Problem set 1: Worked solutions

Statistics and statistical programming
Northwestern University
MTS 525

Aaron Shaw

September 28, 2020

Contents

PC 1: Access and describe a dataset provided in an R library

PC1.1

```
## This first command can install the pacakge. I've commented it out here because it only needs to run
## You can also install packages from the 'Packages' tab of the Rstudio interface.
##
## install.packages("openintro")

library(openintro)
```

```
## Loading required package: airports

## Loading required package: cherryblossom

## Loading required package: usdata
```

```
data("county")
```

PC1.2

```
class(county)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

Notice that the results of that last call to `class()` produced something you might not have expected: three different values! This is because the OpenIntro authors have provided this dataset as something called a "tibble" (that's the `tbl_df` and `tbl` bits) which is basically a souped-up Tidyverse analogue to a Base-R dataframe. You can do some nice things with Tidyverse tibbles. For example, you can invoke them directly and not worry that R is going to print out a massive amonut of data at your console. The output also includes information about the dimensions and the types/classes of the columns/variables in the Tibble. Here's what it looks like:

```
county
```

```
## # A tibble: 3,142 x 15
##    name  state pop2000 pop2010 pop2017 pop_change poverty homeownership
##    <chr> <fct>   <dbl>   <dbl>   <int>      <dbl>   <dbl>         <dbl>
##  1 Auta~ Alab~   43671   54571   55504       1.48    13.7          77.5
##  2 Bald~ Alab~  140415  182265  212628       9.19    11.8          76.7
##  3 Barb~ Alab~   29038   27457   25270      -6.22    27.2          68
##  4 Bibb~ Alab~   20826   22915   22668       0.73    15.2          82.9
##  5 Blou~ Alab~   51024   57322   58013       0.68    15.6          82
##  6 Bull~ Alab~   11714   10914   10309      -2.28    28.5          76.9
##  7 Butl~ Alab~   21399   20947   19825      -2.69    24.4          69
##  8 Calh~ Alab~  112249  118572  114728      -1.51    18.6          70.7
##  9 Cham~ Alab~   36583   34215   33713      -1.2     18.8          71.4
## 10 Cher~ Alab~   23988   25989   25857      -0.6     16.1          77.5
## # ... with 3,132 more rows, and 7 more variables: multi_unit <dbl>,
## #   unemployment_rate <dbl>, metro <fct>, median_edu <fct>,
## #   per_capita_income <dbl>, median_hh_income <int>, smoking_ban <fct>
```

Notice that the Tibble refers to some numeric variables as "" ("doubles") and others as "<int"> ("integers"). The latter maps to the colloquial idea of an integer. A "double" is a programming-language speak for a variable that takes non-integer numeric values.

The results of this call are actually sufficient to answer PC1.3 and PC1.4 below.

PC1.3

If you didn't know or didn't realize that calling a Tibble directly answered this, here are some other ways to find the dimensions of a dataset:

```
dim(county)
```

```
## [1] 3142   15
```

```
nrow(county)
```

```
## [1] 3142
```

```
ncol(county)
```

```
## [1] 15
```

PC1.4

Again, some additional tools/approaches you might use to answer this if the Tibble method isn't available/known to you. Note that you can find out the class for any one variable easily with the `class()` command. Iterating this over the names of all the variables in a dataframe is feasible, but tedious and inefficient, so I provide a more concise method with `lapply()` below:

```
names(county)
```

```
##  [1] "name"              "state"             "pop2000"          
##  [4] "pop2010"           "pop2017"           "pop_change"       
##  [7] "poverty"           "homeownership"     "multi_unit"       
## [10] "unemployment_rate" "metro"             "median_edu"       
## [13] "per_capita_income" "median_hh_income"  "smoking_ban"
```

```
## just an example here:
class(county$poverty)
```

```
## [1] "numeric"
```

```
## lapply() is useful for doing this over all variables in a dataframe/tibble
lapply(county, class)
```

