Binomial distribution, R simulation, function

STA 032: Gateway to data science Lecture 15

Jingwei Xiong

May 5, 2023

Binomial random variable

- The **binomial distribution** gives us the probability of X "successes" from a sequence of n independent Bernoulli trials (size n). This is often denoted binomial(n, p).
- In our example, each student would represent an independent Bernoulli trial (either an e-cig smoker, or not).
    - 1 draw from the binomial distribution is made of 3 independent draws from the Bernoulli distribution
- This distribution involves three assumptions.
    - There is a fixed number n of Bernoulli trials, each of which results in one of two mutually-exclusive outcomes
    - The outcomes of the n trials are independent
    - The probability of success p is the same for each trial

Binomial distribution

- The probability mass function for the binomial distribution is given by $P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}$
- Compare this with $P(Y = y) = p^y (1-p)^{1-y}$ for the Bernoulli distribution.
- First, look at the second part, $p^x (1-p)^{n-x}$. This is just multiplying the right combination of p and $1-p$ as in the previous tables.
    - There will be a total of n terms being multiplied, one probability for each draw of the distribution (each student in this case)
    - For example, if we want the probability of 3 e-cig smokers, $X = 3$, the second part is $p^x (1-p)^{n-x} = 0.2^3 (0.8)^0 = 0.008$, just as in the table.

Binomial distribution

$$P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}$$

- If we want the probability of 2 e-cig smokers and 1 non-smoker, i.e., $x = 2$, the second part is $p^x(1-p)^{n-x} = 0.2^2(0.8)^{3-2} = 0.032$, which is what we see in any single row in which we have two smokers and one non-smoker.
    - This is the probability of any one specific combination of 2 smokers and 1 nonsmoker. Then we need to figure out how many combinations of 2 smokers and 1 nonsmoker we could get.
- The first part, $\binom{n}{x}$, accounts for all the possible ways in which we can have 2 smokers out of 3 people.

Combinations

- The first part, $\binom{n}{x}$, is pronounced "n choose x"
- This is called a **combination**.
- In this particular setting, it is also called the *binomial coefficient*.
- It is a formula for the number of ways to pick x subjects from a larger group of n and is defined as

$$\binom{n}{x} = \frac{n!}{x!(n-x)!}.$$

- n! (pronounced "n factorial") is shorthand for the recursive multiplication $n! = n(n-1)(n-2)\cdots(1)$.
  - $3! = 3(2)(1) = 6$
  - $4! = 4(3)(2)(1) = 24$, and so forth.
  - We define $0! = 1$.

Combinations

- In R, combinations are done using `choose(n, k)`
- Factorials are `factorial(x)`, e.g.,

```
factorial(3)
```

```
[1] 6
```

- How many ways can we pick 3 subjects from a group of 3? Using the binomial coefficient, we see it is $\binom{3}{3} = \frac{3!}{3!0!} = \frac{3(2)(1)}{3(2)(1)1} = 1$. In R,

```
choose(3, 3)
```

```
[1] 1
```

Case Study: E-Cigarettes

- The CDC reports that 19.6% of high school students have smoked e-cigarettes in the past 30 days. We'll round this to 20% for simplicity.
- $P(Y = 1) = P(Smoker) = p = 0.2$ and $P(Y = 0) = 0.8$
- Now suppose we randomly select 3 independent high school students and define a new random variable X representing the number of smokers. X can take the values 0, 1, 2 or 3.
- Let Y_i be the smoking status of the ith student, where $Y_i = 1$ if student i smokes and 0 otherwise.

Y_1	Y_2	Y_3	X	$P(X)$
0	0	0	0	
1	0	0	1	
0	1	0	1	
0	0	1	1	
1	1	0	2	
1	0	1	2	
0	1	1	2	

Combinations

Going back to the table, we see there are 3 ways to pick 2 subjects from a group of 3 students.

Y_1	Y_2	Y_3	X	$P(X)$
0	0	0	0	0.8(0.8)(0.8)=0.512
1	0	0	1	0.2(0.8)(0.8)=0.128
0	1	0	1	0.8(0.2)(0.8)=0.128
0	0	1	1	0.8(0.8)(0.2)=0.128
1	1	0	2	0.2(0.2)(0.8)=0.032
1	0	1	2	0.2(0.8)(0.2)=0.032
0	1	1	2	0.8(0.2)(0.2)=0.032
1	1	1	3	0.2(0.2)(0.2)=0.008

We can also calculate $\binom{3}{2} = \frac{3!}{2!1!} = \frac{3(2)(1)}{(2)(1)(1)} = 3$.

