Tutorial 1: Probability Theory and Distributions

Jan Vogler (jan.vogler@duke.edu)

September 2, 2016

Today's agenda

1. Combinations and permutations
2. Basic probability
3. Conditional probability

1. Combinations and permutations

Combinations and permutations fundamentally rest on the use of factorials.

In R, factorials are written in the following way:

```
factorial(3)  # 3! = 3*2*1 = 6
```

```
## [1] 6
```

```
factorial(5)  # 5! = 5*4*3*2*1 = 120
```

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

There is a second factorial function in R, called the logarithmic factorial.

This function is used for high value factorials because they become increasingly difficult to store.

```
lfactorial(2)  # log(2*1) = log(2) + log(1) = 0.6931472
```

```
## [1] 0.6931472
```

In order to write a permutation with the factorial command, we need to know "n" and "k". Let n=10 and k=7.

```
factorial(10)/factorial(3)  # This is the expression for 10!/3! = 604800
```

```
## [1] 604800
```

In order to write a combination with the factorial command, we need to account for the different orders in which the elements can occur. In the case of n=10 and k=7, there are 7! different combinations in which the elements can occur.

```
factorial(10)/(factorial(3) * factorial(7))  # This is the expression for 10!/(3!*7!) = 120
```

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

As you can see, there are much fewer combinations than permutations, why is this the case?

R has a built-in command for combinations.

```
choose(10, 3)  # If we did everything right, it should return the same value '120'
```

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

2. Basic probability

Let us create a vector that contains all possible outcomes of a die roll.

```
die = c(1, 2, 3, 4, 5, 6)
```

What is the sample space in this case?

Now let's sample without replacement.

```
sample(die, size=1) # Gives you a random number from the die
```

```
## [1] 5
```

```
sample(die, size=2) # Gives you two random numbers from the die
```

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

```
sample(die, size=7) # Returns an error, why?
```

```
## Error in sample.int(length(x), size, replace, prob): cannot take a sample larger than the population
```

Without running the code, what outcome would you expect from the following command?

```
sample(die, size=6)
```

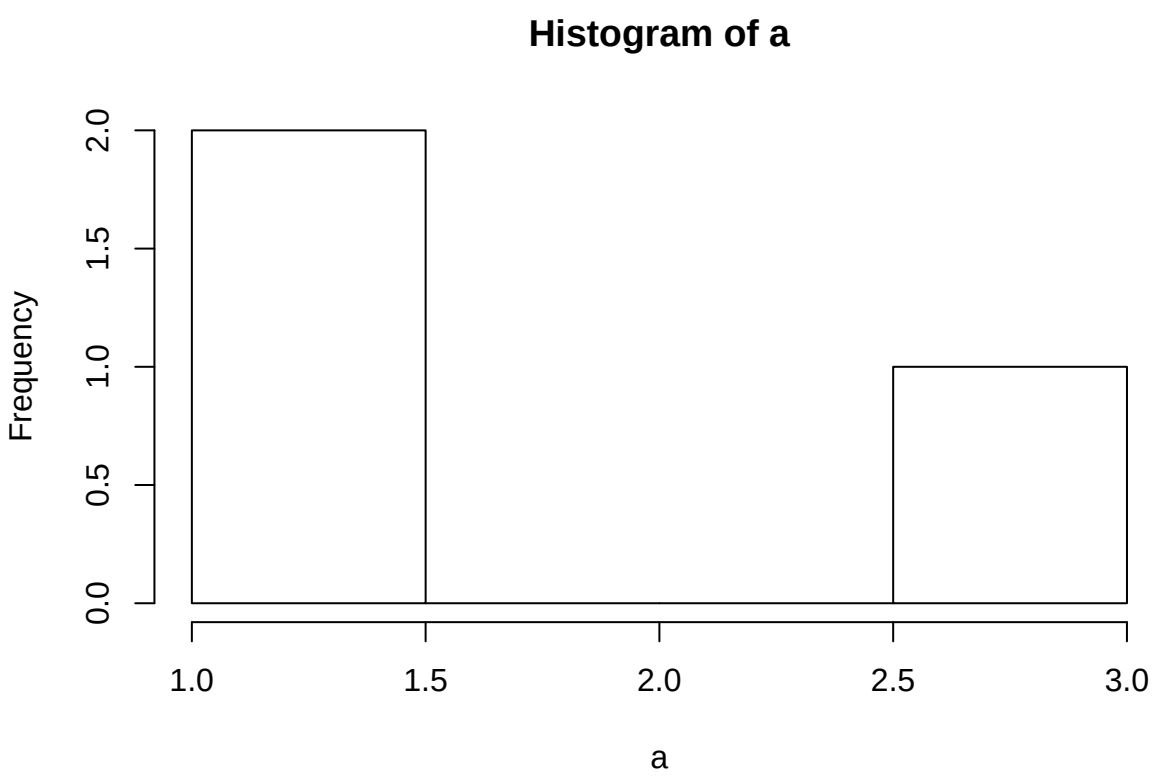
In order to make sure that you can draw n $>$ 6, you need to replace a number after drawing it.

```
sample(die, size = 10, replace = T)
```

```
##  [1] 4 5 2 6 5 3 3 2 6 2
```