```
## $name
## [1] "character"
## 
## $state
## [1] "factor"
## 
## $pop2000
## [1] "numeric"
## 
## $pop2010
## [1] "numeric"
## 
## $pop2017
## [1] "integer"
## 
## $pop_change
## [1] "numeric"
## 
## $poverty
## [1] "numeric"
## 
## $homeownership
## [1] "numeric"
## 
## $multi_unit
## [1] "numeric"
## 
## $unemployment_rate
## [1] "numeric"
## 
## $metro
## [1] "factor"
## 
## $median_edu
## [1] "factor"
##
## $per_capita_income
## [1] "numeric"
##
## $median_hh_income
## [1] "integer"
##
## $smoking_ban
## [1] "factor"
```

PC1.5

For my example, I'll work with the `poverty` variable:

```
length(county$poverty)
```

```
## [1] 3142
```

```
min(county$poverty, na.rm=TRUE) ## That na.rm=TRUE part is crucial!
```

```
## [1] 2.4
```

```
max(county$poverty, na.rm=TRUE)
```

```
## [1] 52
```

```
mean(county$poverty, na.rm=TRUE) ## So many significant digits...
```

```
## [1] 15.96885
```

```
sd(county$poverty, na.rm=TRUE)
```

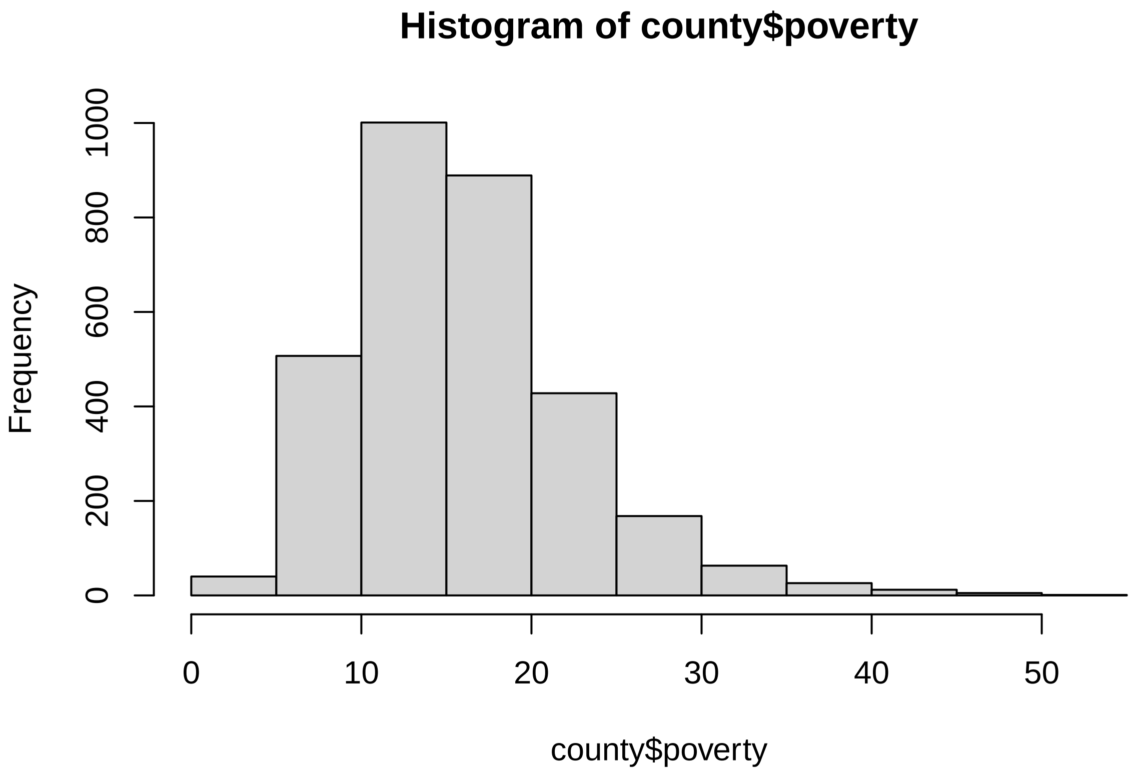
```
## [1] 6.515682
```

