

Computational Statistics - Lab 03

Annalena Erhard (anner218) and Maximilian Pfundstein (maxpf364)

2019-01-26

Contents

1	Question 1: Cluster Sampling	1
2	Question 2: Different distributions	4
3	Source Code	5

1 Question 1: Cluster Sampling

An opinion pool is assumed to be performed in several locations of Sweden by sending interviewers to this location. Of course, it is unreasonable from the financial point of view to visit each city. Instead, a decision was done to use random sampling without replacement with the probabilities proportional to the number of inhabitants of the city to select 20 cities. Explore the file `population.xls`. Note that names in bold are counties, not cities.

Task: Import necessary information to R.

Municipality	Population
Botkyrka	81195
Danderyd	31150
Ekerö	25095
Haninge	76237
Huddinge	95798
Järfälla	65295

Task: Use a uniform random number generator to create a function that selects 1 city from the whole list by the probability scheme offered above (do not use standard sampling functions present in R).

```
# Implementation Max
seed = 12345

get_city_by_urn_wo = function(city_pool) {

  # We take the cumulative sum and then runif from 1 to max(cumulative sum).
  # This way we respect the proportions. As we need every intermediate result,
  # we use a loop
  cumulative_pop_sum = 0

  for (i in 1:nrow(city_pool)) {
    cumulative_pop_sum = cumulative_pop_sum + city_pool$Population[i]
    city_pool$CumSum[i] = cumulative_pop_sum
  }

  # Now we get a random value between 1 to max(cumulative sum). As larger muni-
```

```

# cipoalities have larger ranges, this works as expected
selection =
  floor(runif(n = 1, min = 1, max = city_pool$CumSum[nrow(city_pool)]))

# Return the first city which has a greater CumSum than the selection
return(city_pool[city_pool$CumSum > selection,][1, c(1, 2)])
}

```

Task: Use the function you have created in step 2 as follows:

- Apply it to the list of all cities and select one city
- Remove this city from the list
- Apply this function again to the updated list of the cities
- Remove this city from the list
- ... and so on until you get exactly 20 cities.

Answer: We will combine all of these steps in one function. We're lazy.

```

get_n_cities = function(data, n) {

  # Create a copy to not touch the original data.
  city_pool = data
  selected_cities = data.frame()

  # As long as we don't have n samples, get one and remove it from the pool,
# as we sample without replacement
  while(nrow(selected_cities) < n) {
    selected_city = get_city_by_urn_wo(city_pool)
    selected_cities = rbind(selected_cities, selected_city)
    city_pool = city_pool[!rownames(city_pool) %in% rownames(selected_cities),]
  }

  return(selected_cities)
}

sample = get_n_cities(data, 20)

```

Task: Run the program. Which cities were selected? What can you say about the size of the selected cities?

Answer:

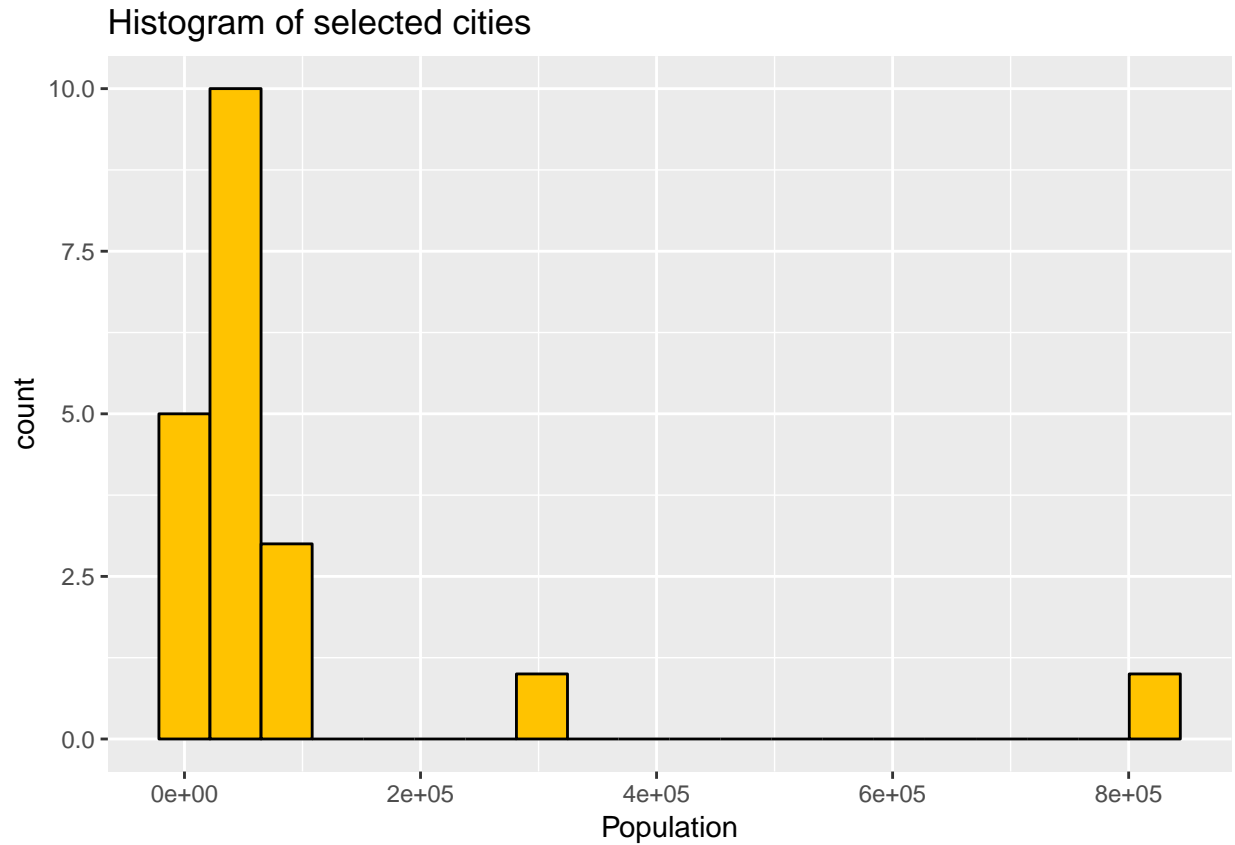
The following cities were selected:

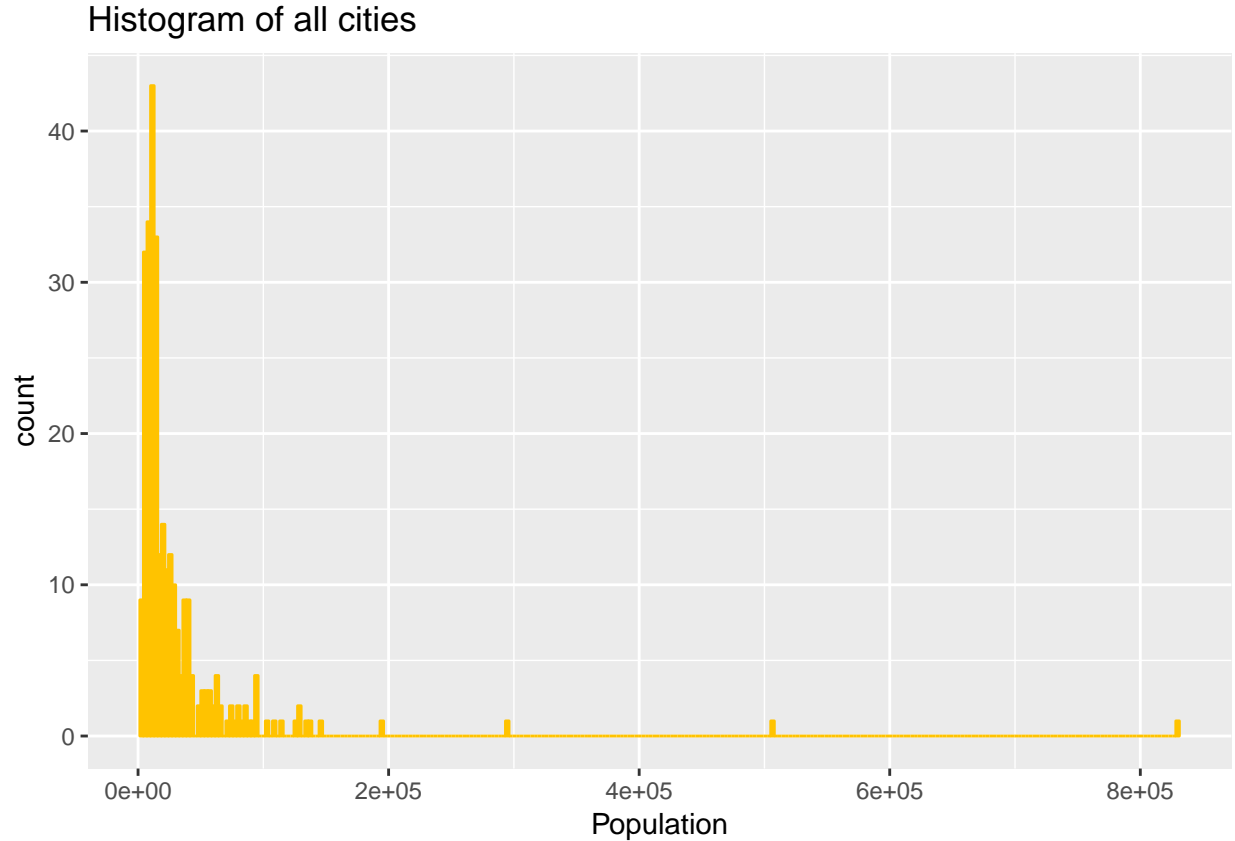
##	Municipality	Population
## 14	Sollentuna	63347
## 136	Alingsås	37515
## 17	Sundbyberg	37722
## 165	Skövde	50984
## 112	Malmö	293909
## 73	Markaryd	9559
## 42	Trosa	11446
## 16	Stockholm	829417
## 82	Kalmar	62388
## 223	Borlänge	48681
## 186	Filipstad	10626
## 7	Lidingö	43445
## 116	Sjöbo	18153

##	287	Piteå	40860
##	191	Karlstad	84736
##	13	Sigtuna	39219
##	20	Täby	63014
##	139	Borås	102458
##	6	Järfälla	65295
##	236	Älvdalen	7288

It can be seen, that mostly cities with a population greater than the mean (32009.25) were drawn.

