Introduction to R

Becky Tang

2023-09-17

Element of the editor

In an R Markdown document, we can interweave code and text. The code goes in a code chunk, and text outside of the code chunks. When you knit, the code is executed. Note that running code is not the same as knitting!

Some coding basics

R as calculator

```
1 + 2
```

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

```
3 * 4
```

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

```
5^2
```

```
## [1] 25
```

```
sqrt(9)
```

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

```
# this calculates e^2
exp(2)
```

```
## [1] 7.389056
```

```
# this calculates 5!
factorial(5)
```

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

```r
# this evaluates the binomial coefficient "5 choose 3"
choose(5, 3)
```

```
## [1] 10
```

R respects PEMDAS:

```r
1 + 1/2
```

```
## [1] 1.5
```

```r
(1+1)/2
```

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

Booleans

Booleans/logicals can be formed with (in)equality symbols:

```r
1 > 2
```

```
## [1] FALSE
```

```r
2 >= -1
```

```
## [1] TRUE
```

```r
3 == 4
```

```
## [1] FALSE
```

Variables

Storing/saving values into variables is achieved by using the syntax `<variable name> <- <value>`. Note that the variable name cannot begin with a number!

Note: storing means the result is not displayed to the user. Type out the variable name after storing to explicitly show the values.

```r
a <- 1
a
```

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

Vectors

Create vectors of values using the concatenate `c()` function:

```
v <- c(1, 4, 9, 16)
v
```

```
## [1]  1  4  9 16
```

The syntax `a:b` creates a sequence of integers from `a` to `b`.

```
c <- 1:4
c
```

```
## [1] 1 2 3 4
```

R is vectorized language. Also, under the hood, booleans/logicals are treated at 1/0 for TRUE/FALSE.

```
c + v
```

```
## [1]  2  6 12 20
```

```
c * v
```

```
## [1]  1  8 27 64
```

```
sqrt(c)
```

```
## [1] 1.000000 1.414214 1.732051 2.000000
```

```
u <- v >= 3
u
```

```
## [1] FALSE  TRUE  TRUE  TRUE
```

```
sum(u)
```

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

Functions

A function takes in some argument(s) as input and returns something back as output. We can write custom functions using `function()`. We need to specify the name of the function for future use, as well as what to output based on inputs.

Suppose a group of k people each choose a number at random from a list of n possibilities. The probability of at least one match is 1 minus the probability of no matches:

$$1 - \frac{n!}{(n-k)!n^k} = 1 - \binom{n}{k}\frac{k!}{n^k}$$

We can write this as a function:

```r
prob_match <- function(n, k){
  1 - choose(n, k) * factorial(k) / n^k
}
```

Note that the function name is `prob_match`, and the inputs are `n` and `k`. The output is the probability for the specific `n` and `k`. For example, probability of at least one match among 3 people choosing a number between 1 and 10 is:

```r
prob_match(10, 3)
```

```
## [1] 0.28
```

Basic plotting

We can plot one variable against another (e.g. input and output of a function) using the `plot()` function. The first argument is the x-axis variable (e.g. input), and the second argument is the y-axis variable (e.g. output).

