Probability, Simulation, Sampling

Dr. Christopher Brown

November 9, 2018

0. Introduction

These notes are written for the Math 331 Data Analysis With R course, and are a quick overview of three areas in mathematics and computer science. The goals of these notes are:

- Introduce probability terminology and theory.
- Introduce the notion of a simulation approach to problem solving and provide examples.
- Give an "under-the-hood" glance at some techniques for algorithmically generating random samples.
- Provide some practical examples for using R for simulations and sampling.
- Provide an example applying these techniques to modeling data.

Where these notes are complete they are not coherent, and where they are coherent they are not complete. Their one saving grace is that they are short or at least no longer than they need to be, and I have no plans to embiggen them anytime soon.

Acknowledgements go out to my former students who have read bits and pieces of these notes over the years: Robert Spangler, Christina Messer, Justine Duke, Lauren Chrislu, Samuel Oh, Diana Salazar, Christopher Kromm, and Laura Willits. Special thanks to Shannon Pesta, whose capstone project started me thinking about whether and how this material should be presented.

1. Probability

Terminology and Notation

An *event space* or *sample space* S is a nonempty set. An *event* is a subset of the given sample space. Visually, we can represent sample spaces as rectangles and events as ovals in them. . . so, your basic Venn diagram.

Probability is a complicated term. In these notes we give two rough definitions of probability, each of which is useful in some scenarios. For more precise mathematical definitions of probability see [1] and [2].

Frequentist definition: Roughly, we determine the *probability of an event* A by repeatedly performing an experiment in which the event A may or may not occur, and the probability is then the ratio of the number of times A occurs to the number of times the experiment is performed. That is, the probability of A is the relative frequency at which A occurs in an experiment repeated many times. For example, if we flip a fair coin 1000 times, we expect about 500 Heads in the flips, so that the relative frequency with which we see Heads will be about 1/2.

Bayesian definition: Roughly, we determine the *probability of an event* A by evaluating the credibility we have in the occurrence of A given our observations of the system. For example, if we have not flipped a coin, we might give equal credibility to Heads and Tails for the next flip. We base this on the fact that most coins in our experience seem to be fiar. On the other hand, if we have already flipped that coin 10 times and seen 10 heads, we might give more credibility to Heads than Tails for the next flip, because we suspect the coin may be biased.

Notation in probability theory is notoriously complex, informal, and nonstandardized. In these notes we'll try to be consistent and explain what we mean by our notation. There will still be times when informality is helpful.

We usually denote events by capital letters like A, B, and so on. We denote the *probability of event* A by $P(A)$.

When A and B are both events, the event "A and B" is the set $A \cap B$. We write $P(A \text{ and } B) = P(A \cap B)$. Many textbooks also write $P(A, B)$ for the probability of A and B; we won't use this notation but we mention it for the sake of connecting with other works.

Two events are *mutually exclusive* when $A \cap B = \emptyset$.

The event "A or B" is the set $A \cup B$ and we write $P(A \text{ or } B) = P(A \cup B)$.

The event "not A" is called the *complement of* A, and is written A^C. If we need to specify the sample space, we may use the set difference notation $S \setminus A$ instead of A^C.

Properties of Probability

Chapters 2 and 3 of [1] contain a standard presentation of the properties of probability. Chapter 1 of [2] contains a more advanced look at this material. Here, we recount only the axioms and the basic propositions.

Suppose that S is a given sample space. Then a probability measure on S is a function P mapping events to real numbers, with P satisfying the following three axioms:

Axiom 1: For every event A, $0 \leq P(A) \leq 1$.

Axiom 2: $P(S) = 1$.

Axiom 3: For any sequence of mutually exclusive events A_1, A_2, ... A_n,...,

$$P\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} P(A_i)$$

From these three axioms, we can then prove the following propositions:

Proposition 4 (Complementation Rule): $P(A^C) = 1 - P(A)$

Proposition 5 (P is Increasing): If $A \subseteq B$ then $P(A) \leq P(B)$.

Proposition 6 (Sum Rule): For all events A and B,

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

Example: Suppose that our experiment is to flip a fair coin twice and record the faces showing on each flip, in order. In this experiment, there are some events - like A="Heads, then Tails" - that are not further divisible into events, because there are no events that are proper nonempty subsets of A. Any event that has no proper nonempty subsets as events is called *atomic*. (There are other events that can be further subdivided, e.g. B="Flip Heads on at least one flip" has A as a proper subset.)

We could list the atomic events for this experiment (there are four, which we might briefly label as HH, HT, TH, and TT). Let's build a probability measure P by assigning $P(A) = 1/4$ for all atomic events A, and then requiring that P satisfy all the probability axioms. Two questions: (1) Can we *actually* require that this P satisfy the probability axioms, or will we encounter an inconsistency? and (2) Can we use this definition of P to compute probabilities of events?

For question (1): Yes, we can. This requires some proof, but the essential observation is that the atomic events are mutually exclusive and union to the sample space S.

For question (2): Yes. For the event B="flip heads on at least one flip", we can compute $P(B)$ in two ways:

Way 1: $B = \{HH, HT, TH\}$ and so $P(B) = P(HH) + P(HT) + P(TH)$ from axiom 3, and so $P(B) = 1/4+1/4+1/4 = 3/4$ from our assignment of probabilities to atomic events.

Way 2: Alternately, we have $B^C = \{TT\}$, so $P(B) = 1 - P(B^C) = 1 - 1/4 = 3/4$.

This is a fairly simple example, but the principle holds for much larger sample spaces. For example, if we are studying five card draw poker probabilities, we would want to consider the sample space with each poker hand as an atomic event. There are

```
choose(52,5)
```

```
## [1] 2598960
```

possible hands, and so each hand would be assigned a probability of 1/2598960.

More generally, whenever we have a sample space that can be expressed as a union of N atomic events, assigning $P(A) = 1/N$ for all atomic events A and then requiring the probability axioms to hold gives a probability measure.

Conditional Probability

The notion of conditional probability is probably the most important concept in probability theory. In essence, with this concept we can express and compute probabilities of events given different collections of information.

If A and B are events, then the *probability of A given that B occurred* is denoted $P(A|B)$. When we know that B occurred, this reduces our sample space to B. A few sketches with Venn diagrams should convince you that

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

This expression can be rearranged to read, equivalently, $P(A \cap B) = P(A|B)P(B)$. Section 3.5 of [1] shows that the function $P(\cdot|B)$ mapping events to real numbers is also a probability measure, in that it satisfies the three axioms above.

Because $P(A \cap B) = P(B \cap A)$, we can now write

Proposition 7 (Bayes' Theorem):

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Proposition 8 (Conditioning on a Partition): If B_1, B_2,...,B_n is a collection of mutually exclusive events whose union is S (a *partition* of S), then

$$P(A) = \sum_{i=1}^{n} P(A|B_i)P(B_i)$$

Two events A and B are said to be *independent* if $P(A|B) = P(A)$. That is, A and B are independent if the computation of the probability of A does not change whether we know B happened or not.

Proposition 9: A and B are independent events if and only if $P(A \cap B) = P(A)P(B)$.

Random Variables

A *random variable* is a function mapping events in a sample space to real numbers. We often think of random variables as measurements of quantities of interest in an experiment. For example, if our experiment is to flip a fair coin 20 times, then one random variable might be X=number of Heads in the 20 flips. Here, the range of X is $\{0, \ldots, 20\}$.

In this example, we denote the event "we saw 1 Heads in the 20 flips" as $X = 1$. The notation $X = 1$ should be interpreted to mean "all of the events in the sample space S which, when measured by X, produce an output of 1". (For those of you with some advanced mathematics under your belt, $X = 1$ is one way to write the inverse image of the output 1 under the function X, i.e. the set $X^{-1}(1)$.) *It is very important to understand that when we write $X = 1$ or $X \leq 10$, these are just notations describing certain events, and we should treat them just as we would any other event!*

As a second example, if our experiment is to repeatedly flip a fair coin until the first occurrence of Heads, then a random variable might be X=number of Tails appearing before the first Heads appears. In this example, the range of X is $\{0, 1, \ldots\}$, and so the range of a random variable need not be finite.

A random variable is *discrete* if its range is made up of numbers separated by gaps, with the smallest gap being positive. For example, if the range of X is $\{1, 2, 3\}$ or

$\{2, 4, 6, 8, \ldots\}$, X is a discrete random variable. Otherwise, the random variable is said to be *continuous*.

Given a discrete random variable X, the *probability mass function* for X is the function $f(x) = P(X = x)$, and the *distribution function* is

$$F(x) = P(X \leq x) = \sum_{z \leq x} f(z)$$

We note in this case that if $x_1 < x_2$ and no value in the range of X is between x_1 and x_2 then we have $F(x_2) - F(x_1) = f(x_2)$. So, we can create the density function f from the distribution and vice versa. For example, suppose we have a random variable X with mass function given by the table

$X =$	$p(x)$
0	0.1
1	0.6
2	0.3

Then we can compute the distribution by computing a cumulative sum.

$X =$	$p(x)$
0	0.1
1	0.1+0.6=0.7
2	0.1+0.6+0.3=1

On the other hand, if we have a random variable X with distribution given by

$X =$	$F(x)$
0	0.1
1	0.7
2	1

then we can compute the mass function by appropriate subtractions (work from the bottom up!).

$X =$	$F(x)$
0	0.1 - 0 = 0.1
1	0.7 - 0.1 =0.6
2	1 - 0.7 = 0.3

Given a continuous random variable X, the *probability density function* for X is the function $f(x)$ such that

$$P(X \leq x) = \int_{-\infty}^{x} f(z)\, dz$$

and the *continuous distribution function* for X is the function

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} f(z)\, dz$$

In the language of calculus, $F'(x) = f(x)$, so the rate of change of the distribution is the density. Thus we can create the density from the distribution and the distribution from the density.

In both the discrete and continuous cases, the distribution is a sort of sum of the density or mass, and the density or mass function is a change in the distribution.

Expectation and Variance

The *expected value of a discrete random variable* X, $E(X)$, is given by the formula

$$E(X) = \sum_{x} xp(x)$$

where p is the probability mass function of X and this sum is computed over all x values in the range of X. For example, if the random variable has mass given by the table

$X =$	$p(x)$
0	0.1
1	0.6
2	0.3

then we may compte the expected value of X as

$$E(X) = 0 * 0.1 + 1 * 0.6 + 2 * 0.3 = 1.2$$

The expected value of X is a measure of the "center", "mean", or "average" of X, in the following sense. Suppose we repeat the experiment generating the random variable X many times and collect the data. We can do this in R by bulding a sample space with the correct proportions of outputs, and then sampling from it. Here, the outputs 0, 1, and 2 should be in the ratios 1:6:3, so we build a sample space with 1 "0", 6 "1"'s, and 3 "2"'s. I somewhat arbitrarily decided to draw a million samples.

```
data <- sample(c(0,1,1,1,1,1,1,2,2,2),1000000,replace=TRUE)
```

(We will look more closely at the sample function in R in the next section. If you are curious now, go look it up!) We can now compute the mean value for this data set.

```r
mean(data)
```

```
## [1] 1.199373
```

We get (a very close approximation to) the expected value. So, the expected value measures the average numerical outcome we should expect to get after many repetitions of the experiment generating X.

We can also compute the expected value of quantities computed from the random variable X. Suppose that $g : \mathbb{R} \to \mathbb{R}$. Then the expected value of $g(X)$ is defined to be

$$E(g(X)) = \sum_x g(x)p(x)$$

where p is the mass function for X and the sum is taken over all x in the range of X. For example,

$$E(X^2) = \sum_x x^2 p(x)$$

.

If we are dealing with a single random variable X, we (by tradition) denote $E(X)$ by the Greek letter *mu*, μ.

Another measurement we can make about a random variable is the average deviation of the values of the random variable from the center, $E(|X - \mu|)$. It turns out that in practice, a more mathematically tractable quantity is the average square deviation of the values of X from its center, called the *variance of* X, denoted $V(X)$, and defined by

$$\sigma^2 = V(X) = E((X - \mu)^2) = \sum_x (x - \mu)^2 p(x)$$

(where σ is the greek letter *sigma* and is used by long tradition). The square root of this quantity has the same physical units as X, and so it is often convenient to compute $\sigma = \sqrt{V(X)}$, which is called the *standard deviation of* X.

Exercises

Exercise 1.1 Suppose that we flip a fair coin 5 times and record the face showing after each flip.

(a) How large is the sample space for this experiment?

(b) Write examples of two elements of the sample space.

(c) How many of the elements of the sample space begin with a Heads?

(d) How many of the elements of this sample space begin with a Tails?

(e) How many of the elements of the sample space begin with a Heads and end with a Tails?

(f) Are the events "Begins with a Heads" and "Begins with a Tails" mutually exclusive?

(g) Are the events "Begins with a Heads" and "Ends with a Tails" mutually exclusive?

Exercise 1.2 For a certain species of cat, the number of young produced each year per mother is a random variable X. Current data suggests that X has the mass function

$X =$	$p(x)$
0	0.3
1	0.5
2	0.15
3	0.05

(a) What is the probability that a mother cat has 2 or more young in a given year?

(b) What is the probability that a mother cat has at least one young in a given year?

(c) What is the expected number of young for a mother cat in a year?

(d) What is the variance in the number of young for a mother cat in a year?

(e) A similar species of cat has Y young per year per mother, where Y has the mass function

$Y =$	$p(y)$
0	0.2
1	0.5
2	0.15
3	0.1
4	0.05

How do the mean and variance of Y compare to X? What about the mass functions would suggest this with no calculations?

(f) Use R to build a sample space for X, draw a large sample, and compute the mean and variance of the sample as we did earlier in the notes. You'll want to use the *mean* and *var* functions in R.

(g) Repeat (f) fpr the random variable Y.

Exercise 1.3 Suppose that X is a random variable that only produces one output, c, and $p(c) = 1$ (so, X isn't very random, is it?). What are the expected value and variance of X?

Exercise 1.4 Suppose that X is a random variable with mass function

$X =$	$p(x)$
0	0.3
1	0.5
2	0.15
3	0.05

(a) Are $X = 0$ and $X = 1$ mutually exclusive events?

(b) Using the third axiom of probability, compute $P(X < 3)$. (c) Using the complementation rule compute $P(X < 3)$.

(c) Is $P(X < 2)$ greater than, less than, or equal to $P(X < 3)$? Try to explain this without resorting to computing these probabilities.

Exercise 1.5 Suppose that X is a random variable with mass function

$X =$	$p(x)$
0	0.3
1	0.5
2	0.15
3	0.05

(a) Compute the probability $P(X \geq 2 | X \neq 0)$.

(b) Compute the probability $P(X = 0 | X < 3)$.

(c) Compute the probability $P(X = 0 | X = 3)$.

(d) Are $X = 0$ and $X = 3$ mutually exclusive events? Explain.

(e) Are $X = 0$ and $X = 3$ independent events? Explain. (f) So, does "mutually exclusive" mean the same thing as "independent"?

Exercise 1.6 Show that if A and B are mutually exclusive events then $P(A \cup B) = P(A) + P(B)$.

Exercise 1.7 Suppose that we flip a fair coin twice in a row, in such a way that the two flips are independent of one another, and thus the events X_1 =Heads on flip 1 and X_2 =Heads on flip 2 are independent events.

(a) What is the probability $P(X_1 \cap X_2)$?

(b) If we continue this on out for a while, what is the probability $P(X_1 \cap \ldots \cap X_n)$?

(c) What is the probability of flipping at least one Tail in n flips? (Hint: the complement of the event "at least one Tail" is "no Tails".)

Exercise 1.8 Suppose that a plane has crashed in one of three regions and we are in charge of running the search. The plane is equally likely to have crashed in any one of the three regions. Region A is mountainous and the probability of finding the plane in a search of A given that it is in A is only 0.2. Regions B and C are grassy hills, and so the probability of finding the plane on a search of B or C given that it crashed in that region is 0.6.

(a) Suppose we first search region B and do not find the plane. Use Bayes' Theorem to update the probabilities that the plane crashed in regions A, B, or C.

(b) Now suppose we search region C and do not find the plane. Again, Use Bayes' Theorem to update the probabilities that the plane crashed in regions A, B, or C.

(c) Which region should we search next? Why?

Exercise 1.9 Suppose that X is a random variable with mass function

$X =$	$p(x)$
0	0.3
1	0.5
2	0.15
3	0.05

(a) Find the distribution function for X.

(b) Compute $P(X \leq 2)$ using the distribution function.

Exercise 1.10 Suppose that X is a random variable with distribution function

$X =$	$F(x)$
0	0.1
1	0.4
2	0.9
3	1

Find the probability mass function for X.

Exercise 1.11 Suppose that X is a continuous random variable with density function

$$f(x) = \begin{cases} 2e^{-2x} & x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

This is the exponential distribution.

(a) Compute the distribution function for X.

(b) Use the distribution function to compute $P(2 \leq X \leq 5)$.

(c) Compute the mean of X.

Exercise 1.12 For each of the following named distributions, (i) look up the distribution on Wikipedia and write a few sentences on what it is used for, then find the distribution in R and (ii) graph the density for a few example parameter sets, (iii) generate a sample from the random variable of size 1000, and (iv) plot the histogram of your sample.

(a) Binomial

(b) Poisson

(c) Negative binomial

(d) Normal

(e) Student t

(f) Exponential

(g) Beta

Exercise 1.13 Prove Proposition 4 from the Probability Axioms.

Exercise 1.14 Prove Proposition 5 from the Probability Axioms.

Exercise 1.15 Prove Proposition 6 from the Probability Axioms.

2. Simulation

When we *simulate* an experiment, we program the rules of an experiment and some initial conditions into a computer, and then we perform the experiment electronically. Generally, simulated experiments are cheap and highly repeatable. However, they do require us to understand the rules for the experiment very well. So for many questions in probability, simulation can be a highly effective tool because in asking the probability question, we define the rules of the experiment! Some questions in the natural, physical, and social sciences can be approached by simulation experiments, but care must be taken in interpreting the results.

Random Number Generators in R

Probability simulations generally require a random number generator. Chapter 2 of [3] contains a wealth of information, both historical and technical, on random number generation in digital computers. This is a beautiful and absolutely essential topic at the boundary of mathematics and computer science.

We'll approach random number generation on a need-to-know basis. For now, we need to know how R can generate random numbers.

