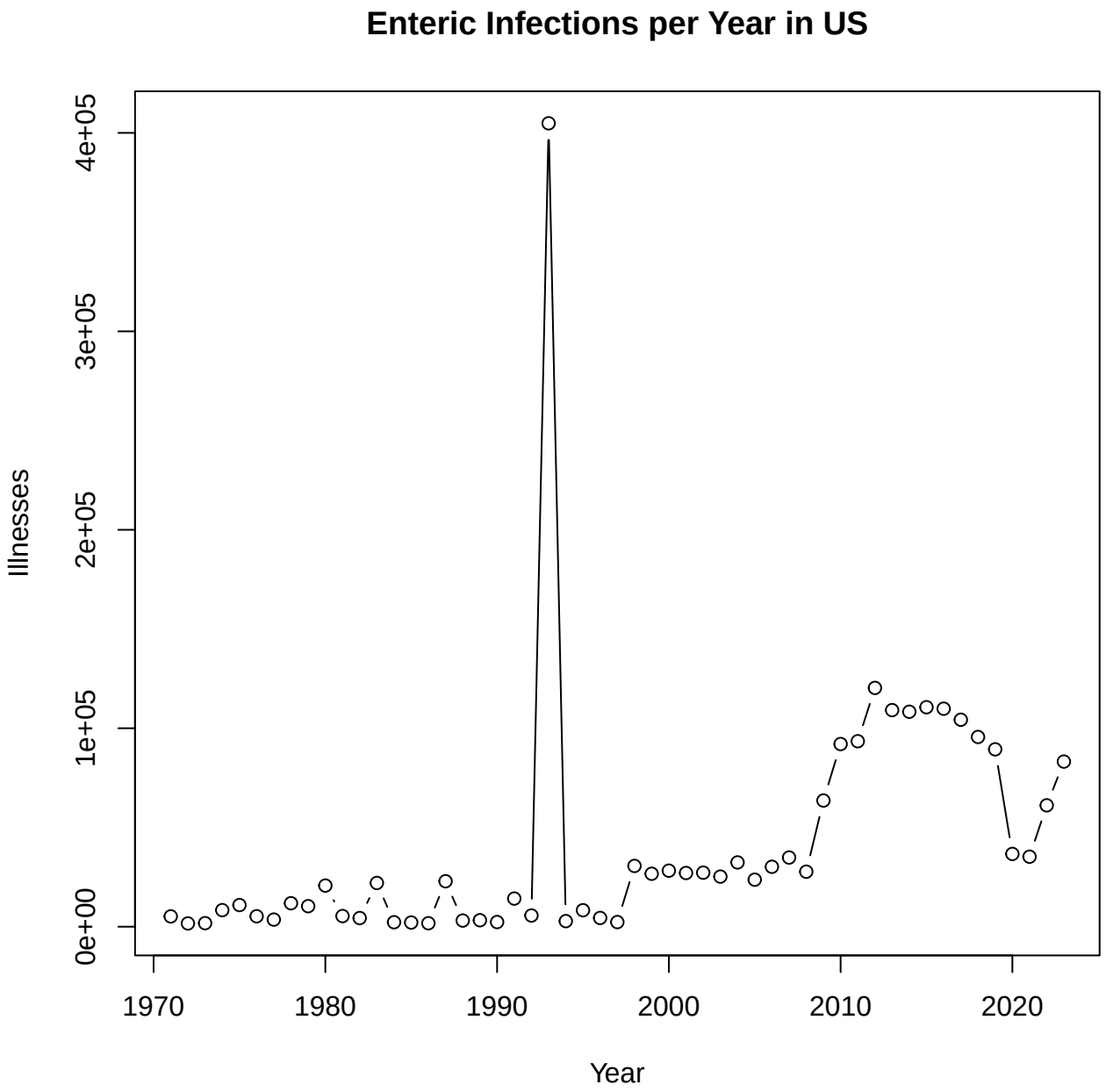
```
plot(yearly.illnesses$Year, # x-values
     yearly.illnesses$Illnesses) # y-values
```