Binomial probabilities

Putting the ingredients together, we have

$$\begin{aligned} P(X = 2) &= \binom{n}{x} p^x (1-p)^{n-x} \\ &= \binom{3}{2} p^2 (1-p)^{3-2} \\ &= 3(0.2)^2 (0.8)^{3-2} \\ &= 3(.032) \\ &= .096 \end{aligned}$$

This is the same as we saw in the table.

Binomial probabilities

- We saw how to calculate $P(X = 2)$. How about $P(X = 3)$?

$$\begin{aligned} P(X = x) &= \binom{n}{x} p^x (1-p)^{n-x} \\ &= \frac{n!}{x!(n-x)!} p^x (1-p)^{n-x} \end{aligned}$$

- Using the binomial distribution, the probability of getting 3 recent e-cig smokers in our sample of 3 students is

$$\begin{aligned} P(X = 3) &= \binom{3}{3} 0.2^3 (1-0.2)^{3-3} \\ &= \frac{3!}{3!(3-3)!} 0.2^3 (1-0.2)^{3-3} \\ &= \frac{3(2)(1)}{3(2)(1)1} 0.2^3 (0.8)^0 = 0.2^3 \\ &= 0.008. \end{aligned}$$

- In this way, we can derive the probability distribution

Bernoulli vs. binomial random variable

- Recall: The **binomial distribution** gives us the probability of X "successes" from a sequence of n independent Bernoulli trials.
- In our example, each student would be an independent Bernoulli trial (either an e-cig smoker, or not).
    - Bernoulli random variable: whether or not a student is a recent e-cigarette smoker
    - Binomial random variable: number of students, out of 3, who are recent e-cigarette smokers
    - One observation of this binomial random variable is composed of 3 Bernoulli trials ("size 3")
- The Bernoulli distribution can be thought of as a special case of the binomial distribution, with 1 "trial"

Mean and variance of binomial random variable

For a binomial random variable with number of trials n and probability of success p, $E(X) = np$ and $Var(X) = n(p)(1-p)$

- In our example, $p = .2$, $n = 3$, so $E(X) = .6$ and $Var(X) = 3 * .2 * .8 = .48$
- The Bernoulli distribution is a special case of the binomial distribution, with $n = 1$. Recall: $E(X) = p$ and $Var(X) = p(1-p)$.

Binomial probabilites in R

- Luckily, we don't have to calculates by hand
- In R, we can use `dbinom()`
- From the help page: `dbinom()` has the arguments `x, size, prob, log = FALSE`
  - `x` are the values of x that we want probabilities for; here it is 0, 1, 2 and 3
  - `size` is the number of Bernoulli trials; here it is 3
  - `prob` is the probability of success; here it is .2

```
dbinom(x = 0:3, size = 3,
       prob = .2)
```

```
[1] 0.512 0.384 0.096 0.008
```

The probability distribution of X, the number of recent e-cig smokers out of three high school students, is now

X	0	1	2	3
$P(X = x)$	0.512	0.384	0.096	0.008

More examples of calculating binomial probabilities

- We might be interested in the probability that there is at least 1 student, out of 500, that has recent e-cigarette use.
- This translates to $P(X \geq 1)$, where X is the random variable representing the number of students, out of 500, with recent e-cigarette use. Here, $n = 500$ and $p = .2$
- Other probabilities that may be of interest include the probability that more than 2 students, out of 500, have recent e-cigarette use.

