Stat 5810: Introduction to R Fall 2020

Name: Niranjan Poudel

Homework Assignment

90 Points — Due befor sem begins

General Instructions

For this fifth homework assignment, you have to create your own RMarkdown (.Rmd) file, based on files from class and from Homework 1, copy the question numbers and the answer options into your .Rmd file, and knit that file into a pdf file. **Alternatively** (and much easier!!!), use this .Rnw file as a template, just fill in the answers into the provided spaces, and knit into a pdf file.

Only the final resulting pdf file (from .Rmd or .Rnw) has to be submitted via Canvas. As previously stated, I would like to encourage potential and current MS and PhD students to work with .Rnw and LaTeX instead of .Rmd.

You need to learn how to write R code that is easily readable for others. There exists *Google's R Style Guide* that summarizes rules for good R style. These rules are accessible at `https://google.github.io/styleguide/Rguide.xml`. In particular, make sure that you always have a space after a comma and that you consistently use the same type of assignment operator, ideally `<-`. Look at the examples on this web page and follow the style whenever you write your own R code from now on.

Do not forget to replace my name and include your name instead! We will print the homeworks, so a homework with no name/my name on it can't be graded!

In all question parts, show your R code and the results!

Make sure that you reactivate all R code that has been commented out/made inactive in this file when you work on the solutions on your side — and copy such code into your Rmd file in case you work with RMarkdown.

(i) (28 Points) **San Francisco Housing Data Revisited:**
In this question, you have to work with actual housing data from the San Francisco area. Each question part **must** be answered by some kind of *apply* function. Other solutions may be possible but will result in **point deductions**.

Show your R code and the final results produced from within R for all question parts!

(a) (3 Points) Copy the San Francisco housing data set (`hw05_SFhousing.rda`) for this homework from Canvas into your local folder for this homework. For each component of the *cities* and *housing* data frames, determine the class. As a reminder, you **must** use one of the apply functions (or otherwise lose points).

Answer:

```
> load(file = "hw04_SFHousing.rda")
> lapply (cities, class)

$longitude
[1] "array"

$latitude
[1] "array"

$county
[1] "factor"

$medianPrice
[1] "array"

$medianSize
[1] "array"

$numHouses
[1] "array"

$medianBR
[1] "array"

> lapply (housing, class)

$county
[1] "factor"

$city
[1] "factor"

$zip
[1] "factor"
```

```
$street
[1] "character"

$price
[1] "numeric"

$br
[1] "integer"

$lsqft
[1] "numeric"

$bsqft
[1] "integer"

$year
[1] "integer"

$date
[1] "POSIXt"  "POSIXct"

$long
[1] "numeric"

$lat
[1] "numeric"

$quality
[1] "factor"

$match
[1] "factor"

$wk
[1] "Date"
```

```
> options(width = 80)  # For pdf display
```

(b) (4 Points) Determine how many missing values were present for each of the variables in *housing*. Hint: Using an orphan function (within the proper apply function) may be most efficient.

Answer:

```
> sapply(X = housing, FUN = function(x) sum(is.na(x)))  # more easy??
```

```
 county    city     zip  street   price      br   lsqft   bsqft    year    date
      0       0       5       0       0       0   21687     426    9202       0
   long     lat quality   match      wk
  23316   23316   23316   23316       0
```

(c) (4 Points) For each county (in *housing*), calculate the median price.

Answer:

```
> tapply(housing$price, housing$county, median)
```

```
      Alameda County  Contra Costa County         Marin County
              510000                466000                739000
        Napa County San Francisco County     San Mateo County
              505000                702000                700000
  Santa Clara County        Solano County        Sonoma County
              582000                380000                476500
```

(d) (4 Points) For each city (in *housing*), calculate the mean price. Display the resulting 10 highest prices in decreasing order.

Answer:

```
> head(sort(tapply(housing$price, housing$city, mean), decreasing = T), n = 10)
```

```
  Los Altos Hills           Atherton      Hillsborough Belvedere/Tiburon
          2393311            2379174           2354199           2217681
       Belvedere               Ross            Diablo Belvedere/tiburon
          2170088            2135883           1973025           1776572
    Monte Sereno     Stinson Beach
          1656639            1640469
```

(e) (3 Points) Look back at the top-10 results from the previous part, in particular at the resulting city names. What seemed to have happened during the data cleaning?

