Lab 1: Data Manipulation, Random Number Generation

July 7, 2020

Today's agenda: Manipulating data objects; using the built-in functions, doing numerical calculations, and basic plots; reinforcing core probabilistic ideas.

0. Open a new R Markdown file; set the output to HTML mode and "Knit". This should produce a web page with the knitting procedure executing your code blocks. You can edit this new file to produce your homework submission.

Background

The exponential distribution is defined by its cumulative distribution function

$$F(x) = 1 - e^{-\lambda x}$$

The R function `rexp` generates random variables with an exponential distribution.

```
rexp(n=10, rate=5)
```

```
##  [1] 0.38340462 0.03866304 0.24814625 0.44634820 0.20847509 0.33775319
##  [7] 0.04744719 0.19942565 0.05813347 0.14102286
```

produces 10 exponentially-distributed numbers with rate (λ) of 5. If the second argument is omitted, the default rate is 1; this is the `standard exponential distribution`.

Part I

1. Generate 200 random values from the standard exponential distribution and store them in a vector `exp.draws.1`. Find the mean and standard deviation of `exp.draws.1`.

```
exp.draws.1 <- rexp(200)
mean(exp.draws.1)
```

```
## [1] 0.9239318
```

```
sd(exp.draws.1)
```

```
## [1] 0.8429488
```

2. Repeat, but change the rate to 0.1, 0.5, 5 and 10, storing the results in vectors called `exp.draws.0.1`, `exp.draws.0.5`, `exp.draws.5` and `exp.draws.10`.

```
exp.draws.0.1 <- rexp(200, rate = 0.1)
mean(exp.draws.0.1)
```

```
## [1] 10.71301
```

```
sd(exp.draws.0.1)
```

```
## [1] 10.48842
```

```r
exp.draws.0.5 <- rexp(200, rate = 0.5)
mean(exp.draws.0.5)
```

```
## [1] 2.268291
```

```r
sd(exp.draws.0.5)
```

```
## [1] 2.114471
```

```r
exp.draws.5 <- rexp(200, rate = 5)
mean(exp.draws.5)
```

```
## [1] 0.1956412
```

```r
sd(exp.draws.5)
```

```
## [1] 0.1952145
```

```r
exp.draws.10 <- rexp(200, rate = 10)
mean(exp.draws.10)
```

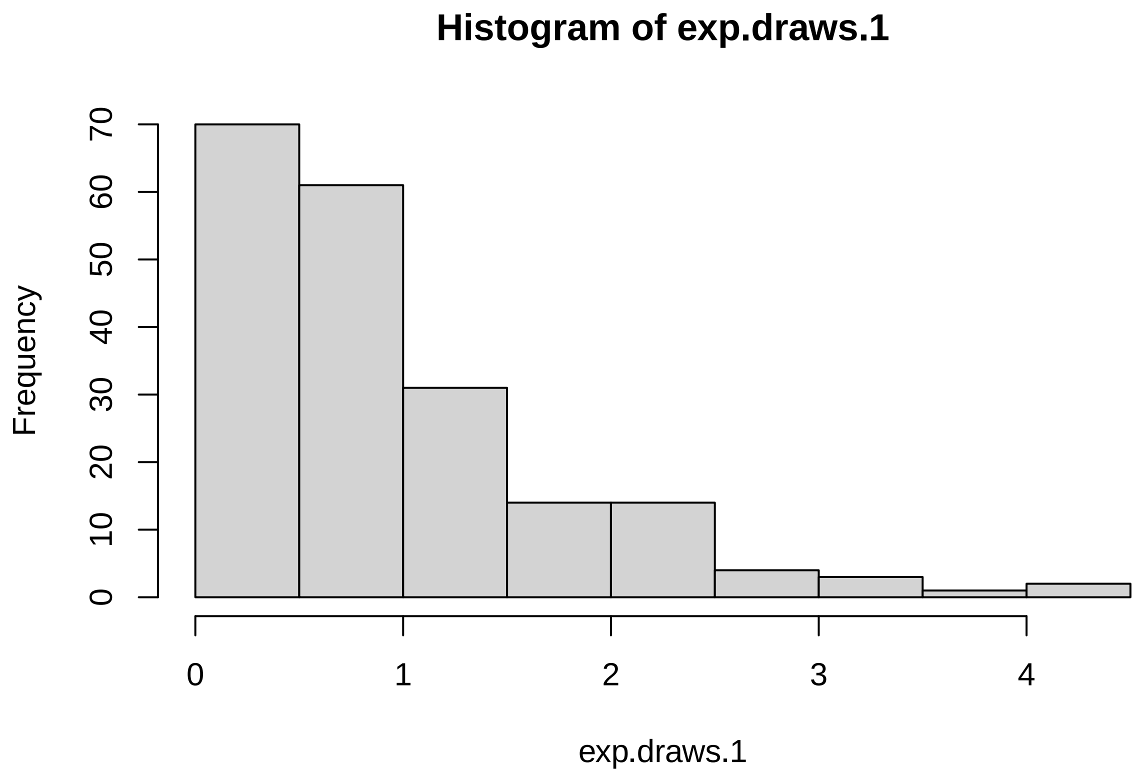
```
## [1] 0.09755564
```

```r
sd(exp.draws.10)
```

