Markdown Exercise

Using a Random Number Generator to create a cleaning schedule

Piet Jonker

2020-10-28

# Notes

*Welcome dear reader*

This document concerns an exercise in writing reproducible code and learning the markdown language in R.

## Objectives

Here are some notes on the objectives of this document.

1. The target audience of this document is regarded as the one and only Gerko Vink and possibily classmates
2. This document should be reproducible and run flawless on any device
3. There should be a glossary, consisting of three parts
   * Abbreviations (in text, e.g. *“RNG”*)
   * Long form (in glossary, e.g. *“Random Number Generator”*)
   * Exhaustive definition (in glossary, e.g. *“A Random Number Generator creates random numbers”*)
4. Easy to read and use
5. Each function will be seperately created and stored in a folder
6. Compatability with PDF, and if the glossary allows it also HTML

# Glossary

RNG

Random Number Generator

Random number generation (RNG) is a process which, through a device, generates a sequence of numbers or symbols that cannot be reasonably predicted better than by a random chance. Random number generators can be true hardware random-number generators (HRNGS), which generate random numbers as a function of current value of some physical environment attribute that is constantly changing in a manner that is practically impossible to model, or pseudo-random number generators (PRNGS), which generate numbers that look random, but are actually deterministic, and can be reproduced if the state of the PRNG is known.

# Introduction

## Background

Students at both selective and nonselective institutions frequently engage in academic procrastination regardless of their gender, race, or learning style. Studies indicate that fear of failure, aversiveness of the task, and fear of social disapproval by peers are primary motives for academic procrastination. found that academic and non-academic tasks should be challenging, yet fun, to heighten the likelihood that they are completed by students.

This document will aim to challenge the author in a fun way, as suggested by . Three methods of which two use a [RNG](#rng) will be deployed to automatically create a cleaning schedule for a six person household with three cleaning tasks. This creates a challenge that is fun and a solution to a relevant problem which the author faces.

Since we’ve established that procrastination is bad, we can postulate that it is desirable to minimize the amount of effort that needs to be undertaken in order to fulfill tasks, such as household chores. In student housing, cleaning schedules are often used as tool to plan divide tasks. However, student housing is a high throughput system where residents often move to other places and new roommates take their place. This immediatly causes the schedule to be outdated. Furthermore, the schedule can be outdated. Both factors cause the schedule to be outdated as we can see in Figure 1.

A new schedule will take effort to make, causing an increase of . To minimalize , it is useful to automate process as much as possible. In order to achieve this goal an approach to automate this process will be presented. This way we follow the guidelines by in the sense that it makes this assignment challenging, fun, but also useful.

It should have the following structure:

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Chore 1 | Chore 2 | Chore 3 |
| Date 1 | Name roommate | Name roommate | Name roommate |
| Date 2 | Name roommate | Name roommate | Name roommate |

# Method

## Deterministic versus Random Number Generator

The Random Number Generator will be used to divide tasks among roommates in a pseudo-random manner. This will then be compared to a deterministic approach so that the two methods can be compared.

First the seed is set so that our results are reproducible. Also, the tidyverse is loaded since we’ll need it later.

library(tidyverse)  
library(knitr)  
set.seed(123)

### Deterministic approach

First a function will be created for the deterministic approach. One pattern will repeat itself for the entire sequence of dates specified. We will do this as following:

createSchedule <- function(startDate = Sys.Date(), # the current date is used as default input  
 endDate = Sys.Date() + 305, #   
 createCSV = "no",   
 roommateNames  
 ) {  
  
 # create a string of dates between startDate and endDate  
 dateString <- seq(from = startDate,  
 to = endDate,  
 by = "days")  
  
 # turn it into dataframe  
 data <- as.data.frame(dateString)  
  
 # add weekdays  
 data <- data %>%  
 mutate(days = weekdays(dateString))  
  
 # filter out fridays  
 data <- data %>%  
 filter(days == "vrijdag" | days == "friday")  
  
 # shuffle the orders of the name strings so that they can be put into dataframe  
 roommateNames2 <- roommateNames[c(5,6,1,2,3,4)]  
 roommateNames3 <- roommateNames[c(3,4,5,6,1,2)]  
  
 # add columns with chores  
 data <- data %>%  
 mutate(gang = rep\_len(roommateNames, length.out = nrow(data)),  
 keuken = rep\_len(roommateNames2, length.out = nrow(data)),  
 badkamer = rep\_len(roommateNames3, length.out = nrow(data))  
 )  
  
 # format dates  
 data <- data %>%  
 mutate(datum = format(dateString, "%d-%m-%y"))  
  
 # delete weekday and dateString column  
 data <- data %>%  
 select(datum,   
 gang,  
 keuken,   
 badkamer)  
   
 if (createCSV == "yes") {  
 write.csv(data, file = "schedule.csv")  
 } else {  
 return(data)  
 }  
   
}

Now, we test the function and show the first 10 rows of the schedule. To do this in a reproducible manner, we specify:

* a start date
* an end date
* the names of my roommates

This will be used throughout this document. Completely by chance, the start date is my birthday and the end date Sinterklaas evening (which is my grandfathers birthday).

startDate <- as.Date("1996-02-16")  
endDate <- as.Date("1996-12-15")   
roommateNames <- c("thijs", "jur", "wies", "hidde", "piet", "jette")  
  
kable(createSchedule(startDate = startDate,  
 endDate = endDate,  
 createCSV = "no",   
 roommateNames = roommateNames)[1:10,])

|  |  |  |  |
| --- | --- | --- | --- |
| datum | gang | keuken | badkamer |
| 16-02-96 | thijs | piet | wies |
| 23-02-96 | jur | jette | hidde |
| 01-03-96 | wies | thijs | piet |
| 08-03-96 | hidde | jur | jette |
| 15-03-96 | piet | wies | thijs |
| 22-03-96 | jette | hidde | jur |
| 29-03-96 | thijs | piet | wies |
| 05-04-96 | jur | jette | hidde |
| 12-04-96 | wies | thijs | piet |
| 19-04-96 | hidde | jur | jette |

