Topic 11: Probability

One of the first and most important points about probability is that probability is always a value between 0 and 1, inclusive. If, for a probability question you get and answer that is less than 0 or greater than 1 then your answer is wrong!

There is a small caution to demonstrate here. R displays numbers that are really close to 0 by using a scientific notation. Look at the following command and the associated console output.

```
3       # probability is always between 0 and 1
4       #
5       # demonstrate a small "catch" to this, namely
6       # that R expresses really small, close to 0,
7       # values in "scientific" form.  For example,
8       # look at the number 0.00000006478.
9  0.00000006478
```

```
>       # probability is always between 0 and 1
>       #
>       # demonstrate a small "catch" to this, namely
>       # that R expresses really small, close to 0,
>       # values in "scientific" form.  For example,
>       # look at the number 0.00000006478.
> 0.00000006478
[1] 6.478e-08
```

The 6.478e-08 means 6.478 times 10 to the minus 8 power. To convert this back to non-scientific form we would need to move the decimal point 8 places to the left.

000000006.478

In statistics when we say something is an experiment we mean that it is a non-deterministic task. It is something that we can repeat again and again, each time getting a result, but where we can never be sure, before we run the experiment, of the exact result that we get.

An experiment can be performed many times. A trial is just one of those times.

```
11      # we will run 4 trials of the experiment of
12      # getting 5 random values between 1 and 99.
13      # Note that if you run this script then you
14      # will get different results than are shown
15      # here. Sort the values to make it easier to
16      # inspect the lists.
17 sort( as.integer( runif( 5, 1, 100)) )
18 sort( as.integer( runif( 5, 1, 100)) )
19 sort( as.integer( runif( 5, 1, 100)) )
20 sort( as.integer( runif( 5, 1, 100)) )
```

```
>       # we will run 4 trials of the experiment of
>       # getting 5 random values between 1 and 99.
>       # Note that if you run this script then you
>       # will get different results than are shown
>       # here. Sort the values to make it easier to
>       # inspect the lists.
> sort( as.integer( runif( 5, 1, 100)) )
[1]  2 48 55 73 79
> sort( as.integer( runif( 5, 1, 100)) )
[1]  8 55 66 77 95
> sort( as.integer( runif( 5, 1, 100)) )
[1]  8 17 44 69 91
> sort( as.integer( runif( 5, 1, 100)) )
[1]  1 10 16 36 49
```

Each trial in an experiment has an outcome. The collection of all possible outcomes is the sample space.

There are times when it is helpful to actually write out a sample space, just so that we can see all of the possibilities and perhaps count how many of certain outcomes appear in the sample space.

Here is the sample space for a game where we have two boxes, a red and a blue one, and in each box we have eight slips of paper, one for each of the values 1 to 8, with the name of the number on the slip. We draw a slip from each box. The outcome is the number of distinct letters in the combined names.

		Number on slip of paper in blue box							
	number distinct letters	one	two	three	four	five	six	seven	eight
number on slip of paper in red box	one	3	5	6	6	6	6	5	7
	two	5	3	6	6	7	6	7	7
	three	6	6	4	7	7	7	7	6
	four	6	6	7	4	7	7	8	9
	five	6	7	7	7	4	6	6	7
	six	6	6	7	7	6	3	6	7
	seven	5	7	7	8	6	6	4	8
	eight	7	7	6	9	7	7	8	5

There are 64 possible outcomes. Exactly 4 of the outcomes are the value 8. This makes the probability of getting an answer of 8 to be 4/64=0.0625. There are exactly 25 outcomes that are 7. Therefore, the probability of getting an answer of 7 is 24/64=0.375. There are 0 outcomes that are 10. Therefore, the probability of getting an answer of 10 is 0/64=0. There are 64 outcomes that are 9 or less, so the probability of getting an answer that is 9 or less is 64/64=1.

There are many times when the possible outcomes in the sample space are made up of separate choices. For example consider the case where we draw a slip of paper from either the blue hat (getting one of the numbers one through eight), flipping a coin (getting a "head" or a "tail"), and picking a fruit from a bowl of 5 different fruits (O=orange, P=pear, A=apple, G=grape, and L=lime). Then our sample space is all the possible triplets of values (a number, a H or T, a fruit). One such outcome would be (4,H,P). We might ask, how many different outcomes are there? That answer would be 8*2*5=80.