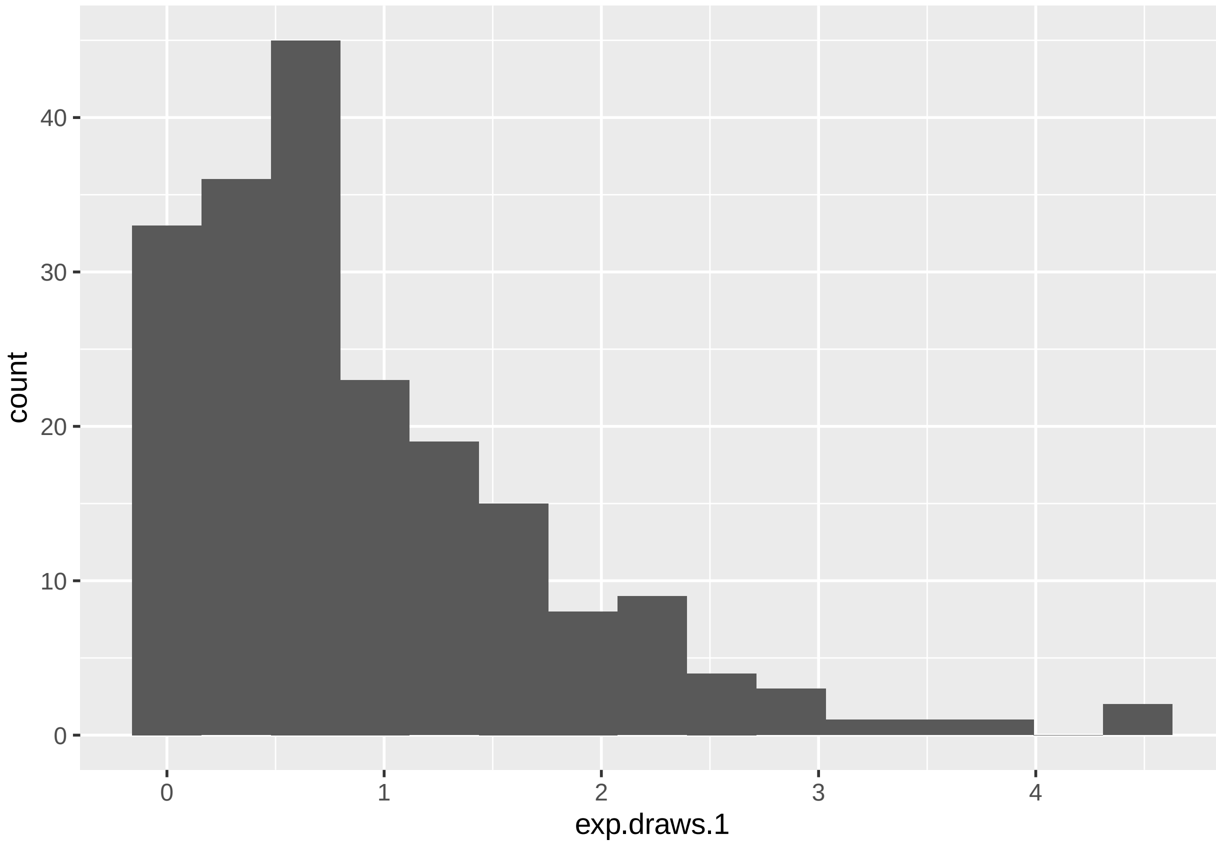
```
## [1] 0.09568435
```

3. The function `plot()` is the generic function in R for the visual display of data. `hist()` is a function that takes in and bins data as a side effect. To use this function, we must first specify what we'd like to plot.
    a. Use the `hist()` function to produce a histogram of your standard exponential distribution.

```r
hist(exp.draws.1) # or use ggplot2
```

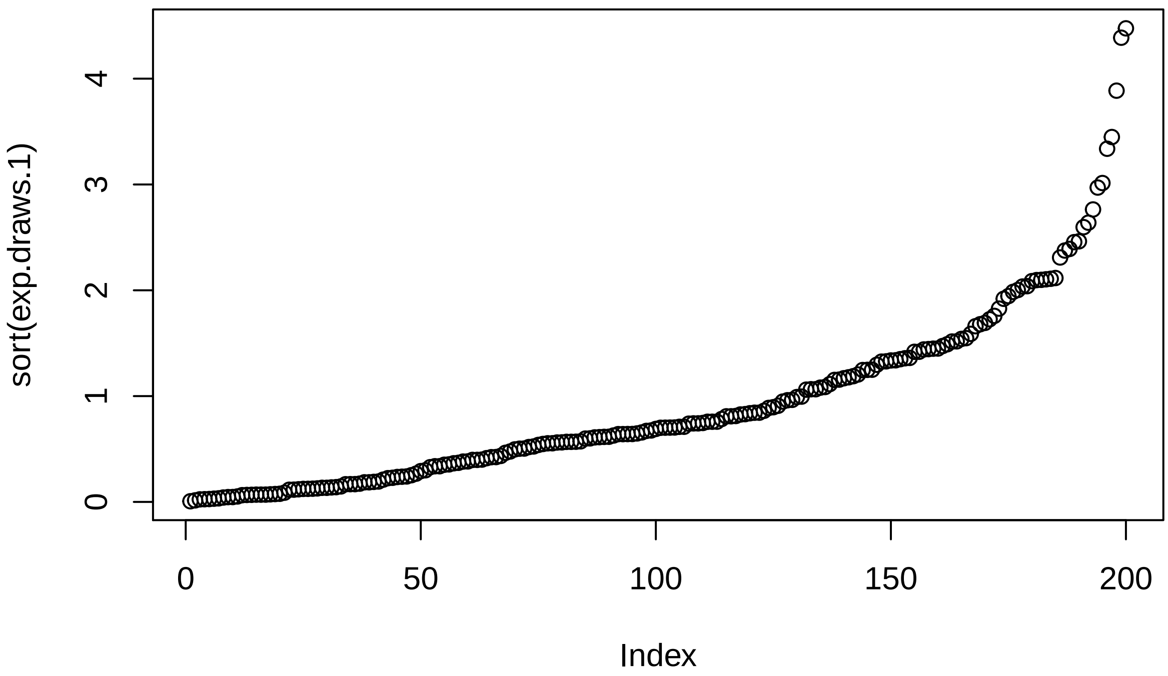



```
tibble(expdraw = exp.draws.1) %>% ggplot(aes(x = exp.draws.1))+
  geom_histogram(bins = 15)
```

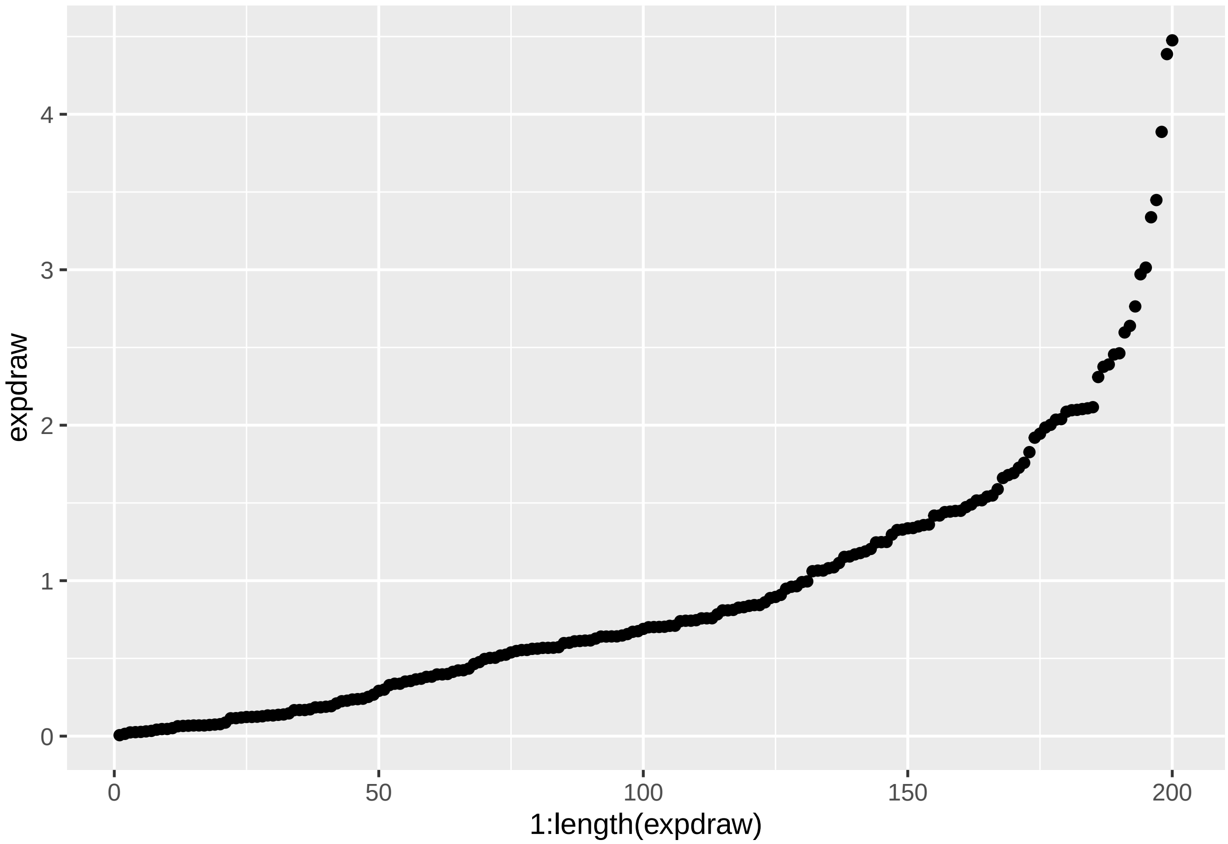



```
b. Use `plot()` with this vector to display the random values from your standard distribution in order.
```

```r
plot(sort(exp.draws.1)) # or use ggplot2
```