Okay. That doesn't look interesting. Let's add the line to indicate it is a time series. To do this, the `plot` function in R has an argument for the *type* of scatterplot, where you can indicate `'l'` for line `'p'` for points, and `'b'` for both. We want both. So we will write `typ = 'b'` in the arguments for the plot. Let's also add axis labels and a title.

```
plot(yearly.illnesses$Year, # x-values
     yearly.illnesses$Illnesses, # y-values
```

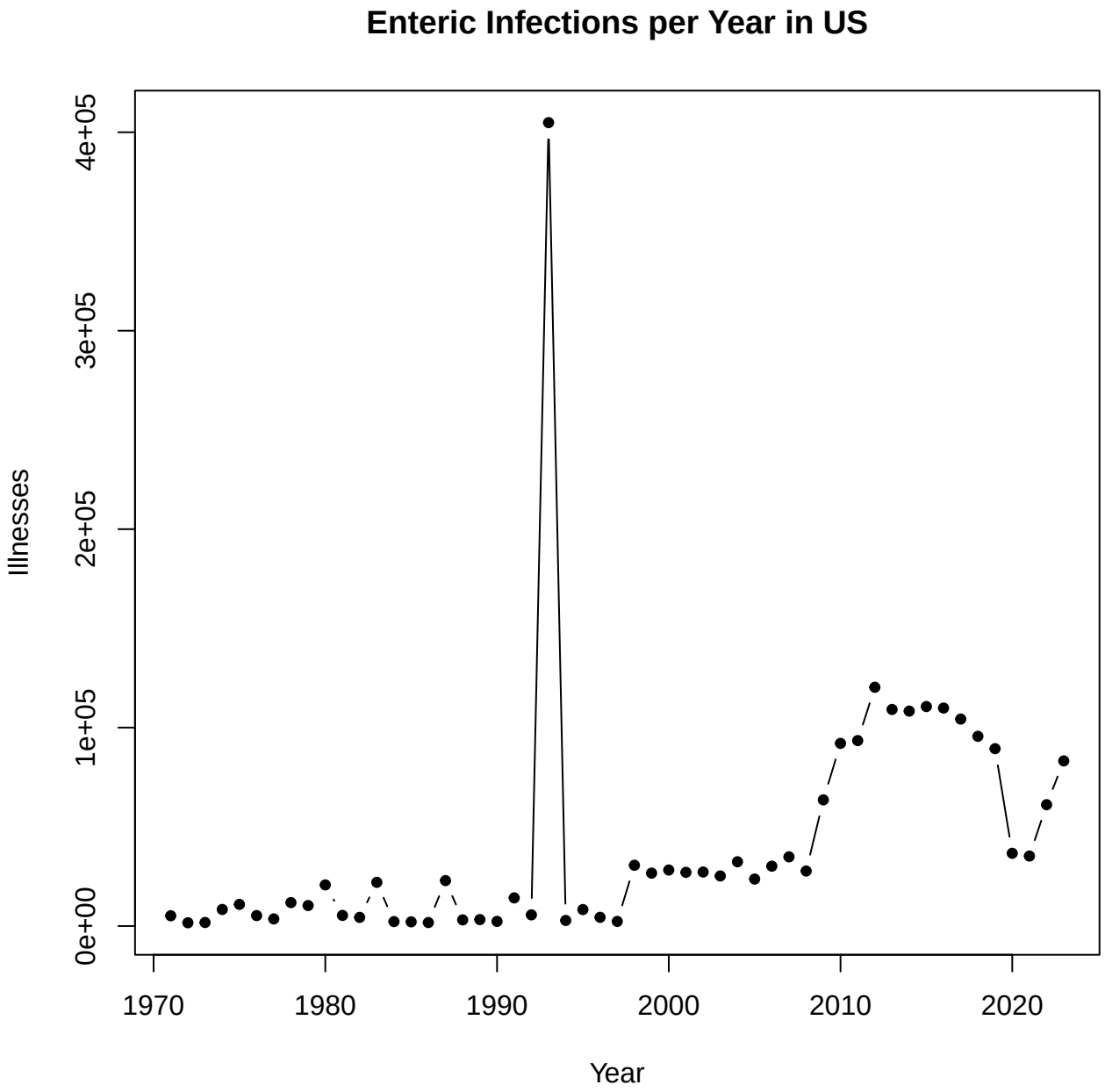
```
     typ = 'b', # make the plot type include both points and lines
     xlab = 'Year', # add x label
     ylab = 'Illnesses', # add y label
     main = 'Enteric Infections per Year in US') # add title
```