Generally, most computer programming languages have some sort of function that randomly selects a number uniformly distributed on the interval $[0, 1]$. R is "distribution-centric", and so names its function after the uniform distribution.

```
runif(1)
```

```
## [1] 0.7175067
```

The argument to this function, 1, tells us that R will return a resulting vector of length 1, i.e. only one randomly generated number. If we need to draw several uniformly distributed random numbers between 0 and 1, we can do so with a single command.

```
runif(20)
```

```
##  [1] 0.119799453 0.402391649 0.008426069 0.442143529 0.609985589
##  [6] 0.229419207 0.371045246 0.108638472 0.036116387 0.916555611
## [11] 0.947631273 0.356663814 0.403518240 0.792584554 0.839324025
## [16] 0.457788503 0.240336855 0.420468332 0.142381704 0.916462546
```

Random number generation in R is quite fast. Try executing something like the following on your own.

```
data <- runif(100000)
```

On my machine, generating these hundred thousand values required about half a second.

Digital random number generators are not really random, but work something like a roulette wheel; if we knew how fast the wheel was spun, how fast the ball was rolled, and all the little irregularities about how this particular roulette wheel spins, then we could use physics to correctly predict the outcome of the roulette spin. That is, roulette wheels are not really random; they are deterministic and have sensitive dependence on initial conditions. Since it is nearly impossible to know the initial conditions precisely enough, the roulette wheel "feels" random. Digital random number generators are similar, but with one important difference. For our digital random number generators, we can set the initial conditions. This means that experiments can be made precisely repeatable. For example, if I execute

```
runif(5)
```

```
## [1] 0.41884572 0.63928478 0.02045085 0.14049758 0.37895680
```

and repeat it

```
runif(5)
```

```
## [1] 0.3428132 0.7644245 0.1091249 0.5727909 0.4963470
```

I don't necessarily get the same values. However, with the set.seed() function, I can set the initial condition - called the "seed" - to a certain value and guarantee a repeatable experiment. Round 1:

```
set.seed(1234)
runif(5)
```

```
## [1] 0.1137034 0.6222994 0.6092747 0.6233794 0.8609154
```

And round 2:

```
set.seed(1234)
runif(5)
```

```
## [1] 0.1137034 0.6222994 0.6092747 0.6233794 0.8609154
```

Other random number generators in R based on common distributions work in much the same way.

Another common task in simulation is to randomly choose some values from a set of values. For example, I may have the data set

```
data <- c('red','blue','green','yellow')
```

and I would like to choose three values randomly. This is called *sampling*, and can be done with or without replacement of values before the next draw. For example, without replacement we have

```
set.seed(1234)
sample(data,3)
```

```
## [1] "red"   "blue"  "green"
```

and with replacement we have

```
set.seed(1234)
sample(data,3,replace = TRUE)
```

```
## [1] "red"   "green" "green"
```

When we replace before drawing again, we can resample from the same small data set many times. For example, if I give a multiple choice exam with twenty questions, each of which has a correct answer of choice a, b, c, or d, I might want to generate the answers randomly to avoid human bias. So I can turn to R and sample:

```
choices <- c('a','b','c','d')
sample(choices, 20, replace = TRUE)
```

```
##  [1] "c" "d" "c" "a" "a" "c" "c" "c" "c" "b" "d" "b" "d" "b" "b" "a" "a"
## [18] "b" "b" "a"
```

If for some reason I want to bias the selection towards choice a, I can increase the number of a's in the drawing pool.

```
choices <- c('a','a','a','b','c','d')
sample(choices, 20, replace = TRUE)
```

```
##  [1] "a" "a" "c" "b" "d" "c" "a" "a" "a" "a" "b" "a" "c" "a" "a" "d" "c"
## [18] "b" "b" "a"
```

Does it look like there are more a's than there ought to be?

A Programming Aside: The Differences Among For Loops, While Loops, and Apply

In simulation, we will often need to repeat an experiment or some action in an experiment many times. There are several common programming structures to accomplish this. In this section I'll take a moment to step to the side and consider these.

When deciding which programming structure to use to repeat an action ("loop"), we need to ask three questions: 1. Before beginning the loop, do we know how many times we need to run the loop? 2. Will any of the repetitions of the loop influence the computation in the *next* repetition of the loop? 3. Will any of the repetitions of the loop influence the computation in the *previous* repetition of the loop?

For us, an answer of *yes* to 3 takes us a little too deeply into a more advanced computer programming topic (recursively defined functions). Also, there are almost no situations we will run into that will give an answer of *yes* to 3. So, let's just assume we don't have to worry about 3! Now on to the more practical concerns...

If the answer to 1 is *no*, we will almost surely need a while loop.

If the answer to 1 is *yes* and the answer to 2 is *yes*, we will almost surely need a for loop.

If the answer to 1 is *yes* and the answer to 2 is *no*, we will be able to use an apply loop. This will happen very frequently for us; we'll see why in the examples in the next section!

Many R functions have a built-in apply loop or can be given one. For example, the squaring function will be automatically applied across all the elements of a vector:

```
x <- c(1,2,5,7)
x^2
```

```
## [1]  1  4 25 49
```

If we define a function of one variable we can always give it this behavior. Here is a function that squares anything less than or equal to three and maps everything else to 0.

```
f <- function(x) {
  if (x>3) {
    return(0)
  } else {
    return(x^2)
```

```
  }
}
f <- Vectorize(f)
x <- c(1,2,5,7)
f(x)
```

```
## [1] 1 4 0 0
```

Here is a great trick, and one that comes up often! I frequently need to perform some experiment which has the form of a function. I need to repeat it some number of times, but I don't actually need to provide any inputs. Since each pass through my loop is independent of all the others, I should be able to avoid a for or a while loop. I can do this with Vectorize by providing a dummy variable for my function. For example, let's suppose I want to flip a fair coin some number of times and record the flips with 1's for Heads and 0's for Tails. I can write my code as follows:

```
run_experiment <- Vectorize(  # Vectorize here!
  function(x=17) {             #dummy variable x=17 here!  You can initalize x to whate
    sample(c(0,1),1)
  }
)
data <- run_experiment(1:20) # run the experiment 20 times and store the results in dat
```

Modularity and Simulation

In computer programming, *modularity* is the programming practice in which programs are broken down into small chunks by using functions and comments, in such a way that the chunks can be individually assessed for performance and the code can be more easily understood.

There are five working principles you should try to adhere to:

1. **Use variables, not constants.** That is, if you need to use the constant "100" in a conceptual way in your code, e.g. as a sample size, then store it in a variable and then use that variable in the code instead of writing the constant 100. Storytime: I once had a student who routinely violated this principle. Whenever he needed to change his constants in his code, he would simply use the Find-Replace function to change all the instances. Well, one day, he need to change a certain constant "2" to a "5" in his code. Find-Replace...and he changed *all* the 2's to 5's, everywhere in his code. He also didn't notice for a while, so undoing was no longer an option. Since he had used the number 2 in some of his variable and function names, and

all of them changed to 5's, his code became unuseable. I have no idea how long it took him to fix this, but I did hear from his other instructors that he skipped classes for an entire week.

2. **Use functions for small, conceptually simple tasks.** It is easier to test code in pieces when tasks have been separated into functions. It is also easier to read code as well. You should use descriptive names for the tasks your functions perform.
3. **Use functions to tie together small tasks into a larger workflow.**
4. **Use functions for repeatability.** Any task that needs to be repeated should be encoded as a function.
5. **Comment to explain code chunks.**

Simulation Example: Dice Throw Probability

Let's first simulate a probability we can compute symbolically, as a check: what is the probability of rolling a sum of 7 when rolling a pair of fair six-sided dice?

Generally, we can fit probability simulations into a common framework:

- Define constants for the experiment.
- Create one or more functions to perform one repetition of an experiment (in this case, one roll of the two dice).
- Create a function or code to perform many experiments and record the results of each experiment.
- Analyze the results of the collection of experiments.

Let's begin by defining constants for the experiment.

```
set.seed(42) # Hitchhiker's Guide
number_of_experiments <- 100000
one_die <- 1:6
```

Next, let's define a function to roll two fair six-sided dice and sum the faces showing.

```
one_roll <- Vectorize(function(x=1) {
  return(sum(sample(one_die,2,replace=TRUE)))
})
```

Before we go on to write the rest of our code, let's test this function.

```
print(one_roll(1))
```

```
## [1] 12
```

And again:

```
print(one_roll(1))
```

```
## [1] 7
```

And you can execute this a few more times to see how it behaves.

```
one_roll(1:10)
```

```
##  [1]  8  6  9  8  8  9  7  7  7 12
```

This gives us a good idea of how we should proceed for our next phase: repeat the experiment many times and record the data.

```
data <- one_roll(1:number_of_experiments)
```

At this point we need to compute the number of 7's we see in the data.

```
number_of_7 <- length(which(data==7))
```

And now we can compute the probability that we roll a 7.

```
number_of_7/number_of_experiments
```

```
## [1] 0.16853
```

Symbolically, we have 6 "7"'s out of 36 outcomes, i.e.

```
6/36
```

```
## [1] 0.1666667
```

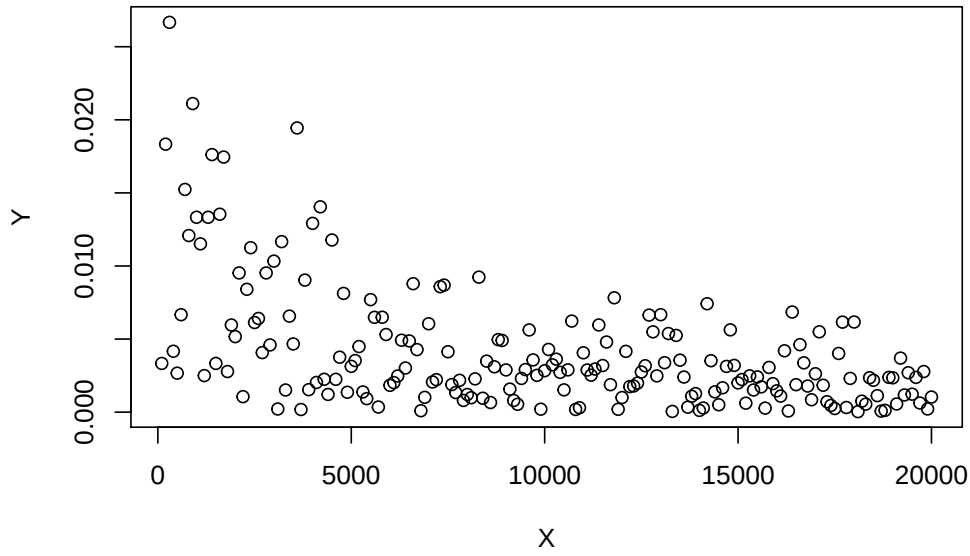
So our estimate is certainly good enough for government work. If we need to how can we improve our estimate?

Let's take a look at how the estimate improves as the number of experiments increases. We'll need a function that runs these experiments.

```
run_experiments <- Vectorize(function(number_of_experiments) {
  data <- one_roll(1:number_of_experiments)
  number_of_7 <- length(which(data==7))
  return(number_of_7/number_of_experiments)
})
```

Now let's run different numbers of experiments and see how good our estimate is.

```
X <- 100*(1:200)
estimates <- run_experiments(X)
Y <- abs(estimates-6/36)
plot(X,Y)
```



We observe two things about this plot. First, as the number of experiments increases, the error in our estimate decreases (with some noise). Second, as the number of experiments increases, further increasing the number of experiments leads to less improvement in the estimate. Thus increasing the number of experiments improves the estimate but with diminishing returns.

Simulation Example: A Walk With a Coin

My son is standing on a long, straight sidewalk running east and west. I give him a fair coin and the following procedure:

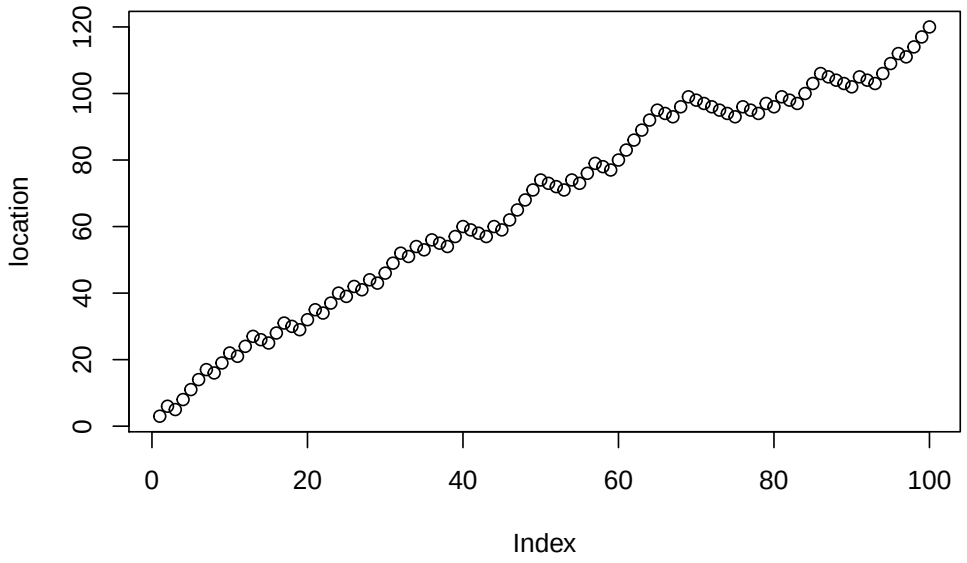
Step 1: Take 1 step east. Step 2: Flip the coin. If Heads, take two more steps east. If Tails, take two steps west.

After 100 steps, where is he likely to be relative to the starting position? We know that he could be anywhere from 100 steps west to 300 steps east, but how likely

are any of those possibilities?

Let's model this with steps east being represented by positive numbers and steps west represented by negative numbers. We can simulate one trip fairly easily.

```
set.seed(42)
coin_flips <- sample(c(-1,1),100,replace=TRUE)
extra_steps <- 2*coin_flips
location <- cumsum(1+extra_steps)
plot(location)
```



The final location is the data we really wish to collect.

```
tail(location,1)
```

```
## [1] 120
```

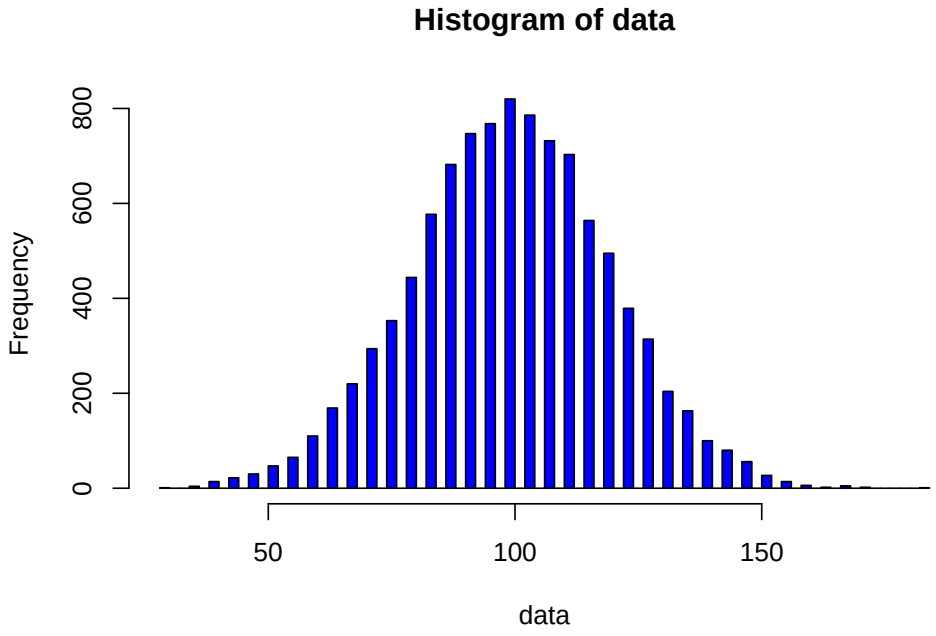
Let's now write a function to perform one trip and then repeat that.

```
set.seed(42)
number_of_trips <- 10000
one_trip <- Vectorize(function(x=1) {
  coin_flips <- sample(c(-1,1),100,replace=TRUE)
  extra_steps <- 2*coin_flips
  location <- cumsum(1+extra_steps)
```

```
  return(tail(location,1))
})
data <- one_trip(1:number_of_trips)
```

Let's examine the data.

```
hist(data,breaks=60,col='blue')
```



How do we interpret this histogram?

Simulation Example: Professor Z's Office Hours

Professor Z tries to show up at his office for appointments on time. He really does! But students and faculty frequently stop him to ask questions or just chat, and he is a typical absent-minded professor, so sometimes he is late.

Estrella is trying to meet Professor Z at his office for an appointment. She really is! But between traffic and campus parking, Estrella might be late.

Professor Z and Estrella are both busy, so they can't wait long for the other person.

(a) Let's assume that Professor Z and Estrella arrive at the appointment at some time uniformly distributed between noon (time 0) and 1:00PM (time

1), and let's assume each will wait for the other for 15 minutes but not longer. Write a simulation to estimate the probability that Professor Z and Estrella actually meet for their appointment.

(b) Now let's assume that Professor Z and Estrella each arrive at the appointment at a time uniformly distributed between noon and 1PM. Let's also assume that each will wait for a uniformly distributed random amount of time between 0 and 20 minutes. What do you expect to happen to the probability the two meet in this scenario, as compared to that in part (a)? Write a simulation to estimate the probability that Professor Z and Estrella actually meet for their appointment.

Part (a)

Define our constants.

```
set.seed(42)
number_of_experiments <- 100000
wait_time <- 15
```

Define functions to perform one repetition of this experiment.

```
one_rep <- Vectorize(function(x=1) {
  arrival_time <- runif(2,min=0,max=60)  # generate two arrival times
  abs(arrival_time[1]-arrival_time[2])<=15
})
```

How can we test the one_rep function? Try it!

Now repeat the experiment many times and record the data.

```
data <- one_rep(1:number_of_experiments)
```

Finally we can analyze the data and compute the probability of interest.

```
length(which(data))/number_of_experiments
```

```
## [1] 0.43483
```

Part (b)

Define our constants.

```
number_of_experiments <- 100000
```

Define functions to perform one repetition of this experiment.

```
one_rep <- Vectorize(function(x=1) {
  arrival_time <- runif(2,min=0,max=60)  # generate two arrival times
  wait_time <- runif(1,min=0,max=20)  # generate one wait time
  abs(arrival_time[1]-arrival_time[2])<=wait_time
})
```

How can we test the one_rep function?