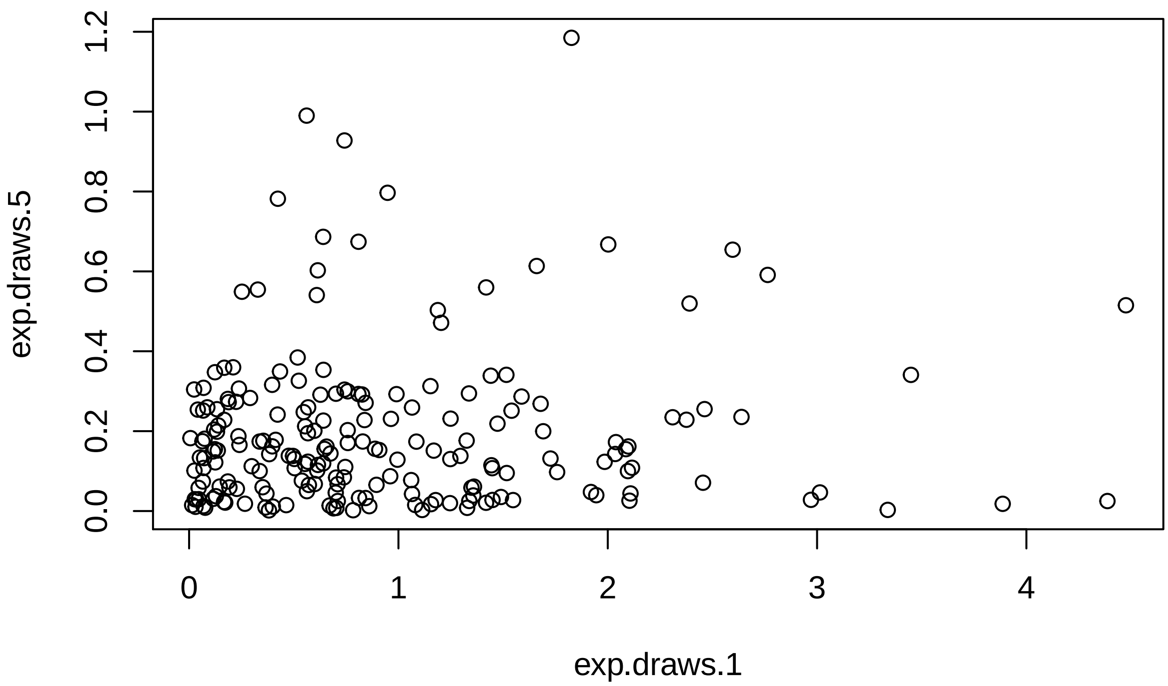



```r
tibble(expdraw = exp.draws.1) %>% arrange(expdraw) %>%
  ggplot(aes(x = 1:length(expdraw), y = expdraw))+
  geom_point()
```

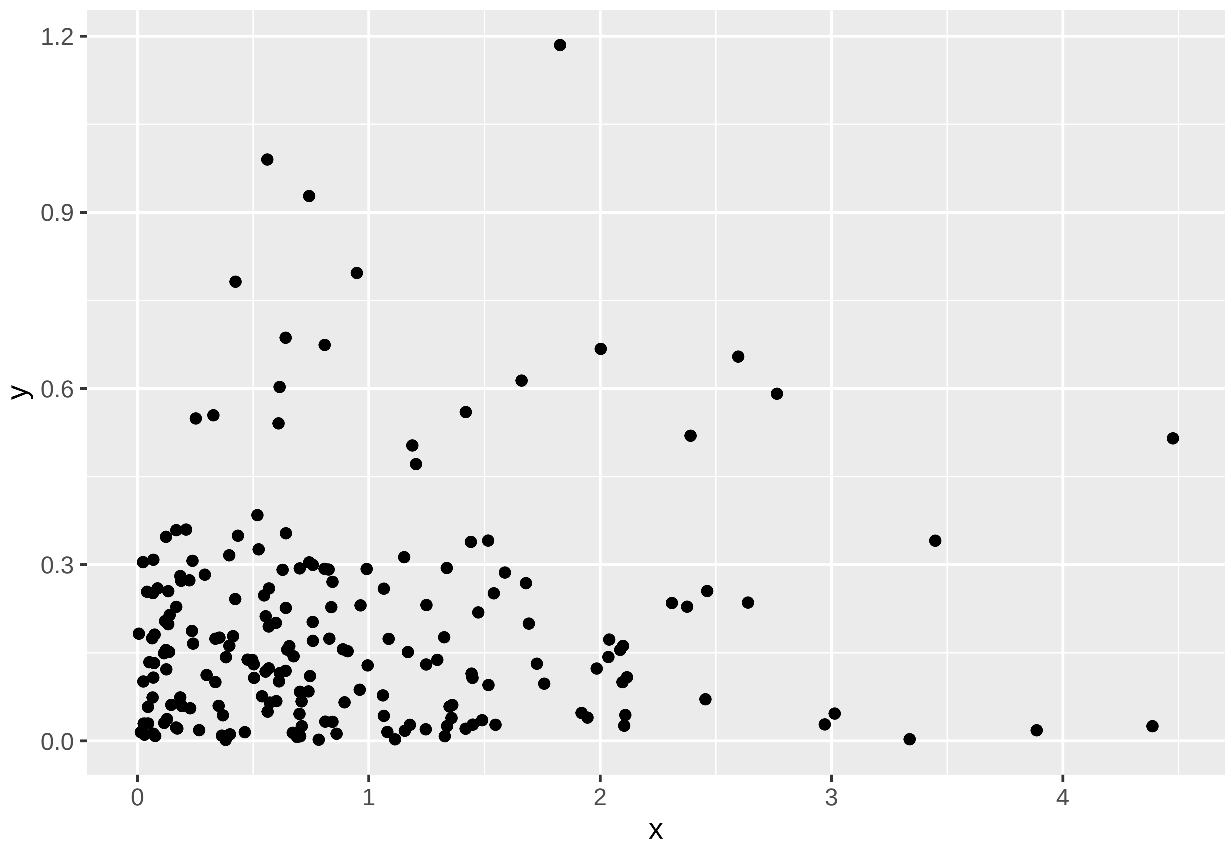



```
c. Now, use `plot()` with two arguments -- any two of your other stored random value vectors -- to creat
```

```r
plot(exp.draws.1, exp.draws.5) # or use ggplot2
```




```
tibble(x = exp.draws.1, y = exp.draws.5) %>%
  ggplot(aes(x = x, y = y))+
  geom_point()
```

