Randomness and Random Sampling

Introduction

Goals

In this tutorial we are going to learn

- What is a pseudo-random generator?
- How to generate pseudo-random numbers
- How to randomly sample from a set of N things
- How to perform repeatable random sequences

using R

Pseudo-random generators

People are bad at generating random numbers because we have all sorts of unconscious biases that favour some number over others. Ideally, we want to observe some random process and extract random numbers from such a measurement. For example tossing some dice. This however, makes for slow random number generation and does not permit much automation. Think how long it takes to draw the Lotto numbers on TV.

In practice we make do with pseudo-random number generators implemented as code that runs on a computer.

> A pseudorandom number generator (PRNG) is an algorithm for generating a sequence of numbers whose properties approximate the properties of sequences of random numbers.

A pseudo-random generator is a process that appears to be random but is not. Pseudo-random sequences typically exhibit statistical randomness while being generated by an entirely deterministic process. That means we can repeatedly generate the same sequence of random numbers if we want to. These are generated by some fixed algorithm, but *appear*, for all practical purposes, to be random.

In R we use the function `runif()` . It is so named because it generates random real-numbers within a range, giving equal probability to every possible number. In this sense it is a Randon UNIForm distribution generator.

Below is the documentation for `runif()`

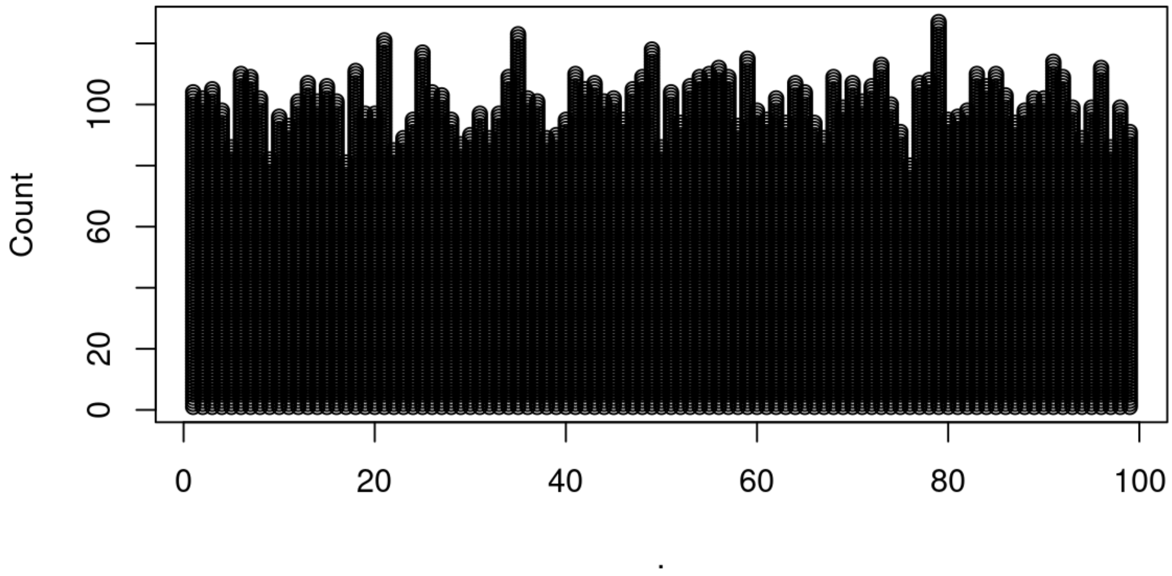
> ```
> runif(n, min = 0, max = 1)
> ```
>
> **Arguments**
>
> - **n** number of observations. If length(n) > 1, the length is taken to be the number required.
> - **min, max** lower and upper limits of the distribution. Must be finite.
>
> If min or max are not specified they assume the default values of 0 and 1 respectively. The uniform distribution has density f(x) = 1/(max-min) for min ≤ x ≤ max.

Plot random numbers

Let’s draw a dotplot of 10,000 random numbers between 1 and 100 to see how good the random number generator really is. If it is good, the dotplot should be quite flat across its range. Just click the “Run Code” button. You will see so many circles that the chart will look black.