```
## And here's a built-in command that covers many of these:
summary(county$poverty)
```

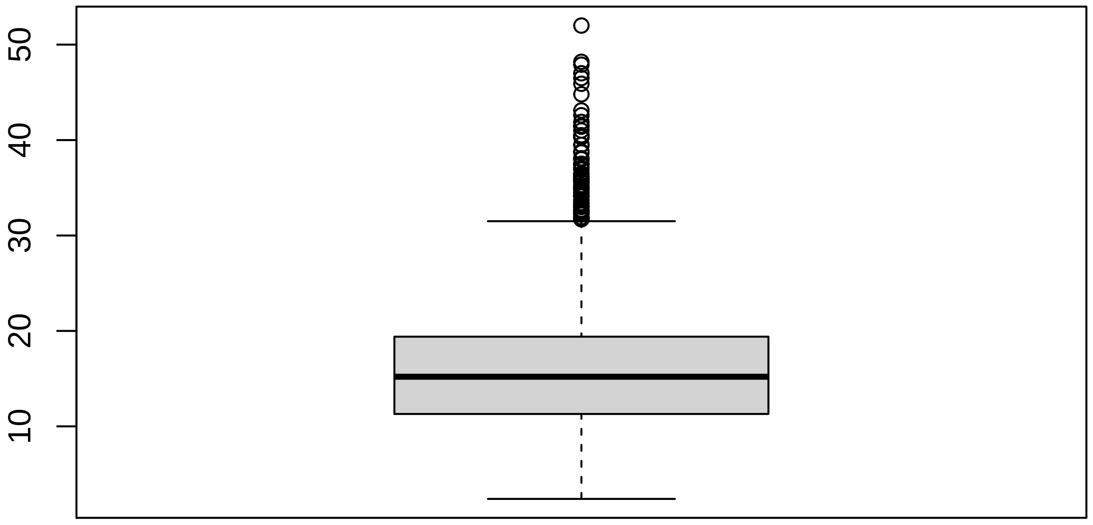
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    2.40   11.30   15.20   15.97   19.40   52.00       2
```

PC1.6

```
hist(county$poverty)
```



```
boxplot(county$poverty)
```



PC1.7

Because each row in the dataset corresponds to a county, the following will summarize the number of counties per state in the data

```
table(county$state)
```

```
##
##          Alabama          Alaska          Arizona
##               67              29               15
##         Arkansas      California         Colorado
##                   75                   58                   64
##          Connecticut             Delaware District of Columbia
##                    8                    3                    1
##              Florida              Georgia               Hawaii
##                   67                  159                    5
##                Idaho             Illinois              Indiana
##                   44                  102                   92
##                 Iowa               Kansas             Kentucky
##                   99                  105                  120
##            Louisiana                Maine             Maryland
##                   64                   16                   24
##        Massachusetts             Michigan            Minnesota
##                   14                   83                   87
##          Mississippi             Missouri              Montana
##                   82                  115                   56
##             Nebraska               Nevada        New Hampshire
##                   93                   17                   10
##           New Jersey           New Mexico             New York
##                   21                   33                   62
##       North Carolina         North Dakota                 Ohio
##                  100                   53                   88
##             Oklahoma               Oregon         Pennsylvania
##                   77                   36                   67
##         Rhode Island       South Carolina         South Dakota
##                    5                   46                   66
##            Tennessee                Texas                 Utah
##                   95                  254                   29
##              Vermont             Virginia           Washington
##                   14                  133                   39
##        West Virginia            Wisconsin              Wyoming
##                   55                   72                   23
```

PC2. Working with a dataset from the web

PC2.1

```r
set.seed(220920) ## A recent date.
sample(x=c(1:20), size=1)
```

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

My dataset number for the rest of this worked solution to the programming challenge is 3. That said, the code should work if you substitute an appropriate group number into the url below.

PC2.2

I'll do this by pointing RStudio directly to the URL for the data using the `url()` and `load()` commands.

```r
load(url("https://communitydata.science/~ads/teaching/2020/stats/data/week_03/group_03.RData"))

ls() # shows me what's available. The counties dataset is still hanging around.
```

```
## [1] "county" "d"
```

```r
head(d)
```

```
## [1]   5.0165660   0.7734509  24.9639274   1.7455026   1.1062885 454.2820677
```

Note: If you downloaded the file, you'll need to point R's `load()` command at the correct file location on your machine. RStudio can be…picky about this sort of thing.