We can modify other parts of the plot to make it easier to see. I like the points to filled in. We can do that with the `pch` plotting parameter which gives different point shapes and fillings. I like `pch = 19`. We can also make the plots a bit smaller with the argument `cex`

```
plot(yearly.illnesses$Year, # x-values
     yearly.illnesses$Illnesses, # y-values
     typ = 'b', # make the plot type include both points and lines
     xlab = 'Year', # add x label
     ylab = 'Illnesses', # add y label
     main = 'Enteric Infections per Year in US', # add title
```

```
    cex = 0.75, # Points 90% of their default size
    pch = 19) # Filled circles
```




Something clearly happened in 1993. It says that 40,000 people got sick from food-borne or water-borne illnesses that year. I wonder what happened. Let's see.

```
NORS.df[which(NORS.df$Illnesses == max(NORS.df$Illnesses)), ]
```

```
##       Year Month     State Primary.Mode               Etiology
## 65821 1993     3 Wisconsin        Water Cryptosporidium parvum
##       Serotype.or.Genotype Etiology.Status                Setting Illnesses
## 65821                           Confirmed Community/municipality    403000
##       Hospitalizations Info.On.Hospitalizations Deaths Info.On.Deaths
## 65821               NA                       NA     50             NA
##       Food.Vehicle Food.Contaminated.Ingredient IFSAC.Category Water.Exposure
```

```
## 65821                                                          Drinking water
##       Water.Type Animal.Type
## 65821  Community
```

Hmm. It says there was a *Cryptosporidium parvum* outbreak in Wisconsin that sickened 403,000 people and killed 50 people. A quick google search shows that this indeed happened. The city of Milwaukee's water treatment plant had bad filtration and about a quarter of the 1.6 million residents got sick.