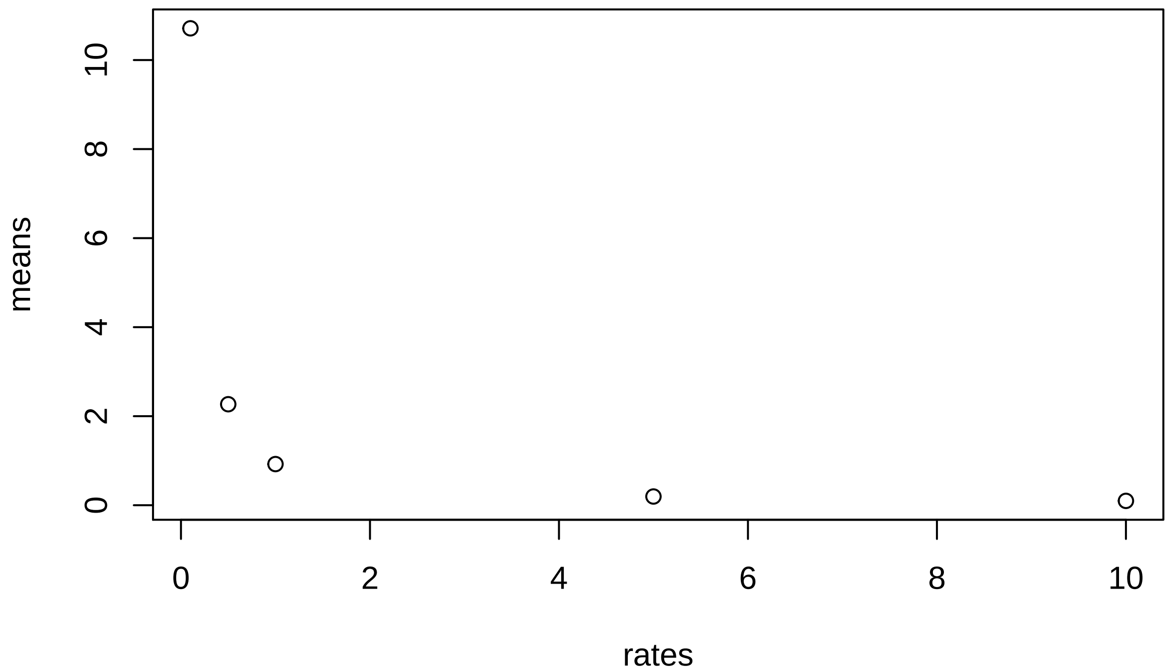


4. We'd now like to compare the properties of each of our vectors. Begin by creating a vector of the means of each of our five distributions in the order we created them and saving this to a variable name of your choice. Using this and other similar vectors, create the following scatterplots:
   a. The five means versus the five rates used to generate the distribution.

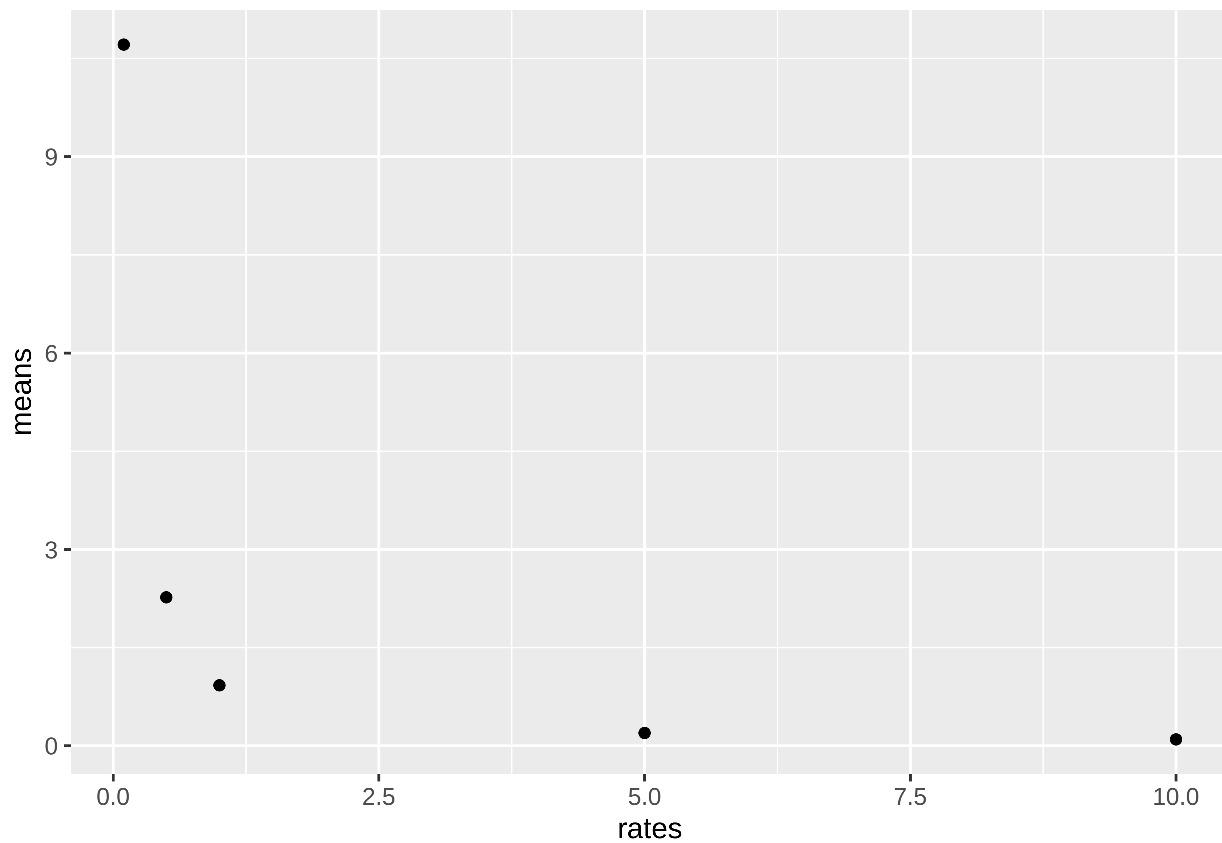
```
exprn <- tibble(exp.1 = exp.draws.1, exp.0.1 = exp.draws.0.1, exp.0.5 = exp.draws.0.5, exp.5 = exp.draws
emeans <- tibble(rates = c(1, 0.1, 0.5, 5, 10), means = colMeans(exprn), stds = apply(exprn, 2, sd))
rord <- order(emeans$rates)
emeans <- emeans[rord,]
# after chapter 4
emeans1 <- exprn %>% gather(key = 'draws', value = 'value') %>%
  group_by(draws) %>% summarize(means = mean(value), stds = sd(value)) %>% arrange(desc(means)) %>% ung
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
plot(means ~ rates, data = emeans)
```