Answer:

The top ten cities are displayed above, the highest price was at Los Altos Hills with mean price of 2399911.**To comment on what happened during data cleaning should i not know what data looked like before cleaning?**

(f) (4 Points) Zip codes for San Francisco cover the range 94102 through 94134 — see `http://www.city-data.com/zipmaps/San-Francisco-California.html`. First, create a logical vector called SFZip that is TRUE when a zip code (in *housing*) belongs to San Francisco. Be careful when converting from factor to integer. Likely, your first attempt will not work. Try converting to character first — and then converting to integer. How many of the observations fall into the SF area (according to their zip codes)? (Note: You don't have to use apply in this part.)

Answer:

```
> SFZip <- as.logical(as.integer(as.character(housing$zip)) >= 94102
+                     & as.integer(as.character(housing$zip)) <= 94134)
> summary(SFZip)

   Mode   FALSE    TRUE    NA's
logical  273367    8134       5

> table(SFZip)

SFZip
 FALSE   TRUE
273367   8134
```

(g) (6 Points) Using *housing*, compare the average number of bedrooms (br) inside and outside of San Francisco. What do you notice? Provide a clear interpretation of this numerical output!

Answer:

```
> tapply(housing$br, SFZip, mean, na.rm = T)

   FALSE     TRUE
3.043085 2.369560
```

The outout shows that the avergae number of bedrooms inside San Fransciso is 2.370 and the average number of bedrooms outside San Fransico is 3.043 which suggest the housing in San Franscio are smaller compared to housing outside of San Franscio and suggest San Fransico might be a crowded city.

(ii) (34 Points) n **Factorial — Functions and Benchmarking:**
Recall that $n!$ (read: "n factorial") is defined as follows: $0! = 1$ and $n! = n \cdot (n-1)!$

Show your R code and the final results produced from within R for all question parts!

(a) (6×3 Points $= 18$ Points) Write six R functions called `myFactorial1` through `myFactorial6` that calculate $n!$ in six different ways. You can assume that n is a valid positive integer or equal to zero (if not, just let R produce its own error message). These six functions should have the following properties:

i. `myFactorial1` should be based on the existing `factorial` function.

Answer:

```
> myFactorial1 <- function(n) {
+   # Computes the factorial of zero and positive integer
+   #
+   # Args:
+   #   n : Positive integer or zero
+   #
+   # Returns:
+   #   The factorial of the number n
+   print(factorial(n))
+ }
> myFactorial1(0)
[1] 1
> myFactorial1(1)
[1] 1
> myFactorial1(5)
[1] 120
> myFactorial1(10)
[1] 3628800
> myFactorial1(100)
[1] 9.332622e+157
```

ii. `myFactorial2` should be based on the `prod` function.

Answer:

```
> myFactorial2 <- function(n) {
+   # Computes the factorial of zero and positive integer
+   #
+   # Args:
+   #   n : Positive integer or zero
+   #
+   # Returns:
+   #   The factorial of the number n
+   print(prod((n:0)[ - (n + 1)]))
+ }
> myFactorial2(0)
[1] 1
> myFactorial2(1)
[1] 1
> myFactorial2(5)
[1] 120
> myFactorial2(10)
[1] 3628800
> myFactorial2(100)
[1] 9.332622e+157
```

iii. `myFactorial3` should be based on a for–loop that multiplies the previous result with the current loop index.

Answer:

```
> myFactorial3 <- function(n) {
+   # Computes the factorial of zero and positive integer
+   #
+   # Args:
+   #   n : Positive integer or zero
+   #
+   # Returns:
+   #   The factorial of the number n
+   fac <- 1
+   for (i in 1:n) {
+     fac<- fac * i
+   }
+   if (fac == 0){
+     fac = 1
+   }
+   print(fac)
+ }
> myFactorial3(0)
[1] 1
> myFactorial3(1)
[1] 1
> myFactorial3(5)
[1] 120
> myFactorial3(10)
[1] 3628800
> myFactorial3(100)
[1] 9.332622e+157
```

iv. `myFactorial4` should be based on a while–loop that checks whether n has been surpassed. If not, the previous result should be multiplied with the current counter.