We can see an example of this in R.

```
22      # Set up three different choices
23 blue_hat <- c(1,2,3,4,5,6,7,8)
24 coin <- c("H","T")
25 fruit <- c("O","P","A","G","L")
26      # Get thee sample space of an item
27      # taken from each choice
28 expand.grid( blue_hat, coin, fruit )
```

```
>      # Set up three different choices
> blue_hat <- c(1,2,3,4,5,6,7,8)
> coin <- c("H","T")
> fruit <- c("O","P","A","G","L")
>      # Get thee sample space of an item
>      # taken from each choice
> expand.grid( blue_hat, coin, fruit )
   Var1 Var2 Var3
1     1    H    O
2     2    H    O
3     3    H    O
4     4    H    O
```

Continued below:

5	5	H	O	25	1	T	P	45	5	T	A	65	1	H	L
6	6	H	O	26	2	T	P	46	6	T	A	66	2	H	L
7	7	H	O	27	3	T	P	47	7	T	A	67	3	H	L
8	8	H	O	28	4	T	P	48	8	T	A	68	4	H	L
9	1	T	O	29	5	T	P	49	1	H	G	69	5	H	L
10	2	T	O	30	6	T	P	50	2	H	G	70	6	H	L
11	3	T	O	31	7	T	P	51	3	H	G	71	7	H	L
12	4	T	O	32	8	T	P	52	4	H	G	72	8	H	L
13	5	T	O	33	1	H	A	53	5	H	G	73	1	T	L
14	6	T	O	34	2	H	A	54	6	H	G	74	2	T	L
15	7	T	O	35	3	H	A	55	7	H	G	75	3	T	L
16	8	T	O	36	4	H	A	56	8	H	G	76	4	T	L
17	1	H	P	37	5	H	A	57	1	T	G	77	5	T	L
18	2	H	P	38	6	H	A	58	2	T	G	78	6	T	L
19	3	H	P	39	7	H	A	59	3	T	G	79	7	T	L
20	4	H	P	40	8	H	A	60	4	T	G	80	8	T	L
21	5	H	P	41	1	T	A	61	5	T	G				
22	6	H	P	42	2	T	A	62	6	T	G				
23	7	H	P	43	3	T	A	63	7	T	G				
24	8	H	P	44	4	T	A	64	8	T	G				

We see all 80 of the different possible outcomes.

	Blue Hat	Coin	Fruit	
number of choices	8	2	5	
Size of sample space	8 times	2 times	5 equals	80