Let's create several samples and look at their distribution.

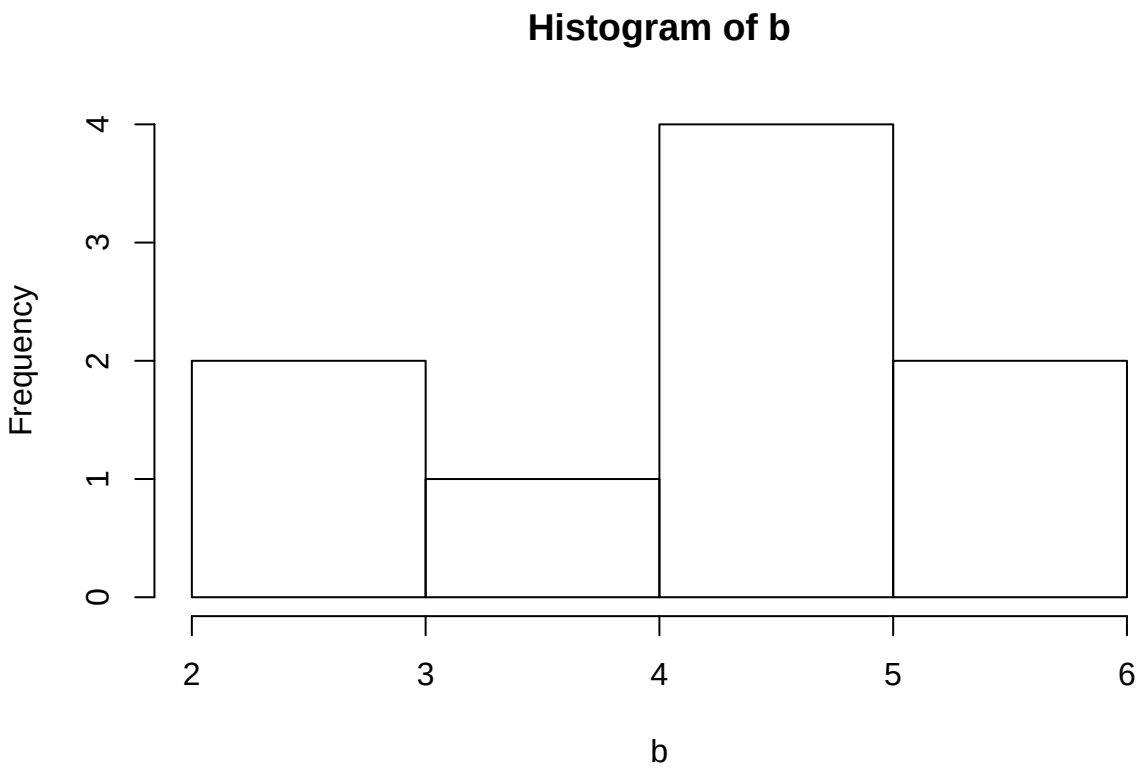
```
a = sample(die, size = 3, replace = T)
hist(a)  # Gives you a histogram
```




```
mean(a)
```

```
## [1] 1.666667
```

