Combinatorics

by Andri Signorell

Helsana Versicherungen AG, Health Sciences, Zurich
HWZ University of Applied Sciences in Business Administration, Zurich

andri@signorell.net

August 14, 2015

0. Key concepts

- **Permutation:** arrangement in some order.
- **Ordered versus unordered samples:** In ordered samples, the order of the elements in the sample matters; e.g. digits in a phone number, or the letters in a word. In unordered samples the order of the elements is irrelevant; e.g. elements in a subset, fruits in a fruit salad or lottery numbers.
- **Samples with replacement versus samples without replacement:** In the first case, repetition of the same element is allowed (e.g. numbers in a license plate); in the second, repetition not allowed (as in a lottery drawing—once a number has been drawn, it cannot be drawn again).

This document demonstrates the calculations with an example of the letters “a”, “b”, “c” and “d”, so n = 4. The formulas for the number of the permutations and combinations are given, as well as an approach how to construct the respective samples.

```
library(DescTools)

## Loading required package: manipulate

x <- letters[1:4]
n <- length(x)
m <- 2
```

1. Some basic number functions

In base R there are some basic number functions missing, found in other applications. In DescTools there’s the function `Primes(x)`, returning all the primes up to a given bound x. `IsPrime` checks if a number x is a prime number.
`Factorize` splits a number in its prime bases and returns the number and the specifically used power.
`GCD` returns the greatest common divisor and `LCM` the least common multiple of a vector of numbers.

```
# Find all prime numbers less than n.
Primes(n=20)

## [1]  2  3  5  7 11 13 17 19
```

```r
set.seed(23)
(x <- sample(1:20, 5))

## [1] 12  5  6 13 14


# check if the elements in x are prime numbers
IsPrime(x)

## [1] FALSE  TRUE FALSE  TRUE FALSE

# compute the prime factorizations of integers n
Factorize(n=c(56, 42))

## $`56`
##      p m
## [1,] 2 3
## [2,] 7 1
##
## $`42`
##      p m
## [1,] 2 1
## [2,] 3 1
## [3,] 7 1

# calculate the greatest common divisor of the given numbers
GCD(64, 80, 160)

## [1] 16

# calculate the least common multiple
LCM(10, 4, 12, 9, 2)

## [1] 180

# calculate the first 12 Fibonacci numbers
Fibonacci(1:12)

## [1]   1   1   2   3   5   8  13  21  34  55  89 144
```

2. Permutations of n objects

The number of permutation of n objects is calculated as: n!
We can use the base function `factorial` to calculate the total number of permutations. For creating the whole dataset of the permutations, there's the function `Permn` in DescTools.

```r
factorial(n)

## [1] 24

# generate all permutations
Permn(x)

##       [,1] [,2] [,3] [,4]
##  [1,] "a"  "b"  "c"  "d"
##  [2,] "b"  "a"  "c"  "d"
##  [3,] "b"  "c"  "a"  "d"
##  [4,] "a"  "c"  "b"  "d"
##  [5,] "c"  "a"  "b"  "d"
##  [6,] "c"  "b"  "a"  "d"
##  [7,] "b"  "c"  "d"  "a"
##  [8,] "a"  "c"  "d"  "b"
```

```
##  [9,] "c"  "a"  "d"  "b"
## [10,] "c"  "b"  "d"  "a"
## [11,] "a"  "b"  "d"  "c"
## [12,] "b"  "a"  "d"  "c"
## [13,] "c"  "d"  "a"  "b"
## [14,] "c"  "d"  "b"  "a"
## [15,] "a"  "d"  "b"  "c"
## [16,] "b"  "d"  "a"  "c"
## [17,] "b"  "d"  "c"  "a"
## [18,] "a"  "d"  "c"  "b"
## [19,] "d"  "a"  "b"  "c"
## [20,] "d"  "b"  "a"  "c"
## [21,] "d"  "b"  "c"  "a"
## [22,] "d"  "a"  "c"  "b"
## [23,] "d"  "c"  "a"  "b"
## [24,] "d"  "c"  "b"  "a"
```

3. Ordered samples of size m, without replacement, from n objects

The number is calculated as: $$n \cdot (n-1) \ldots (n-m+1) = \frac{n!}{(n-m)!} = {}_n P_r$$