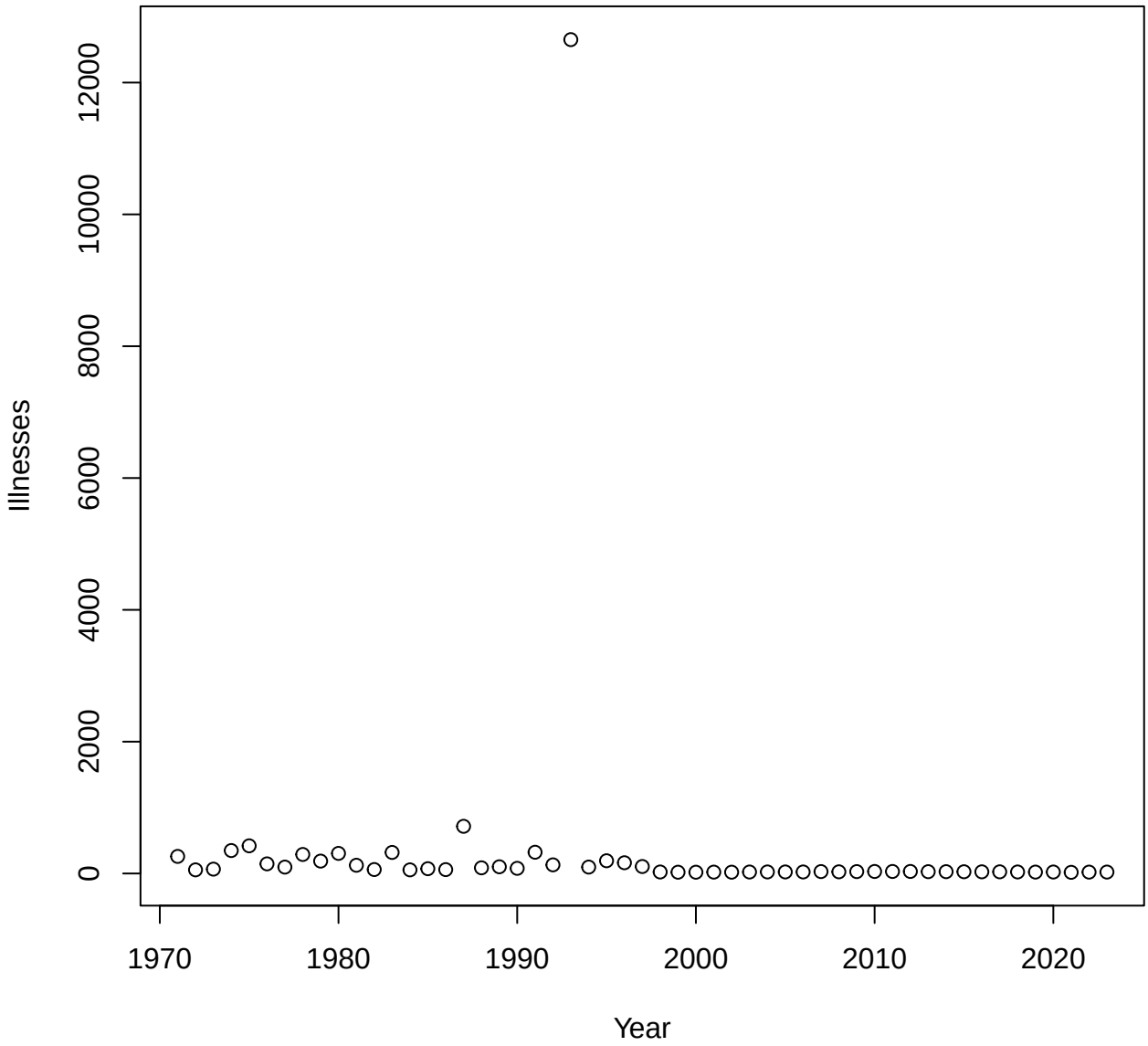
What about the size of the average outbreak over time? Another time when we can use `aggregate`.

```
(mean.outbreak.size <- aggregate(Illnesses ~ Year, NORS.df, mean))
```

```
##    Year   Illnesses
## 1  1971   259.20000
## 2  1972    55.00000
## 3  1973    66.07407
## 4  1974   348.08333
## 5  1975   419.69231
## 6  1976   146.19444
## 7  1977    96.91892
## 8  1978   288.34146
## 9  1979   187.87273
## 10 1980   304.73529
## 11 1981   124.60465
## 12 1982    59.73973
## 13 1983   319.24638
## 14 1984    54.78049
## 15 1985    72.72414
## 16 1986    58.66667
## 17 1987   715.75000
## 18 1988    85.94444
## 19 1989   101.71875
## 20 1990    78.16667
## 21 1991   322.36364
## 22 1992   130.95349
## 23 1993 12651.37500
## 24 1994    97.00000
## 25 1995   193.02326
## 26 1996   162.96296
## 27 1997   105.72727
## 28 1998    22.61743
## 29 1999    19.38299
## 30 2000    19.27099
## 31 2001    20.89884
## 32 2002    19.83965
## 33 2003    22.40195
## 34 2004    23.43271
## 35 2005    23.31496
## 36 2006    23.04427
## 37 2007    28.83127
## 38 2008    25.34004
## 39 2009    28.35950
## 40 2010    30.33718
## 41 2011    30.30253
## 42 2012    30.73384
## 43 2013    27.45509
```

```
## 44 2014    27.53355
## 45 2015    26.24585
## 46 2016    25.36320
## 47 2017    25.14127
## 48 2018    22.92136
## 49 2019    21.28554
## 50 2020    22.32582
## 51 2021    17.47621
## 52 2022    20.76503
## 53 2023    21.65669
```

```
plot(mean.outbreak.size)
```



