STAT 4234/5234 Survey Sampling: Introduction to R, part 2

Using R to draw random samples

Suppose we have a population of values, let's say there are five of them.

```
> y <- c(70, 72, 75, 78, 73)
> y
[1] 70 72 75 78 73
```

The R function `sample` can be used to get an SRS of size n. Here are random samples of size $n = 2$ and $n = 3$.

```
> set.seed(5234)
> sample(y, 2)
[1] 73 75
> sample(y, 3)
[1] 72 75 73
```

I used the `set.seed` function so I would get the same sample every time — good for handout writing but not always desirable. If I wanted an SRSwR I could use

```
> sample(y, 2, replace=T)
[1] 73 70
> sample(y, 3, replace=T)
[1] 72 78 73
> sample(y, 3, replace=T)
[1] 70 73 75
> sample(y, 3, replace=T)
[1] 78 70 78
```

In some instances it may be desirable to sample from the indices $1, 2, \ldots, N$ rather than the y_i directly.

```
> set.seed(5234)
> select <- sample(1:5, 2)
> y[select]
[1] 73 75
> select <- sample(1:5, 3)
> y[select]
[1] 72 75 73
```

All possible samples from an SRS

R is a functional language: you write functions, then execute them with arguments of your choice. Here is a very simple function.

```
> myadd <- function(a, b=3)
+ {
+  out <- a + b
+  out
+ }
```

The function is called `myadd`. It has two arguments `a` and `b`. If you don't set `b`, then it is set to its default value of 3, but you must give a value of `a`. The left curly bracket `{` opens the function and the right curly bracket `}` closes it. The code in the function uses the argument to compute something. Variables like `out` that are computed inside the function are generally not visible outside the function. The last line of the function is the returned value, in this case the sum of the arguments.

```
> myadd(a=2, b=6)
[1] 8
> myadd(2, 6)
[1] 8
> myadd(a=2)
[1] 5
> myadd(2)
[1] 5
> out
Error: object 'out' not found
```

Functions can be useful for organizing your work. For example, the function below finds all possible SRSs of size $n = 2$ and computes summary statistics for each. It prints them, then computes and prints more summary statistics. After the printing is done I don't want the function to return anything, so the last line of the function is `invisible()`.

```
> # R code available on Courseworks, under 'Examples'
>
> allsamp2 <- function(y)
+ {
+  out <- NULL
+  N <- length(y)
+  ybar.U <- mean(y)
+  S2 <- var(y)
+  for(i in 1:(N-1))
+  {
+   for(j in (i+1):N)
+   {
+    ysamp <- y[c(i,j)]
+    ybar <- mean(ysamp)
+    sqerror <- (ybar - ybar.U)^2
+    out <- rbind(out, c(i=i, j=j, ybar=ybar, sqerror=sqerror))
+   }}
+  print(out)
+  cat("Population ybar.U   =", ybar.U, "\n")
+  cat("Population variance =", S2, "\n")
+  cat("Average of the ybar =", mean(out[,3]), "\n")
+  cat("Mean squared error  =", mean(out[,4]), "\n")
+  cat("S2/n * (1-f)        =", (S2/2) * (1 - 2/N), "\n")
+  invisible()
+ }
```

This function uses lots of buit-in R functions, like `length` to get the length of a vector, and `mean` to compute the mean, `var` for variance (dividing by $n-1$ by default). Inside the for-loops we computed each sample mean and squared error versus the true population mean $\bar{y}_U$. We use `print` to print these values, then use `cat` to do more printing.