Binomial distribution: $P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}$

- Quick trick: $P(X \geq 1) = 1 - P(X = 0)$ (you either have no students with recent e-cigarette use, or one or more students with recent e-cigarette use)
- Similarly, $P(X > 2) = 1 - P(X = 0) - P(X = 1) - P(X = 2)$

More examples of calculating binomial probabilities

Binomial distribution: $P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}$

We thus have

$$
\begin{aligned}
P(X \geq 1) &= 1 - P(X = 0) \\
&= 1 - \binom{500}{0} 0.2^0 (0.8)^{500-0} \\
&= 1 - 0.8^{500} \\
&= 1 \\
P(X > 2) &= 1 - P(X = 0) - P(X = 1) - P(X = 2) \\
&= 1 - 0.8^{500} - \binom{500}{1} 0.2^1 (0.8)^{500-1} - \binom{500}{2} 0.2^2 (0.8)^{500-2} \\
&= 1 - 0.8^{500} - 500(0.2)(0.8)^{500-1} - \frac{500!}{2!(500-2)!} 0.2^2 (0.8)^{500-2} \\
&= 1 - 0.8^{500} - 500(0.2)(0.8)^{500-1} - \frac{500(500-1)}{2} 0.2^2 (0.8)^{500-2} \\
&= 1
\end{aligned}
$$

In R

- Given the large number of students (500), it is almost certain that at least 1 or 2 would have recent e-cigarette use.
- We might be interested instead in $P(X \geq 100)$, the probability that at least four-fifths of the students have recent e-cigarette use
- As we can see, this quickly becomes intractable. Luckily, we can do this in R

```
sum(dbinom(x = 100:500, size = 500, prob = .2))
```

```
[1] 0.5178363
```

- We can also use the `pbinom()` function, which gives us $P(X \leq x)$ directly

```
1 - pbinom(q = 99, size = 500, prob = .2)
```

```
[1] 0.5178363
```

Sampling from a binomial distribution in R

- Recall draws from the Bernoulli distribution:
  - For any value of $0 \leq p \leq 1$ (including .5), we can use the `rbinom()` function
  - `rbinom()` has the arguments `n, size, prob`, where `n` is the number of draws, and `prob` is the probability of success.
  - `size` is the "number of trials (zero or more)" -- for the Bernoulli distribution, we use `size = 1`.

Sampling from a binomial distribution in R

- For the binomial distribution, `size` is the number of Bernoulli trials; in the e-cigarette example, `size = 3`
- **NOTE**: we use n to denote the number of Bernoulli trials; in R this is the `size` argument

Description	R
Number of draws/samples from Binomial	`n`
Number of Bernoulli trials, size n	`size`
Probability of success, p	`prob`

- It should now make sense that for draws from the Bernoulli distribution, we use the argument `size = 1`.

Sampling from a binomial distribution in R

Example: 100 draws from the binomial distribution; each draw is made of 3 Bernoulli trials (size 3); probability of success = .2

```
set.seed(0) # so results are reproducible
inputP <- .2
binomDraws <- rbinom(n = 100, size = 3, prob = inputP)
binomDraws
```

```
  [1] 2 0 0 1 2 0 2 2 1 1 0 0 0 1 0 1 0 1 2 0 1 2 0 1 0 0 0 0 0 1 0 0 1 0 0 1
 [38] 1 0 1 0 1 1 1 1 1 1 0 0 1 1 0 1 0 0 0 0 0 1 1 0 2 0 0 0 1 0 0 1 0 1 0 1
 [75] 0 0 1 1 0 1 2 0 1 0 0 1 0 1 0 0 0 0 0 1 1 1 1 0 0 1
```

Sample mean

- Recall: for a binomial random variable with number of trials n and probability of success p, $E(X) = np$ and $Var(X) = n(p)(1-p)$
- In our example, $p = .2$, $n = 3$, so $E(X) = .6$ and $Var(X) = 3 * .2 * .8 = .48$
- We can calculate the sample mean from our sample of 100:

```
mean(binomDraws)
```

```
[1] 0.56
```

R Functions for Probability Distributions

Every distribution that R handles has four functions. There is a root name, for example, the root name for the normal distribution is `norm`. This root is prefixed by one of the letters.

- `p` for "probability", the cumulative distribution function (c. d. f.)
- `q` for "quantile", the inverse c. d. f.
- `d` for "density", the density function (p. m. f. or p. d. f.)
- `r` for "random", a random generator having the specified distribution.

For the normal distribution, these functions are `pnorm`, `qnorm`, `dnorm`, and `rnorm`. For the binomial distribution, these functions are `pbinom`, `qbinom`, `dbinom`, and `rbinom`. And so forth.

Here you can find the distribution and their function names. We won't have time to explain all of them, you can look up the help function **?** for their arguments.