Task : Plot one histogram showing the size of all cities of the country. Plot another histogram showing the size of the 20 selected cities. Conclusions?





Answer: We see that both histograms have a similar shape. This means that our sampling function is taking the size of the city into account and we created a similar distribution with a smaller subset. That is exactly what we needed to perform the opinion pool.

2 Question 2: Different distributions

The double exponential (Laplace) distribution is given by formula

$$DE(\mu, \alpha) = \frac{\alpha}{2} e^{-\alpha|x-\mu|}$$

Task: Write a code generating double exponential distribution $DE(0, 1)$ from $Unif(0, 1)$ by using the inverse CDF method. Explain how you obtained that code step by step. Generate 10000 random numbers from this distribution, plot the histogram and comment whether the result looks reasonable.

Task: Use the Acceptance/rejection method with $DE(0, 1)$ as a majorizing density to generate $\mathcal{N}(0, 1)$ variables. Explain step by step how this was done. How did you choose constant c in this method? Generate 2000 random numbers $\mathcal{N}(0, 1)$ using your code and plot the histogram. Compute the average rejection rate R in the acceptance/rejection procedure. What is the expected rejection rate ER and how close is it to R ? Generate 2000 numbers from $\mathcal{N}(0, 1)$ using standard `rnorm()` procedure, plot the histogram and compare the obtained two histograms.

3 Source Code

```
knitr::opts_chunk$set(echo = TRUE, cache = FALSE, include = TRUE, eval = TRUE)
library(knitr)
library(readxl)
library(ggplot2)

# reading the data in -----
data = read.csv("population.csv", sep = ";", encoding = "latin1")

kable(head(data))

# Implementation Max
seed = 12345

get_city_by_urn_wo = function(city_pool) {

  # We take the cumulative sum and then runif from 1 to max(cumulative sum).
  # This way we respect the proportions. As we need every intermediate result,
  # we use a loop
  cumulative_pop_sum = 0

  for (i in 1:nrow(city_pool)) {
    cumulative_pop_sum = cumulative_pop_sum + city_pool$Population[i]
    city_pool$CumSum[i] = cumulative_pop_sum
  }

  # Now we get a random value between 1 to max(cumulative sum). As larger muni-
  # cipalities have larger ranges, this works as expected
  selection =
    floor(runif(n = 1, min = 1, max = city_pool$CumSum[nrow(city_pool)]))

  # Return the first city which has a greater CumSum than the selection
  return(city_pool[city_pool$CumSum > selection,][1, c(1, 2)])
}

# Comments:
# - Why n if the function is supposed to select one?
# - This does not take into account that cities with a bigger population are more
#   likely to be selected
# - round() sometimes rounds downwards, sometimes upwards, for correct partitioning
#   floor() or ceil() is better
# - Setting the seed for each drawing will result in the same number every time
#   if the external seed supplier does not change it!

sample_wo_replacement = function(n, data, seed){
  samples = numeric()
  i = 1
  while (length(samples) < n) {
    set.seed(seed)
    a = round(runif(n = 1, min = 1, max = nrow(data)))
```

```

    while (a %in% samples) {
      set.seed(seed)
      a = round(runif(n = 1, min = 1, max = nrow(data)))
    }
    samples[i] = a
    i = i+1
  }
  return(samples)
}

get_n_cities = function(data, n) {

  # Create a copy to not touch the original data.
  city_pool = data
  selected_cities = data.frame()

  # As long as we don't have n samples, get one and remove it from the pool,
# as we sample without replacement
  while(nrow(selected_cities) < n) {
    selected_city = get_city_by_urn_wo(city_pool)
    selected_cities = rbind(selected_cities, selected_city)
    city_pool = city_pool[!rownames(city_pool) %in% rownames(selected_cities),]
  }

  return(selected_cities)
}

sample = get_n_cities(data, 20)

city_pool = data
selected_cities = data.frame()

selected_city_id = sample_wo_replacement(1, all_cities, 12345)

selected_cities = rbind(city_pool[selected_city_id,])
city_pool = all_cities[-selected_city,]

while (length(selected_cities) < 21) {
  selected_city_id = sample_wo_replacement(1, all_cities, 12345)
  all_cities = all_cities[-selected_city,]
  selected_cities = c(selected_cities, selected_city)
}

print(selected_cities)

sample

```

```
ggplot(sample)+  
  geom_histogram(aes(x = Population), bins = nrow(sample), color = "black", fill = "#FFC300") +  
  ggtitle("Histogram of selected cities")  
  
ggplot(data)+  
  geom_histogram(aes(x = Population), bins = nrow(data), color = "#FFC300", fill = "#FFC300") +  
  ggtitle("Histogram of all cities")
```