Also, a clarifying point: if you're compiling your own RMarkdown scripts, you will need to load the dataset explicitly (not just open the file with RStudio). When RMarkdown tries to "knit" the .Rmd file into HTML or whatever, it is as if you are running the entire contents of the .Rmd script in an entirely new RStudio environment. This means if you don't load something explicitly within the .Rmd script RStudio will not know where to find it.

PC2.3

I'll do all the ones I asked for as well as some I didn't (just so you have an example what the commands look like)

```
min(d)
```

```
## [1] -42.34089
```

```
max(d)
```

```
## [1] 33720.15
```

```
mean(d)
```

```
## [1] 831.6718
```

```
sd(d)
```

```
## [1] 3775.512
```

```
## extras
median(d)
```

```
## [1] 5.183996
```

```
var(d) ## variance. compare to the standard deviation
```

```
## [1] 14254490
```

```
IQR(d) ## interquartile range
```

```
## [1] 76.51708
```

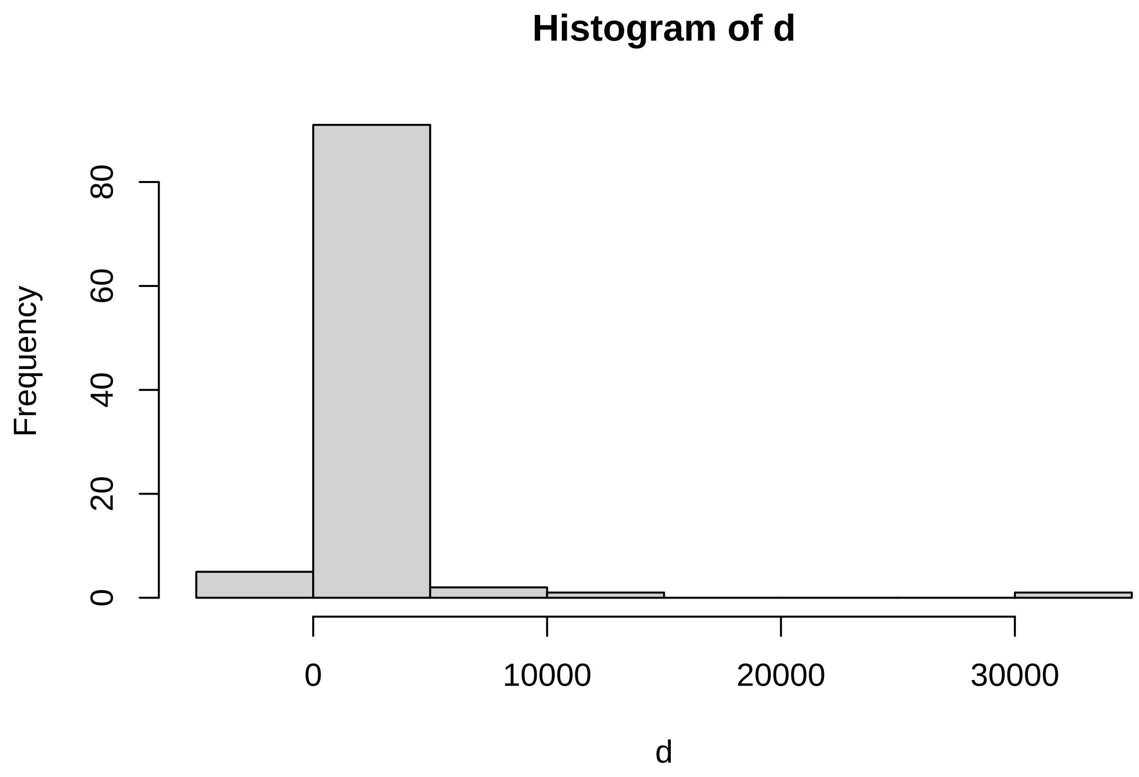
```
## A handy Base-R command that does a bunch of these at once:
summary(d)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##   -42.34     0.71     5.18   831.67    77.22 33720.15
```