Case study: Simulate the chicken farm

In a chicken farm, suppose for each day, each chicken has a probability $p_1 = 0.2$ to spawn an egg. Independently to the spawn egg, each chicken has a probability $q = 0.2$ to become chicken product (die). And for each day, each egg has a probability $p_2 = 0.5$ to hatch a chicken. How can we simulate the chicken and egg numbers for the chicken farm?

Assumptions:

- In the beginning of each day, we record the number of chicken n_c and eggs n_e.
- In that day, the number of eggs spwan will follow a binomial distribution: $e \sim Binomial(n_c, p = p_1)$
- In that day, the number of chicken died will follow a binomial distribution: $c_1 \sim Binomial(n_c, p = q)$
- In that day, the number of chicken hatched will follow a binomial distribution: $c_2 \sim Binomial(n_e, p = p_2)$

So at the end of that day (beginning of next day), we will have $n_c - c_1 + c_2$ chicken, $n_e + e - c_2$ eggs

Simulate the binomial distribution

- Step 1: Set initial values and the parameters:

```
# Set initial values
n_c = 50 # number of chickens
n_e = 20 # number of eggs
# Set probabilities
p1 = 0.2 # probability of egg spawn
q = 0.2 # probability of chicken death
p2 = 0.5 # probability of egg hatching
# Set number of days to simulate
n_days = 365
```

- Step 2: Create vectors to store results

```
# Create vectors to store results
chicken_counts = numeric(n_days)
egg_counts = numeric(n_days)
```

- Step 3: Given the last day result, simulate next day
    - From the help page: `rbinom()` has the arguments `n, size, prob`
    - `n` are the number of samples we want to obtain.
    - `size` is the number of Bernoulli trials;
    - `prob` is the probability of success;

```
# In the beginning, we have n_c chickens, n_e eggs
n_c = 50
n_e = 20
# Calculate number of eggs spawned and number of chicken deaths
eggs_spawned = rbinom(1, n_c, p1)
chicken_deaths = rbinom(1, n_c, q)
# Calculate number of chicken hatched from eggs
chicken_hatched = rbinom(1, n_e, p2)
# Calculate new chicken and egg counts
n_c = n_c - chicken_deaths + chicken_hatched
n_e = n_e + eggs_spawned - chicken_hatched
```

With this code we update the simulation result for 1 day.

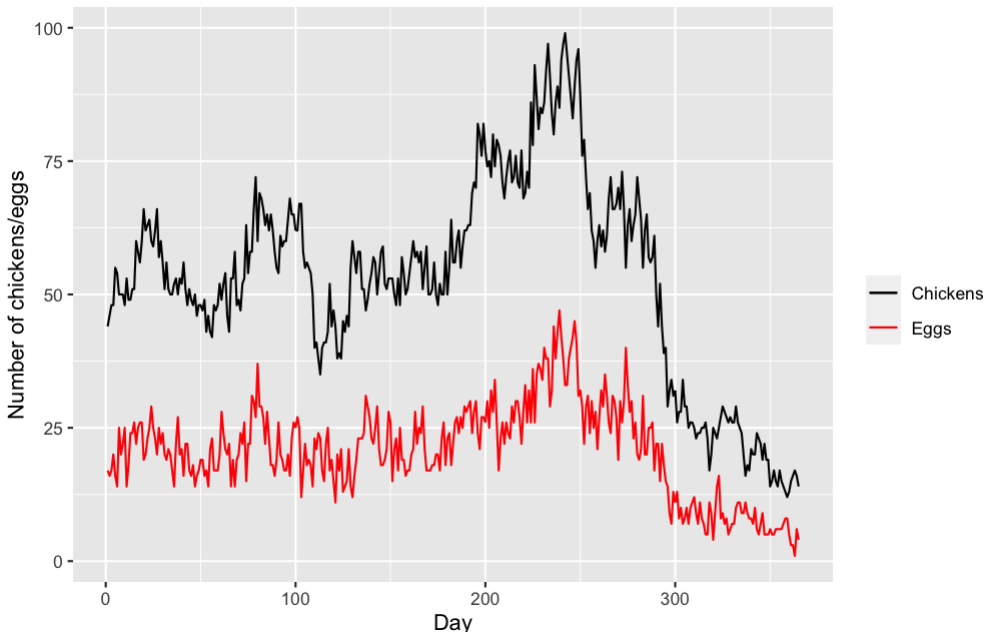
- Step 4: Wrap the procedure into a for loop
  - In R we use `for` loop to run commands multiple times.
  - Syntax: `for(control_variable in loop_vector){commands}`
  - `control_variable` will change over loops, iterate in the `loop_vector`

```
n_c = 50
n_e = 20
for (i in 1:n_days) {
  # Calculate number of eggs spawned and number of chicken deaths
  eggs_spawned = rbinom(1, n_c, p1)
  chicken_deaths = rbinom(1, n_c, q)

  # Calculate number of chicken hatched from eggs
  chicken_hatched = rbinom(1, n_e, p2)

  # Calculate new chicken and egg counts
  n_c = n_c - chicken_deaths + chicken_hatched
  n_e = n_e + eggs_spawned - chicken_hatched

  # Store results
  chicken_counts[i] = n_c
  egg_counts[i] = n_e
}
```