There's no function in R which would directly calculate this, but we can use the function `choose` and multiply the result with m!. This is implemented so in `DescTools::CombN`.
For creating the whole dataset of the permutations, there's the function `CombSet` in DescTools, which has an argument m for defining the size of the subset to be drawn and where the replacement and order arguments can be set.

```
CombN(n, m)

## [1] 12

# generate all samples
CombSet(x, m, repl=FALSE, ord=TRUE)

##       [,1] [,2]
##  [1,] "a"  "b"
##  [2,] "b"  "a"
##  [3,] "a"  "c"
##  [4,] "c"  "a"
##  [5,] "a"  "d"
##  [6,] "d"  "a"
##  [7,] "b"  "c"
##  [8,] "c"  "b"
##  [9,] "b"  "d"
## [10,] "d"  "b"
## [11,] "c"  "d"
## [12,] "d"  "c"
```

4. Unordered samples of size m, without replacement, from a set of n objects

The number is calculated as: $$\binom{n}{m} = \frac{{}_n P_r}{m!} = \frac{n \cdot (n-1) \ldots (n-m+1)}{m!} = \frac{n!}{m! \cdot (n-m)!}$$

That's exactly what `choose` returns.
For creating the whole dataset of the combinations, again the function `CombSet` can be used.

```
choose(n, m)

## [1] 6

# generate all samples
CombSet(x, m, repl=FALSE, ord=FALSE)

##      [,1] [,2]
## [1,] "a"  "b"
## [2,] "a"  "c"
## [3,] "a"  "d"
## [4,] "b"  "c"
## [5,] "b"  "d"
## [6,] "c"  "d"
```

5. Ordered samples of size m, with replacement, from n objects

The number is calculated as: n^m
This can directly be entered as `n^m` into R.
For creating the whole dataset of the combinations, again the function `CombSet` can be used. This solution is based on Rs `expand.grid` and `replicate` functions.

```
n^m

## [1] 16

# or as alternative
CombN(x, m, repl=TRUE, ord=TRUE)

## [1] 16

# generate all samples
CombSet(x, m, repl=TRUE, ord=TRUE)

##       [,1] [,2]
##  [1,] "a" "a"
##  [2,] "b" "a"
##  [3,] "c" "a"
##  [4,] "d" "a"
##  [5,] "a" "b"
##  [6,] "b" "b"
##  [7,] "c" "b"
##  [8,] "d" "b"
##  [9,] "a" "c"
## [10,] "b" "c"
## [11,] "c" "c"
## [12,] "d" "c"
## [13,] "a" "d"
## [14,] "b" "d"
## [15,] "c" "d"
## [16,] "d" "d"
```

6. Unordered samples of size m, with replacement, from a set of n objects

The number is calculated as: $\binom{n+m-1}{m} = \frac{(n+m-1)!}{m! \cdot (n-1)!}$

Here again we can use the function `choose` with the appropriate arguments.

Creating the whole dataset of the combinations, needed the most sophisticated approach. The idea is to start with the dataset from the ordered samples of size m, without replacement (4.) and keep only the unique entries (concerning the containing letters). This is encapsulated in the function `CombSet`.

Again for the numbers either the base R solution with `choose` or the `CombN` from DescTools can be used.

```
choose(n + m - 1, m)

## [1] 10

# or as alternative
CombN(x, m, repl=TRUE, ord=FALSE)

## [1] 10

# generate all samples
CombSet(x, m, repl=TRUE, ord=FALSE)

##       [,1] [,2]
##  [1,] "a"  "a"
##  [2,] "a"  "b"
##  [3,] "a"  "c"
##  [4,] "a"  "d"
##  [5,] "b"  "b"
##  [6,] "b"  "c"
##  [7,] "b"  "d"
##  [8,] "c"  "c"
##  [9,] "c"  "d"
## [10,] "d"  "d"
```

7. Generate all possible subsets

A list with all the subsets that can be built based on the elements given in x can be created by setting the argument m in `CombSet` to a vector of all desired lengths. (For all subsets set m=1:4 in the following example)

```
CombSet(letters[1:4], m=2:3)

## [[1]]
## [,1] [,2]
## [1,] "a"  "b"
## [2,] "a"  "c"
## [3,] "a"  "d"
## [4,] "b"  "c"
## [5,] "b"  "d"
## [6,] "c"  "d"
##
## [[2]]
## [,1] [,2] [,3]
## [1,] "a"  "b"  "c"
## [2,] "a"  "b"  "d"
## [3,] "a"  "c"  "d"
## [4,] "b"  "c"  "d"
```

8. Pairs from two different sets

If the pairs between two different sets should be found, there's the function `CombPairs`.

```
CombPairs(letters[1:3], letters[4:6])

##   Var1 Var2
## 1    a    d
## 2    b    d
## 3    c    d
## 4    a    e
## 5    b    e
## 6    c    e
## 7    a    f
## 8    b    f
## 9    c    f
```

9. References

Wollschläger, D. (2010, 2012) Grundlagen der Datenanalyse mit R, Springer, Berlin.