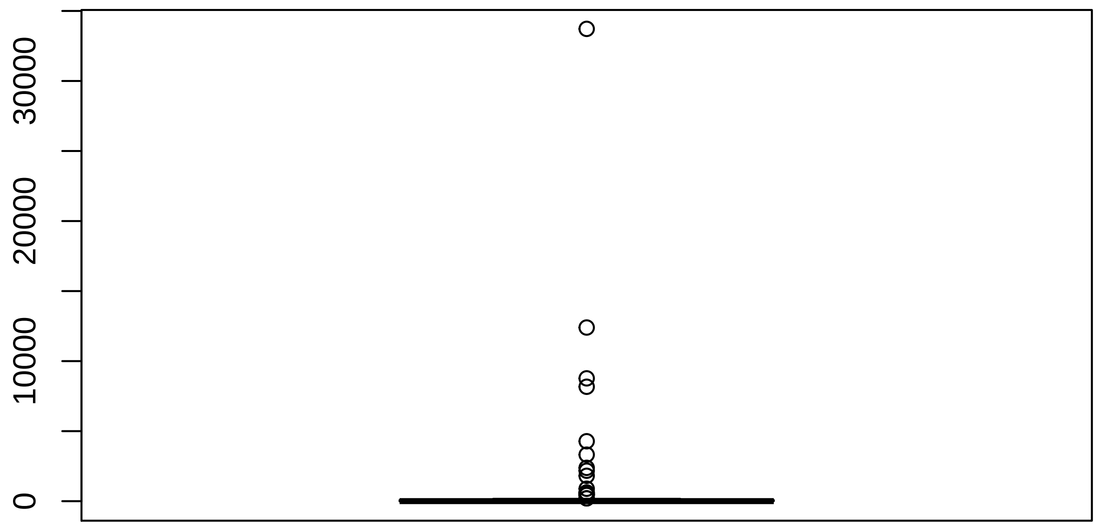
PC2.4

Graphing using R's built-in functions:

```
hist(d)
```



```r
boxplot(d)
```



PC2.5

```r
d[d < 0] <- NA
```

Fwiw, I want to point out that this last line of code is a little “fast and loose.” I have reused my original variable name when I recode the negative values to `NA`. This means that my copy of `d` has been changed and the only way I can restore it is by reading in the raw data again (which would, in turn, overwrite my recoded copy). It is often a good idea to check your work after recoding. You could do that here by “stashing” the original variable in a new object/name (e.g., run something like `d.orig <- d` before running the line above). Then after you’ve recoded the negative values, you could run whatever tests/comparisons you like to make sure the recoding is correct before deleting the original data (which can be done with `rm(d.orig)`).

Onwards to the new mean and standard deviation:

```
mean(d, na.rm=T) # R can understand "T" in place of TRUE
```

```
## [1] 876.5269
```

```
sd(d, na.rm=T)
```

```
## [1] 3869.37
```

Both values here are larger than before I removed my negative values. This is pretty intuitive in the case of the mean, but maybe less so with the standard deviation since the range of the values in the recoded variable is smaller. Can you explain how the standard deviation would increase?

PC2.6

```
d.log <- log1p(d) # Note: I use log1p() because some values are very close to zero

mean(d.log, na.rm=T)
```

```
## [1] 2.845581
```