A different situation is where we might just choose a fruit from our selection. However, instead of just choosing one fruit we will choose three of them. (We will do this without replacing any of the fruit that we have chosen, building our outcome to hold three fruit items.) This is a different process. We will have 5 choices for our first pick, but only 4 for the second, and then only three choices for the third. Thus the number of different triplets of fruit would be 5*4*3=60. This is a situation of finding the number of permutations of things. In permutations the order of the choice matters. We can get R to find all of the permutations of our five fruits although we will have to import a few new tools to do this..

```
30      # look at permutations
31      # we need to load a special package to get
32      # the function we want.
33  install.packages("gtools")
34  library(gtools)
35
```

```
> # look at permutations
> # we need to load a special package to get
> # the function we want.
> install.packages("gtools")
WARNING: Rtools is required to build R packages but is not currently installed.
 Please download and install the appropriate version of Rtools before proceedin
g:

https://cran.rstudio.com/bin/windows/Rtools/
Installing package into 'C:/Users/rpala/Documents/R/win-library/4.0'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.0/gtools_3.8.2.zip'
Content type 'application/zip' length 336236 bytes (328 KB)
downloaded 328 KB

package 'gtools' successfully unpacked and MD5 sums checked
```

```
The downloaded binary packages are in
        C:\Users\rpala\AppData\Local\Temp\RtmpATr7Ny\downloaded_packages
> library(gtools)
```

```
    # recall the values in fruit
fruit
    # Now that we have the library installed we
    # can use the function permutations to get the
    # permutations of our 5 fruits taken 3 at a time.
permutations( 5, 3, fruit )
```

```
>     # recall the values in fruit
> fruit
[1] "O" "P" "A" "G" "L"
>     # Now that we have the library installed we
>     # can use the function permutations to get the
>     # permutations of our 5 fruits taken 3 at a time.
> permutations( 5, 3, fruit )
      [,1] [,2] [,3]
 [1,] "A"  "G"  "L"
 [2,] "A"  "G"  "O"
 [3,] "A"  "G"  "P"
 [4,] "A"  "L"  "G"
 [5,] "A"  "L"  "O"
 [6,] "A"  "L"  "P"
 [7,] "A"  "O"  "G"
 [8,] "A"  "O"  "L"
 [9,] "A"  "O"  "P"
[10,] "A"  "P"  "G"
[11,] "A"  "P"  "L"
[12,] "A"  "P"  "O"
[13,] "G"  "A"  "L"
[14,] "G"  "A"  "O"
[15,] "G"  "A"  "P"
[16,] "G"  "L"  "A"
[17,] "G"  "L"  "O"
[18,] "G"  "L"  "P"
[19,] "G"  "O"  "A"
[20,] "G"  "O"  "L"
[21,] "G"  "O"  "P"
[22,] "G"  "P"  "A"
[23,] "G"  "P"  "L"
[24,] "G"  "P"  "O"
[25,] "L"  "A"  "G"
[26,] "L"  "A"  "O"
[27,] "L"  "A"  "P"
[28,] "L"  "G"  "A"
[29,] "L"  "G"  "O"
[30,] "L"  "G"  "P"
[31,] "L"  "O"  "A"
[32,] "L"  "O"  "G"
[33,] "L"  "O"  "P"
[34,] "L"  "P"  "A"
[35,] "L"  "P"  "G"
[36,] "L"  "P"  "O"
[37,] "O"  "A"  "G"
[38,] "O"  "A"  "L"
[39,] "O"  "A"  "P"
[40,] "O"  "G"  "A"
[41,] "O"  "G"  "L"
[42,] "O"  "G"  "P"
[43,] "O"  "L"  "A"
[44,] "O"  "L"  "G"
[45,] "O"  "L"  "P"
[46,] "O"  "P"  "A"
[47,] "O"  "P"  "G"
[48,] "O"  "P"  "L"
[49,] "P"  "A"  "G"
[50,] "P"  "A"  "L"
[51,] "P"  "A"  "O"
[52,] "P"  "G"  "A"
[53,] "P"  "G"  "L"
[54,] "P"  "G"  "O"
[55,] "P"  "L"  "A"
[56,] "P"  "L"  "G"
[57,] "P"  "L"  "O"
[58,] "P"  "O"  "A"
[59,] "P"  "O"  "G"
[60,] "P"  "O"  "L"
```

We compute the number of permutations of 8 things taken 3 at a time as 8*7*6 = 336, because we have 8 choices for the first item, bit only 7 remaining choices for the second, and just 6 remaining for the third.

Take a small diversion here. we write 8*7*6*5*4*3*2*1 as 8! which is read as 8 factorial. We read 6! as six factorial and we know that 6!=6*5*4*3*2*1=720. We can even get R to do this with the factorial() function.

```
    # demonstrate factorial()
factorial(8)
factorial(6)
```

```
>     # demonstrate factorial()
> factorial(8)
[1] 40320
> factorial(6)
[1] 720
```

In general terms, for the number n we have

n! = n*(n-1)*(n-2)*(n-3)*...*3*2*1

However, there are some special cases. First 1! = 1 and, more surprising, 0! is, by definition, 1.

```
    # look at the surprising case
factorial( 0)
```

```
>     # look at the surprising case
> factorial( 0)
[1] 1
```

Now that we know about factorials we can observe that

$$8*7*6 = \frac{8*7*6*5*4*3*2*1}{5*4*3*2*1} = \frac{8!}{5!} = \frac{8!}{(8-3)!}$$

This generalizes to the formula for the number of permutations of n things taken r at a time:

$$_nP_r = \frac{n!}{(n-r)!}$$

We could use the factorial() function and this formula to find, in R, the number of permutations of 19 things taken 4 at a time.

```
46      # find the number of permutations of 19 things
47      # taken 4 at a time
48  factorial( 19 ) / factorial( 19-4 )
```

```
>       # find the number of permutations of 19 things
>       # taken 4 at a time
> factorial( 19 ) / factorial( 19-4 )
[1] 93024
```