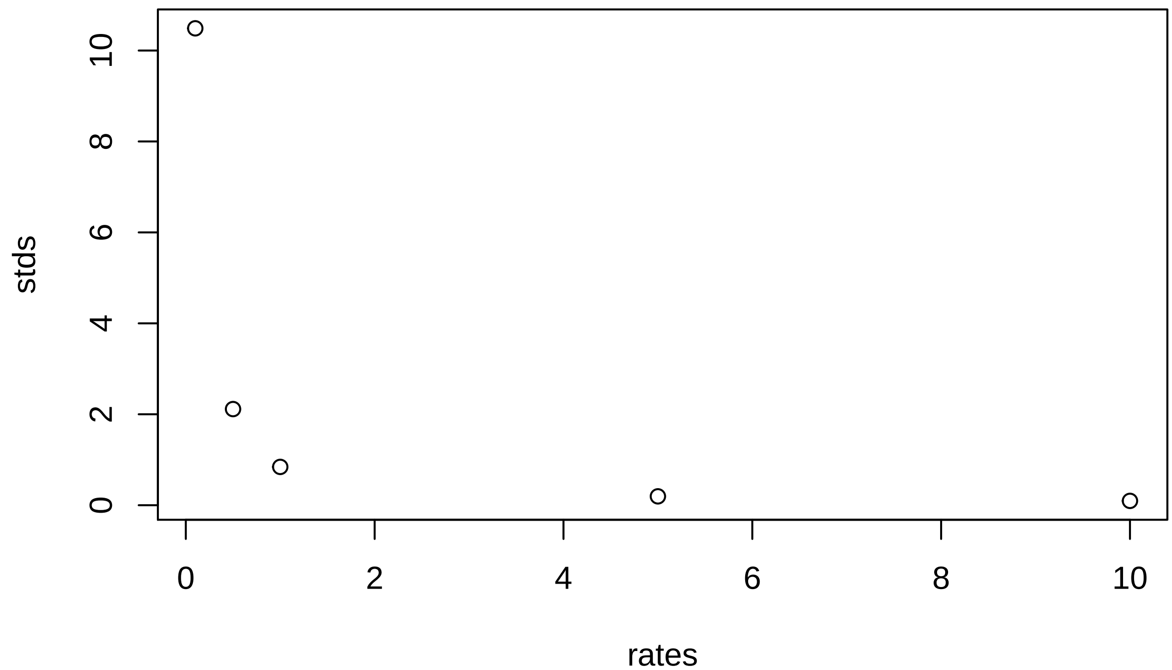



```r
# or using ggplot2
emeans1 %>% ggplot(aes(x = rates, y = means))+
  geom_point()
```

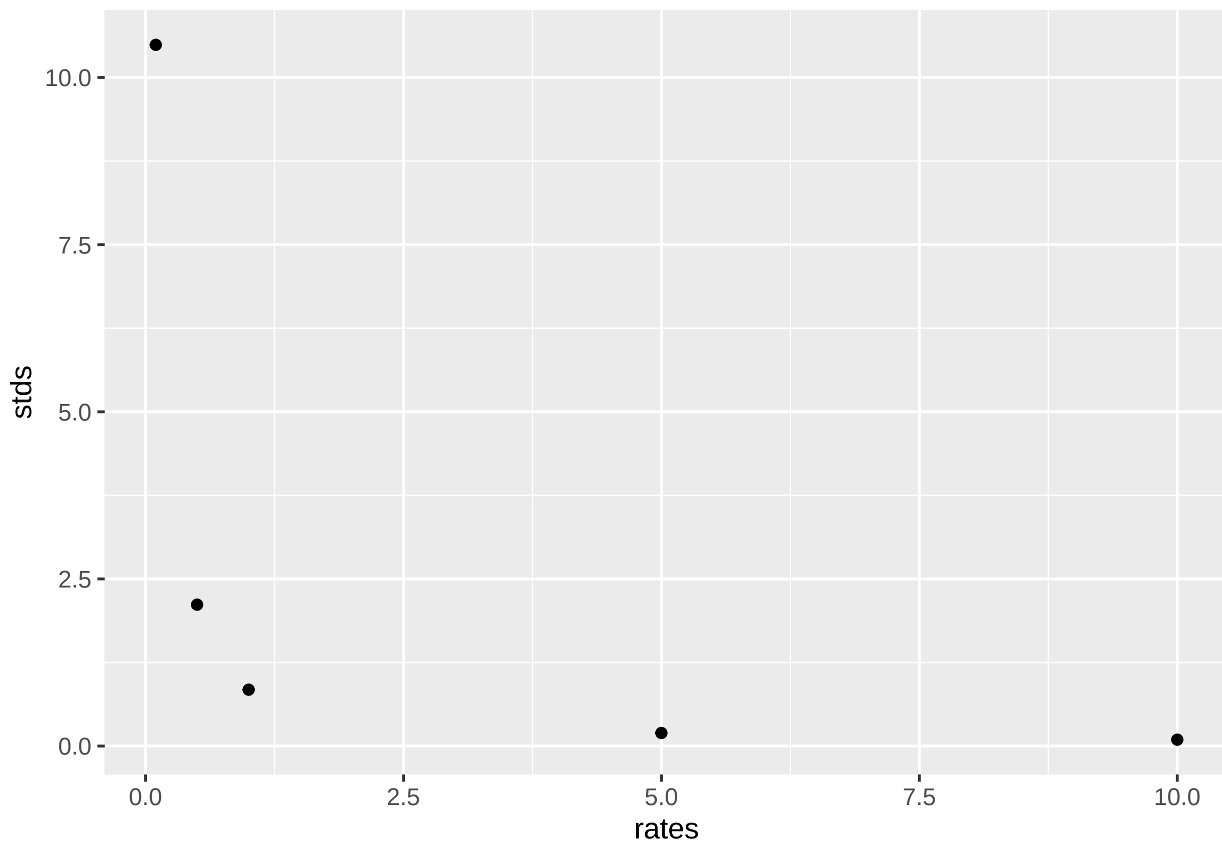


b. The standard deviations versus the rates.

```
plot(stds ~ rates, data = emeans)
```