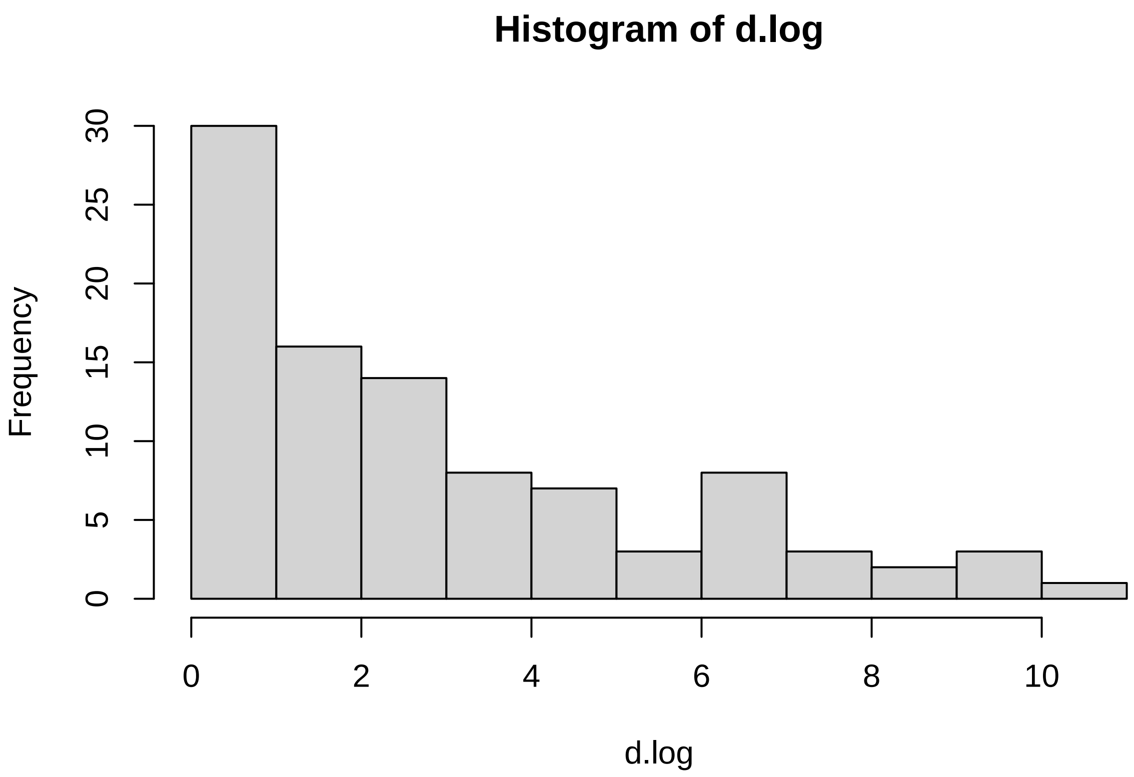
```
median(d.log, na.rm=T)
```

```
## [1] 2.098321
```

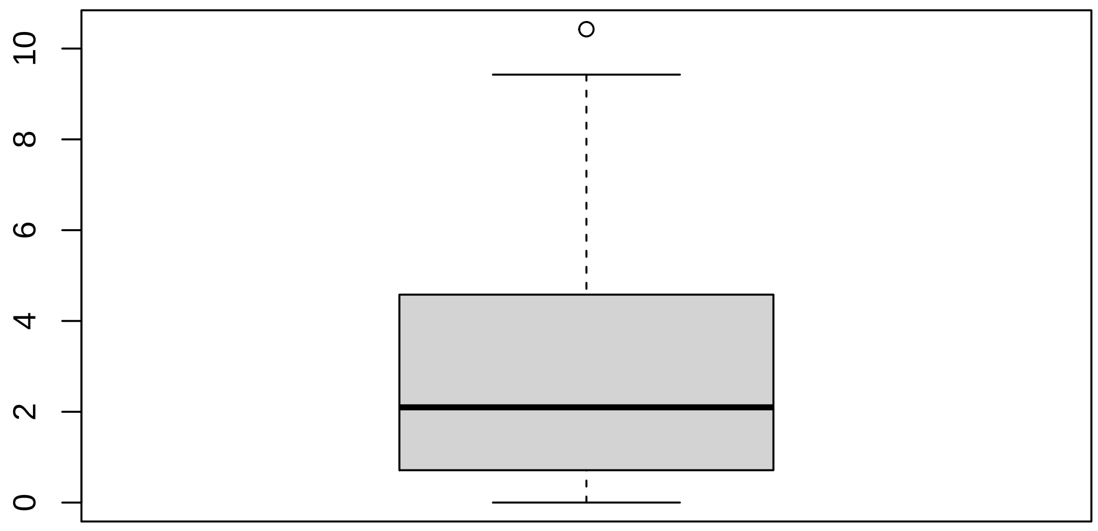
```
sd(d.log, na.rm=T)
```

```
## [1] 2.667537
```

```
hist(d.log)
```



```
boxplot(d.log)
```