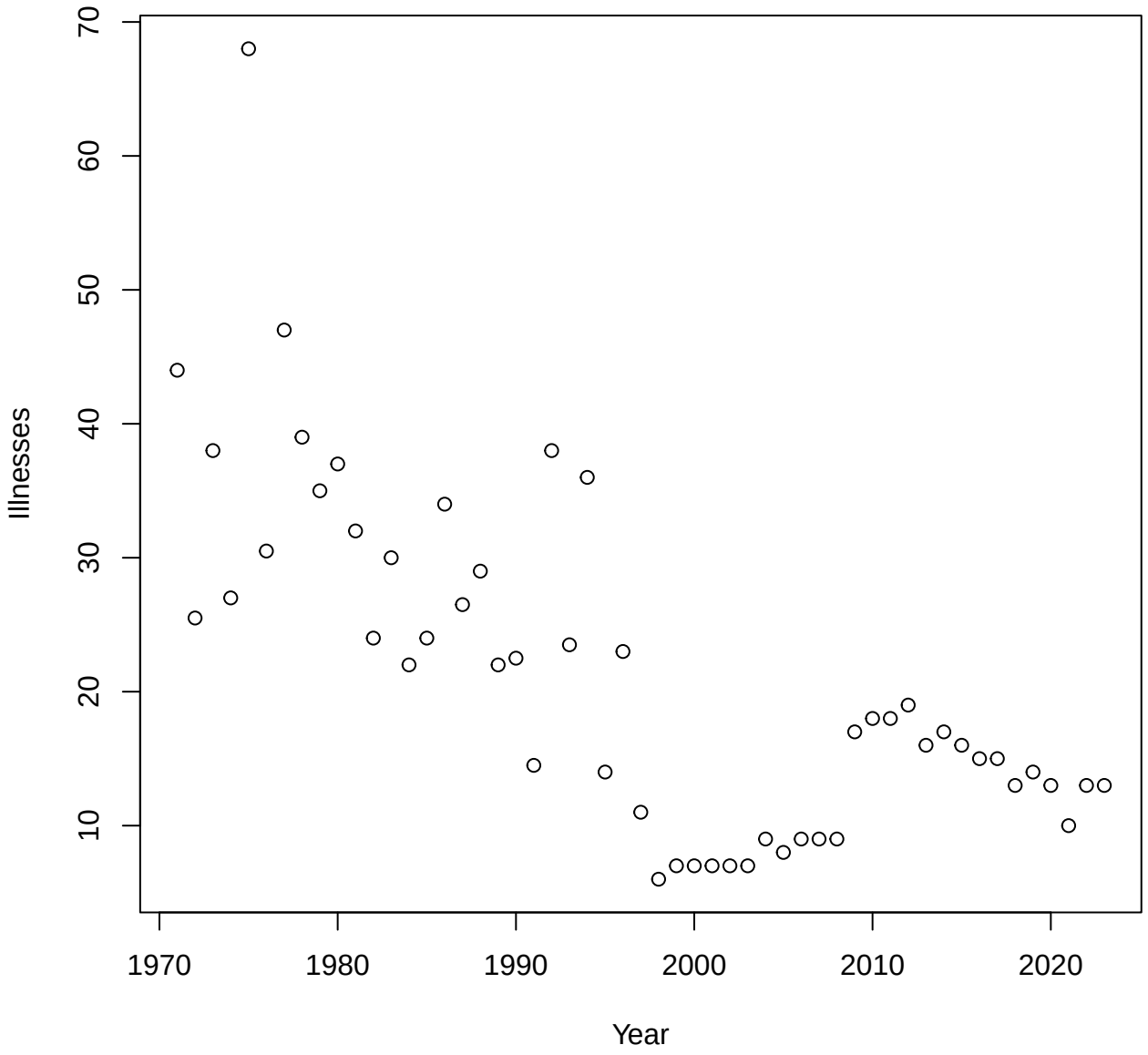

Hmm. That one outbreak in 1993 dominates everything. Here is a good use for the median.

```
(median.outbreak.size <- aggregate(Illnesses ~ Year, NORS.df, median))
```

```
##    Year Illnesses
## 1  1971      44.0
```