R-code Start Over Run Code

```
library(TeachingDemos)
runif(n = 10000, min = 1, max = 100) %>%
  floor() %>%
  dots()
```

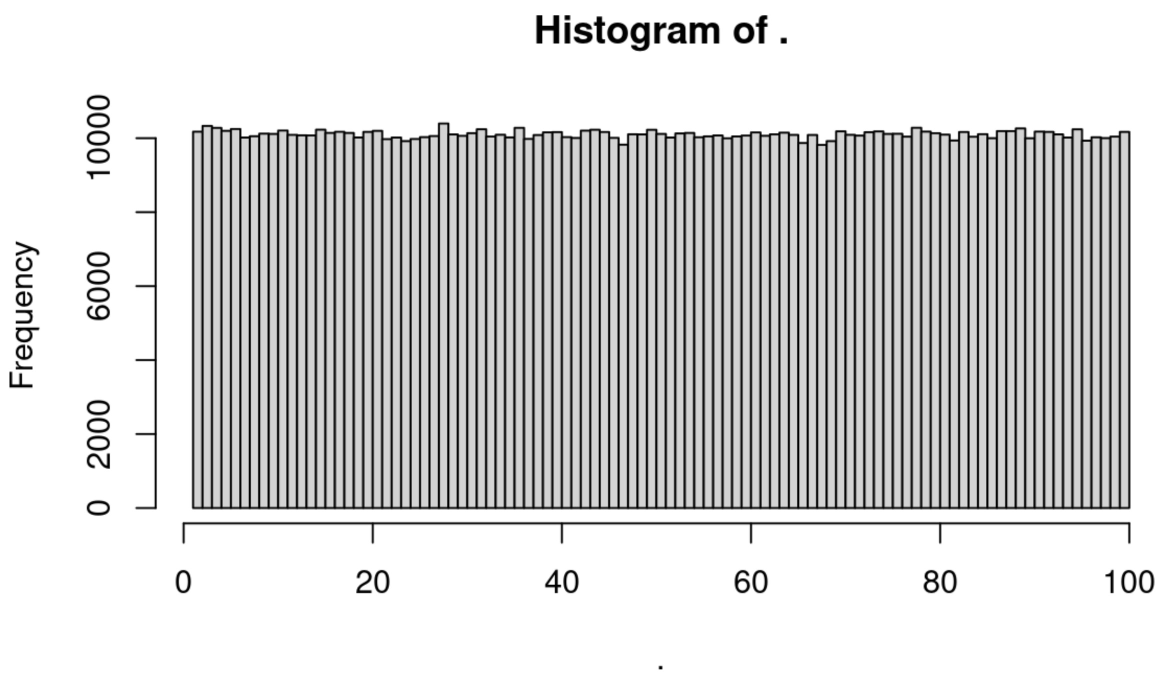


From the dotplot the pseudo-random numbers seem to be fairly evenly allocated across the range. We can chart this faster using a histogram, since it does not have 10,000 circle to draw. Let’s push the number of random numbers to 1 million. Please change the 10,000 to 1,000,000 (without the commas) i.e. add two extra zeros.

R-code Start Over Run Code Submit Answer

```
runif(n = 1000000, min = 1, max = 100) %>%
  hist(breaks = 100)

```



Cool job! To be clear, you just generated **1 million** random numbers between 1 and 100 and plotted their frequency histogram.

I hope you are impressed

Simple Random Sampling

Now that you know how to generate random numbers, how do we perform random sampling?

For example, suppose there are N cases in a population, but you want a sample of n of them, where n < N

To do this fairly, every case is given a $\frac{n}{N}$ chance of being selected for the sample. Sampling can be performed **without replacement** i.e., one deliberately avoids choosing any member of the population more than once, or **with replacement** where such duplicates are allowed. You will encounter *bootstrap* *sampling* later in this course, which is an example of sampling **with replacement**.

- Sampling done **without replacement** is not strictly independent sampling. Each case selection depends on what sampling has preceded it to avoid duplicates.
- For a small sample from a large population, sampling **without replacement** is approximately the same as sampling **with replacement**, since the probability of choosing the same individual twice is low.

Sampling without replacement

Let's do some simple random sampling (without replacement) using R.

