R Tutorial-02

T. Scofield

In this "issue":

- Loading packages in quarto
- Simulation of the sum of two dice
- Tutorial on making (small) data frames

You may click here to access the .qmd file.

Loading packages in quarto

First up, separate from the other things I want to do in this document, I load all the packages I think I'll need. In fact, it is completely irrelevant, so far as the rendering/compilation process goes, whether you have loaded the package into the console. Since I intend to use commands (`resample()` and `tally()`) the way they are implemented in **mosaic**, I am loading it now.

```
# One must include desired packages in a quarto document separately
# regardless of whether the package is loaded for use in the console.
require(mosaic)
```

```
Loading required package: mosaic
```

```
Registered S3 method overwritten by 'mosaic':
  method                           from
  fortify.SpatialPolygonsDataFrame ggplot2
```

```
The 'mosaic' package masks several functions from core packages in order to add
additional features.  The original behavior of these functions should not be affected by this
```

```
Attaching package: 'mosaic'

The following objects are masked from 'package:dplyr':

    count, do, tally

The following object is masked from 'package:Matrix':

    mean

The following object is masked from 'package:ggplot2':

    stat

The following objects are masked from 'package:stats':

    binom.test, cor, cor.test, cov, fivenum, IQR, median, prop.test,
    quantile, sd, t.test, var

The following objects are masked from 'package:base':

    max, mean, min, prod, range, sample, sum
```

```
require(gridExtra)
```

```
Loading required package: gridExtra


Attaching package: 'gridExtra'

The following object is masked from 'package:dplyr':

    combine
```

```
require(Lock5withR)
```

```
Loading required package: Lock5withR
```

```
require(fosdata)
```

```
Loading required package: fosdata
```

Loading these packages caused a lot of *junk* messages I really don't want in my .pdf. They can be suppressed (and I will do so in the future) by adding this line at the top of the relevant code block.

```
#| include: false
```

Simulation of the sum of two dice:

```
numTrials = 10000
die = c(1:6)
manyRolls = resample(die, size = numTrials) + resample(die, size=numTrials)
tally(~ manyRolls)
```

```
manyRolls
   2    3    4    5    6    7    8    9   10   11   12
 278  529  842 1086 1386 1636 1388 1183  828  591  253
```

The `tally()` command above tells us

- what values (sum of two dice) occurred (in the simulation), and
- how often

Said another way, `tally()` shows us the (approximate) **distribution** of the variable `X` = sum of two dice. There are other ways to display a distribution; this one gives the **frequencies** (or counts) of the various rolls, so is called a **frequency table**. You can choose, instead, to produce a relative frequency table, merely adding a switch to the above command:

```
tally(~ manyRolls, format="proportion")
```

```
manyRolls
     2      3      4      5      6      7      8      9     10     11     12
0.0278 0.0529 0.0842 0.1086 0.1386 0.1636 0.1388 0.1183 0.0828 0.0591 0.0253
```

I've touted simulation (such as that above using `resample()`) as a way to discover the probabilities of various events. Recall that, in class, we found theoretical probabilities for the rolls `X = 2, 3, 4, ..., 12`. Let's see if the fractions `1/36, 2/36, ...` are close to the simulated relative frequencies:

```
theoreticalProbabilities = c(1:6,5:1)/36
theoreticalProbabilities
```

```
 [1] 0.02777778 0.05555556 0.08333333 0.11111111 0.13888889 0.16666667
 [7] 0.13888889 0.11111111 0.08333333 0.05555556 0.02777778
```

They are a *near* match. Keep in mind that every simulation (i.e., every new run of 10000 trials) will produce different relative frequencies than the time before unless we control the seed of the random number generator. So the best we can expect is for simulation to closely match the theoretical values (in cases where have those theoretical values).

Tutorial on making (small) data frames

You might have occasion to build a data frame with R commands. The basic command structure goes this way:

```
data.frame(colName1 = vector1, colName2 = vector2, ...)
```

Say you're a fan of Calvin men's soccer team. A few weeks into the 2024 season you want to make a data frame containing game results. Including only results up through Sept. 15, 2024, the rubric above suggest this command:

```
cms24 = data.frame(
  date = c("08/30", "08/31", "09/06", "09/07", "09/13", "09/14"),
  opponent = c("LeTourneau Univ", "Lake Forest College", "Ohio Wesleyan Univ",
               "Ohio Northern Univ", "Pacific Lutheran Univ", "Willamette Univ"),
  gameSite = c(rep(c("home","away"), times=c(2,4))),
  us = c(4,2,1,1,1,1),
  them=c(0,1,2,5,2,0),
  result=c("win","win","loss","loss","loss","win")
)
```

This data frame, I called it `cms24`, behaves just like other data frames—like `iris`, for instance.

```
nrow(cms24)
```

```
[1] 6
```

```
names(cms24)
```

```
[1] "date"     "opponent" "gameSite" "us"       "them"     "result"
```

```
cms24
```

```
   date              opponent gameSite us them result
1 08/30       LeTourneau Univ     home  4    0    win
2 08/31   Lake Forest College     home  2    1    win
3 09/06    Ohio Wesleyan Univ     away  1    2   loss
4 09/07    Ohio Northern Univ     away  1    5   loss
5 09/13 Pacific Lutheran Univ     away  1    2   loss
6 09/14       Willamette Univ     away  1    0    win
```