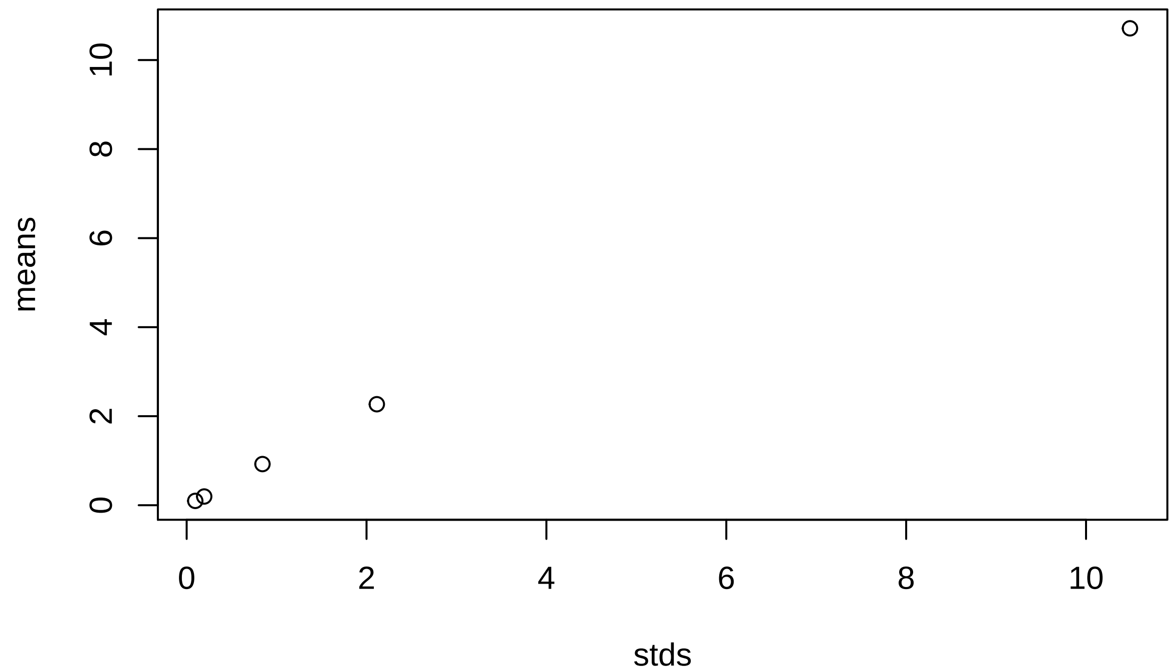



```
# or using ggplot2
emeans1 %>% ggplot(aes(x = rates, y = stds))+
  geom_point()
```

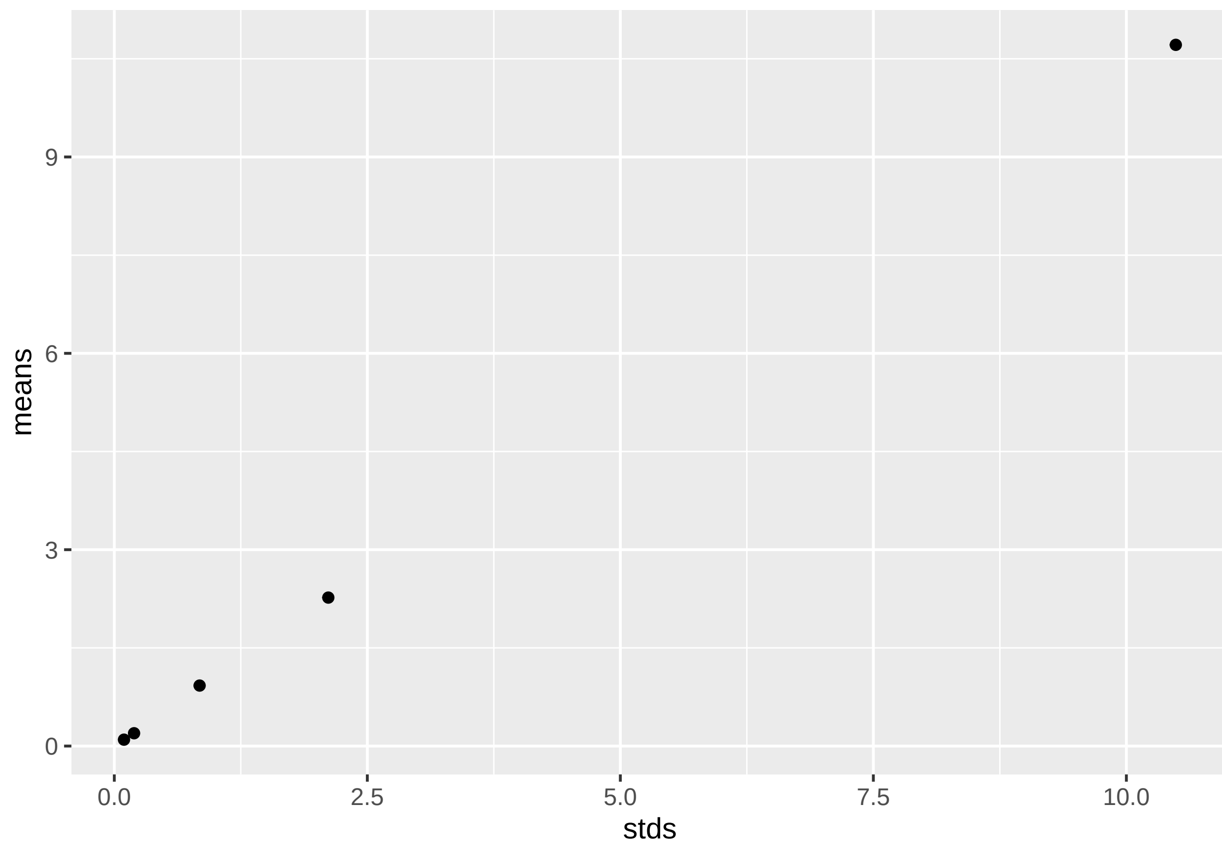


c. The means versus the standard deviations.

```r
plot(means ~ stds, data = emeans)
```




```
# or using ggplot2
emeans1 %>% ggplot(aes(x = stds, y = means))+
  geom_point()
```



For each plot, explain in words what's going on.