Here we iterate the `i` from 1,2,... to `n_days`, for each day we simulate and update the result, and then store the result into the `i`th location of the result

- Step 5: visualize the result

```
df = data.frame(chicken_counts = chicken_counts,
                egg_counts = egg_counts,
                day = 1:n_days)

# Plot results using ggplot2
ggplot(df, aes(x = day)) +
  geom_line(aes(y = chicken_counts, color = "Chickens")) +
  geom_line(aes(y = egg_counts, color = "Eggs")) +
  xlab("Day") + ylab("Number of chickens/eggs") +
  scale_color_manual(name = "", values = c("Chickens" = "black", "Egg
```



Function

An R function is a self-contained block of code that performs a specific task and can be executed repeatedly with different inputs.

Functions are defined using the **function** keyword and have a specific structure, including a name, a set of arguments, and a code block.

The basic structure of an R function looks like this:

```
function_name = function(arg1, arg2, ...){
  # code block
  return(output)
}
```

```r
function_name = function(arg1, arg2, ...){
  # code block
  return(output)
}
```

- **function_name** is the name of the function and can be any valid R object name.
- **arg1, arg2, ...** are the arguments or inputs of the function, which can be used to pass data into the function and specify how it should behave.
- The code block is the set of instructions that the function will execute when it is called.
- The return statement is used to specify the output or result of the function, which can be used in other parts of your code. If there is no return statement, the last line will be returned.

Benefits of using functions

One of the best ways to improve your reach as a data scientist is to write functions. Functions allow you to automate common tasks in a more powerful and general way than copy-and-pasting. Writing a function has three big advantages over using copy-and-paste:

1. You can give a function an evocative name that makes your code easier to understand.

2. As requirements change, you only need to update code in one place, instead of many.

3. You eliminate the chance of making incidental mistakes when you copy and paste (i.e. updating a variable name in one place, but not in another).

When should you write a function?

You should consider writing a function whenever you've copied and pasted a block of code more than twice (i.e. you now have three copies of the same code). For example, take a look at this code. What does it do?

```
df = data.frame(
  a = rnorm(10),
  b = rnorm(10),
  c = rnorm(10),
  d = rnorm(10)
)

df$a = (df$a - min(df$a, na.rm = TRUE)) /
  (max(df$a, na.rm = TRUE) - min(df$a, na.rm = TRUE))
df$b = (df$b - min(df$b, na.rm = TRUE)) /
  (max(df$b, na.rm = TRUE) - min(df$a, na.rm = TRUE))
df$c = (df$c - min(df$c, na.rm = TRUE)) /
  (max(df$c, na.rm = TRUE) - min(df$c, na.rm = TRUE))
df$d = (df$d - min(df$d, na.rm = TRUE)) /
  (max(df$d, na.rm = TRUE) - min(df$d, na.rm = TRUE))
```

You might be able to puzzle out that this rescales each column to have a range from 0 to 1. But did you spot the mistake? I made an error when copying-and-pasting the code for **df$b**: I forgot to change an **a** to a **b**. Extracting repeated code out into a function is a good idea because it prevents you from making this type of mistake.

To write a function you need to first analyse the code. How many inputs does it have?

```
(df$a - min(df$a, na.rm = TRUE)) /
  (max(df$a, na.rm = TRUE) - min(df$a, na.rm = TRUE))
```

This code only has one input: df$a. (If you're surprised that TRUE is not an input, you can explore why in the exercise.)