Answer:

```
> myFactorial4 <- function(n) {
+   # Computes the factorial of zero and positive integer
+   #
+   # Args:
+   #   n : Positive integer or zero
+   #
+   # Returns:
+   #   The factorial of the number n
+   fac <- 1
+   i <- 1
+   while (i <= n) {
+     fac = i * fac
+     i = i + 1
+   }
+   print(fac)
+ }
> myFactorial4(0)
[1] 1
> myFactorial4(1)
[1] 1
> myFactorial4(5)
[1] 120
> myFactorial4(10)
[1] 3628800
> myFactorial4(100)
[1] 9.332622e+157
```

v. `myFactorial5` should be done in a recursive way, using the fact that $\text{myFactorial5}(n) = n \cdot \text{myFactorial5}(n-1)$.

Answer:

```
> myFactorial5 <- function(n) {
+   # Computes the factorial of zero and positive integer
+   #
+   # Args:
+   #   n : Positive integer or zero
+   #
+   # Returns:
+   #   The factorial of the number n
+   if (n == 0){
+     return(1)
+   } else {
+     return(n * myFactorial5(n - 1))
+   }
+ }
> myFactorial5(0)
[1] 1
> myFactorial5(1)
[1] 1
> myFactorial5(5)
[1] 120
> myFactorial5(10)
[1] 3628800
> myFactorial5(100)
[1] 9.332622e+157
```

vi. `myFactorial6` should be similar to `myFactorial5`, but now write the recursion as myFactorial6(n) = myFactorial6$(n-1) \cdot n$.

Answer:

```
> myFactorial6 <- function(n) {
+   # Computes the factorial of zero and positive integer
+   #
+   # Args:
+   #   n : Positive integer or zero
+   #
+   # Returns:
+   #   The factorial of the number n
+  if (n == 0){
+     return(1)
+   } else {
+     return(myFactorial5(n - 1) * n)
+   }
+ }
> myFactorial6(0)
[1] 1
> myFactorial6(1)
[1] 1
> myFactorial6(5)
[1] 120
> myFactorial6(10)
[1] 3628800
> myFactorial6(100)
[1] 9.332622e+157
```

(b) (8 Points) Conduct a microbenchmark experiment for your previously created 6 functions: Calculate 100! for each of your 6 functions, using `times = 500`. Assign the results to a variable called `mBench` and show the numerical results.

Answer:

```
> library(microbenchmark)
> set.seed(550)
> mBench <- microbenchmark(myFactorial1(100), myFactorial2(100),
+                          myFactorial3(100), myFactorial4(100),
+                          myFactorial5(100), myFactorial6(100),
+                          times = 500)

> print(mBench)

Unit: microseconds
              expr    min     lq     mean median      uq      max neval
 myFactorial1(100) 18.489 20.196 23.12915 21.334 23.0410   81.920   500
 myFactorial2(100) 20.196 22.187 25.87972 23.325 25.1735   84.196   500
 myFactorial3(100) 20.765 22.471 26.82465 23.894 25.8850  194.275   500
 myFactorial4(100) 23.609 25.316 29.04280 26.738 28.4440  185.174   500
 myFactorial5(100) 33.849 35.556 62.99457 42.667 46.3650 9565.000   500
 myFactorial6(100) 33.849 35.840 43.74052 42.098 46.9340  122.311   500
```

(c) (2 Points) What is the class of `mBench`?

Answer:

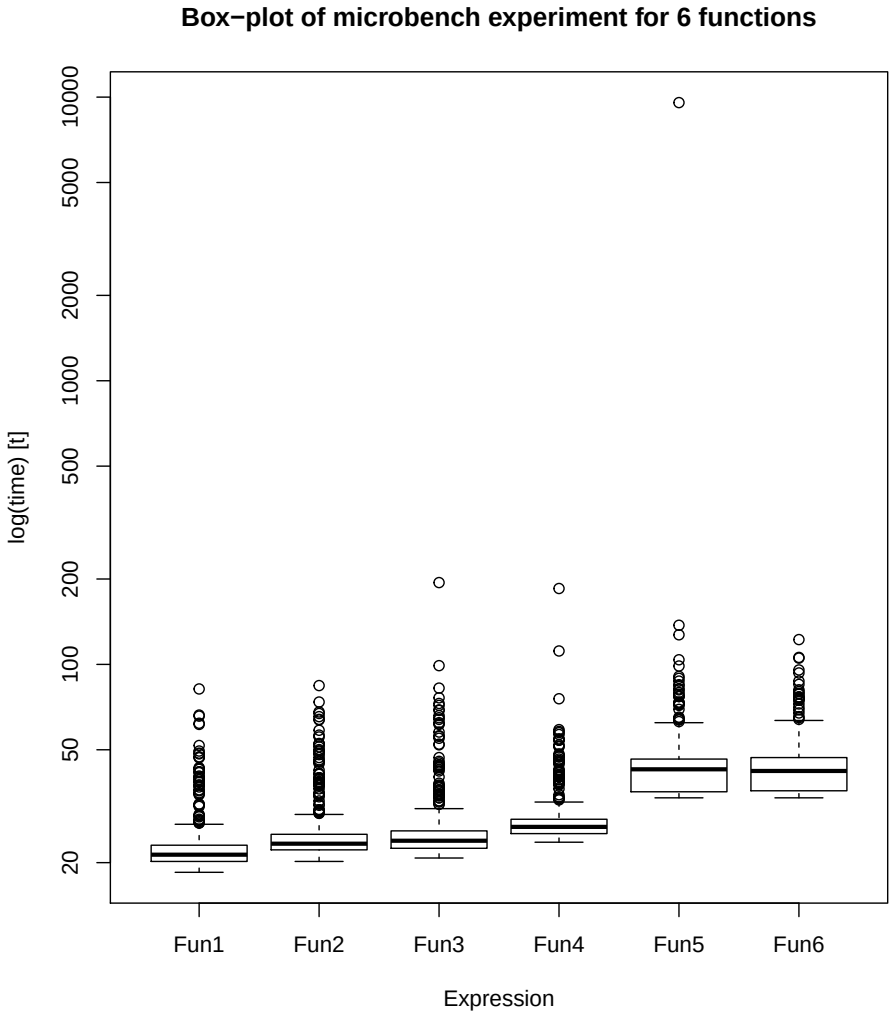
```
> str(mBench)

Classes 'microbenchmark' and 'data.frame':        3000 obs. of  2 variables:
 $ expr: Factor w/ 6 levels "myFactorial1(100)",..: 4 6 5 5 2 5 6 3 1 2 ...
 $ time: num  185174 122311 41530 36409 73672 ...
```

mBench has two classes as per the "str()" function dataframe and microbenchmark.

(d) (2 Points) Create side–by–side boxplots of `mBench`.

Answer:

```
> boxplot(mBench, names = c("Fun1", "Fun2", "Fun3", "Fun4", "Fun5", "Fun6"),
+         main = 'Box-plot of microbench experiment for 6 functions')
```



(e) (4 Points) Summarize the results from your microbenchmark experiment. Focus on the medians! Which function(s) for the calculation of $n!$ are fastest, which are slowest, which are about the same? Write about 3 to 5 sentences.

Answer: We have created six different functions to calculate the factorials for different numbers, Further, microbenchmark experiment was conducted to see the execution time of these functions. Basically the fastest one when looking at median was the function created by inbuilt factorial function. the two recursive functions almost took around equal time and were the slowest, among them the sixth function was the slowest one. The second third and fourth one were little slower than the inbuilt factorial function. (But the time taken basically also depends upon how we code inside the functions as

well).

(iii) (28 Points) **Central Limit Theorem Simulations:**
This question requires you to do some simulations to illustrate the Central Limit Theorem (CLT).

Show your R code and the final results produced from within R for all question parts!

(a) (15 Points) Write a function to simulate the sampling distribution of the mean, when the population has a binomial distribution. You can use the code below to help write your function:

```
> # number of samples
> nreps <- 10000
> # number of observations in each sample
> nobs <- 100
> # number of trials in each observation
> trials <- 5
> # probability of "success" in each trial
> ps <- 0.1
> # generate data
> binomSamps <- lapply(rep(nobs, nreps), rbinom,
+                      size = trials, prob = ps)
> # compute sample means
> mymeans <- sapply(binomSamps, mean)
> # draw a histogram
> hist(mymeans, freq = FALSE)
> # draw a normal probability plot
> qqnorm(mymeans)
> # calculate the mean of all sample means
> mean(mymeans)
```