Part II

5. R's capacity for data and computation is large to what was available 10 years ago.
   a. To show this, generate 1.1 million numbers from the standard exponential distribution and store them in a vector called `big.exp.draws.1`. Calculate the mean and standard deviation.

```
big.exp.draws.1 <- rexp(1100000)
mean(big.exp.draws.1)
```

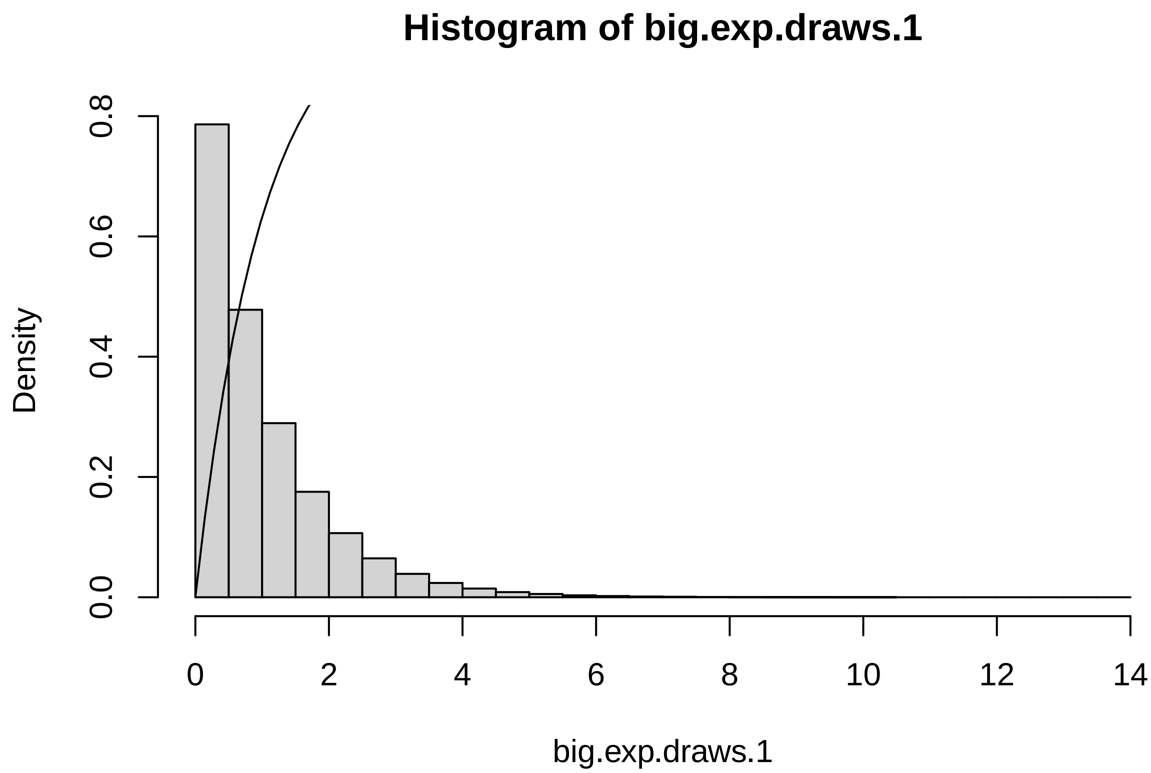
```
## [1] 1.000194
```

```
sd(big.exp.draws.1)
```

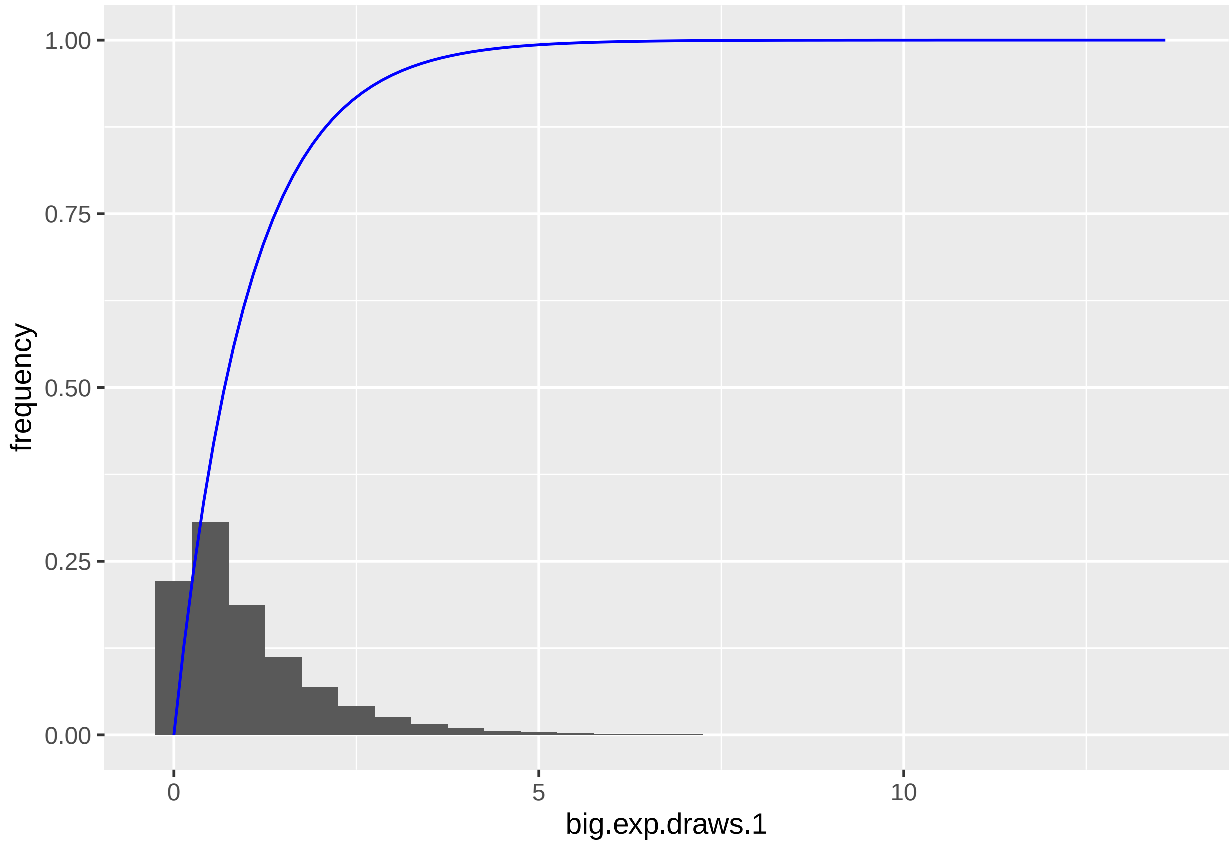
```
## [1] 0.9999471
```

```
b. Plot a histogram of `big.exp.draws.1`.  Does it match the function $1-e^{-x}$?  Should it?
```

```
hist(big.exp.draws.1, probability = T)
afun <- function(x) 1-exp(-x)
curve(afun, add = T)
```




```r
# or use ggplot2
tibble(expdraw = big.exp.draws.1) %>% ggplot()+
  geom_histogram(aes(x = big.exp.draws.1, y = (..count..)/sum(..count..)), binwidth = 0.5)+
  ylab('frequency')+
  stat_function(fun = afun, col = 'blue')
```



c. Find the mean of all of the entries in `big.exp.draws.1` which are strictly greater than 1. You may r

```
big2 <- big.exp.draws.1 > 1
mean(big.exp.draws.1[big2])
```

```
## [1] 2.000028
```