To make the inputs more clear, it's a good idea to rewrite the code using temporary variables with general names. Here this code only requires a single numeric vector, so I'll call it x:

```
x = df$a
(x - min(x, na.rm = TRUE)) / (max(x, na.rm = TRUE) - min(x, na.rm = T
```

There is some duplication in this code. We're computing the range of the data three times, so it makes sense to do it in one step:

```
x = df$a
xmin = min(x, na.rm = TRUE)
xmax = max(x, na.rm = TRUE)
(x - xmin) / (xmax - xmin)
```

Or use the `range` function:

```
rng = range(x, na.rm = TRUE)
(x - rng[1]) / (rng[2] - rng[1])
```

Remarks: **Pulling out intermediate calculations into named variables** is a **good practice** because it makes it more clear what the code is doing.

Now that I've simplified the code, and checked that it still works, I can turn it into a function:

```
rescale01 = function(x) {
  rng = range(x, na.rm = TRUE)
  return(
    (x - rng[1]) / (rng[2] - rng[1])
    )
}
rescale01(c(0, 5, 10))
```

```
[1] 0.0 0.5 1.0
```

There are three key steps to creating a new function:

1. You need to pick a name for the function. Here I've used rescale01 because this function rescales a vector to lie between 0 and 1.

2. You list the inputs, or arguments, to the function inside function. Here we have just one argument. If we had more the call would look like function(x, y, z).

3. You place the code you have developed in body of the function, a { block that immediately follows function(...).

Note the overall process: I only made the function after I'd figured out how to make it work with a simple input. **It's easier to start with working code and turn it into a function**; it's harder to create a function and then try to make it work.

At this point it's a good idea to check your function with a few different inputs:

```
rescale01(c(-10, 0, 10))
```

```
[1] 0.0 0.5 1.0
```

```
rescale01(c(1, 2, 3, NA, 5))
```

```
[1] 0.00 0.25 0.50   NA 1.00
```

Now we can simplify the original example now that we have a function:

```
df$a = rescale01(df$a)
df$b = rescale01(df$b)
df$c = rescale01(df$c)
df$d = rescale01(df$d)
```

Compared to the original, this code is easier to understand and we've eliminated one class of copy-and-paste errors. There is still quite a bit of duplication since we're doing the same thing to multiple columns. We'll learn how to eliminate that duplication in **iteration**.

Another advantage of functions is that if our requirements change, we only need to make the **change in one place**. For example, we might discover that some of our variables include infinite values, and rescale01() fails:

```
x = c(1:10, Inf)
rescale01(x)
```

```
 [1]   0   0   0   0   0   0   0   0   0   0 NaN
```

Because we've extracted the code into a function, we only need to make the fix in one place:

```
rescale01 = function(x) {
  rng = range(x, na.rm = TRUE, finite = TRUE)
  #because when finite = TRUE, range will return the max which is not
  (x - rng[1]) / (rng[2] - rng[1])
}
rescale01(x)
```

```
 [1] 0.0000000 0.1111111 0.2222222 0.3333333 0.4444444 0.5555556 0.6666667
 [8] 0.7777778 0.8888889 1.0000000       Inf
```

This is an important part of the "**do not repeat yourself**" (or DRY) principle. The more repetition you have in your code, the more places you need to remember to update when things change (and they always do!), and the more likely you are to create bugs over time.

