MAS6024, Assignment2

180128022

29 October 2018

Ans. (i) Here, we create a function called defective_leaflets that returns the proportion of defective leaflets in a random sample.

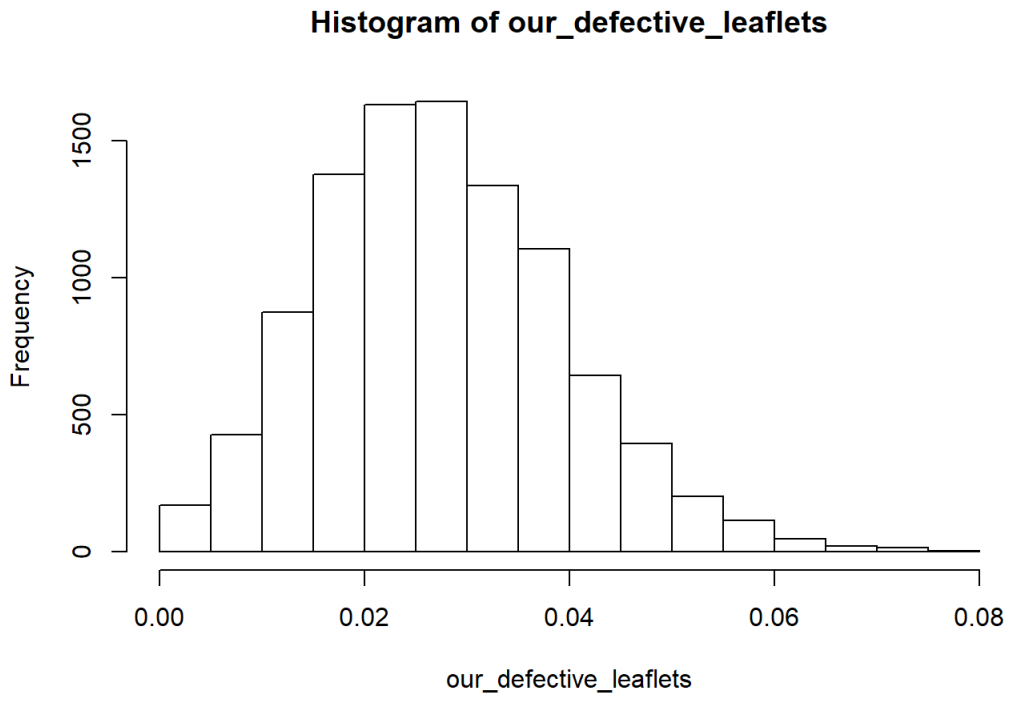
```
defective_leaflets <- function(N1, N2, p1, p2, n){
  # use of random binomial distribution to generate the following
  N_1_leaflets <- rbinom(n = N1, size=1, prob = p1) # number of leaflets from  1
  N_2_leaflets <- rbinom(n = N2, size=1, prob = p2) # number of leaflets from  2
  N_i_leaflets <- append(N_1_leaflets, N_2_leaflets) # append the values
  N_i_leaflets_shuffled <- sample(N_i_leaflets) # shuffle them
  proportion_of_sampled_N_i_leaflets <- sample(x = N_i_leaflets_shuffled, size = n, replace = FALSE, prob =
NULL) # sample n of these without replacement
  return(sum(proportion_of_sampled_N_i_leaflets)/n) # proportion of defectives
}
```

Ans. (ii) Assuming that N1 = 10000 , N2 = 20000, p1 = 0.05, p2 = 0.02 and n = 200, we repeat the above process 10000 times using the replicate function and plot the histogram of the probability estimates we get. From the histogram, we can say that the probability distribution of the proportion of defective leaflets that the managing director will observe is normal distribution as it is a bell-curve with mean greater than 0.

```
cat(defective_leaflets(10000, 20000, 0.05, 0.02, 200))
```

```
## 0.02
```

```
our_defective_leaflets<- replicate(10000, defective_leaflets(10000, 20000, 0.05, 0.02, 200))
hist(our_defective_leaflets)
```



Ans. (iii) Now, we create a function called monte_carlo_simulation to solve the problem using Monte Carlo simulation as N1+N2 leaflets are not independent. The function takes in the value of N1, N2, E (range of error for the absolute difference of p and p_hat where p is per-leaflet probability of the internal print devices as p1 = p2 = p, and p_hat is the estimate of p based on the random sample), p and x (number of replications, the reason we are taking this in the input function is because we want to cherry-pick the values of 'x' , which is under our control).