Now repeat the experiment many times and record the data.

```
data <- one_rep(1:number_of_experiments)
```

Finally we can analyze the data and compute the probability of interest.

```
length(which(data))/number_of_experiments
```

```
## [1] 0.29411
```

Does this probability agree with your expectation?

Simulation Example: Optimization

Suppose that we are trying to maximize the function $f(x) = x^5 e^{-2x}$ on the interval $[0, 5]$. We could take a calculus approach, but why do that when we have a computer in front of us? We could just evaluate $f(x)$ for LOTS of x values and see which gives the largest output.

```
set.seed(42)
inputs <- runif(10000,min=0,max=5)
f <- function(x) {
  return(x*(1-x))
}
outputs <- f(inputs)
```

The maximum value for this function on $[0, 5]$ is

```
max(outputs)
```

```
## [1] 0.2499996
```

which occurs at x-value

```
index <- which.max(outputs)
inputs[index]
```

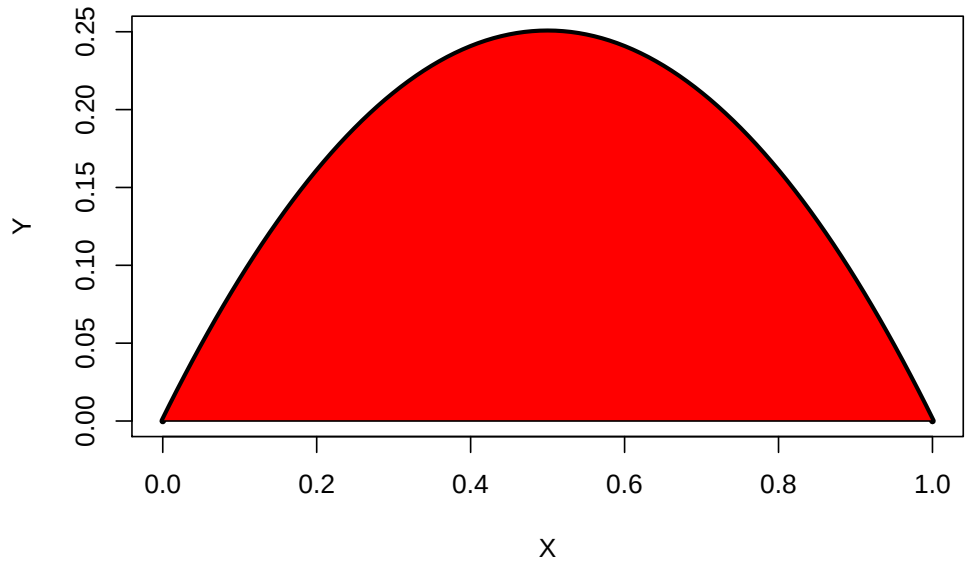
```
## [1] 0.5006234
```

Can you convince yourself that this is (approximately) the right answer?

Simulation Example: Area Under a Curve

Let's suppose that we want to compute the area under the curve $f(x) = x(1-x)$ on the interval $[0, 1]$. Graphically, we are trying to compute the area shown below:

```
X <- seq(from=0,to=1,by=0.01)
f <- Vectorize(
  function(x) {
    return(x*(1-x))
  }
)
Y <- f(X)
plot(X,Y,type='l',lwd=4)
XX <- c(X,rev(X))
YY <- c(Y,rep(0,length(X)))
polygon(XX,YY,col='red')
```



So, how can we do this? The thinking is to throw darts randomly at the graph

above, compute the proportion of the darts that land in the area, and then use that to compute the area of the shaded region.

First, let's get the coordinates of our darts.

```
xmin <- 0
xmax <- 1
ymin <- 0
ymax <- 0.3
number_of_darts <- 10000
X <- runif(number_of_darts,min=xmin,max=xmax)
Y <- runif(number_of_darts,min=ymin,max=ymax)
```

Let's also compute the area of the image.

```
image_area <- (xmax-xmin)*(ymax-ymin)
image_area
```

```
## [1] 0.3
```

We now need to check to see which darts fall in the shaded region. A dart will fall in the region exactly when the y coordinate of the dart is smaller than the y coordinate of the graph at that x coordinate, i.e. exactly when $y \leq f(x)$.

```
darts_in_region <- length(which(Y <= f(X)))
```

We now compute the proportion of darts in the shaded region.

```
darts_in_region/number_of_darts
```

```
## [1] 0.5542
```

So about 55.4% of the darts fell in the shaded region, which tells us that the shaded region takes up about 55.4% of the image area 0.3. We can then estimate the area of the shaded region as

```
darts_in_region/number_of_darts * image_area
```

```
## [1] 0.16626
```

It's worth noting that elementary calculus tells us that the exact answer is

```
integrate(f,lower=0,upper=1)
```

```
## 0.1666667 with absolute error < 1.9e-15
```

So our darts estimate is quite good. Computationally, it does not beat the standard numerical integration techniques as long as we have a formula for the function or some way to evaluate it, but it may be helpful in other contexts.

Parameter Estimation and Confidence Intervals

A *parameter* for a distribution is one of a set of constants that fully describe the distribution. For example, a normally distributed random variable is fully described by two parameters, the mean μ and the variance σ^2, and we have an expression for the density function f of the normal distribution with these parameters:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

As a second example, a Poisson distributed random variable X with rate parameter λ has the density function

$$p(n) = P(X = n) = \frac{\lambda^n}{n!} e^{-\lambda}$$

(The mean and variance of the Poisson distribution are expressed in terms of the rate parameter, as $\mu = \lambda$ and $\sigma^2 = \lambda$.)

A fundamental problem in statistics is to estimate a given parameter for a given distribution, based on available data. Estimates come in three flavors:

- Point estimate - Quick and dirty. A point estimate is the "best guess" for a parameter value we can make from available data.
- Interval estimate - A little more information is available. An interval estimate of a parameter uses data and a specified confidence level - e.g. 95% - to construct an interval. This interval has the property that we are 95% certain that it contains the true value of the parameter.
- Density estimate - Lots of information is available. A density estimate of a parameter treats the parameter as a random variable and uses data to construct a density function for the values of the parameter. From this we may construct a confidence interval, a point estimate, or simply present the density function as our estimate. Density estimates are most often used in Bayesian analysis.

[11] contains a concise but thorough introduction to point and interval estimate construction. You can read about kernel density estimation at the Wikipedia page, and [12] describes its construction and use in Bayesian analysis.

In some sense we've already been constructing density estimates for parameters. Each time we simulated a probability problem and plotted the histogram... density estimate! (More or less.) Let's look at two examples in which we produce all three

estimates from data. In the first example we will assume that the data follows a certain distribution, and in the second we will not.

Example

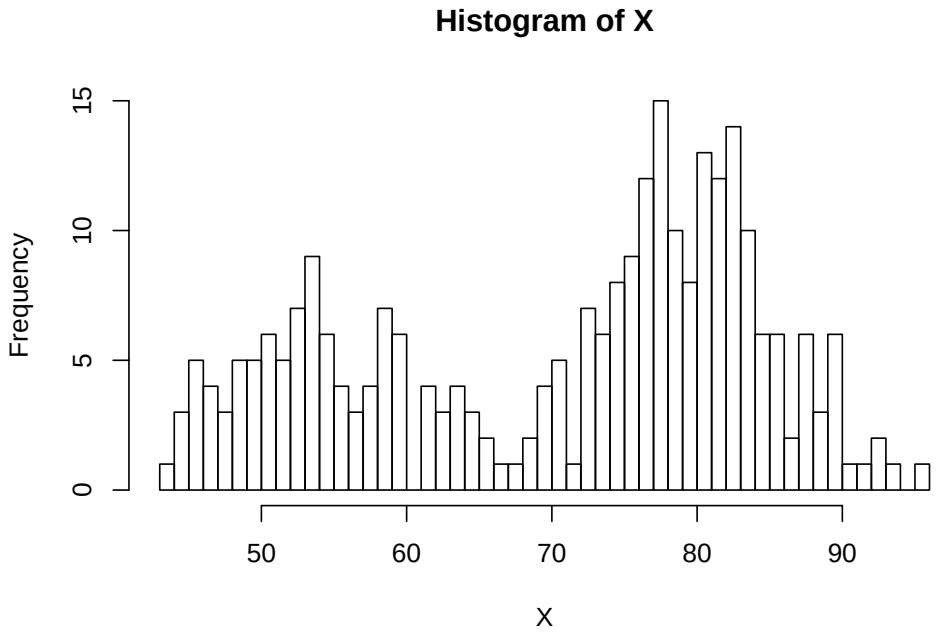
Let's load the Old Faithful data set.

```
data(faithful)
head(faithful)
```

```
##   eruptions waiting
## 1     3.600      79
## 2     1.800      54
## 3     3.333      74
## 4     2.283      62
## 5     4.533      85
## 6     2.883      55
```

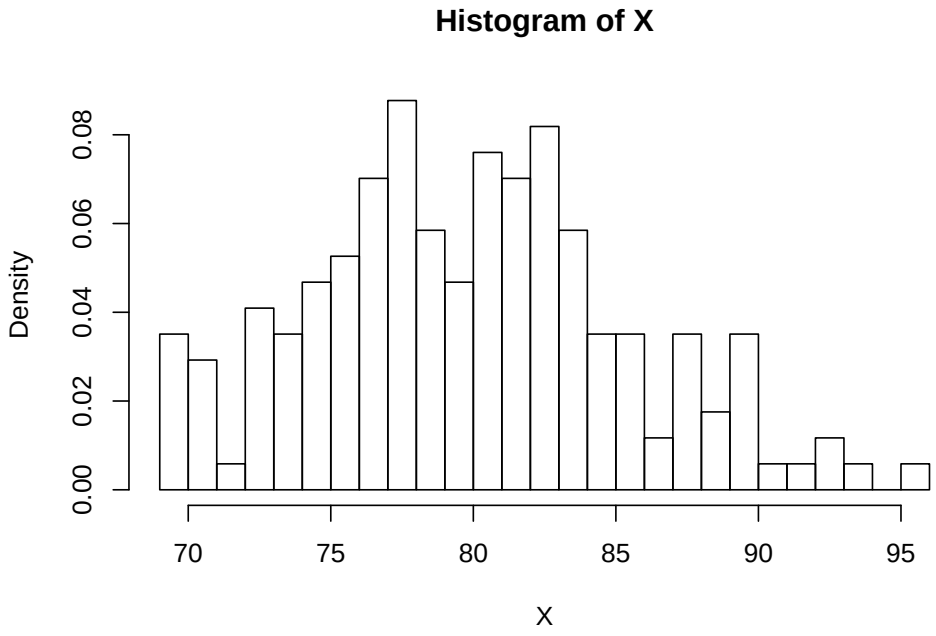
In this example let's only examine the *waiting* variable (waiting times between geyser eruptions in minutes).

```
X <- faithful$waiting
hist(X,breaks=60)
```



This data appears to be a mixture of two normal distributions, broken roughly at 68. Let's pull out the higher normal curve:

```
X <- X[X>68]
hist(X,breaks=30,freq=FALSE)
```

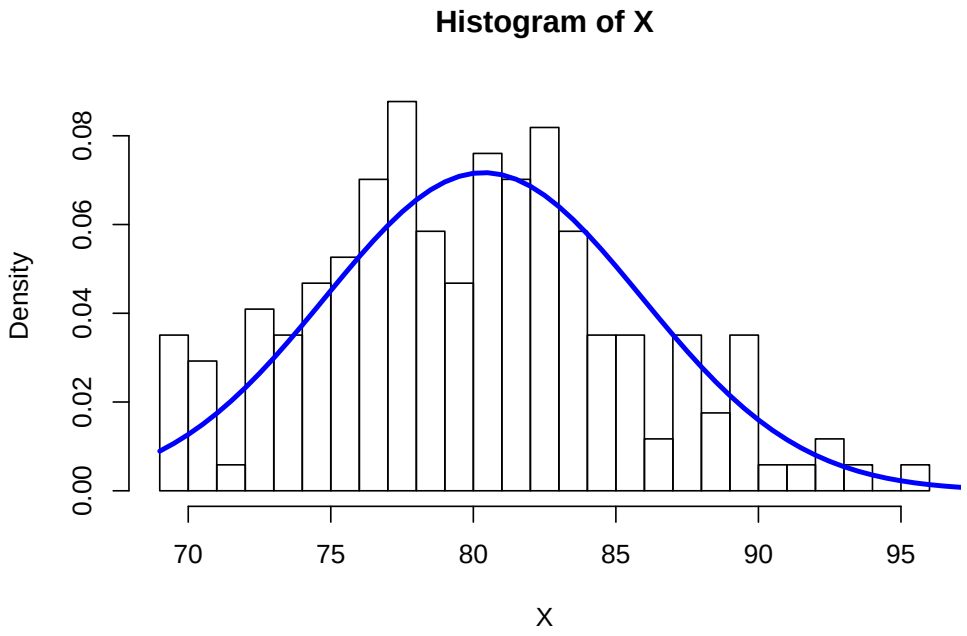


We can now estimate the mean and standard deviation of this density function.

```
mu <- mean(X)
sigma <- sd(X)
```

We now have the parameters to describe the normal density function.

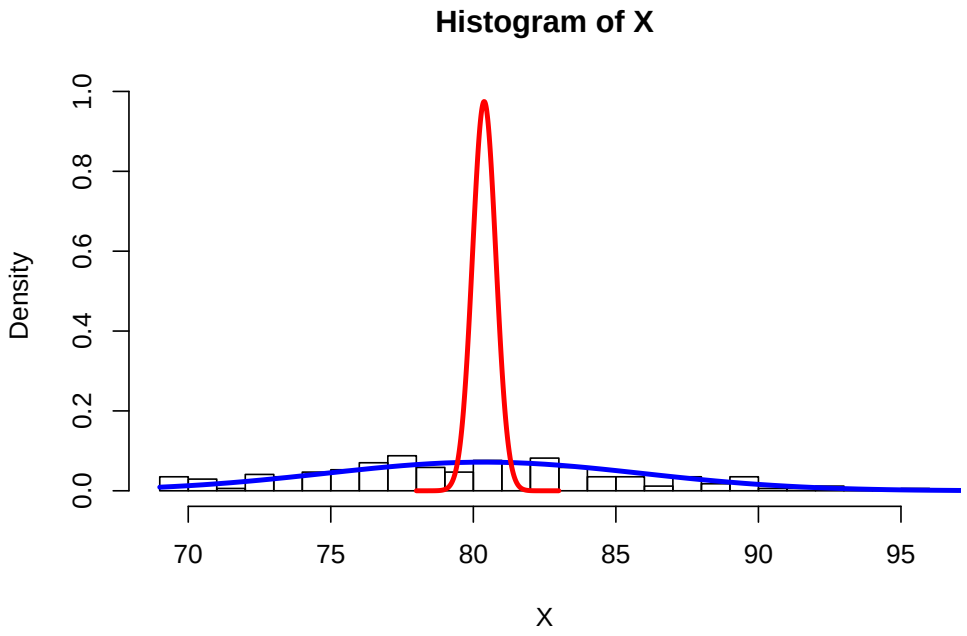
```
hist(X,breaks=30,freq=FALSE)
x <- seq(from=69,to=100,by=0.5)
y <- dnorm(x,mean=mu,sd=sigma)
lines(x,y,col='blue',lwd=3)
```



Not a terrible approximation! The mean is our point estimate of the waiting time. So we would say that on average, we expect about 80.3567251 minutes to elapse between geyser eruptions.

It's important to understand that our plot above is *not* the distribution of the mean waiting time random variable. To generate *that* distribution we first need to run an experiment in which we draw samples and compute means.

```
sample_size <- length(X)
one_mean <- Vectorize(function(x=1) {
  return(mean(sample(X,sample_size,replace=TRUE)))
})
sample_means <- one_mean(1:sample_size)
meanmu <- mean(sample_means)
meansigma <- sd(sample_means)
xmean <- seq(from=78,to=83,by=0.01)
ymean <- dnorm(xmean,mean=meanmu,sd=meansigma)
hist(X,breaks=30,freq=FALSE,ylim=c(0,max(ymean)))
lines(x,y,col='blue',lwd=3)
lines(xmean,ymean,col='red',lwd=3)
```



Note that the density for the sample mean is *much* more tightly clustered.