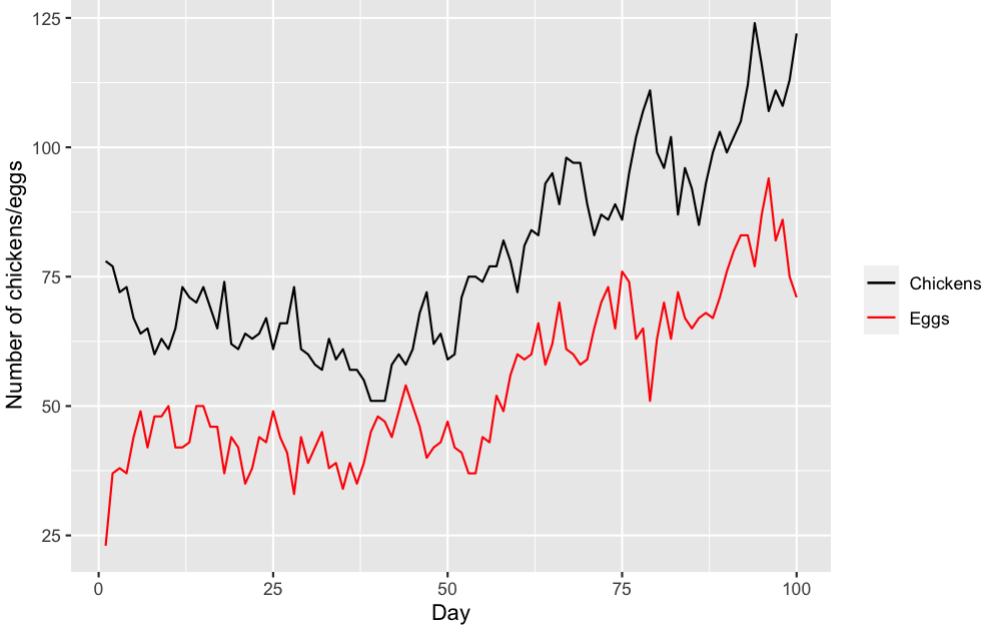
Function for Simulating the chicken farm

Now we can try to wrap all of the simulation procedure into one function.

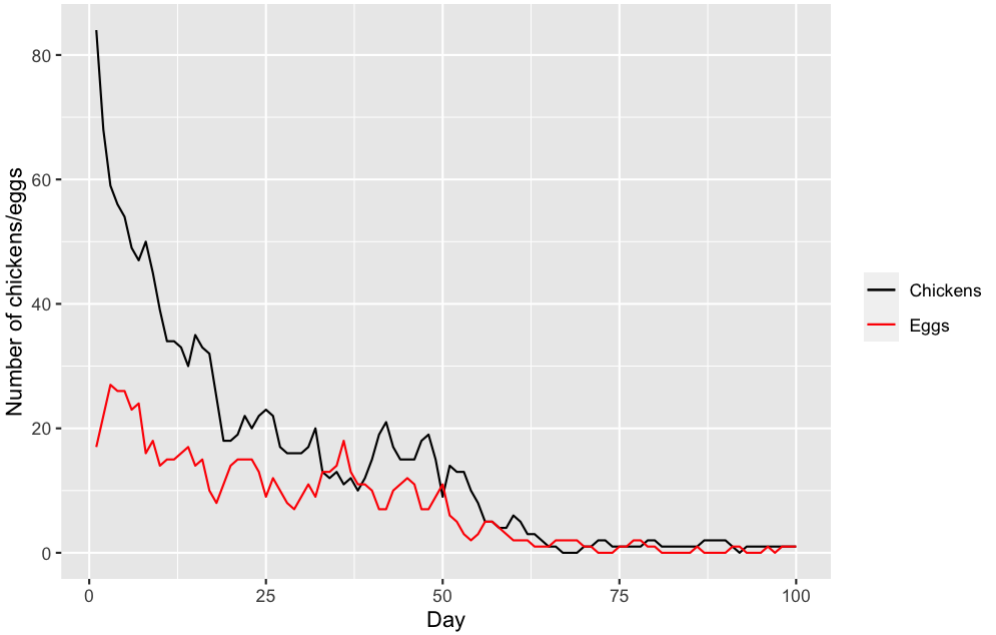
```
chicken_farm = function(initial_chicken, initial_egg,
                        p_spawn, p_death, p_hatch, n_days){
  n_c = initial_chicken # number of chickens
  n_e = initial_egg # number of eggs
  p1 = p_spawn # probability of egg spawn
  q = p_death # probability of chicken death
  p2 = p_hatch # probability of egg hatching
  # Create vectors to store results
  chicken_counts = numeric(n_days)
  egg_counts = numeric(n_days)

  # Simulate each day
  for (i in 1:n_days) {
    # Calculate number of eggs spawned and number of chicken deaths
    eggs_spawned = rbinom(1, n_c, p1)
    chicken_deaths = rbinom(1, n_c, q)

    # Calculate number of chicken hatched from eggs
    chicken_hatched = rbinom(1, n_e, p2)

    # Calculate new chicken and egg counts
    n_c = n_c - chicken_deaths + chicken_hatched
    n_e = n_e + eggs_spawned - chicken_hatched

    # Store results
    chicken_counts[i] = n_c
    egg_counts[i] = n_e
  }
  df = data.frame(chicken_counts = chicken_counts,
                  egg_counts = egg_counts,
                  day = 1:n_days)
  return(df)
}
```

Now we have the function to generate the dataframe used for plotting. And you can easily change the parameter for the simulation.