Suppose we have a dataset called "population" with cases labelled 1 to 100,000 in sequence. We need to perform simple random sampling *without replacement* from this dataset, so that our sample has exactly 30 cases. N = 100000, n = 30

The code below will list the first 20 cases in the "population" dataset.

R-code Start Over Run Code

```
head(population, n = 20)


```

```
       id      var1       var2
1   case1 0.8740369 0.74238856
2   case2 0.2483726 0.31558021
3   case3 0.4885991 0.58819326
4   case4 0.6227667 0.13611296
5   case5 0.9026309 0.74777059
6   case6 0.6688046 0.31734833
7   case7 0.3578356 0.02493506
8   case8 0.4592608 0.30964444
9   case9 0.4716946 0.14532138
10 case10 0.9303656 0.04529568
11 case11 0.5148335 0.29236205
12 case12 0.1218752 0.70418891
13 case13 0.5854816 0.10238669
14 case14 0.1099663 0.60458678
15 case15 0.9049738 0.48299812
16 case16 0.2573018 0.24702173
17 case17 0.5231224 0.51947733
18 case18 0.5546247 0.91699598
19 case19 0.3910285 0.11679053
20 case20 0.2808143 0.33121649
```

The code snippet below, selects 30 case identifiers randomly sampled from dataset "population." We can view these case ids. Each time you press the "Run Code" button, a different random selection is generated.

R-code Start Over Run Code Submit Answer

```
sampleIds <- sample(population$id, size = 30, replace = FALSE)
sampleIds
```

```
 [1] "case88479" "case71169" "case3784"  "case42173" "case83088" "case92833"
 [7] "case86182" "case71956" "case7330"  "case24720" "case46821" "case54612"
[13] "case88743" "case34467" "case232"   "case627"   "case24746" "case83593"
[19] "case27704" "case23651" "case56729" "case50600" "case66523" "case17940"
[25] "case68461" "case61699" "case67678" "case47079" "case81035" "case87950"
```

Magnificent! Each time you press the ‘Submit answer’ button, a different random selection is generated. Notice that the case-ids are no longer in ascending order; the order is randomised as it would be if the case-ids were randomly drawn from a hat.

In order to create our sample we need to select complete cases from the populations where the case-ids are the ones randomly selected.

R-code Start Over Run Code Submit Answer

```
library(dplyr)
sampleIds <- sample(population$id, size = 30, replace = FALSE)
sample <- filter(population, id %in% sampleIds)
sample
```

```
5  case12860 0.529807206 0.74373508
6  case16164 0.385641751 0.26586726
7  case16796 0.639295481 0.09762875
8  case17110 0.682280011 0.50634584
9  case18855 0.504349580 0.90609615
10 case25821 0.003252191 0.37639076
11 case27097 0.275156441 0.03020367
12 case36954 0.379811054 0.95992725
13 case39440 0.531638961 0.27031262
14 case48717 0.232375630 0.61399314
15 case49255 0.896834094 0.52839058
16 case51763 0.307972340 0.58534781
17 case59293 0.544621327 0.60656351
18 case64137 0.493802811 0.48137320
19 case66259 0.959763828 0.08449366
20 case67588 0.434468342 0.39665457
21 case73456 0.688844635 0.33105698
22 case74215 0.548374127 0.68030400
23 case75269 0.060311805 0.65642588
24 case78664 0.344958598 0.45116594
25 case78981 0.059421705 0.49567383
26 case79506 0.987560845 0.11221285
27 case88845 0.090279895 0.93882692
28 case89345 0.292485036 0.60772052
29 case89969 0.066906967 0.43285011
30 case99748 0.699397395 0.07183330
```

Wicked smaht! Using `filter()` has restored the order of cases.

Sampling with replacement

Let's try this again *with replacement*.

Suppose we have a dataset called "population" with cases labelled 1 to 100,000 in sequence. We need to perform simple random sampling *with replacement* from this dataset, so that our sample has exactly 30 cases. N = 100000, n = 30

The code snippet below, selects 30 case identifiers randomly sampled from dataset "population." We can view these case ids. Please change the replace parameter from FALSE to TRUE.