### RNG approach 1

In this version, we use the sample function to sample one name per chore per week, The sample function uses the [RNG](#rng). This approach might not be the most convenient, since it is possible for one person to have all the chores in a given week.

createScheduleFull <- function(startDate = Sys.Date(),  
 endDate = Sys.Date() + 305,  
 createCSV = "no",   
 roommateNames  
 ) {  
  
 # create a string of dates between startDate and endDate  
 dateString <- seq(from = startDate,  
 to = endDate,  
 by = "days")  
  
 # turn it into dataframe  
 data <- as.data.frame(dateString)  
  
 # add weekdays  
 data <- data %>%  
 mutate(days = weekdays(dateString))  
  
 # filter out fridays  
 data <- data %>%  
 filter(days == "vrijdag" | days == "friday")  
   
 data <- data %>%   
 rowwise() %>%   
 mutate(gang = sample(roommateNames, 1, replace = TRUE),  
 keuken = sample(roommateNames, 1, replace = TRUE),  
 badkamer = sample(roommateNames, 1, replace = TRUE)  
 )  
   
 # format dates  
 data <- data %>%  
 mutate(datum = format(dateString, "%d-%m-%y"))  
  
 # delete weekday column and rename  
 data <- data %>%  
 select(datum,   
 gang,  
 keuken,   
 badkamer)  
   
 if (createCSV == "yes") {  
 write.csv(data, file = "schedule.csv")  
 } else {  
 return(data)  
 }  
   
}

Again, we test the function and show the first 10 rows

kable(createScheduleFull(startDate = startDate,  
 endDate = endDate,  
 createCSV = "no",   
 roommateNames = roommateNames)[1:10,])

|  |  |  |  |
| --- | --- | --- | --- |
| datum | gang | keuken | badkamer |
| 16-02-96 | wies | jette | jette |
| 23-02-96 | jette | thijs | wies |
| 01-03-96 | wies | jur | jette |
| 08-03-96 | jur | piet | piet |
| 15-03-96 | jur | piet | wies |
| 22-03-96 | jette | hidde | jette |
| 29-03-96 | wies | piet | jur |
| 05-04-96 | piet | jur | piet |
| 12-04-96 | hidde | thijs | piet |
| 19-04-96 | jette | thijs | wies |

### RNG approach 2

Now, we create the another function which uses the sample function to pick roommates for chores. Every other week three names from the string of six names will be randomly assigned for a task. The next week, the three remaining names from that sample will be assigned. This way, a random sample is taken, but we avoid the problem of someone having multiple chores in one week.

createScheduleSemi <- function(startDate = Sys.Date(),  
 endDate = Sys.Date() + 305,  
 createCSV = "no",   
 roommateNames  
 ) {  
  
 # create a string of dates between startDate and endDate  
 dateString <- seq(from = startDate,  
 to = endDate,  
 by = "days")  
  
 # turn it into dataframe  
 data <- as.data.frame(dateString)  
  
 # add weekdays  
 data <- data %>%  
 mutate(days = weekdays(dateString))  
  
 # filter out fridays  
 data <- data %>%  
 filter(days == "vrijdag" | days == "friday")  
  
 # draw a random sample, fill the first row with the first three elements of the sampled vector and the row after with the remaining three elements of the sampled vector  
 for(i in seq(from = 1, to = nrow(data), by = 2)){  
 sample <- sample(roommateNames, 6)  
 data[i,3] <- sample[1]  
 data[i,4] <- sample[2]  
 data[i,5] <- sample[3]  
 data[i+1, 3] <- sample[4]  
 data[i+1, 4] <- sample[5]  
 data[i+1, 5] <- sample[6]  
 }  
   
 # rename columns for clarity  
 data <- data %>%   
 rename(gang = V3,  
 keuken = V4,  
 badkamer = V5)  
   
 # format dates  
 data <- data %>%  
 mutate(datum = format(dateString, "%d-%m-%y"))  
  
 # delete weekday column and rename  
 data <- data %>%  
 select(datum,   
 gang,  
 keuken,   
 badkamer)  
   
 if (createCSV == "yes") {  
 write.csv(data, file = "schedule.csv")  
 } else {  
 return(data)  
 }  
   
}

Again, we test the function and show the first 10 rows

kable(createScheduleSemi(startDate = startDate,  
 endDate = endDate,  
 createCSV = "no",   
 roommateNames = roommateNames)[1:10,])

|  |  |  |  |
| --- | --- | --- | --- |
| datum | gang | keuken | badkamer |
| 16-02-96 | piet | wies | thijs |
| 23-02-96 | jette | jur | hidde |
| 01-03-96 | thijs | jur | hidde |
| 08-03-96 | jette | wies | piet |
| 15-03-96 | hidde | jette | thijs |
| 22-03-96 | jur | wies | piet |
| 29-03-96 | thijs | wies | hidde |
| 05-04-96 | piet | jette | jur |
| 12-04-96 | hidde | jette | jur |
| 19-04-96 | thijs | piet | wies |

We have now showed three methods to create schedules, where two use a RNG. I would really like to do an analysis go into depth on the statistics of the approaches such as double or triple assignments in a week. Also, I could have done some form of simulation to see if the approaches are fair when done X amount of times.

Unfortunetaly, most of my time went into creating the bookdown/markdown format, so I’ll save this part for a later endeavour. Thanks for reading!