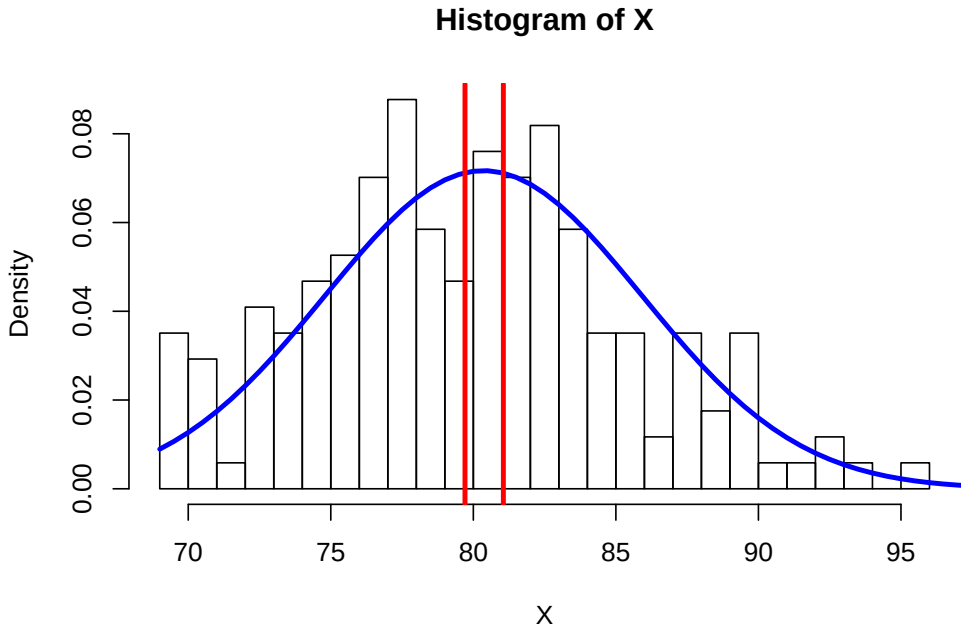
At this point we need to decide on a confidence level; let's take 90%. So we want to leave out 10% of the sample mean data at the ends (the more extreme values). Typically we produce a balanced interval estimate, and so we would want to trim off 5% of the data at each end. For the density function, this translates into finding the values corresponding to the 5th and 95th percentiles.

```
interval90 <- qnorm(c(.05,.95),mean=meanmu,sd=meansigma)
interval90
```

```
## [1] 79.70699 81.05338
```

We can visualize our confidence interval as

```
hist(X,breaks=30,freq=FALSE)
x <- seq(from=69,to=100,by=0.5)
y <- dnorm(x,mean=mu,sd=sigma)
lines(x,y,col='blue',lwd=3)
abline(v=interval90[1],col='red',lwd=3)
abline(v=interval90[2],col='red',lwd=3)
```



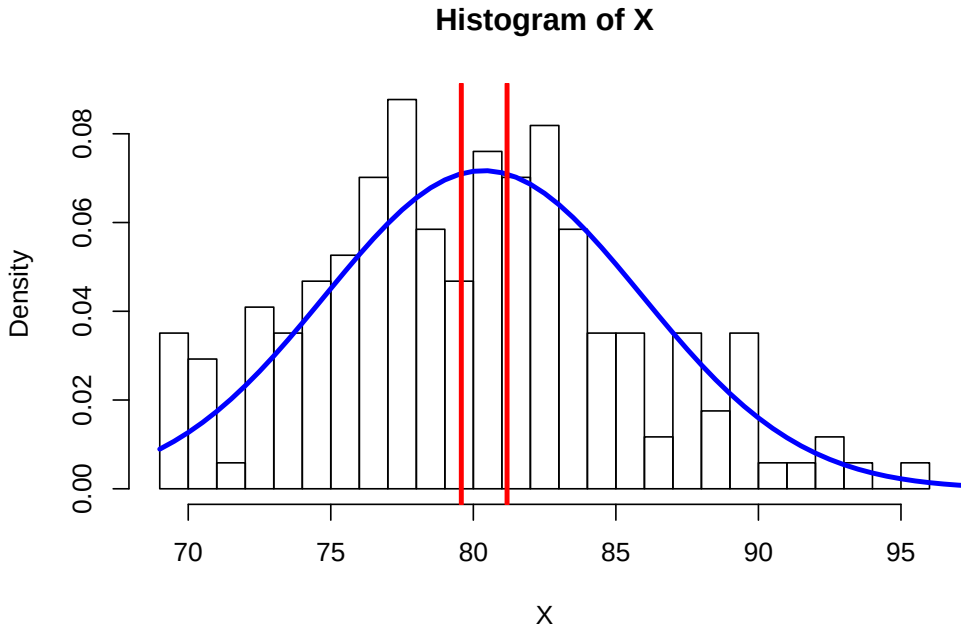
If we then wish to change our confidence, we can do so easily. For example, we can construct a 95% confidence interval by trimming off 2.5% of the data at the left and right.

```
interval95 <- qnorm(c(.025,.975),mean=meanmu,sd=meansigma)
interval95
```

```
## [1] 79.57802 81.18235
```

Note that the 95% confidence interval is a little wider than the 90% confidence interval.

```
hist(X,breaks=30,freq=FALSE)
x <- seq(from=69,to=100,by=0.5)
y <- dnorm(x,mean=mu,sd=sigma)
lines(x,y,col='blue',lwd=3)
abline(v=interval95[1],col='red',lwd=3)
abline(v=interval95[2],col='red',lwd=3)
```

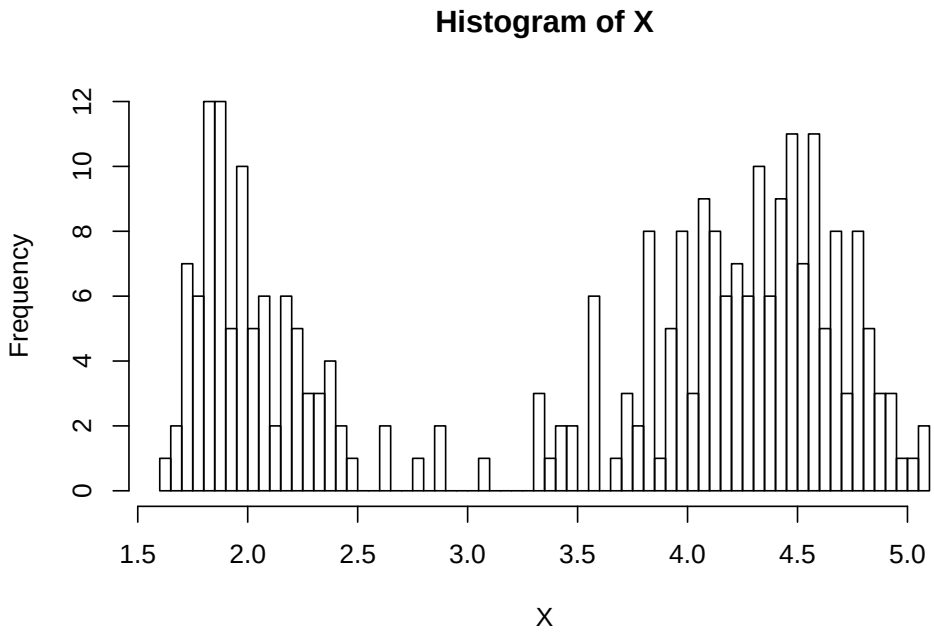


To be more confident that the parameter value falls in our constructed interval, we need to construct a wider interval.

Example

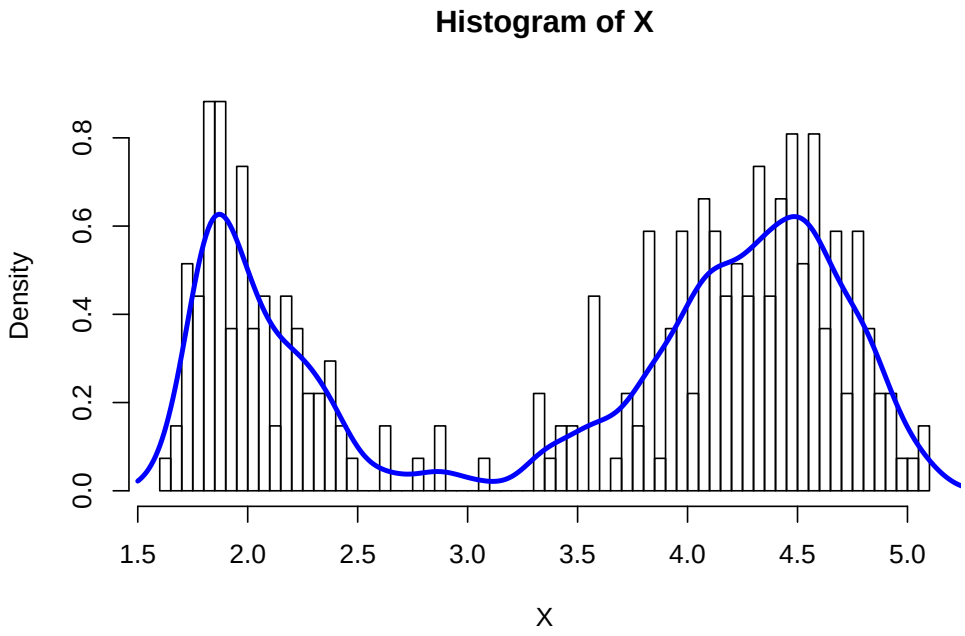
Now let's examine the Old Faithful data set again, this time focusing on the eruptions variable.

```
data(faithful)
X <- faithful$eruptions
hist(X,breaks=60)
```



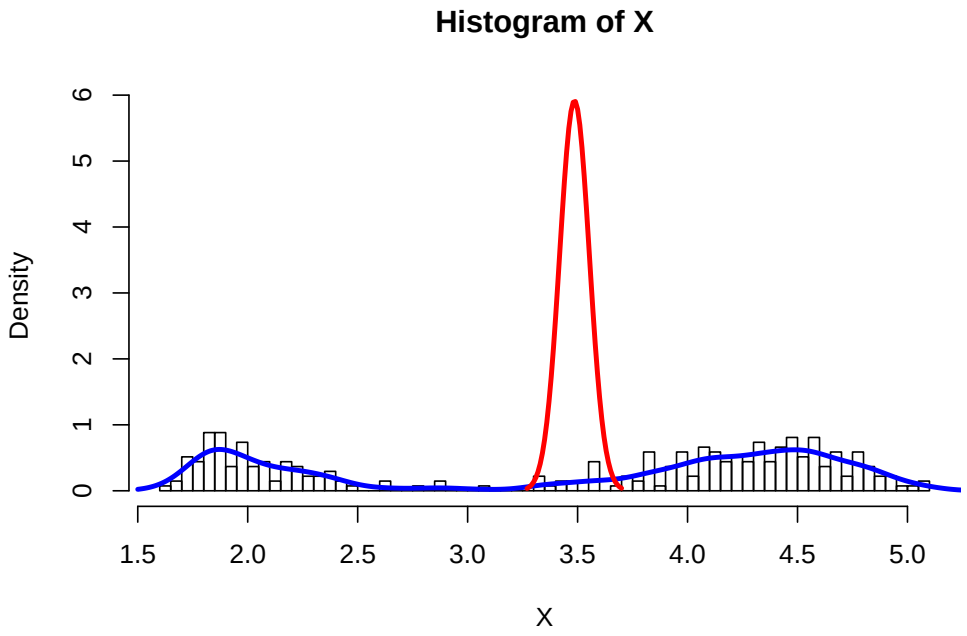
Again, we see a density function that seems to be a mixture of two normal distributions. This time, instead of focusing only on one, we will produce a numerical estimate of the density function.

```
dX <- density(X,adjust=0.3)
x <- seq(from = 1.5,to = 5.5,by=0.01)
f <- approxfun(dX$x,dX$y)
y <- f(x)
hist(X,breaks=60,freq=FALSE)
lines(x,y,col='blue',lwd=3)
```



Let's simulate our sample mean.

```
sample_size <- length(X)
one_mean <- function() {
  return(mean(sample(X,sample_size,replace=TRUE)))
}
sample_means <- sapply(1:1000,function(z) one_mean())
meanmu <- mean(sample_means)
meansigma <- sd(sample_means)
xmean <- seq(from=3.27,to=3.7,by=0.01)
ymean <- dnorm(xmean,mean=meanmu,sd=meansigma)
hist(X,breaks=60,freq=FALSE,ylim=c(0,max(ymean)))
lines(x,y,col='blue',lwd=3)
lines(xmean,ymean,col='red',lwd=3)
```



As in the prevous example we can see that the data for the sample means is much more tightly clustered.

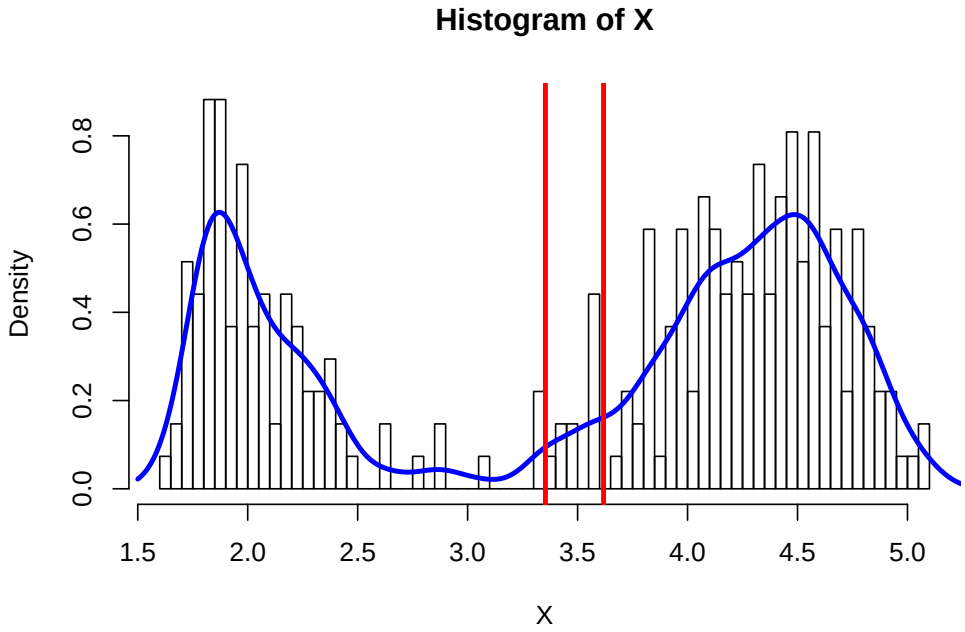
To produce a 95% confidence interval we now need to trim off the left and right 2.5% of the more extreme data points.

```
interval <- qnorm(c(.025,.975),mean=meanmu,sd=meansigma)
interval
```

```
## [1] 3.353679 3.618116
```

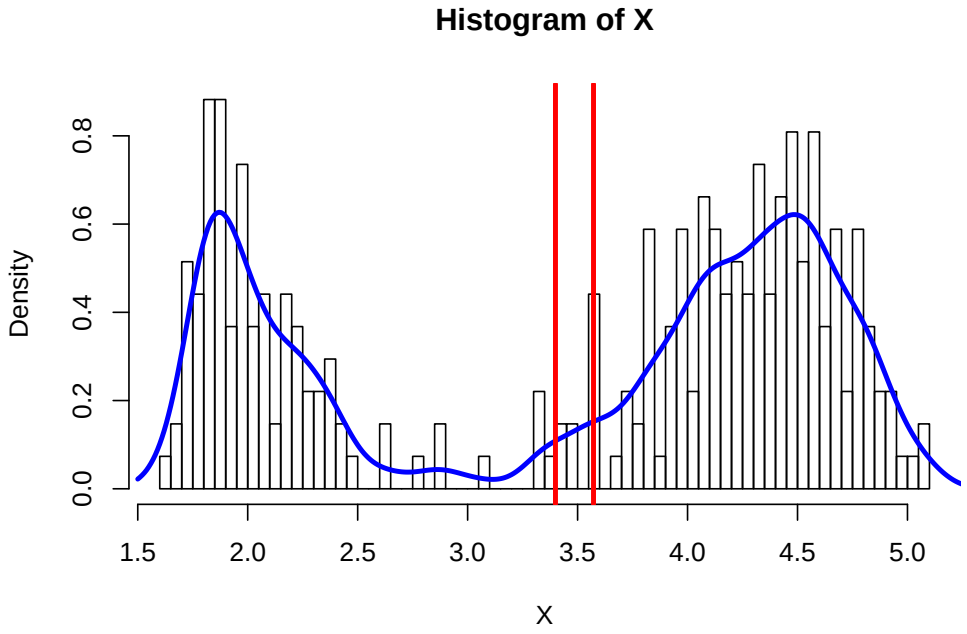
we can visualize our estimate as:

```
hist(X,breaks=60,freq=FALSE)
lines(x,y,col='blue',lwd=3)
abline(v=interval[1],col='red',lwd=3)
abline(v=interval[2],col='red',lwd=3)
```



We could use an interval estimate with a lower confidence instead. The 80% confidence interval is:

```
interval <- qnorm(c(.1,.9),mean=meanmu,sd=meansigma)
hist(X,breaks=60,freq=FALSE)
lines(x,y,col='blue',lwd=3)
abline(v=interval[1],col='red',lwd=3)
abline(v=interval[2],col='red',lwd=3)
```



We can see that our interval estimate does not include a location where the data is heavily clustered. Generally for this sort of distribution it is not a good idea to treat the mean of the whole distribution as a parameter of importance; it is typically better to model this distribution as a mixture of distributions and then model each separately. But for the purposes of an example, this works well enough!

The procedure we have used to estimate our confidence intervals in these examples is known as the *bootstrap*, invented by Bradley Efron. See [13] for the original source, and chapters 10 and 11 of [14] for an updated account with historical notes.

Exercises

Exercise 2.1 Bad Driving! A group of five roommates collectively have 10 car accidents in one year. One of the roommates - Kevin, it's always Kevin! - is responsible for 6 of the 10 accidents. His roommates tell him to start taking the bus, but Kevin claims it is just bad luck, and that he is no worse a driver than any of the others.

(a) Let's test Kevin's claim. Assume that each of the 10 accidents was equally likely to be caused by any one of the five roommates. Write a simulation to estimate the probability that Kevin causes at least 6 of the 10 car accidents in a year. If this probability is very small, then we have to say that EITHER we just observed a very unlikely event, OR our assumption that each of the 10 accidents was equally likely to be caused by any roommate is false. It's silly to assume we observe an unlikely event... so we would conclude that our assumption was false.

(b) How would your answer change if Kevin caused 3 of the 10 accidents?

(c) How would your answer change if Kevin caused 8 of the 10 accidents?

Exercise 2.2 More driving! You pull up to a red stoplight behind some other cars. How long will it take before you start moving again? Assume that the number of cars ahead of you at the red light is Poisson distributed with mean 3, the wait time for each car including yours to begin moving after the car ahead has begun moving is uniformly distributed between 2 and 10 seconds, and the time for the red light to change to green is uniformly distributed between 0 and 60 seconds. Give your expected wait time, the variance of your wait time, and produce a 95% confidence interval for your mean wait time.

Exercise 2.3 Let's imagine a building with 10 floors (including ground level) and one elevator, and not very many people going in or out. In this building the elevator is fairly primitive and obeys the following rules:

- Once the elevator is called from a certain floor, the elevator heads towards that floor and does not stop at any floors in between.
- The elevator parks on whatever floor it last visits and waits for the next call. That is, the elevator does not by default move to the ground floor, the tenth floor, or a middle floor to park and wait; it just waits in place.
- The elevator takes about 10 seconds to get from one floor to the next floor above or below. So for example, the elevator takes 30 seconds to travel from floor 1 (the ground floor here in the US) to floor 4, because it has three 10 second transitions to make: 1 to 2, 2 to 3, and 3 to 4.

We can also make the following assumptions about the people in the building and how they use the elevator:

- People use the elevator so infrequently that we can assume that no one calls the elevator while anyone else is using it. That is, we assume that you can push the call button, wait for the elevator, and then compete your ride, all without anyone else calling the elevator.
- People tend to move from any given floor in the building to any other given floor with equal likelihood. So the elevator is equally likely to be parked on any floor.

George, located on floor 2, wants to visit his friend Marvin, located on floor 9.

(a) What is the probability that George's total travel time, from pushing the call button to arriving on floor 9, exceeds 100 seconds?

(b) For the trip from floor 2 to 9, what is the expected travel time and the variance in the travel time?

(c) What if there is a second elevator, and whenever a call button is pushed, the closest elevator always responds? How does this change your answer to part (b)?

(d) You can extend this elevator problem in many ways to improve the realism. For example, the elevator is realistically less likely to be parked on floor 2 or 3 if there are accessible stairs. Write a probability distribution for the parked location of the elevator, and then recompute your answer to (b) with this new distribution.