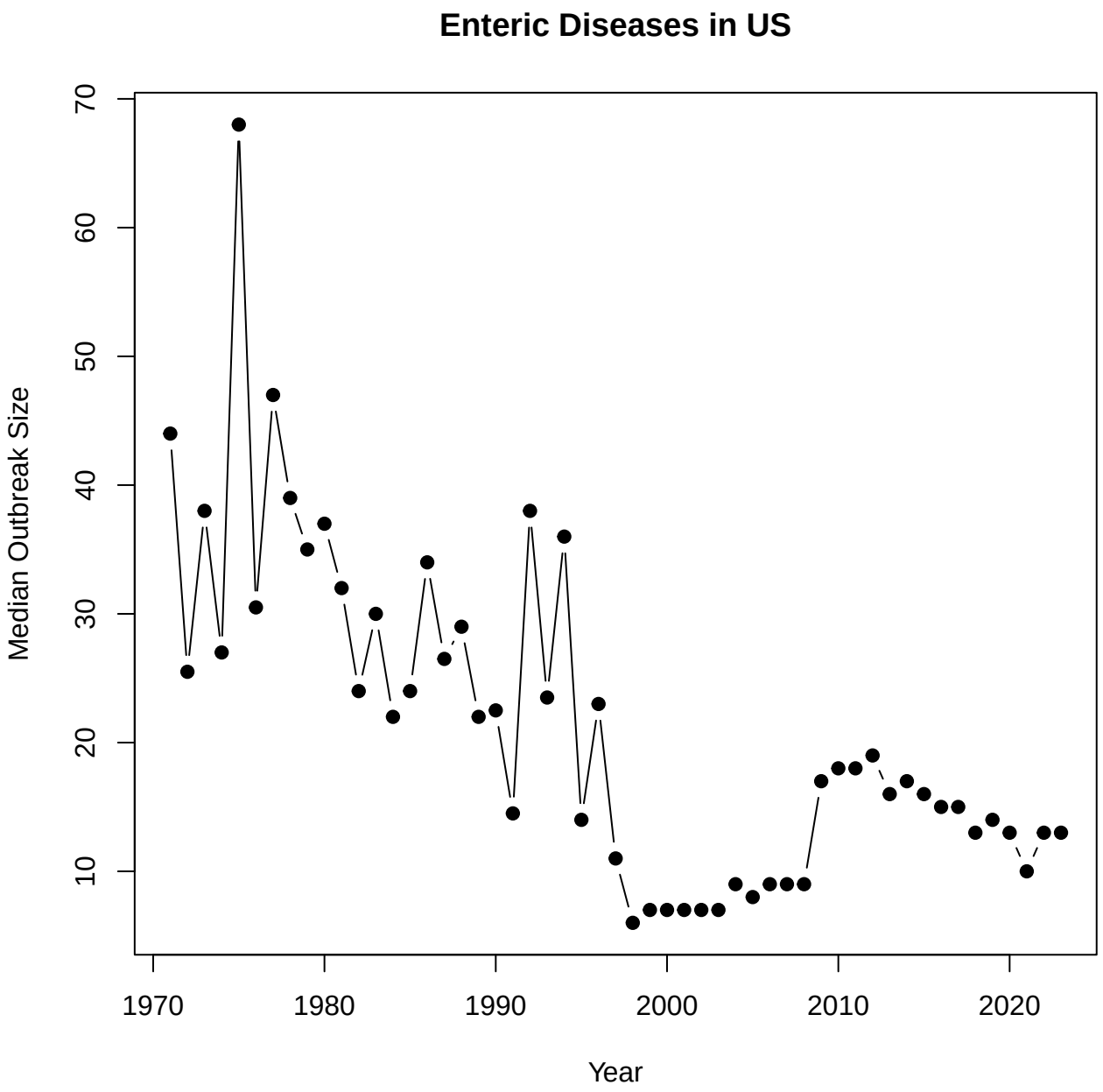
```
## 2  1972      25.5
## 3  1973      38.0
## 4  1974      27.0
## 5  1975      68.0
## 6  1976      30.5
## 7  1977      47.0
## 8  1978      39.0
## 9  1979      35.0
## 10 1980      37.0
## 11 1981      32.0
## 12 1982      24.0
## 13 1983      30.0
## 14 1984      22.0
## 15 1985      24.0
## 16 1986      34.0
## 17 1987      26.5
## 18 1988      29.0
## 19 1989      22.0
## 20 1990      22.5
## 21 1991      14.5
## 22 1992      38.0
## 23 1993      23.5
## 24 1994      36.0
## 25 1995      14.0
## 26 1996      23.0
## 27 1997      11.0
## 28 1998       6.0
## 29 1999       7.0
## 30 2000       7.0
## 31 2001       7.0
## 32 2002       7.0
## 33 2003       7.0
## 34 2004       9.0
## 35 2005       8.0
## 36 2006       9.0
## 37 2007       9.0
## 38 2008       9.0
## 39 2009      17.0
## 40 2010      18.0
## 41 2011      18.0
## 42 2012      19.0
## 43 2013      16.0
## 44 2014      17.0
## 45 2015      16.0
## 46 2016      15.0
## 47 2017      15.0
## 48 2018      13.0
## 49 2019      14.0
## 50 2020      13.0
## 51 2021      10.0
## 52 2022      13.0
## 53 2023      13.0
```

```
plot(median.outbreak.size)
```