Your function should have the following form:

```
centralLimit <- function(nreps, nobs, trials, ps,
                         hist = FALSE, qqnorm = FALSE, ...) {
...
}
```

The arguments of this function are:

- nreps is the number of samples
- nobs is the number of observations in each sample
- trials is the number of binomial trials in each observation
- ps is the probability of "success" in each trial

When the user sets `hist = TRUE`, the function draws a histogram of the observed sampling distribution of the mean. In this case, the ... refers to graphical options that are to be passed directly to the histogram function, e.g., titles, number of histogram bins, etc. When the user sets `qqnorm = TRUE`, the function draws a normal Q–Q plot. No additional arguments should be passed on to this function.

Finally, your function should return the mean of the nreps sample means.

Answer:

```
> CentralLimit <- function(nreps, nobs, trials, ps,
+                          hist = FALSE, qqnorm = FALSE, xlab = " ") {
+   # Computes the mean of the random samples, plots the histogram
+   # and qqlot as well to look at the distribution of the meams of
+   # randolmly generated samples
+   #
+   # Args:
+   #   nreps  : number of samples
+   #   nobs   : number of observation in each sample
+   #   trials : number of binomial trials in each observation
+   #   ps     : probability of success in each trial
+   #   hist   : Plots the histogram if TRUE is passed
+   #   qqnorm : Plots the normal plot if TRUE argument is passed
+   #
+   # Returns:
+   #   Returns the mean of all the samples, histogram and normal
+   #   and normal plot if arguments for the plot are TRUE
+   set.seed(4550)
+   binomSamps <- lapply(rep(nobs, nreps), rbinom,
+                        size = trials, prob = ps)
+   mymeans <- sapply(binomSamps, mean)
+
+
+   if(hist == TRUE) {
+     hist <-  hist(mymeans, freq = FALSE, xlab = xlab)
+   }
+
+   if(qqnorm == TRUE) {
+     normal <-   qqnorm(mymeans)
+   }
+   # return(invisible(list(mean(mymeans),hist,normal)))
+   return(mean(mymeans))  # Works for our question
+ }
```