This problem is based on Donald Knuth's article about the elevator frustrations experienced by George Gamow (physicist) and Marvin Stern (mathematician) when travelling to each other's offices. See [9] for details about the theoretical analysis.

Exercise 2.4 It's registration time for classes! There are two sections of elementary physics, each with 40 seats available.

(a) If students register randomly for section 01 and section 02, how many seats will be left in one section when the other section fills up? Give the expected value and variance for the number of seats remaining in the open section.

(b) Now let's suppose there are two types of students. Type A register in the section with the most students already registered, assuming that the other students know something about that section. Type B register in the section with the fewest students already registered, wanting more room to stretch out. About 60% of students are Type A and 40% are type B. Students

register for elementary physics one at a time, in a random order. How many seats will be left in one section when the other fills up in this scenario? Give the expected value and variance for the number of seats.

This problem is based on an infamous paper [10] by Donald Knuth about toilet paper dispensers.

Exercise 2.5 Mark-Recapture Population Estimation. We would like to estimate the population size of a certain species of animals in a certain region. Let's call the unknown population size N. We perform the following procedure to gather data and perform an estimate:

1. Capture S_1 of the species and mark them in some way.
2. Release the captured animals and wait a sufficient time so they can "mix back in" to the general population.
3. Capture S_2 of the species and count the number in this sample that have already been marked, M.
4. With certain assumptions in place, we reason that the proportion of marked individuals in the sample is the same as the proportion of marked individuals in the population:

   $$\text{Proportion marked in sample 2} = \frac{M}{S_2} = \frac{S_1}{N} = \text{proportion marked in population}$$

   Solving for N gives

   $$\hat{N} = \frac{S_1 S_2}{M}$$

   (We write $\hat{N}$ to denote that this is an estimate and not the true value.)

(a) Suppose that the population size is actually $N_{actual} = 1000$. Suppose that our sample sizes are $S_1 = S_2 = 50$, and assume that all animals are equally likely to be captured during each sampling. Use a simulation to find the expected value and variance for $\hat{N}$. Is the expected value what you would expect it (or want it) to be?

(b) Now, still using $S_1 = S_2 = 50$, assume that all animals are equally likely to be captured in the first round, but suppose that the animals which were captured in the first round are wary of humans and are only half as likely as the uncaptured animals to be captured in the second round. Use a simulation to find the expected value and variance for $\hat{N}$. (You'll need to work out the probabilities for recapture for marked and unmarked animals.)

For information about mark-recapture methods, visit the Wikipedia page.

Exercise 2.6 Parrondo's Paradox. Let's suppose that you begin with an amount of money M. Here are two games you can play:

Game A: Flip a coin that is biased to come up Heads 45% of the time and Tails 55% of the time. If you flip Heads you win \$1 and if you flip Tails you lose \$1.

Game B: You have two biased coins. If M is a multiple of 3, then you flip coin 1, which is biased to come up Heads 5% of the time and Tails 95% of the time (as in Game A, Heads means you win \$1 and Tails means you lose \$1). If M is not a multiple of 3, then you flip coin 2, which is biased to come up Heads 70% of the time and Tails 30% of the time (ditto on payouts).

(a) Explain why repeatedly playing Game A is a losing game, i.e. you expect your amount of money M to decline to 0.

(b) Is repeatedly playing Game B a losing game? Perform a simulation to find out!

(c) Now suppose that you repeatedly play Game A or Game B and you randomly flip back and forth between them each round. Is this a losing game? Perform a simulation to find out!

This problem is known as Parrondo's Paradox, first formulated in 1996 by Juan Parrondo. If we expect to lose each game in repeated play, how can we expect to win when randomly switching between these losing games?

Exercise 2.7 (This exercise involves a bit of guesswork and algebra, so it is a bit more challenging.) Suppose that we are randomly sampling 20 values without replacement from the numbers 1, 2,...,N, where N is a value greater than 100. We measure M = the maximum value in the sample.

(a) Create a simulation that measures M.

(b) If $N = 150$, what is the expected value of M?

(c) Repeat (b) for \$N=\$200,250,...,950,1000.

(d) Make a plot of the expected value of M versus N. From your plot guess a simple relationship of the form $M = f(N)$.

(e) Invert your function, solving for N, to obtain a formula of the form $N = f^{-1}(M)$. This formula tells you how to compute an estimator for the maximum value in the pool, $\hat{N} = f^{-1}(M)$.

(f) Use your formula to compute your estimator $\hat{N}$ by simulating M once for each of $N = 150, 200, \ldots, 1000$ and computing $\hat{N}$ for each of those M values. How good is your estimator?

This is a very simple version of the German tank problem. During World War II, Allied spies were trying to estimate the number of German tanks produced in certain factories. Being very precision minded, German engineers put a serial number on each tank component, numbering them 1 through N. By seeing a few serial numbers and using an appropriate estimator, the Allies were able to accurately estimate numbers of tanks produced and take appropriate precautions.

Exercise 2.8 Use a simulation to find the area under the graph of the function $f(x) = e^{-x^2}$ between $x = 0$ and $x = 3$, using 10^2, 10^3, 10^4, and 10^5 darts. How does your area estimate change as you increase the number of darts?

Exercise 2.9 In the previous exercise you probably chose your image window to be something like $0 \leq x \leq 3$, $0 \leq y \leq 3$ (why?). But what if you had chosen your window to be $0 \leq x \leq 3$, $0 \leq y \leq 3$? Repeat the previous exercise with this new window, and keep track of the percentage of rejected darts for each estimate. Can you see why it is valuable to have a tight window around the area you want to estimate?

Exercise 2.10 Use the bootstrap to give a 95% confidence interval estimate for the mean of the Speed variable in the morley dataset.

Exercise 2.11 Use the bootstrap to give a 95% confidence interval estimate for the mean level of Lake Huron, in the LakeHuron dataset.

3. Sampling

In this section we will dig a little more deeply into how random number generators for various distributions can be programmed. This section is a little sparser, but contains links to lots of useful web content.

Distributions in R

Here is some information about the "built-in" distributions in R: https://stat.ethz.ch/R-manual/R-devel/library/stats/html/Distributions.html . In an introductory statistics class, you will probably have encountered the binomial, Poisson, normal, and chi-squared distributions, at minimum. Be sure that you understand the purpose of the methods associated with each distribution.

The Monte Carlo Method

The Monte Carlo method is really not a method, but an entire class of computational algorithms that focus on answering questions through simulation. Read more here: https://en.wikipedia.org/wiki/Monte_Carlo_method . References [3] and [4] contain some excellent material discussing Monte Carlo methods. Reference [4] contains code and exercises written in Matlab; Python users using the numpy and pylab packages will find this easily readable. [5] is a more advanced mathematical text, aimed at clarifying the connections between Monte Carlo methods and the Bayesian approach to statistical analysis. [6] moves in the computational direction instead, clarifying the connections between Monte Carlo methods and the Bayesian approach using Python and the pymc3 package.

In the simulation section of these notes we have seen examples of estimating probabilities through simulation and optimizing functions through simulation. However, we have not really seen simulations used to generate samples from distributions of continuous random variables. How do computer programs work "under the hood"? How does R generate 10000 draws from a beta distribution? We'll tackle this in the next two sections.

Box-Ticket Sampling

Suppose that we have a discrete random variable with a known distribution function. Then in many cases we can sample from this distribution by creating a

"box" filled with "tickets" and repeatedly draw tickets with replacement to form the sample.

For example, suppose that we have a random variable with density

$X =$	$f(x)$
0	0.1
1	0.3
2	0.5
3	0.1

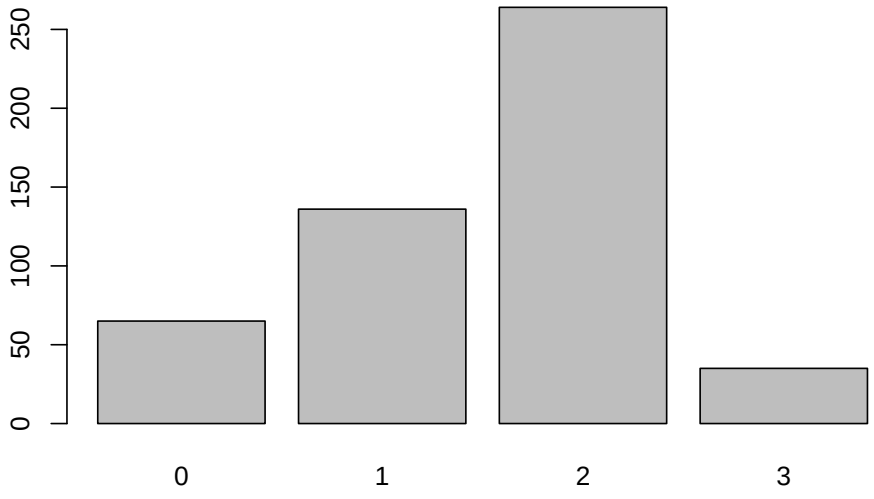
Then we can build a box with 100 tickets, 10 of which are 0's, 30 of which are 1's, 50 of which are 2's, and 10 of which are 3's. In R:

```
box <- c(rep(0,10),rep(1,30),rep(2,50),rep(3,10))
```

Note that the proportion of the each outcome's tickets in the box is the same as the proportions of that outcome in the probability mass table.

To draw a sample from the box of a given size (with or without replacement), we can use the sample function.

```
data <- sample(box,500,replace=TRUE)
barplot(table(data))
```



This is a very useful technique when we have a data set and we want to simulate a sample from the population. For example,

```
data(Loblolly)
X <- as.numeric(as.character(Loblolly$Seed))
```

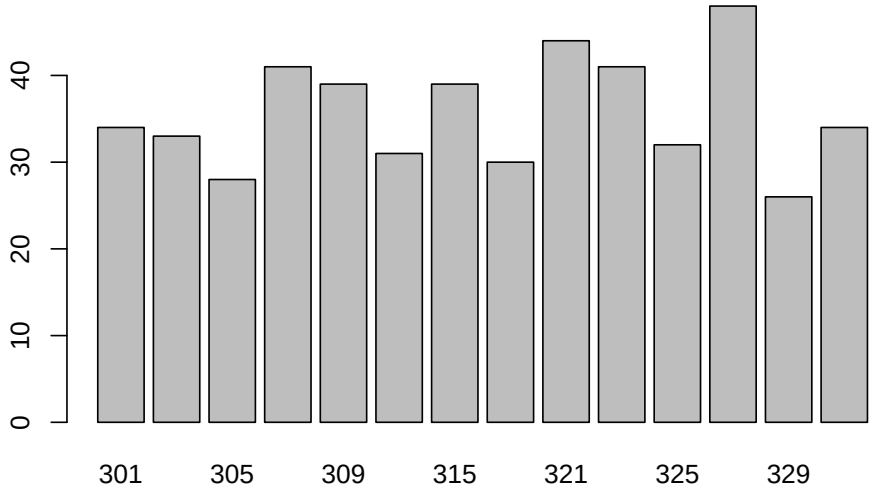
X is a random variable with

```
length(unique(X))
```

```
## [1] 14
```

unique outcomes. If we wish to simulate a sample of size 500 from this distribution, we can simply use sample:

```
data <- sample(X,500,replace=TRUE)
barplot(table(data))
```



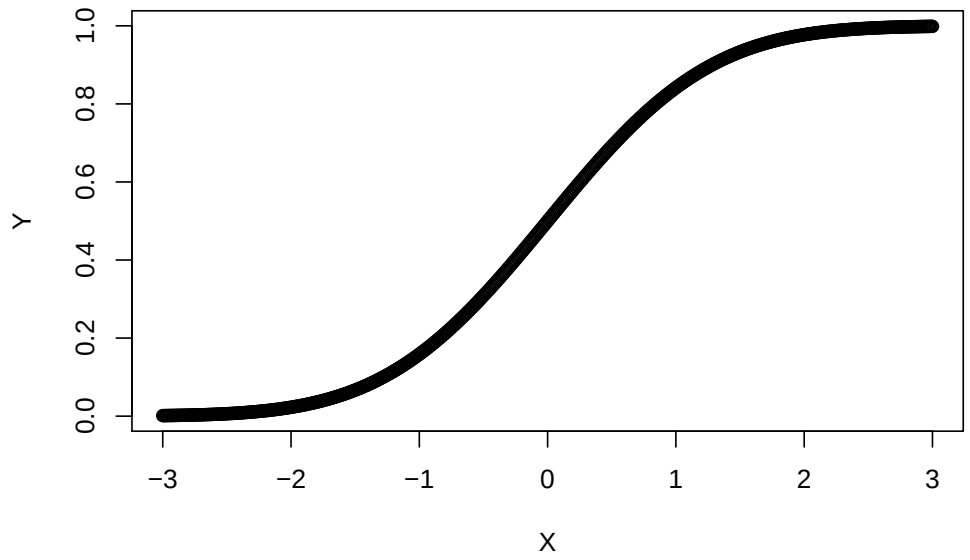
This is a tremendously useful tool, but we can't "see" what R is doing with the sample function. This next section delves into that a bit.

Inversion Sampling

Suppose that X is a continuous random variable with distribution function $F(x) = P(X \leq x)$. Then F is an increasing function of x, and will have an "s shape",similar

to the following.

```
X <- seq(from=-3,to=3,by=0.01)
Y <- pnorm(X)
plot(X,Y)
```



These curves are sometimes called *s-shaped* curves, informally.

Let's note three facts about this distribution function F. First,

$$\lim_{x \to -\infty} F(X) = 0$$

Second,

$$\lim_{x \to \infty} F(x) = 1$$

And third, F is an invertible function.

Inversion sampling uses these three facts and the random number generator for the uniform distribution on $[0, 1]$. The process is as follows:

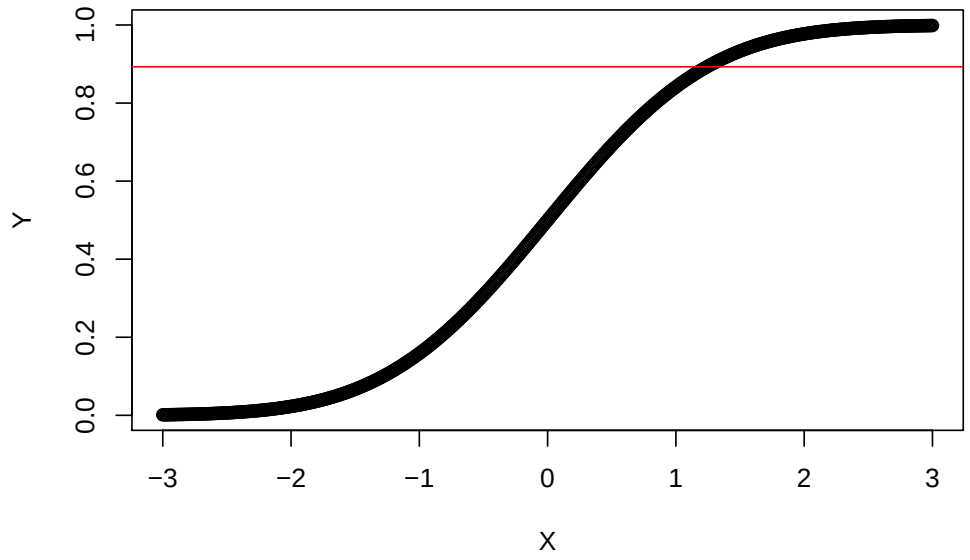
Set-up: You'll need your distribution function written in some format (vectors of x and y coordinates are easiest). For this example I'll take mine to be written in the vectors of X and Y coordinates used to generate the graph above.

Step 1: Draw a random number from the uniform distribution on $[0, 1]$. This will represent a random output of your distribution function, so I will call mine y.

```
y <- runif(1)
```

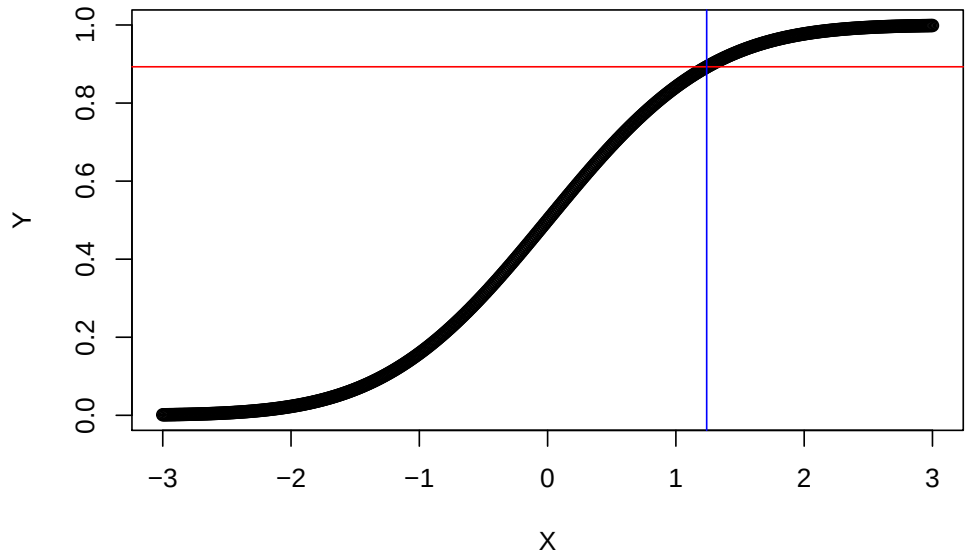
Step 2: We think of plotting y on the vertical axis in the graph above and then finding the x value corresponding to it.

```
set.seed(42)
plot(X,Y)
abline(h=y,col='red')
```



Typically this involves some sort of search through a list. We can visualize this as finding the x-coordinate in the distribution corresponding to the y coordinate we drew.

```
location <- which.min(abs(Y-y))
the_draw <- X[location]
plot(X,Y)
abline(h=y,col='red')
abline(v=the_draw,col='blue')
```

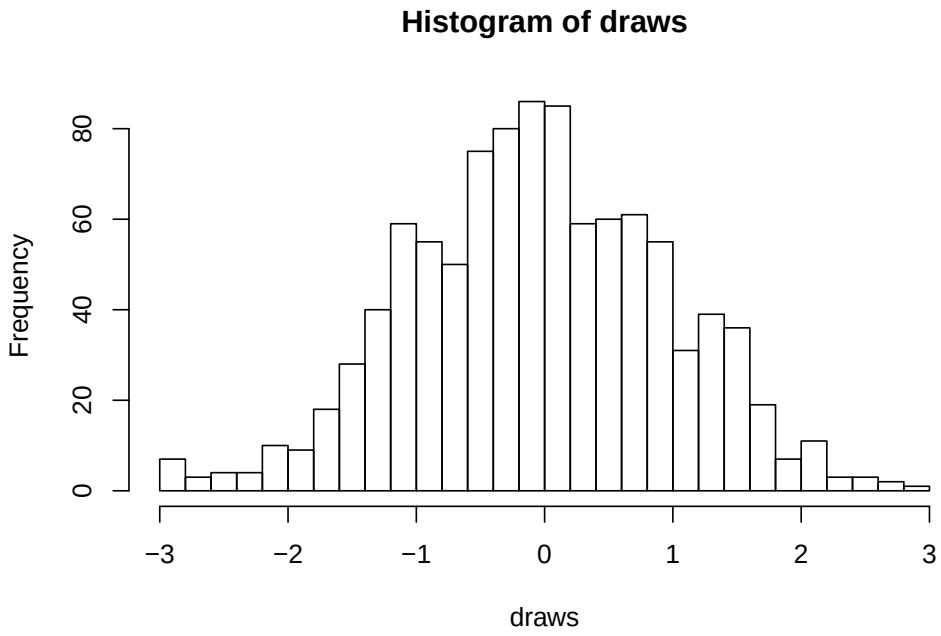


Here we have drawn a sample value of $x=$1.24. If we repeat this many times, we generate many draws from the distribution.

```
set.seed(42)
y <- runif(1000)
find_X <- function(y) {
  location <- which.min(abs(Y-y))
  return(X[location])
}
find_X <- Vectorize(find_X)
draws <- find_X(y)
```

Our data is in the variable draws; what does it look like?

```
hist(draws,breaks=40)
```



Our sample seems to fall roughly into the shape of a normal distribution. Because our random variable was normal, this seems like a good confirmation of this sampling technique.