Let's make it look nicer.

```
plot(median.outbreak.size, typ = "b", pch = 19, xlab = "Year", ylab = "Median Outbreak Size",
    main = "Enteric Diseases in US")
```




**Checkpoint 8: Make a plot of the median outbreak size in North Carolina over time.**

## 4. Plotting 3 characteristics: Heatmaps and Contour plots

These are different types of plots that can be used to visualize 3 characteristics. These fall into categories of contour plots and heatmaps. The idea is that the x and y locations are the plot have information, but information is also encoded in color (for a heatmap) or encoded in a height, visualized with contours (the lines you see on topographical maps).

It's easiest to illustrate contour plots and heatmaps with topographical data. We will use a built-in dataset from R: the Maunga Whau (Mt Eden) Volcano in Auckland, New Zealand.

```
# load the data
data(volcano)
```

```r
# the contour plot
contour(volcano)
```

```r
# the filled contour plot
filled.contour(volcano)
```

```r
# the basic heatmap
image(volcano)
```

The basic heatmap can be done with the function `image` in R, but it is not very ideal. A better function comes with the `fields` package. First, you have to install a package into R. Then, you have to tell R to load the library for that package. Once a package is installed, you don't need to install it again (unless you update R, which is why it is good to have a logical test in your code like the one I have below). However, you do need to load the library every time you want to use it.

```r
# This line of code does a logical test to see you have have a package
# installed, and if you don't, it installs it:
if (("fields" %in% installed.packages()) == FALSE) {
    install.packages("fields", dependencies = TRUE)
}

# load library
library(fields)
```

```
## Loading required package: spam
```

```
## Spam version 2.10-0 (2023-10-23) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.
```

```
##
## Attaching package: 'spam'
```

```
## The following objects are masked from 'package:base':
##
##     backsolve, forwardsolve
```

```
## Loading required package: viridisLite
```

```
##
## Try help(fields) to get started.
```

```r
# In Rstudio, if you click on 'Global Environment' under the 'Environment' tab,
# you can see that the package 'fields' is now loaded.

# Notice that this function automatically plots a legend, and has a better
# color scheme:
image.plot(volcano)
```

For more ways to plot a volcano: http://cran.r-project.org/web/packages/plot3D/vignettes/volcano.pdf

## 5. Saving Plots as Output to .pdf

Wow I really like that volcano plot! I'd like to include it in my publication. It is easy to save pub-quality images in R, although sometimes you have to play around to get the size and margins right.

To save an image in R, you have to open a "device" (in this case a .pdf), write the (several lines of) code to make the plot, and then close the device.

```
# let's read about the pdf function
`?`(pdf)
```

```
# open the device using the pdf command
pdf(file = "MyVolcanoPlot.pdf", width = 7, height = 5)
image.plot(volcano)  # in this case we only have one line of code

# close the device
dev.off()
```

```
## pdf
##   2
```

**Checkpoint 9: Construct a good time series of the median enteric disease outbreak size in New York over time. Save the figure as a .pdf.**