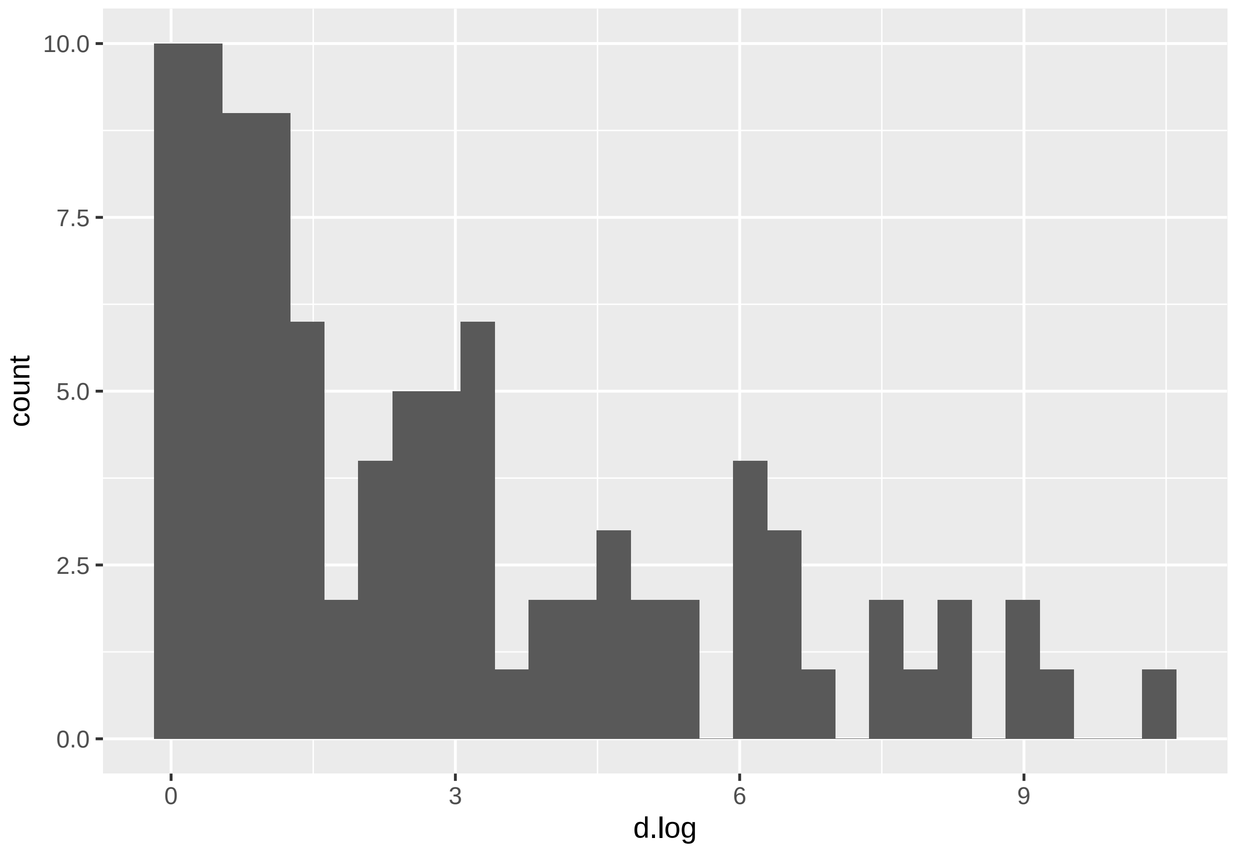
Next week, I will introduce the `ggplot2` package in the R tutorial. For now, I'll reproduce the historgram and boxplot I made above in ggplot2 and leave these examples here for future reference. Graphing a single vector in ggplot2 is a bit weird, so this looks a little different than the examples from the R lecture:

```r
## if you haven't already you'll need to make sure ggplot2 is installed
## install.packages("ggplot2")

library(ggplot2)
p <- ggplot() + aes(d.log)
p + geom_histogram()
```