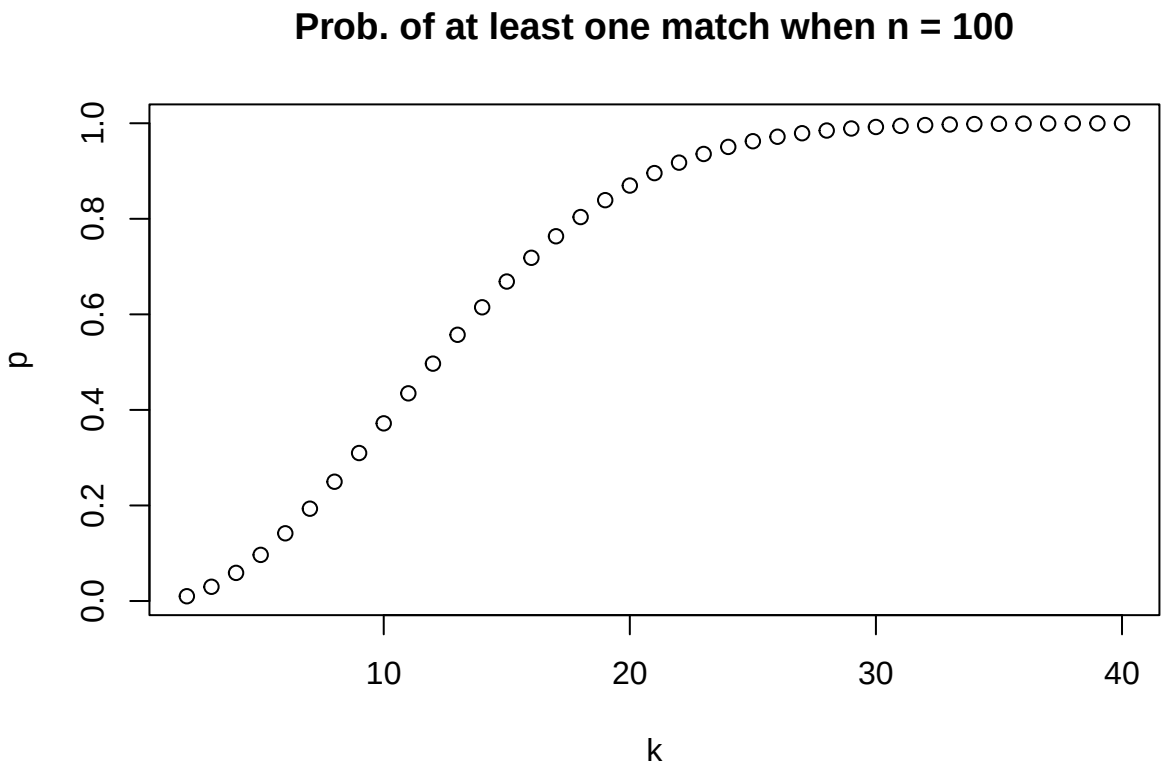
For example, suppose we want to see the outputs of `prob_match` for picking one number out of $n = 100$ possible numbers, for groups of size $k = 2, 3, \ldots, 40$.

We begin by creating a vector of inputs:

```r
k <- 2:40
p <- prob_match(100, k)
```

Then we pass `k` and `p` into the plot function, plus an optional title:

```r
plot(k, p, main = "Prob. of at least one match when n = 100")
```



Sampling and simulating

The `sample()` function generates random values from a pool of values contained in a specific vector. You can twist many knobs to get sampling with or without replacement, uneven probabilities, and different sized samples.

```
x <- 1:10
sample(x)
```

```
##  [1]  6  7  9  1  8  3  2  5  4 10
```

```
sample(x, size = 5)
```

```
## [1] 1 7 4 6 2
```

```
sample(x, size = 5, replace = T)
```

```
## [1] 10 10  2  6  7
```

Card matching problem

We can use the `sample()` function to simulate the card matching problem. Here, we have $n = 10$ cards. We can ask what is the probability of winning the game (i.e. at least one card match)?

This is a single game:

```
n <- 10
deck <- 1:n
game <- sample(deck)
game == deck
```

```
##  [1] FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE
```

```
sum(game == deck)
```

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

We can use the `replicate()` function to repeatedly simulate the game for $N = 1000$ times. Notice that both of the following are acceptable, and that the only difference is where we check whether we have a match:

```
r <- replicate(10000, sum(sample(deck) == deck))
sum(r >= 1)/ 10000
```

```
## [1] 0.6288
```

```
r2 <- replicate(10000, sum(sample(deck) == deck) >= 1)
sum(r2) / 10000
```

```
## [1] 0.6347
```