Example

How does this work for discrete random variables and for data? As an example, let's look at the yearly sunspot data set in R.

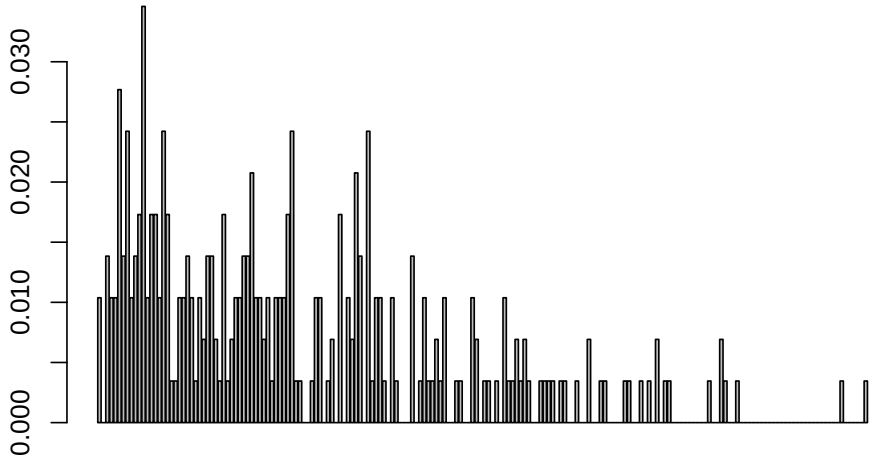
```
data(sunspot.year)
spots <- ceiling(sunspot.year)
head(spots)
```

```
## [1]  5 11 16 23 36 58
```

This is just the number of sunspots recorded each year, in order from 1700 to 1988. Let's first find the mass function.

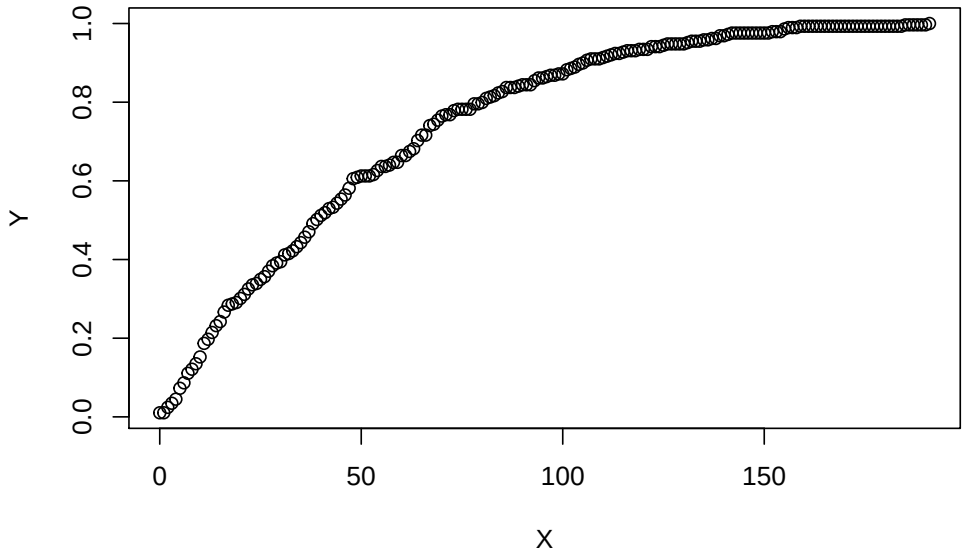
```
countem <- Vectorize(function(x) {
  sum(spots==x)
})
X <- 0:max(spots)
temp <- countem(X)
f <- temp/sum(temp)
```

```
barplot(f)
```



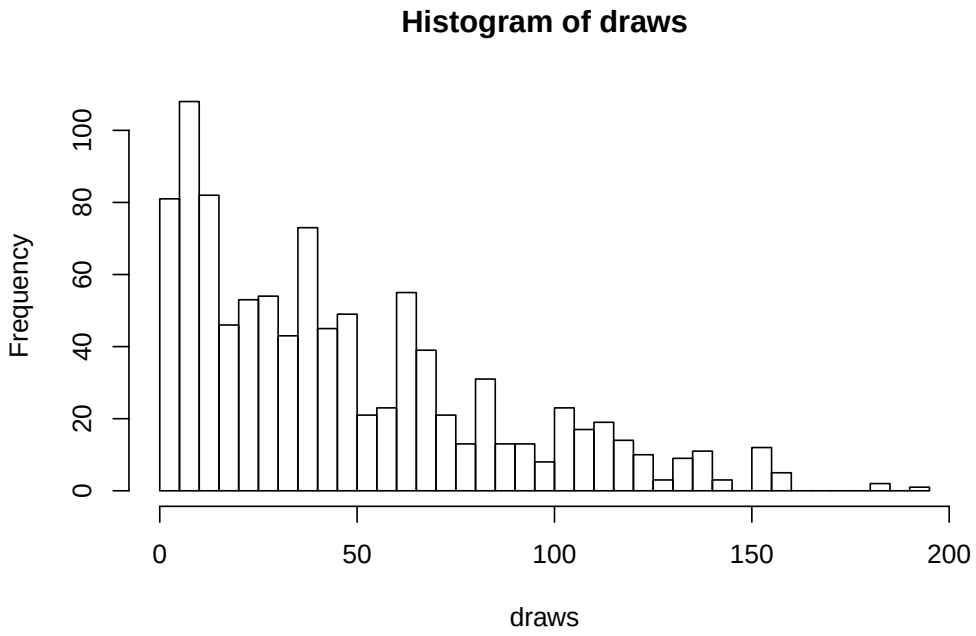
We can now find the distribution function by summing.

```
Y <- cumsum(f)
plot(X,Y)
```



And now that we have the distribution function we can perform inversion sampling as we did for the normal distribution.

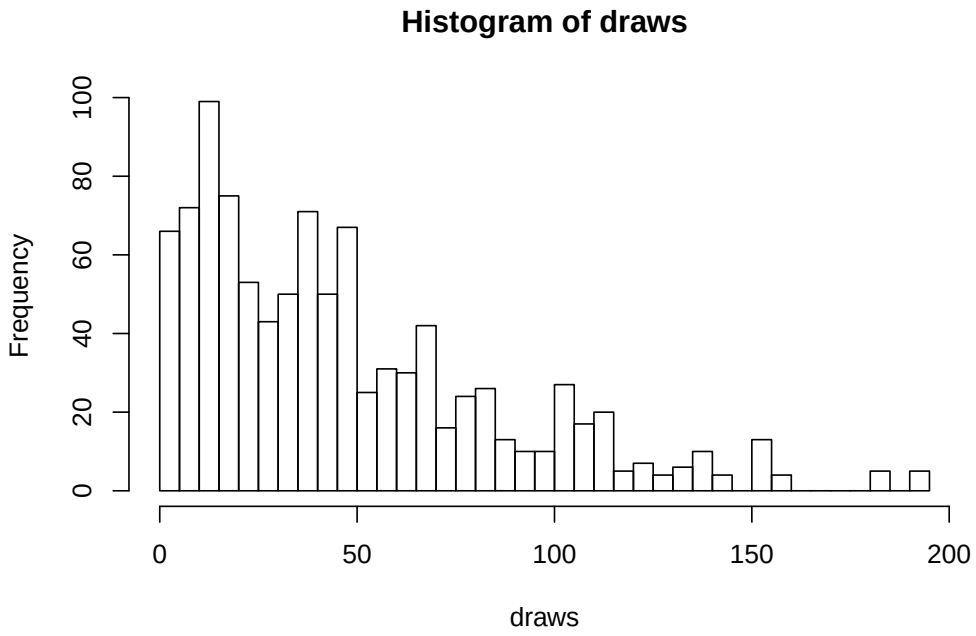
```
set.seed(42)
y <- runif(1000)
find_X <- Vectorize(function(y) {
  location <- which.min(abs(Y-y))
  return(X[location])
})
draws <- find_X(y)
hist(draws,breaks=60)
```



This looks similar to our original distribution of sunspots, so it seems good.

However, I have to stress that this is definitely not the most efficient way for you to sample from data representing a discrete random variable; it's a "hypothetically, we could..." discussion. The easiest approach is to just use the sample function on your original data set, as we did in the box-ticket model section! The procedure we ran through is more or less what R does under the hood with the sample function.

```
draws=sample(spots,1000,replace=TRUE)
hist(draws,breaks=60)
```



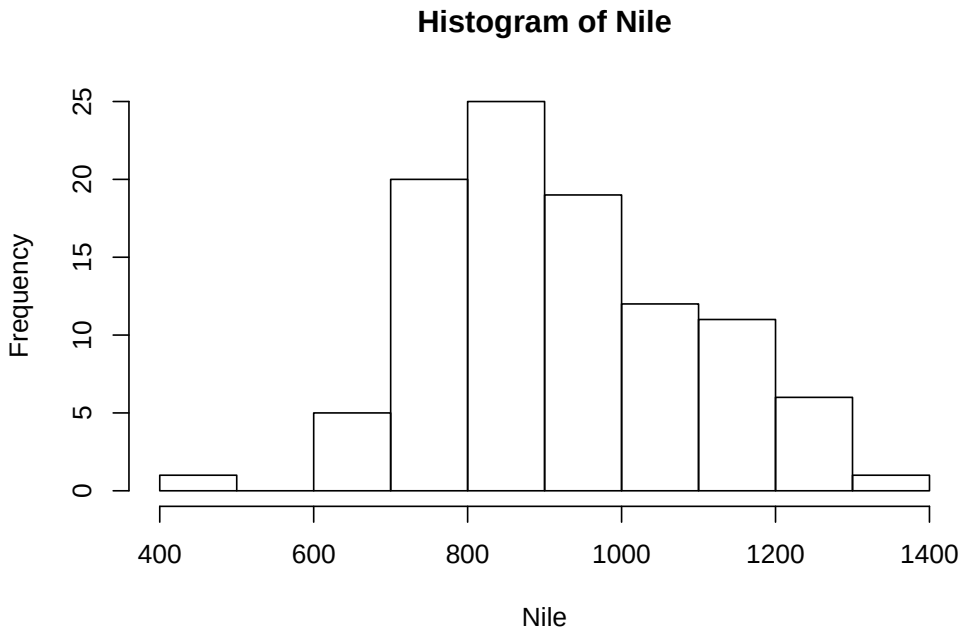
Example

Okay, but what if our random variable is continuous, we have some data sampled from it but no formula, and we wish to sample from the random variable? Let's examine the Nile River water flow data set.

```
data(Nile)
head(Nile)
```

```
## [1] 1120 1160  963 1210 1160 1160
```

```
hist(Nile)
```

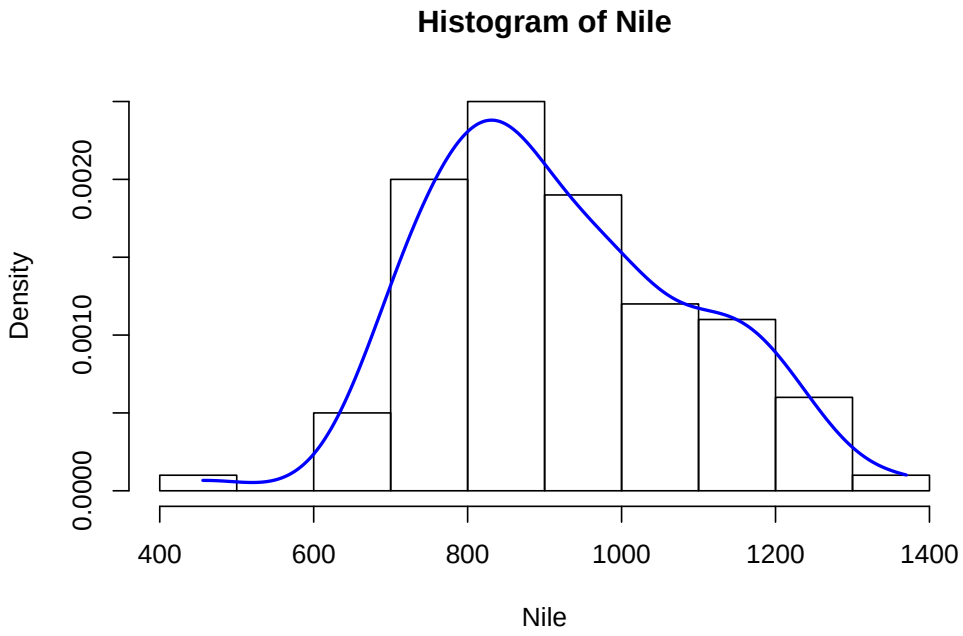


We can fit a smooth density function to the Nile data. (Use ?density in R to read more about this function.)

```
Nile_density <- density(Nile)
```

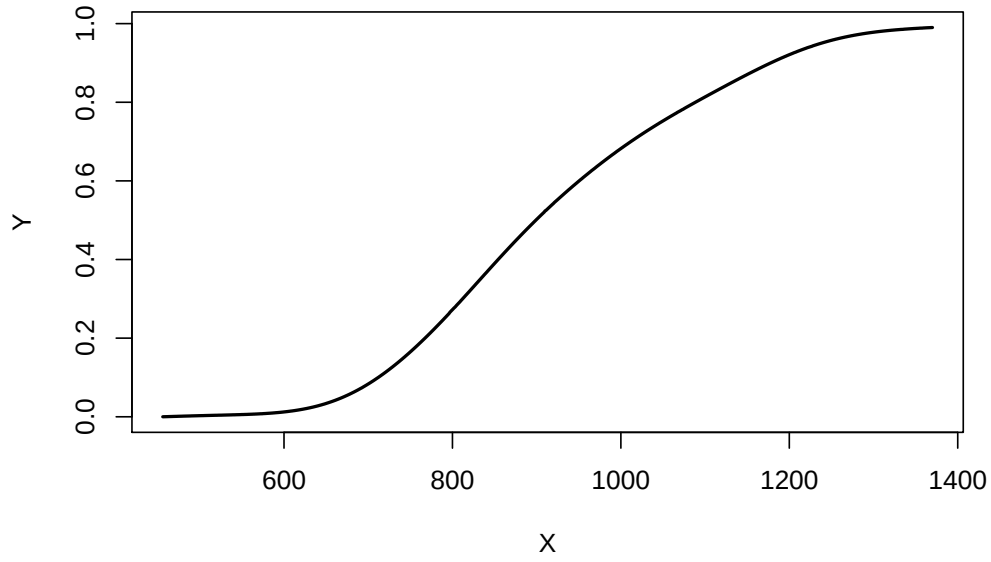
We need to create an approximating function and then create the vectors of inputs and outputs for inversion sampling. We have to be a bit careful here, because we want our density function to be 0 outside of a certain range; thus the definition of f in the code below.

```
Nile_approx <- approxfun(Nile_density)
f <- Vectorize(function(x){
  if (x<min(Nile) | x>max(Nile)) {
    r <- 0
  } else {
    r <- Nile_approx(x)
  }
  return(r)
})
X <- min(Nile):max(Nile)
Y <- sapply(X,function(z) integrate(f,lower=-Inf,upper=z,stop.on.error = FALSE)$value)
hist(Nile,freq=FALSE)
lines(X,f(X),col='blue',lwd=2)
```



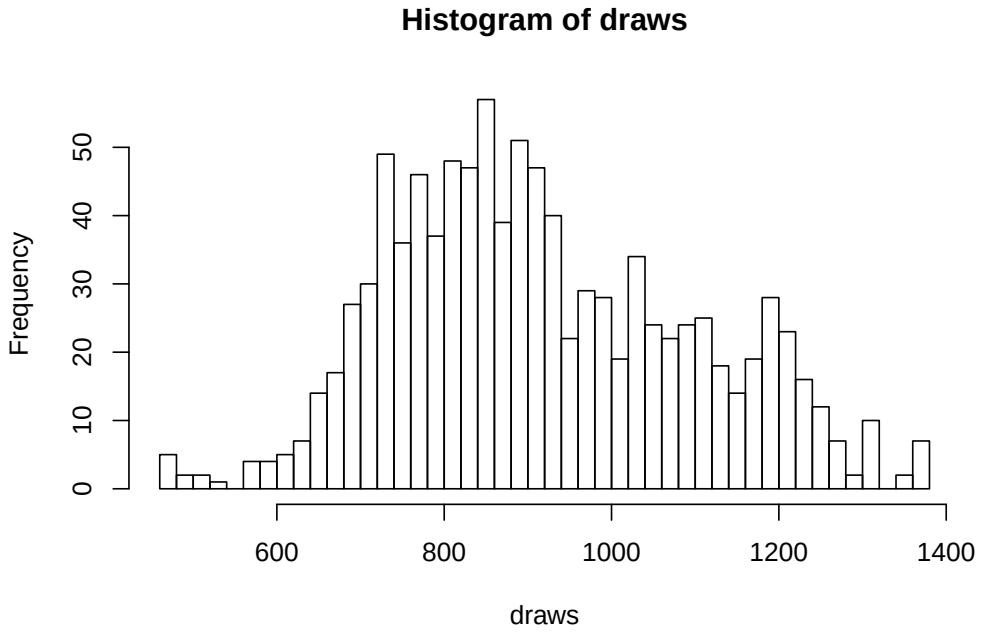
And here is the distribution:

```
plot(X,Y,type='l',lwd=2)
```



We can now sample from this distribution function using inversion sampling.

```
set.seed(42)
y <- runif(1000)
find_X <- Vectorize(function(y) {
  location <- which.min(abs(Y-y))
  return(X[location])
})
draws <- find_X(y)
hist(draws,breaks=60)
```



The benefit of this approach over sampling from the original data set is that the approximating density function has smoothed out the data. We now get some data points in the gaps in between the values in the original data set. This fits our mental model of flow rates of the Nile; there's no reason that the river may have a flow rate of 702 one year and 714 another year, but never 710. If we sample from the original data set we'll never draw a sample value of 710 because 710 is not a data point, but with the smoothed density, we might.