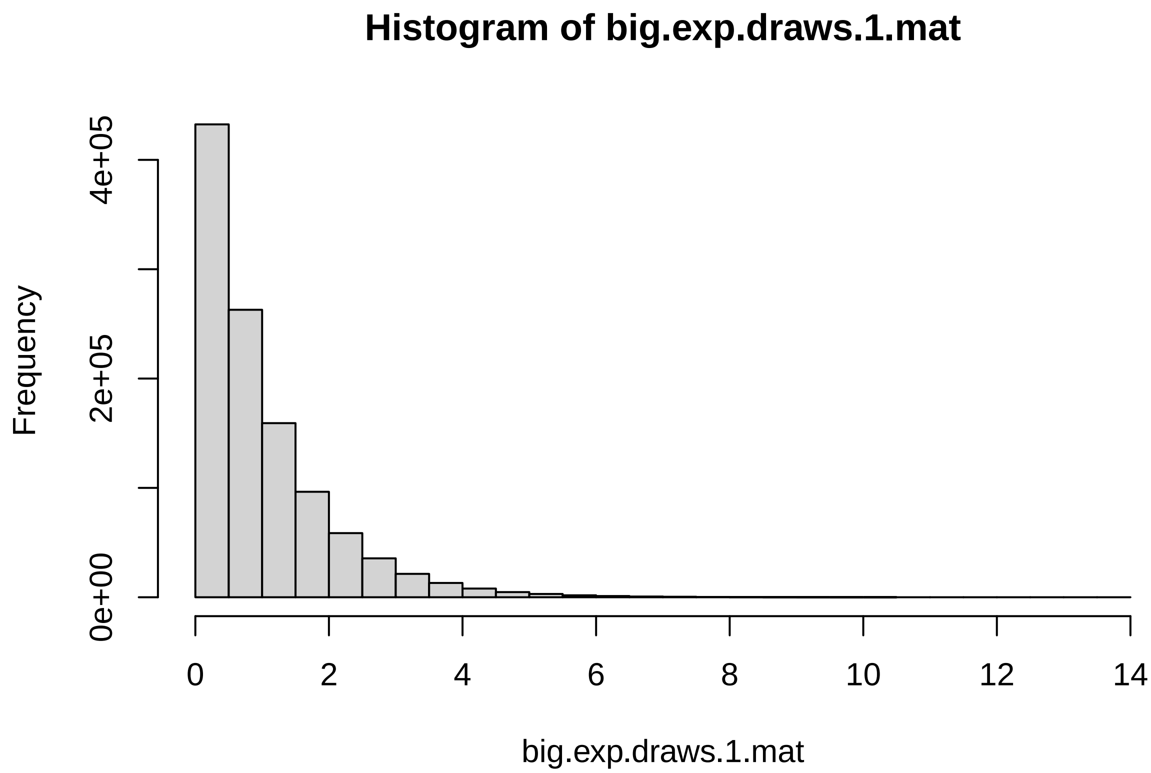
```
# or use dplyr after chapter 4
tibble(draw = big.exp.draws.1) %>% filter(draw > 1) %>% summarize(mean = mean(draw))
```

```
## # A tibble: 1 x 1
##    mean
##   <dbl>
## 1  2.00
```

d. Create a matrix, `big.exp.draws.1.mat`, containing the the values in

`big.exp.draws.1`, with 1100 rows and 1000 columns. Use this matrix as the input to the `hist()` function and save the result to a variable of your choice. What happens to your data?

```
big.exp.draws.1.mat <- matrix(big.exp.draws.1, nrow = 1100)
bighist <- hist(big.exp.draws.1.mat)
```



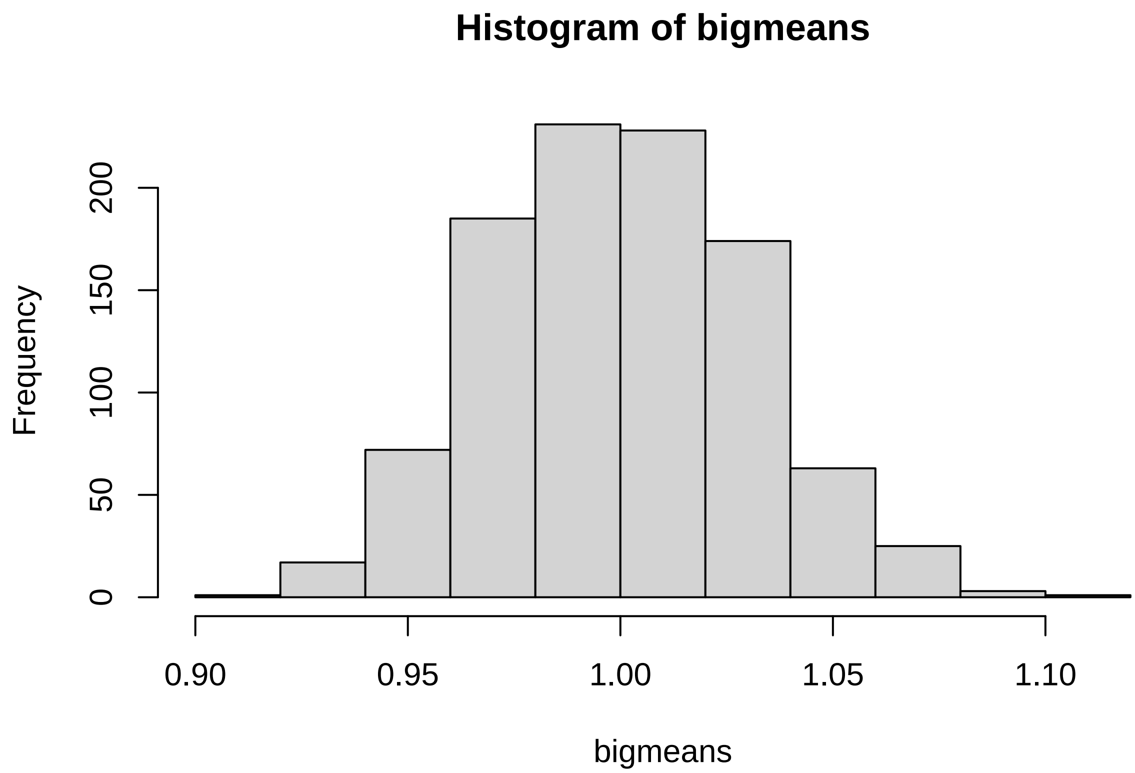
e. Calculate the mean of the 371st column of `big.exp.draws.1.mat`.

```
mean(big.exp.draws.1.mat[,371])
```

```
## [1] 1.011575
```

f. Now, find the means of all 1000 columns of `big.exp.draws.1.mat` simultaneously. Plot the histogram o

```
bigmeans <- colMeans(big.exp.draws.1.mat)
hist(bigmeans)
```




```
g. Take the square of each number in `big.exp.draws.1`, and find the mean of this new vector.  Explain t
```

```r
bigsqrt <- sqrt(big.exp.draws.1)
mean(bigsqrt)
```

```
## [1] 0.8864046
```