R-code Start Over Run Code Submit Answer

```
sampleIds <- sample(population$id, size = 30, replace = TRUE)
sampleIds
```

```
 [1] "case25196" "case37763" "case98539" "case95899" "case6144"  "case6261"
 [7] "case62423" "case50490" "case77009" "case74009" "case53634" "case84359"
[13] "case19055" "case95429" "case37956" "case32178" "case33722" "case27065"
[19] "case35717" "case24163" "case94118" "case28046" "case9647"  "case78338"
[25] "case42088" "case83990" "case31838" "case81457" "case26044" "case82861"
```

Splendid! Each time you press the ‘Submit answer’ button, a different random selection is generated.

With replacement sampling allows for a case to be picked, more than once. We can use the `duplicated()` to determine whether duplicates have appeared in the sample.

In order to create our sample we need to select complete cases from the populations where the case-ids are the ones randomly selected. Try several presses of the “Run Code” button to see if a duplicate appears.

R-code Start Over Run Code

```
sampleIds <- sample(population$id, size = 30, replace = TRUE)
any(duplicated(sampleIds))
```

```
[1] FALSE
```

The chances of duplicates are too low to encounter easily.

Please edit the code below so that the sample size becomes 10,000. We should see some duplicates now.

R-code Start Over Run Code Submit Answer

```
sampleIds <- sample(population$id, size = 10000, replace = TRUE)
any(duplicated(sampleIds))
```

```
[1] TRUE
```

Swell job! Now we are seeing duplicates in the sample of 10,000.

Duplicates

Repeatable Sequences

As you now know, computers generate pseudo-random numbers rather than true random numbers. One consequence of this is that we can control the sequence of random numbers that we generate. The code below will generate 5 random real-numbers between 1 and 100. But with a bit of trickery anyone can predict exactly what these 5 random numbers will be. They will be:

```
## 80.89838 81.54644 22.52098 56.10587 1.27716
```

See if this is right by clicking "Run Code".

R-code Start Over Run Code

```
set.seed(86645)
runif(n = 5, min = 1, max = 100)

```

```
[1] 80.89838 81.54644 22.52098 56.10587  1.27716
```

As you may have guessed the presence of the `set.seed()` was responsible for making this sequence repeatable. The actual value (e.g. 86645) is not important. If the generator is given a integer "seed", the sequence of random numbers will be repeatable whenever that seed is used again.

Consider the following two variants of code. You may need to press the continue button to see into the second tab.

No resetting With resetting

Firstly, two sets without resetting of the random-generator seed.

R-code Start Over Run Code

```
set.seed(1234)
runif(n = 5, min = 1, max = 100)
runif(n = 5, min = 1, max = 100)
```

```
[1] 12.25664 62.60764 61.31820 62.71456 86.23062
```

```
[1] 64.39075  1.94008 24.02250 66.94229 51.91086
```

The two random number sequences are quite different.

Secondly, two sets with resetting of the random-generator seed.

R-code Start Over Run Code

```
set.seed(914)
runif(n = 5, min = 1, max = 100)
set.seed(914)
runif(n = 5, min = 1, max = 100)
```

```
[1]  7.179692 96.816651 80.802243 19.403945 15.692503
```

```
[1]  7.179692 96.816651 80.802243 19.403945 15.692503
```

The two random number sequences are exactly the same.

Wrap-up

Repeatability

The use of `set.seed()` is an important trick. It allows us to write code that involves randomness, in such a way as to be repeatable. The basis of science is repeatability. However we must remember that this particular repeatability is a feature of the imperfect behaviour of pseudo-random generators. Nature, which has the potential for true randomness, is nowhere near as repeatable as this trick might lead you to believe.

Code

In this tutorial you have been exposed to the following R functions:

Function	**Package**	**Description**
`any()`	base	whether any values are TRUE
`data.frame()`	base	rectangular data set
`duplicated()`	base	whether case is duplicated
`floor()`	base	truncate to integer
`paste0()`	base	concatenate text
`sample()`	base	random sampling
`set.seed()`	base	set the random seed
`hist`	graphics	chart a histogram
`runif()`	stats	random uniform distribution
`filter`	dplyr	choose certain rows
`dots()`	TeachingDemos	dot chart

In case this on-line tutorial is not available in future, you may want to keep a PDF copy of this material for reference purposes.

Export as PDF

Topic not yet completed...