This is all well and good, but some of us do not want to have to remember the formula. To that end we have some functions that will help. We will load the functions into the environment.

```
49      # load all of the related functions into the
50      # environment
51  source("../combinations.R")
```

```
>       # load all of the related functions into the
>       # environment
> source("../combinations.R")
>
```

Now the environment looks like:

Global Environment ▾	
Values	
blue_hat	num [1:8] 1 2 3 4 5 6 7 8
coin	chr [1:2] "H" "T"
fruit	chr [1:5] "O" "P" "A" "G" "L"
Functions	
nCr	function (n, r)
nPr	function (n, r)
num_comb	function (n, r)
num_perm	function (n, r)

Two of the newly added functions can help us. nPr() and num_perm() both compute the number of permutations of n things taken r at a time. They happen to compute the answer in different ways. One is not better than the other. (See the web page to check out the different approaches.) We will use both on the previous problem, just as an example.

```
52      # compute the answer to the previous problem
53      # using the two functions nPr() and num_perm().
54  nPr( 19, 4)
55  num_perm( 19, 4 )
```

```
>       # compute the answer to the previous problem
>       # using the two functions nPr() and num_perm().
> nPr( 19, 4)
[1] 93024
> num_perm( 19, 4 )
[1] 93024
```

One thing to notice about permutations is that order is important. In the list of the permutations of the fruits we find all of the following triplets:

```
 [1,] "A"  "G"  "L"
 [4,] "A"  "L"  "G"
[13,] "G"  "A"  "L"
[16,] "G"  "L"  "A"
[25,] "L"  "A"  "G"
[28,] "L"  "G"  "A"
```

Notice that there are 6 versions here. This is not surprising since there are 6 arrangements (i.e., permutations) of 3 things taken 3 at a time. In fact, for any triple in the list of permutations above there are 5 other triples with exactly the same fruits, just in a different order.

A different experiment would be to take our fruit, again select 3 items, but this time we do not keep track of the order in which the items are chosen. Now, order is not important. In this scheme the values AGL, ALG, GAL, GLA, LAG, and LGA are considered identical. It is as if we are making a fruit salad. We do not care which fruit comes first. In the end the selections are just mixed up. This is an example of looking at combinations. Because we loaded "gtools" before, we have access to the function combinations(). We will get the combinations of the 5 fruits taken 3 at a time.

```
56      # get the combinations of the 5 fruits taken
57      # 3 at a time
58  combinations( 5, 3, fruit)
```

```
>       # get the combinations of the 5 fruits taken
>       # 3 at a time
> combinations( 5, 3, fruit)
      [,1] [,2] [,3]
 [1,] "A"  "G"  "L"
 [2,] "A"  "G"  "O"
 [3,] "A"  "G"  "P"
 [4,] "A"  "L"  "O"
 [5,] "A"  "L"  "P"
 [6,] "A"  "O"  "P"
 [7,] "G"  "L"  "O"
 [8,] "G"  "L"  "P"
 [9,] "G"  "O"  "P"
[10,] "L"  "O"  "P"
```

Remember that each of the combinations just given would produce 6 permutation. Therefore, the number of combinations will be equal to the number of permutations of 5 things taken 3 at a time divided by the number of permutations of 3 things taken 3 at a time. The formula, in general terms, for the number of combinations of n things taken r at a time is given as

$$\binom{n}{r} = {}_nC_r = \frac{n!}{(n-r)!*r!}$$

That means that we could solve the problem of finding the number of combintions of 17 things taken 6 at a time using the factorial() function.

```
59      # find the number of combinations of 17 things
60      # taken 6 at a time
61  factorial( 17 ) /(factorial(17-6)*factorial(6))
```

```
>       # find the number of combinations of 17 things
>       # taken 6 at a time
> factorial( 17 ) /(factorial(17-6)*factorial(6))
[1] 12376
```

But that requires us to remember the formula. Instead, we could use either nCr() or num_comb(), functions we loaded before, to get that number.

```
62      # do this with the functions nCr and num_comb
63  nCr( 17, 6 )
64  num_comb( 17, 6 )
```

```
>       # do this with the functions nCr and num_comb
> nCr( 17, 6 )
[1] 12376
> num_comb( 17, 6 )
[1] 12376
```