Rejection Sampling

Since we already have a sampling method, why introduce another? Well, inversion sampling works quite efficiently for a single random variable but not for multivariable scenarios.

Instead, we'll use a technique called *rejection sampling* (or sometimes "acceptance-rejection sampling"). This technique is quite similar to the darts-throwing technique we used to estimate integrals earlier. The idea is that we will throw a "dart" into a region with a density function. If the dart falls under the graph of the density function, we will accept the appropriate coordinates of the dart as a sample point, and if the dart fals above the graph of the density function we will reject the dart for our sample and move on to another.

Let's begin by defining a density function. We start with a shape.

```
f_temp <- function(x1,x2) {
  if (x1>=0 & x1<=2 & x2>=0 & x2<=2) {
    return(5*exp(-x1-x2))
  } else {
    return(0)
  }
}
```

We want the volume under this to be 1. Let's throw darts to find the current volume!

```
set.seed(42)
number_of_darts <- 100000
X1 <- runif(number_of_darts,min=0,max=2)
X2 <- runif(number_of_darts,min=0,max=2)
Y <- runif(number_of_darts,min=0,max=5)
outputs <- sapply(1:length(X1),function(j) f_temp(X1[j],X2[j]))
volume <- length(which(Y<=outputs))/number_of_darts * 2*2*5
volume
```

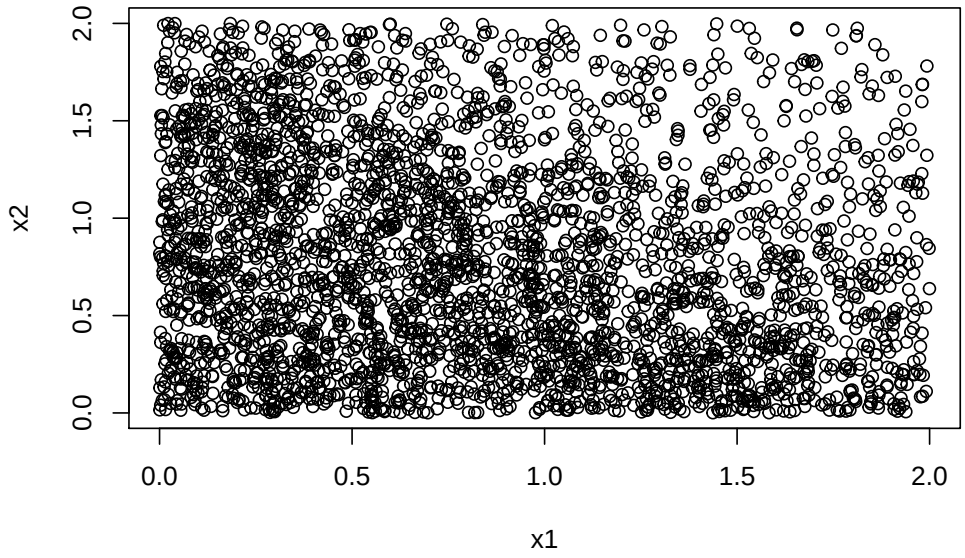
```
## [1] 3.7334
```

And now we can scale our function so that it is a proper density function, i.e. integrates to 1.

```
f <- function(x1,x2) {
  return(f_temp(x1,x2)/volume)
}
```

At this point we have a density function and we can perform rejection sampling. Much of this code will look similar to the code we used to create darts. Pay attention to the y values, though! Let's draw a sample of 1000 points.

```
set.seed(42)
number_of_darts <- 5000
X1 <- runif(number_of_darts,min=0,max=2)
X2 <- runif(number_of_darts,min=0,max=2)
Y <- runif(number_of_darts,min=0,max=1)
outputs <- sapply(1:length(X1),function(j) f_temp(X1[j],X2[j]))
keepers <- which(Y<=outputs)
x1 <- X1[keepers]
x2 <- X2[keepers]
plot(x1,x2)
```



Note the heavier clustering near $(0, 0)$. The density function is maximal there, and so we should expect tighter clustering in our sample.

There are a number of ways to improve a rejection sampling scheme. Chapter 2 in [3] has more details on rejection sampling, and chapter 2 in [4] has several examples of designing sampling algorithms using inversion schemes in some cases and rejection sampling in others.

Exercises

Exercise 3.1 Draw a sample of size 100 without replacement from the *stations* variable of the *quakes* dataset in R. Display your sample and the stations variable using barplots. How similar are they?

Exercise 3.2 Repeat the process from the Inversion Sampling section to draw a sample of size 1000 from the Student t distribution.

Exercise 3.3 Repeat the process from the Inversion Sampling section to draw a sample of size 1000 from the random variable with density function

$$f(x) = \begin{cases} \frac{3}{8}x^2 & 0 \leq x \leq 2 \\ 0 & \text{otherwise} \end{cases}$$

Exercise 3.4 Use rejection sampling to draw a sample of size 1000 from the random variable with density function

$$f(x) = \begin{cases} \frac{3}{8}x^2 & 0 \leq x \leq 2 \\ 0 & \text{otherwise} \end{cases}$$

4. An Application to Data Modeling

In this section we present a longer example applying the ideas in sections 1 through 3. One common problem we face is finding a probability distribution that best models some given data set. Why do we want to do this? Probability distributions have different features, and finding that our data set is similar to a certain distribution can help us understand the processes that generated the data and make better predictions about future samples drawn from the same population.

So, given some univariate (one variable) data set, our goal is to find a distribution that is a good model for our data, or at least better than other models we have proposed so far. We need some ingredients: * Candidate distributions: What distributions seem to be good models for the data? Ideally a good model is one that fits the data reasonably well and one that we understand well. * Metric for assessing the "goodness" of the model. A metric, in mathematical terms, is a function taking two inputs and returning two outputs which behaves like a distance function. So we are really looking for some way of measuring distance between our data and our candidate models. The model that is closest to the data is our best fit. * Procedure for applying the metric.

The example data set

For our first example we consider the .

```
wage <- read.csv("wage.csv")
wage <- wage$wage
```

This data set is univariate with 3000 entries.

Obtaining candidate distributions

Finding candidate distributions involves a combination of background knowledge, persistence, consultation of journal articles and experts, and luck. It's worth noting that there are still ongoing debates about the best models for certain famous data sets, like the distribution of numbers of friends users have on Facebook.

But the starting point for identifying candidate distributions is usually the same: make lots of graphs! We begin with a histogram of the data.

```
hist(wage,breaks=30)
```

The histogram tells us that we may be dealing with two distributions mixed together. Let's select only the lower distribution to work with for now.

```
wage <- wage[wage<240]
hist(wage,breaks=30)
```

Our metrics for goodness-of-fit

One metric for evaluating the goodness-of-fit of a proposed distribution to data is the Kolmogorov-Smirnov (KS) test. The KS test takes as inputs two sets of data and produces a probability measuring how likely it is that the two samples came from the same distribution. As an example, let's draw samples from the same exponential distribution.

```
set.seed(42)
sample_size <- 200
rate <- 2
X <- rexp(sample_size,rate = rate)
Y <- rexp(sample_size,rate = rate)
ks.test(X,Y)
```

```
##
##  Two-sample Kolmogorov-Smirnov test
```

```
##
## data:  X and Y
## D = 0.075, p-value = 0.6272
## alternative hypothesis: two-sided
```

Here, the p-value is relatively high, so our conclusion is that it is reasonably likely that the sample were drawn from the same distribution; seems like a good conclusion to make, since they were! As a second example, let's draw sample from exponential distributions with different rates.

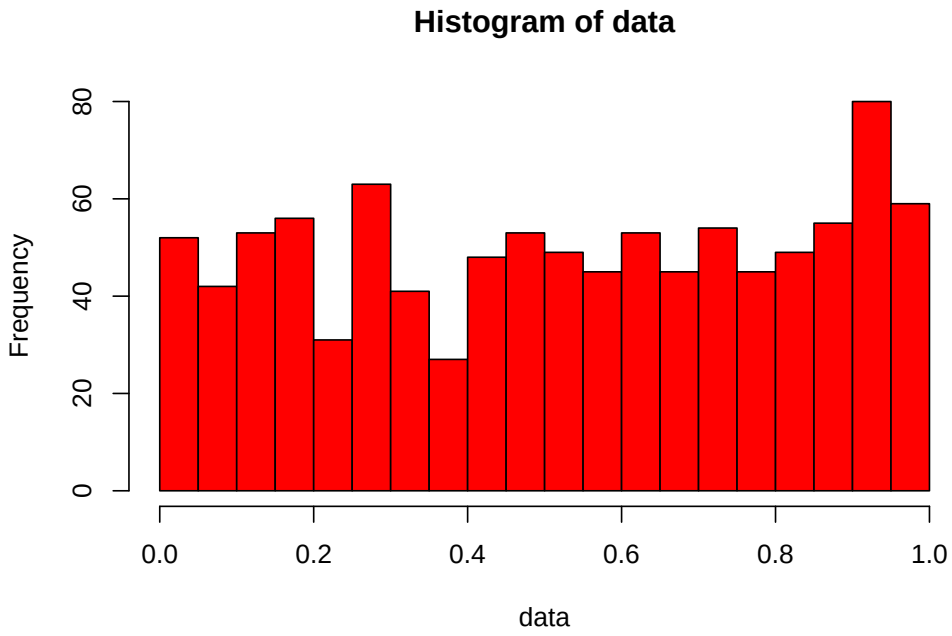
```
set.seed(42)
sample_size <- 200
X <- rexp(sample_size,rate = 2)
Y <- rexp(sample_size,rate = 3)
ks.test(X,Y)
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  X and Y
## D = 0.18, p-value = 0.003068
## alternative hypothesis: two-sided
```

Here, we see a very low p-value and so we conclude that it is very unlikely that the samples were drawn from the same distribution; again, great conclusion since they were not!

Unfortunately, the KS test can be highly sensitive to differences in distributions which may not make much difference in practical modeling, and which may arise simply from random chance in sampling. As an example, consider a normal distribution and a t distribution with the same mean and variance. We draw two samples of size 500, use the KS test, and then repeat this experiment 500 times.

```
set.seed(42)
sample_size <- 500
run_experiment <- Vectorize(function(x=1) {
  X <- rnorm(sample_size)
  Y <- rnorm(sample_size)
  ks.test(X,Y)$p.value
})
data <- run_experiment(1:1000)
hist(data,breaks=30,col='red',
     xlim=c(0,1))
```



It looks like we produce p-values all over the interval! What percent of them are "mistakes"? We usually use a boundary of p=0.10, with p-values lower indicating that the distributions are different. In our experiment,

```
length(which(data<0.1))/length(data)
```

```
## [1] 0.094
```

we have a little more than 9% errors. This is not great performance.

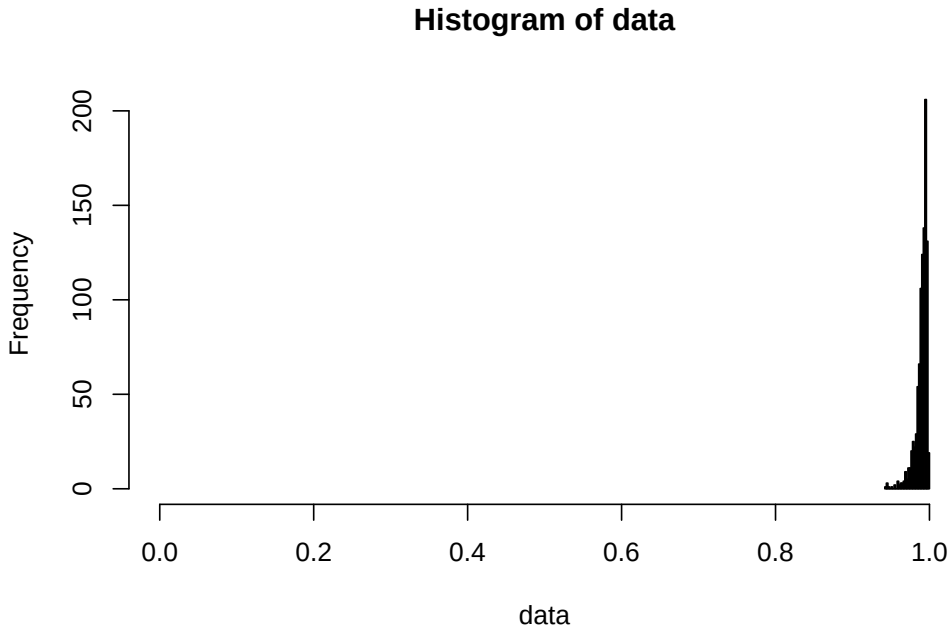
So additionally, we will look at plots comparing the histograms of the data and the candidate sample, and a qq-plot comparing the two. We will also fit a regression line to the qq-plot and measure the R^2 value as a metric for goodness-of-fit.

Procedure for applying the metric

We proceed as follows: 1. Propose a distribution or family of distributions. 2. Use the maximum likelihood estimator (MLE) to estimate the parameters for the proposed distribution from our data. Substituting these parameters into the candidate distribution then produces the best possible distribution for our data *from that family*. This best-among-family distribution is our *candidate distribution*. 3. Draw a sample from the candidate distribution, whose size equals the size of the original data set. 4. Perform the KS test. 5. If our p-value is greater than 0.1, we will consider it reasonable that the data could have come from the candidate distribution, and if less than 0.1, then it is not reasonable. 6. As noted above, we may have to take a "good enough for government work" view of our model fit. To visualize how well the candidate distribution matches the data, we provide a function to plot the histograms overlaying one another and the qq-plot, *twoplot*. We also provide a function that returns the R^2 value from the regression of the qq-plot of the samples, *rsquared*.

How much variation do we get in the R^2 measurement? Let's run the same experiment we did for the KS metric.

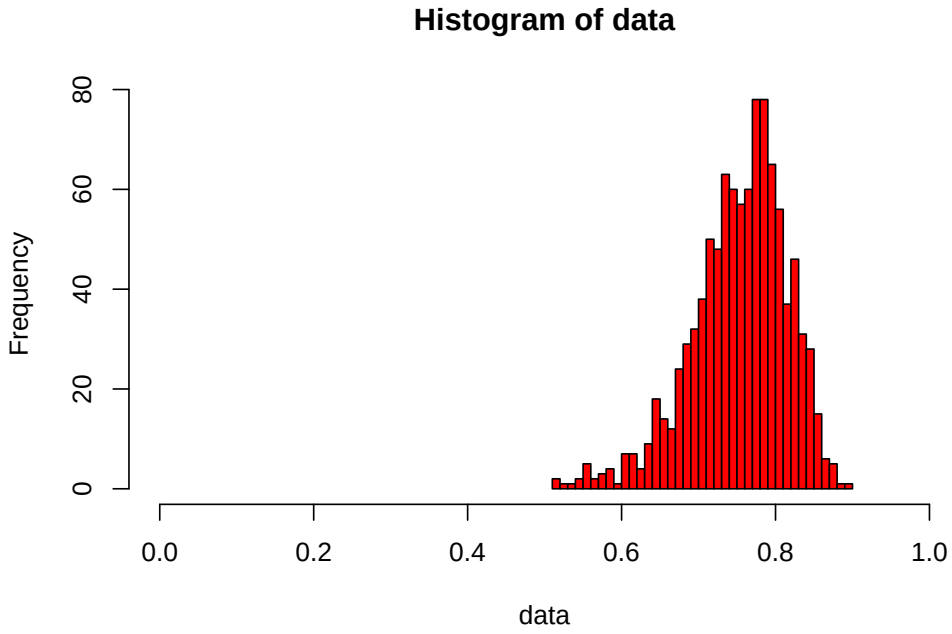
```
set.seed(42)
sample_size <- 500
run_experiment <- Vectorize(function(x=1) {
  X <- rnorm(sample_size)
  Y <- rnorm(sample_size)
  rsquared(X,Y)
})
data <- run_experiment(1:1000)
hist(data,breaks=30,col='red',
     xlim=c(0,1))
```



Here, we see very tight clustering near 1, and so we feel that this is a more stable measurement.

Let's try this for distributions that are not similar. How much variation do we get in the R^2 measurement? Let's run the same experiment we did for the KS metric.

```
set.seed(42)
sample_size <- 500
run_experiment <- Vectorize(function(x=1) {
  X <- rnorm(sample_size)
  Y <- rexp(sample_size,rate=0.2)
  rsquared(X,Y)
})
data <- run_experiment(1:1000)
hist(data,breaks=30,col='red',
     xlim=c(0,1))
```



Here we see our data clustered much lower. So for the same or similar distributions we'll expect to see R^2 measurements in the mid to high 90%'s.