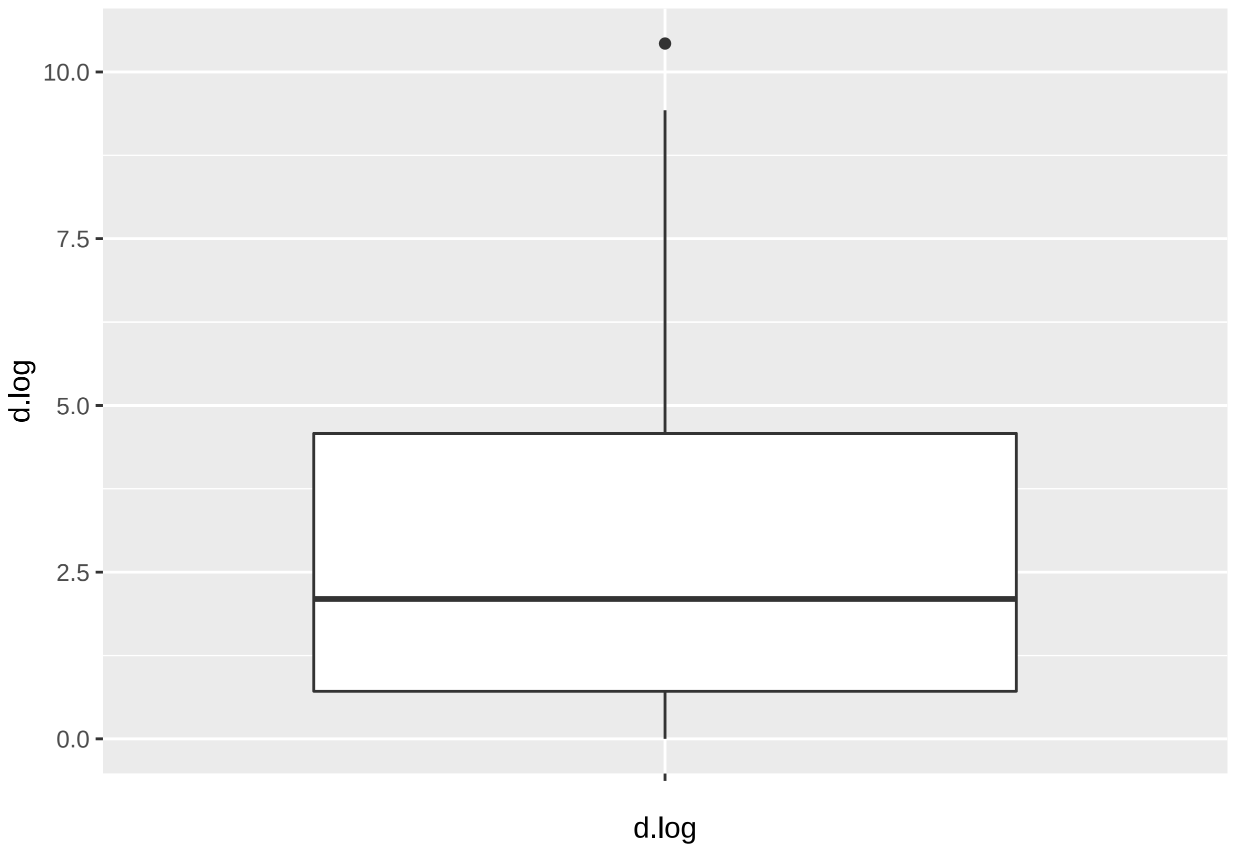
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 5 rows containing non-finite values (stat_bin).
```



```
p + geom_boxplot(aes(x="", y=d.log))
```

```
## Warning: Removed 5 rows containing non-finite values (stat_boxplot).
```



Note that ggplot2 generates a warning about 5 "non-fininte values." In this case, that is due to the 5 `NA` values I created when I recoded the negatives, so I don't need to worry about the warning.

Statistical questions

SQ1

A compelling answer to this depends on the variable you chose. For the one I looked at in my example code (`poverty`) the data is somewhat right skewed, but not much. In this case, the mean and standard deviation should represent the central tendency and spread of the variable pretty well. If your variable was different (e.g., one of the population or income measures), it would probably be good to also examine and report the median and interquartile range. See `OpenIntro` chapter 2 for more on distinctions/reasons behind this.

SQ2

Definitely not. The data is very skewed with a long (positive) tail, suggesting that median and interquartile range would be better measures to report.

SQ3

The cleaned and log-transformed version of the variable has no negative values and is far less right-skewed than the original. You should absolutely prefer the cleaned variable with no negative values since they resulted from a coding error (see the text of PC2.5). However, a preference for the log-transformed or untransformed variable likely depends on the situation. It's almost always good to report descriptive statistics for "raw," untransformed versions of your variables. However, you might prefer the log-transformed version for developing a stronger set of intuitions about your data, comparing with prior work that uses log-transformed versions of a similar measure, or conducting certain statistical tests (we'll talk more about this later in the course).