If this 'probability_of_difference' is greater than 0.95, we are going to return the smallest sample size.

```
monte_carlo_simulation <- function(N1, N2, E, p, x){
  m <-0 # initialise m
  probability <-1 # initialise probability
  condition_of_n <- eval(probability > 0.95) # check if true or false
  while(condition_of_n == T){
    m<-m+1 # increase m by 1
    p_hat<-replicate(n = x, defective_leaflets(N1, N2, p, p, m)) # find out p_hat
    difference_of_p <- abs(p - p_hat)
    probability_of_difference <- length(difference_of_p[difference_of_p<E])/x # find Pr(|p - p_hat| < 0.05)
    if (probability_of_difference > 0.95){
      return (m)
    }
  }
  return (m)
}
```

Ans. (iv) We use our above function to find the smallest sample size needed to ensure that p_hat is within 0.05 of p with probability exceeding 0.95.

Case 1: Taking the values - N1 = 10000, N2 = 20000, E = 0.05, p = 0.1, x = 10.

```
monte_carlo_simulation(10000, 20000, 0.05, 0.1, 10)
```

```
## [1] 49
```

Case 2: Taking the values - N1 = 10000, N2 = 20000, E = 0.05, p = 0.1, x = 100.

```
monte_carlo_simulation(10000, 20000, 0.05, 0.1, 100)
```

```
## [1] 101
```

Case 3: Taking the values - N1 = 10000, N2 = 20000, E = 0.05, p = 0.1, x = 1000.

```
monte_carlo_simulation(10000, 20000, 0.05, 0.1, 1000)
```

```
## [1] 127
```

Now, we can see that our sample sizes have better estimation, but the computation time increases with the increase of replications. So, we need to make a trade-off. We take the number of replications as 550 (average of the two better values of replication).

Case 4 : Taking the values - N1 = 10000, N2 = 20000, E = 0.05, p = 0.1, x = 550.

```
monte_carlo_simulation(10000, 20000, 0.05, 0.1, 550)
```

```
## [1] 115
```

Ans. (v) We can check how our smallest sample size gets affected by:

1. Making p constant and changing the value of E (epsilon)
2. Making E constant and changing the value of p

We set a seed value to see the changes in an accurate way. We take the number of replications to be 5.

Case 1 :

A. We will decrease the value of E while p remains constant

When E = 0.03

```
  set.seed(180128022) # seed value is set to my registration number
  monte_carlo_simulation(10000, 20000, 0.03, 0.1, 5)
```

```
## [1] 54
```

When E = 0.01

```
  set.seed(180128022)
  monte_carlo_simulation(10000, 20000, 0.01, 0.1, 5)
```

```
## [1] 341
```

B. We will increase the value of E while p remains constant

When E = 0.07

```
  set.seed(180128022)
  monte_carlo_simulation(10000, 20000, 0.07, 0.1, 5)
```

```
## [1] 21
```

When E = 0.1

```
  set.seed(180128022)
  monte_carlo_simulation(10000, 20000, 0.1, 0.1, 5)
```

```
## [1] 16
```

Case 2 :

A. We will decrease the value of p while E remains constant

When p = 0.05

```
  set.seed(180128022)
  monte_carlo_simulation(10000, 20000, 0.05, 0.05, 5)
```

```
## [1] 31
```

When p = 0.01

```
  set.seed(180128022)
  monte_carlo_simulation(10000, 20000, 0.05, 0.01, 5)
```

```
## [1] 1
```

B. We will increase the value of E while p remains constant

When p = 0.18

```
  set.seed(180128022)
  monte_carlo_simulation(10000, 20000, 0.05, 0.18, 5)
```

```
## [1] 75
```

When p = 0.2

```
  set.seed(180128022)
  monte_carlo_simulation(10000, 20000, 0.05, 0.2, 5)
```

```
## [1] 76
```

We can therefore observe:

1. Keeping p constant, the sample size is inversely proportional to the value of E.
2. Keeping E constant, the sample size is directly proportional to the value of p.

Apart from the two factors listed above, we know that the values of standard deviation and mean can also influence the value of the sample size as standard deviation and mean depend on n and p. Also, the change in the number of replications can have an impact on the value of n.