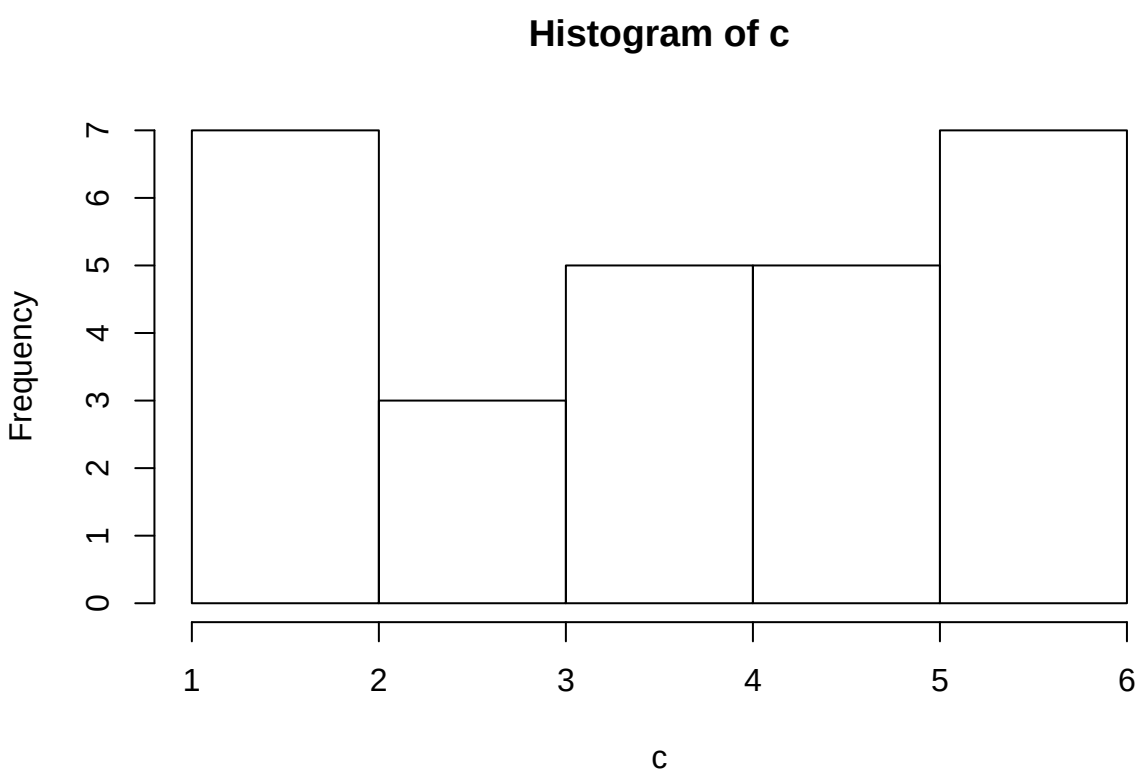
```
b = sample(die, size = 9, replace = T)
hist(b)
```




```
mean(b)
```

```
## [1] 4.555556
```

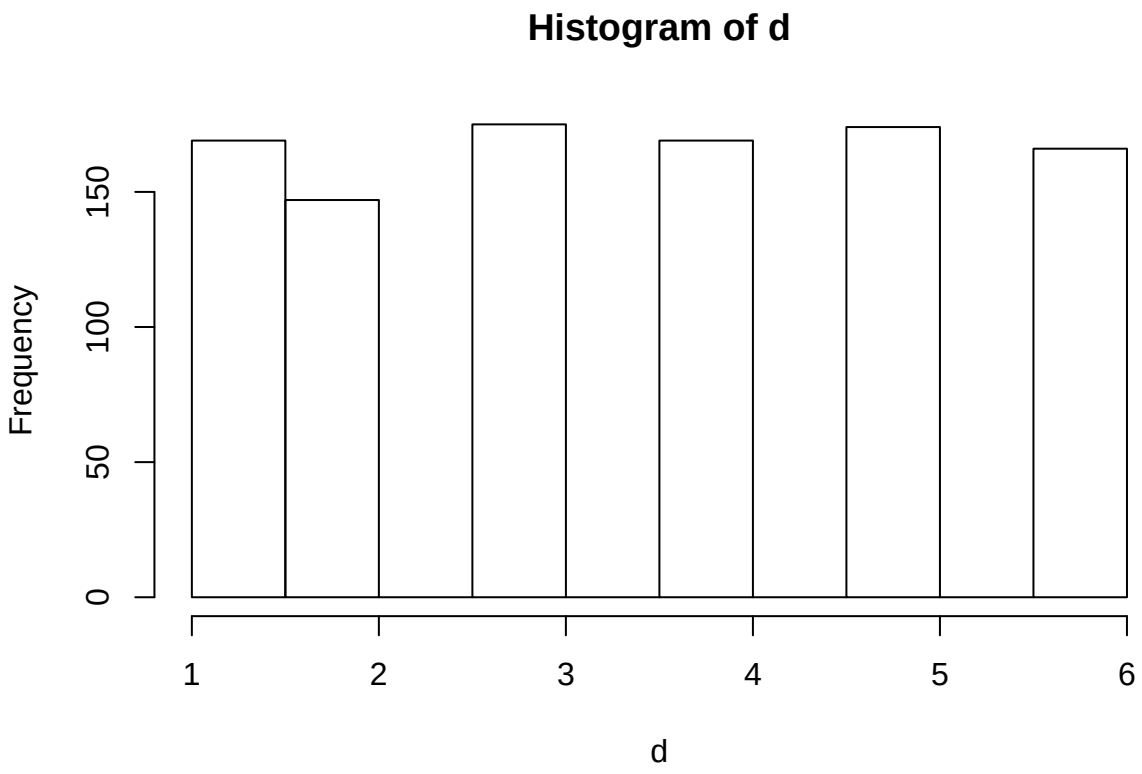
```
c = sample(die, size = 27, replace = T)
hist(c)
```




```r
mean(c)
```

```
## [1] 3.851852
```

```r
d = sample(die, size = 1000, replace = T)
hist(d)
```




```
mean(d)
```

```
## [1] 3.53
```