Candidate: Normal distribution

We'll propose a normal distribution. Let's estimate the parameters.

```
mu <- mean(wage)
sigma <- sd(wage)
```

Next we draw our sample from the candidate distribution.

```
sample_size <- length(wage)
candidate <- rnorm(sample_size,mean=mu,sd=sigma)
```

At this point we want to compare our data to our sample. Using the KS metric, we have

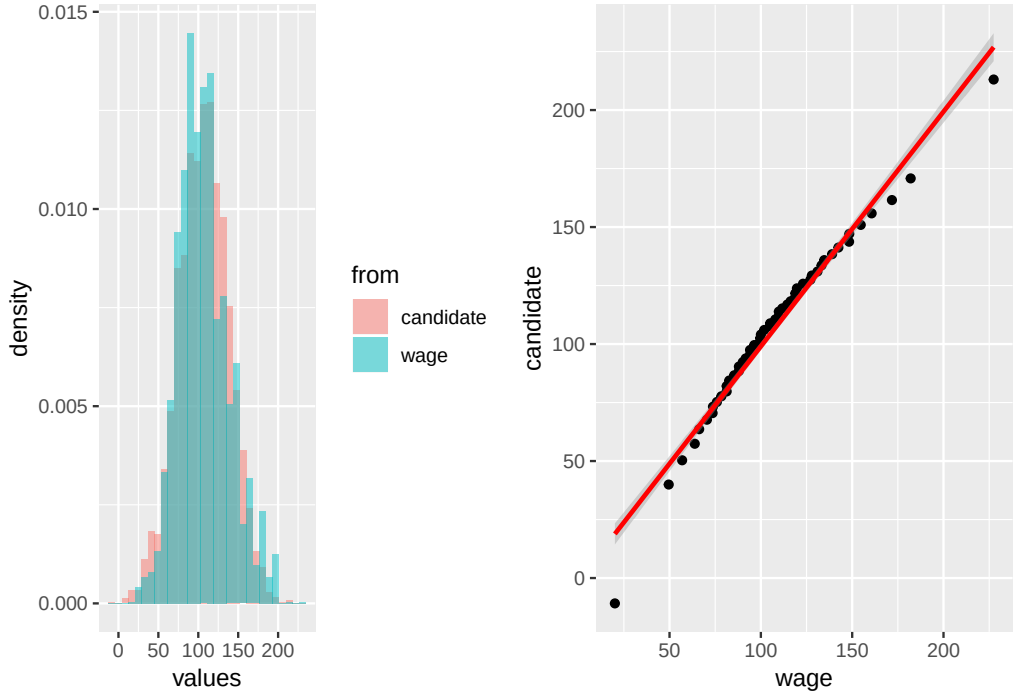
```
ks.test(wage,candidate)
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
```

```
## data:  wage and candidate
## D = 0.058199, p-value = 0.000101
## alternative hypothesis: two-sided
```

and thus we might want to conclude that the distributions are different. But let's look at the plots before we jump to any conclusions.

```
plot_two(wage,candidate)
```



The histograms may be similar enough for our application, and the qq-plot is quite reasonable. Our R^2 metric is

```
rsquared(wage,candidate)
```

```
## [1] 0.9732546
```

The R^2 value is quite high indicating a very good fit. Thus we may want to accept this distribution. Let's look at some other candidates.

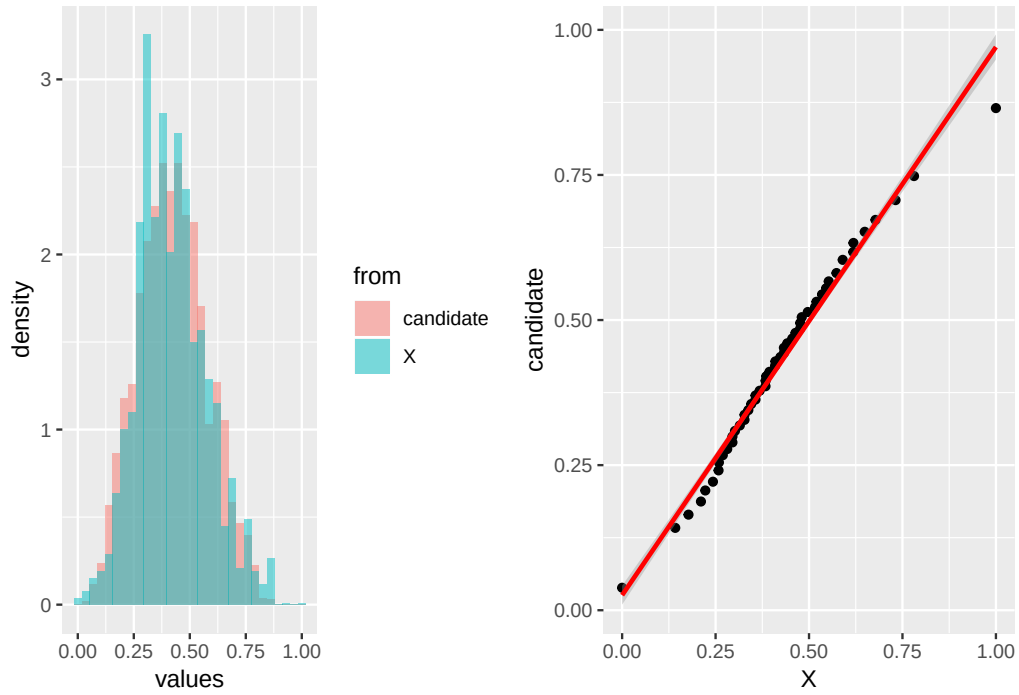
Candidate: Beta distribution

Next, we propose a beta distribution.

```
X <- (wage-min(wage))/(max(wage)-min(wage))
mu <- mean(X)
va <- var(X)
sample_size <- length(X)
alpha <- mu*(mu*(1-mu)/va-1)
beta <- (1-mu)*(mu*(1-mu)/va-1)
candidate <- rbeta(sample_size,shape1=alpha,shape2=beta)
ks.test(X,candidate)
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  X and candidate
## D = 0.06128, p-value = 3.445e-05
## alternative hypothesis: two-sided
```

```
plot_two(X,candidate)
```




```
rsquared(X,candidate)
```

```
## [1] 0.982849
```

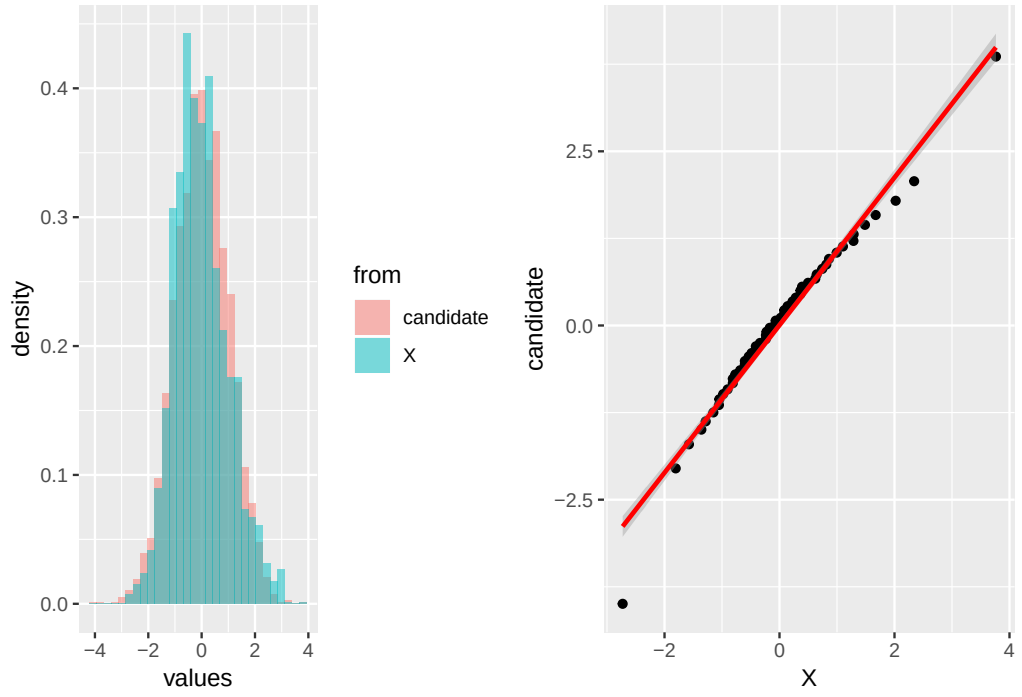
Candidate: Student t distribution

Next, we propose a Student t distribution.

```
X <- wage
mu <- mean(X)
va <- var(X)
X <- (X-mu)/sqrt(va)
sample_size <- length(X)
candidate <- rt(sample_size,df=sample_size-1)
ks.test(X,candidate)
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  X and candidate
## D = 0.071893, p-value = 5.549e-07
## alternative hypothesis: two-sided
```

```
plot_two(X,candidate)
```



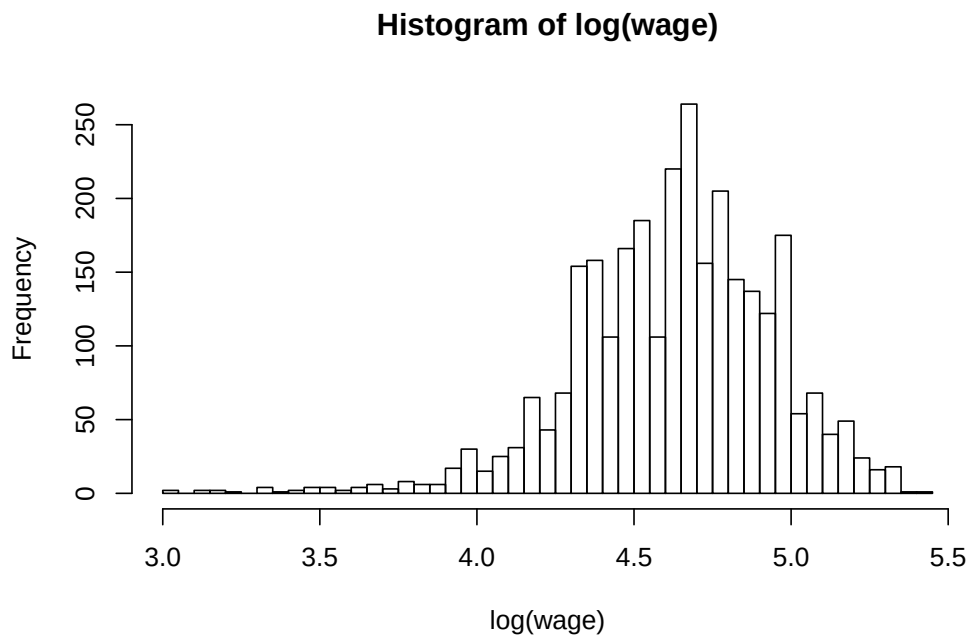

```
rsquared(X,candidate)
```

```
## [1] 0.9734295
```

Candidate: Lognormal distribution

Next, we propose a lognormal distribution.

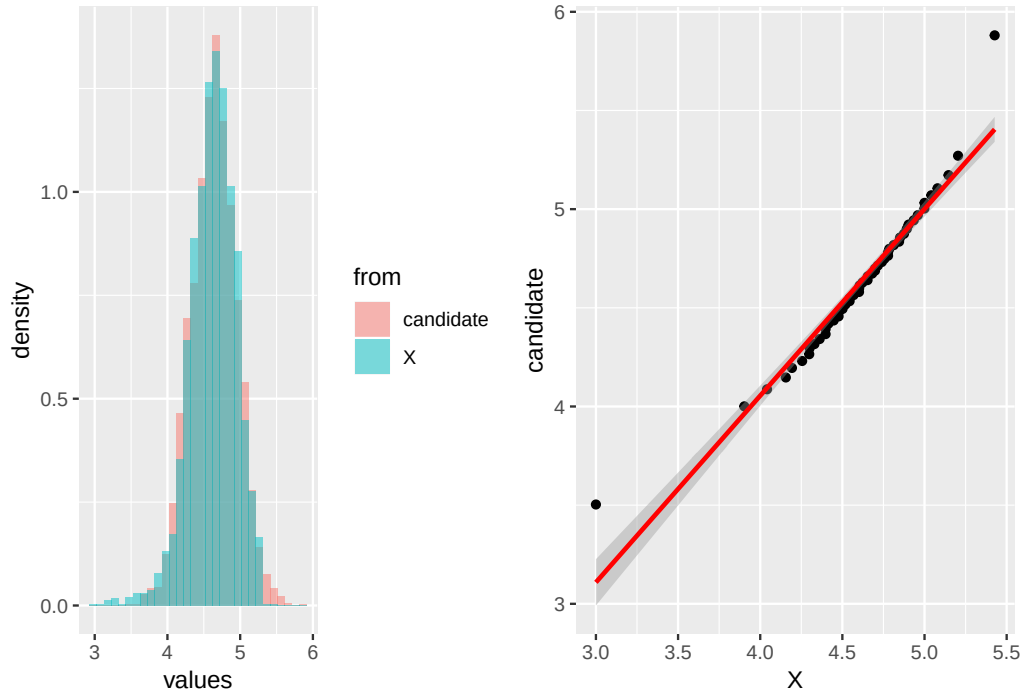
```
hist(log(wage),breaks=60)
```




```
X <- log(wage)
mu <- mean(X)
sigma <- sd(X)
sample_size <- length(X)
candidate <- rnorm(sample_size,mean=mu,sd=sigma)
ks.test(X,candidate)
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  X and candidate
## D = 0.032865, p-value = 0.08526
## alternative hypothesis: two-sided
```

```
plot_two(X,candidate)
```




```
rsquared(X,candidate)
```

```
## [1] 0.936554
```

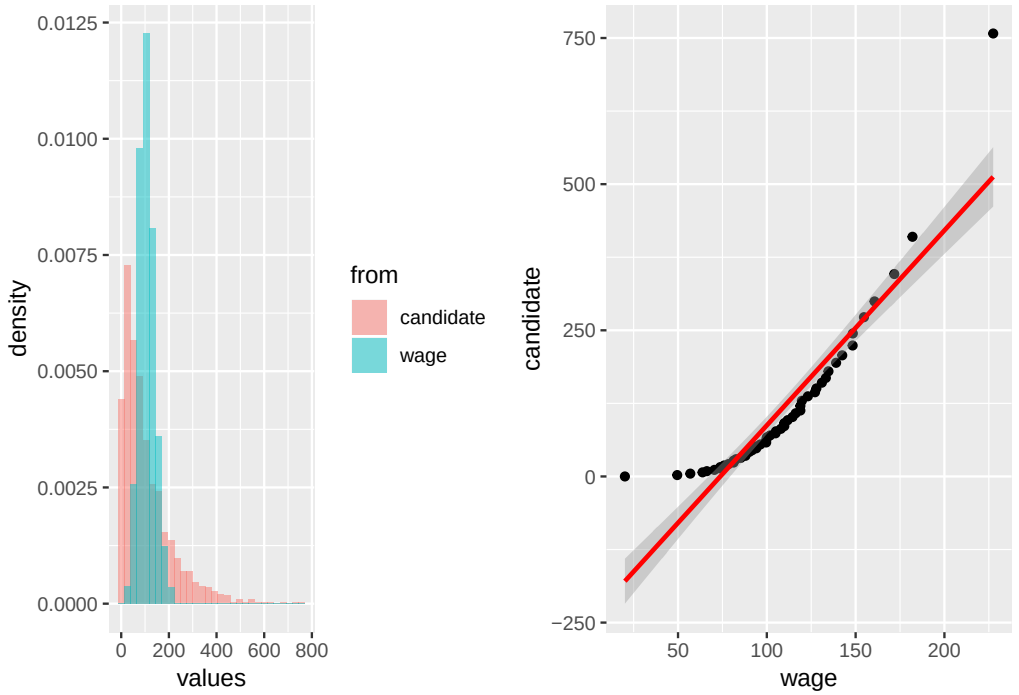
Candidate: Exponential distribution

Next, we propose an exponential distribution. We expect this to be a poor fit.

```
sample_size <- length(wage)
candidate <- rexp(sample_size,rate=1/mean(wage))
ks.test(wage,candidate)
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  wage and candidate
## D = 0.38959, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

```
plot_two(wage,candidate)
```




```
rsquared(wage,candidate)
```

```
## [1] 0.8472446
```

We have a histogram that is visually inappropriate and a R^2 value that is remarkably lower than the scores for our other candidates.

Conclusions?

We found several distributions that seem reasonable models and we proposed one that does not. We can see, comparing the KS test values, that the KS metric tends to reject proposed models often and does not distinguish well among them. Our graphical tests together with the R^2 value for the qq-plot tend to reject proposed models that are obviously incorrect, but may have some problems distinguishing the best model among a group of acceptable models.

For this problem, the Beta distribution seems to perform best. The Student t distribution performs nearly as well, and may be preferable as a model because of its familiarity from the introductory statistics course.

For more to read along these lines, [15] examines the difficulties of fitting power law models to given data sets and then asks whether the power law models are good for each data set. This is a classic paper, and presents a challenging but rewarding read.

Resource [16] is helpful in selecting models to propose for distributions, or just for getting more familiar with univariate distributions.

[14] is a detailed and technical look at the evolution of computational statistics over the past half century

References

[1] Ross. **A First Course in Probability: Fifth Edition**. Prentice-Hall, 1998.

[2] Grimmett, Stirzaker. **Probability and Random Processes: Third Edition**. Oxford University Press, 2001.

[3] Rubinstein, Kroese. **Simulation and the Monte Carlo Method: Third Edition**. Wiley, 2017.

[4] Shonkwiler, Mendivil. **Explorations in Monte Carlo Methods**. Springer, 2009.

[5] Gamerman, Lopes. **Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference, Second Edition**. Chapman and Hall/CRC, 2006.

[6] Pilon. **Probabilistic Programming and Bayesian Methods for Hackers**. Find this at https://github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers.

[7] Denny, Gaines. **Chance in Biology: Using Probability to Explore Nature**. Princeton University Press, 2000.

[8] Grimmett, Welsh. **Probability, An Introduction: Second Edition**. Oxford University Press, 2014.

[9] Knuth. *The Gamow–Stern elevator problem.* J. Recr. Math 2 (1969): 131-137.

[10] Knuth. *The toilet paper problem.* The American Mathematical Monthly 91.8 (1984): 465-470.

[11] Wasserman. **All of Statistics: A Concise Course in Statistical Inference**. Springer, 2004.

[12] Kruschke. **Doing Bayesian Data Analysis, Second Edition.** Academic Press, 2014.

[13] Efron. *Non-parametric estimates of standard error: the jackknife, the bootstrap, and other methods.* Biometrika, vol 68 issue 3, 1981, pages 589-599.

[14] Efron, Hastie. **Computer Age Statistical Inference: Algorithms, Evidence, and Data Science**. Cambridge University Press, 2016.

[15] Clauset, Shalizi, Newman. *Power-Law Distributions in Empirical Data.* SIAM Review, vol. 51, no. 4 (December 2009), pp 661-703.

[16] Leemis. *Univariate Distribution Relationships.* Available online at http://www.math.wm.edu/~leemis/chart/UDR/UDR.html .