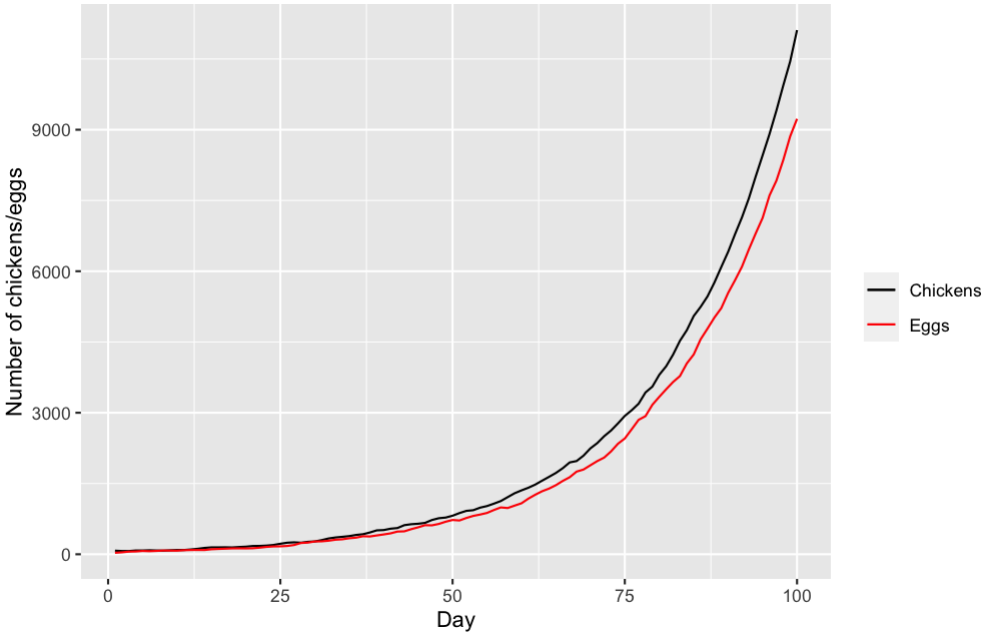
```
chicken_farm(initial_chicken = 100, initial_egg = 0,
  p_spawn = 0.2, p_death = 0.2,
  p_hatch = 0.3, n_days = 100) %>%
  ggplot(aes(x = day)) +
  geom_line(aes(y = chicken_counts, color = "Chickens")) +
  geom_line(aes(y = egg_counts, color = "Eggs")) +
  xlab("Day") + ylab("Number of chickens/eggs") +
  scale_color_manual(name = "", values = c("Chickens" = "black", "Egg
```



```
chicken_farm(initial_chicken = 100, initial_egg = 0,
  p_spawn = 0.15, p_death = 0.2,
  p_hatch = 0.3, n_days = 100) %>%
  ggplot(aes(x = day)) +
  geom_line(aes(y = chicken_counts, color = "Chickens")) +
  geom_line(aes(y = egg_counts, color = "Eggs")) +
  xlab("Day") + ylab("Number of chickens/eggs") +
  scale_color_manual(name = "", values = c("Chickens" = "black", "Egg
```



```r
chicken_farm(initial_chicken = 100, initial_egg = 0,
  p_spawn = 0.3, p_death = 0.2,
  p_hatch = 0.3, n_days = 100) %>%
  ggplot(aes(x = day)) +
  geom_line(aes(y = chicken_counts, color = "Chickens")) +
  geom_line(aes(y = egg_counts, color = "Eggs")) +
  xlab("Day") + ylab("Number of chickens/eggs") +
  scale_color_manual(name = "", values = c("Chickens" = "black", "Egg
```



Summary

- Common probability distributions
  - Bernoulli
  - Binomial distribution
- R simulation
- For loop
- Functions