Looking at plot "d", what can you tell about the distribution of the die roll?

Let us define an event, A, which is that you draw an even number. What is the theoretical probability of A - Pr(A)?

How would we write a function that allows us to input a number of rolls and returns the proportion of rolls in which event A happens?

```
x = which(d%%2 == 0)  # This command allows you to find all elements that have a remainder of 0 when di
length(x)/1000
```

```
## [1] 0.482
```

```
evennum = function(rolls) {
    die = c(1, 2, 3, 4, 5, 6)
    sample1 = sample(die, size = rolls, replace = T)
    event = which(sample1%%2 == 0)
    proportion = length(event)/rolls
    print(proportion)
}

evennum(1000)  # Our theoretical expectation would be 0.5
```

```
## [1] 0.494
```

Let us define a second event B, which is that you draw a number smaller than 4

How would we write a function for that?

```
smaller4 = function(rolls) {
    die = c(1, 2, 3, 4, 5, 6)
    sample1 = sample(die, size = rolls, replace = T)
    event = which(sample1 < 4)
    proportion = length(event)/rolls
    print(proportion)
}

smaller4(1000)  # Our theoretical expectation would be 0.5
```

```
## [1] 0.499
```

Remember that Pr(A union B) = Pr(A) + Pr(B) - Pr(A intersection B). Let us try to find Pr(A union B) for events A and B as defined above.

Theoretically all numbers but 5 meet our condition, so theoretically we would expect the Pr(A union B) to be $5/6 \sim 0.833$.

Let us write an (overly) complex function that illustrates all of this.

```
probunionAB = function(rolls) {
    die = c(1, 2, 3, 4, 5, 6)
    sample1 = sample(die, size = rolls, replace = T)
    event1 = which(sample1%%2 == 0)  # This is A
    event2 = which(sample1 < 4)  # This is B
    event3 = which(sample1%%2 == 0 & sample1 < 4)  # This is A intersection B
    proportion = (length(event1)/rolls + length(event2)/rolls - length(event3)/rolls)
    print(proportion)
}

probunionAB(1000)  # Returns a number very close to 0.833 for large n
```

```
## [1] 0.824
```

3. Conditional probability

Next we will consider the Monty Hall problem. This will be a bonus problem on the homework.

There are three possible locations of the prize:

```
prizeoptions = c(1, 2, 3)  # Define a vector that has all three locations in them - door 1, 2, and 3
prize = sample(prizeoptions, size = 1)  # Randomly draw a location of the prize
prize  # Where is it?
```

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

Let us assume that you also choose a door at random at first:

```r
doorchosen = sample(prizeoptions, size = 1)  # You can draw from the same vector because you also draw
doorchosen
```

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

What happens if your door equals the location of the prize? Monty Hall will show you one of the other two doors at random

If "doorchosen" is equal to "prize", the following happens:

```r
doorshown = sample(prizeoptions[-prize], size = 1)
doorshown
```

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

What happens if your door does not have the prize in it? That means your door is an empty door. Then, Monty Hall will show you the only remaining door that is also empty. Think about how that could be written in R code.

For your homework: Write a function in which you input the number of trials. The function should execute the above steps and always make you switch to the other door after Monty has shown you an emtpy one.

The function should give you the proportion of times that you were right in "n" trials. This won't be easy, you have to apply most things you have learned today.