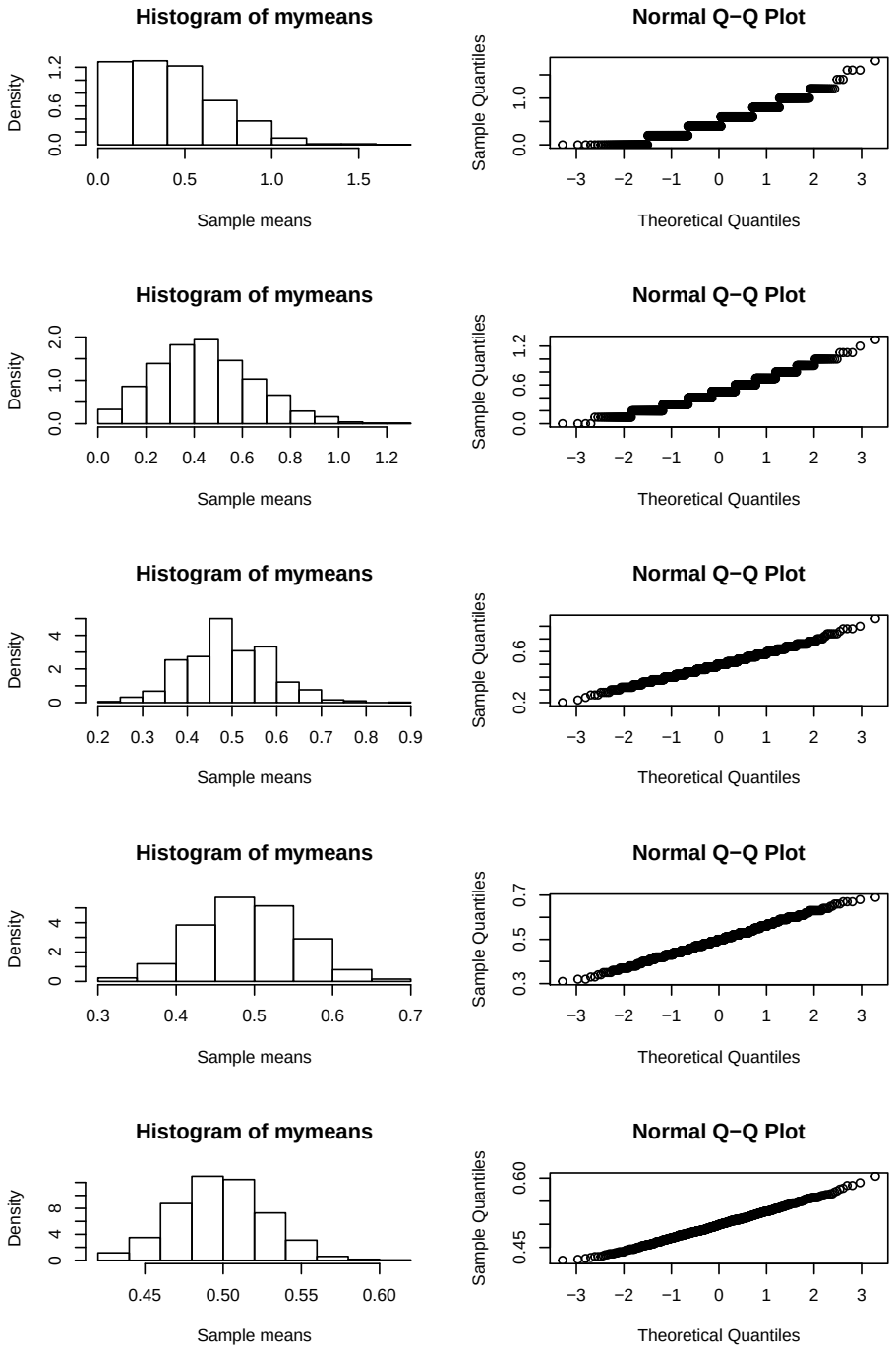
(b) (8 Points) Call your function with nreps = 10000, trials = 5, ps = 0.1, and nobs = 5, 10, 50, 100, and 500. Work with `sapply` to call your function 5 times, once for each value of nobs. Check carefully that nobs is passed on properly to your function (revisit in L15c_loops.R what can go wrong and how to do this correctly)!!! Pass on `xlab = "Sample means"` to provide a more meaningful label for the x–axis in your histogram (but not in your Q–Q Plot).

To guarantee reproducibility, reset your random number generator, e.g., using the last few digits of your A–number, e.g., `set.seed(1234)`.

Finally, draw all of the plots (10 overall) in a single figure. To do so, first set `par(mfrow = c(5, 2))` in your code chunk before you call your centralLimit function. When you do this within RStudio, make sure that your "Plots" window is high (and wide) enough, or otherwise, you will get the following error message `Error in plot.new() : figure margins too large`.

Answer:

```
> par(mfrow = c(5, 2))
> mapply(CentralLimit, nreps = 1000,
+        nobs =  c(5, 10, 50, 100, 500),
+        trials = 5,
+        ps = 0.1,
+        hist = TRUE,
+        qqnorm = TRUE,
+        xlab = "Sample means")
[1] 0.508000 0.493100 0.496520 0.497830 0.498976
```


Histogram of mymeans
Density
Sample means
Normal Q−Q Plot
Sample Quantiles
Theoretical Quantiles
Histogram of mymeans
Density
Sample means
Normal Q−Q Plot
Sample Quantiles
Theoretical Quantiles
Histogram of mymeans
Density
Sample means
Normal Q−Q Plot
Sample Quantiles
Theoretical Quantiles
Histogram of mymeans
Density
Sample means
Normal Q−Q Plot
Sample Quantiles
Theoretical Quantiles
Histogram of mymeans
Density
Sample means
Normal Q−Q Plot
Sample Quantiles
Theoretical Quantiles

(c) (5 Points) Comment on the results, i.e., when does the CLT seem to hold and when not? Answer in 2 or 3 sentences.

Answer:

From my understanding central limit theorem says if we take large number of random samples from a population with replacement then the mean of the sample will be normally distributed. This theorem hold good if the sample size is greater than 30, but if the population is normally distributed it hold for smaller sample size. If we look into our results (histogram and qqlot) as the number of observation for each samples increases, we see more and more normally distibuted which stands with the central limit theorem. With less number of observation its hard for CLM to hold, but impressively with 10 number of observations too it seems to be improved by quite a lot.