Calling this function with the vector `y` defined previously:

```
> allsamp2(y)
      i j ybar sqerror
 [1,] 1 2 71.0    6.76
 [2,] 1 3 72.5    1.21
 [3,] 1 4 74.0    0.16
 [4,] 1 5 71.5    4.41
 [5,] 2 3 73.5    0.01
 [6,] 2 4 75.0    1.96
 [7,] 2 5 72.5    1.21
 [8,] 3 4 76.5    8.41
 [9,] 3 5 74.0    0.16
[10,] 4 5 75.5    3.61
Population ybar.U   = 73.6
Population variance = 9.3
Average of the ybar = 73.6
Mean squared error  = 2.79
S2/n * (1-f)        = 2.79
```

Sampling distribution of the sample mean

```
> mean.simulation <- function(data,sizes=c(2,5,10,25,50),n.samp=1000,replace=F)
+ {
+  par(mfrow=c(2,3))
+  hist(data, main="Population", freq=F)
+  for(size in sizes){
+   val <- rep(NA, n.samp)
+   for(j in 1:n.samp)
+   { val[j] <- mean(sample(data, size, replace=replace)) }
+   hist(val, main=paste("Size =", size), freq=F)
+  }
+  invisible()
+ }
```

The data for this example come from the population of the 100 largest US cities. You can find the datafile `citypopulation.csv` on the Coureworks, save it to your Data folder. Then you can read it into R as follows.

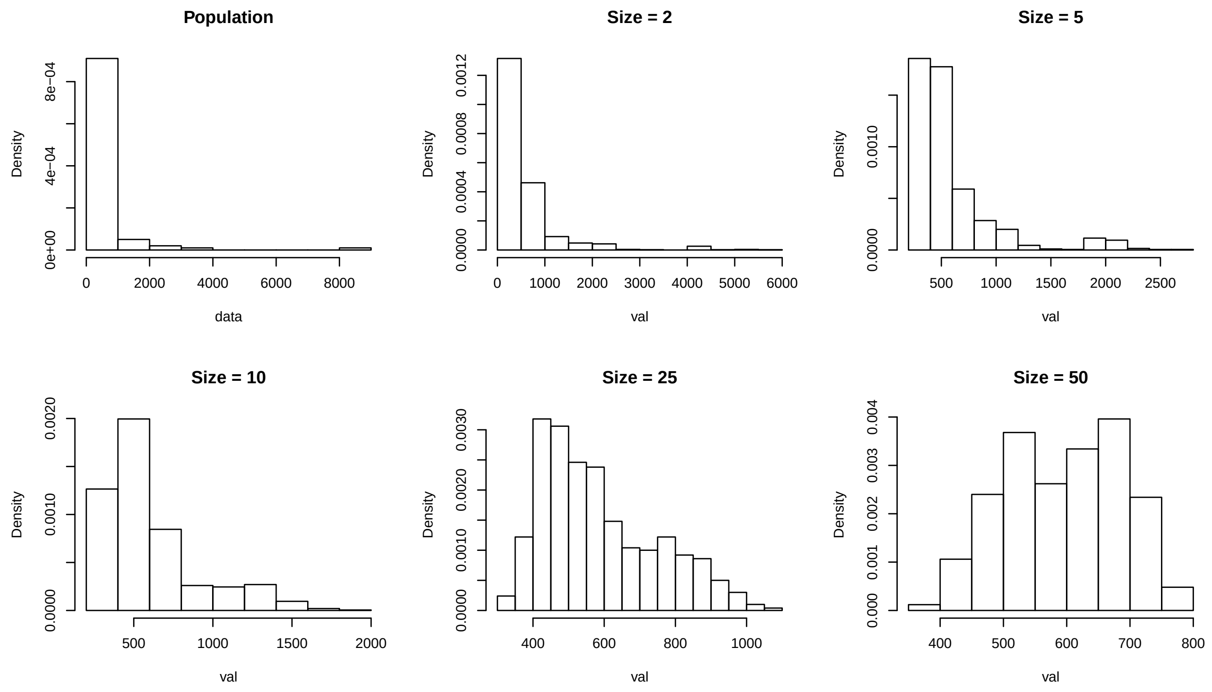
```
> Data <- read.csv("~/Data/citypopulation.csv", header=T)
> Data
                                City Rank Population
1                 New York, New York    1    8213839
2            Los Angeles, California    2    3794640
3                 Chicago, Illinois    3    2824584
4                     Houston, Texas    4    2076189
5        Philadelphia, Pennsylvania    5    1517628
6                   Phoenix, Arizona    6    1476331

etc.
```

Study the sampling distribution of the sample mean for $n = 1, 2, 5, 10, 25, 50$.

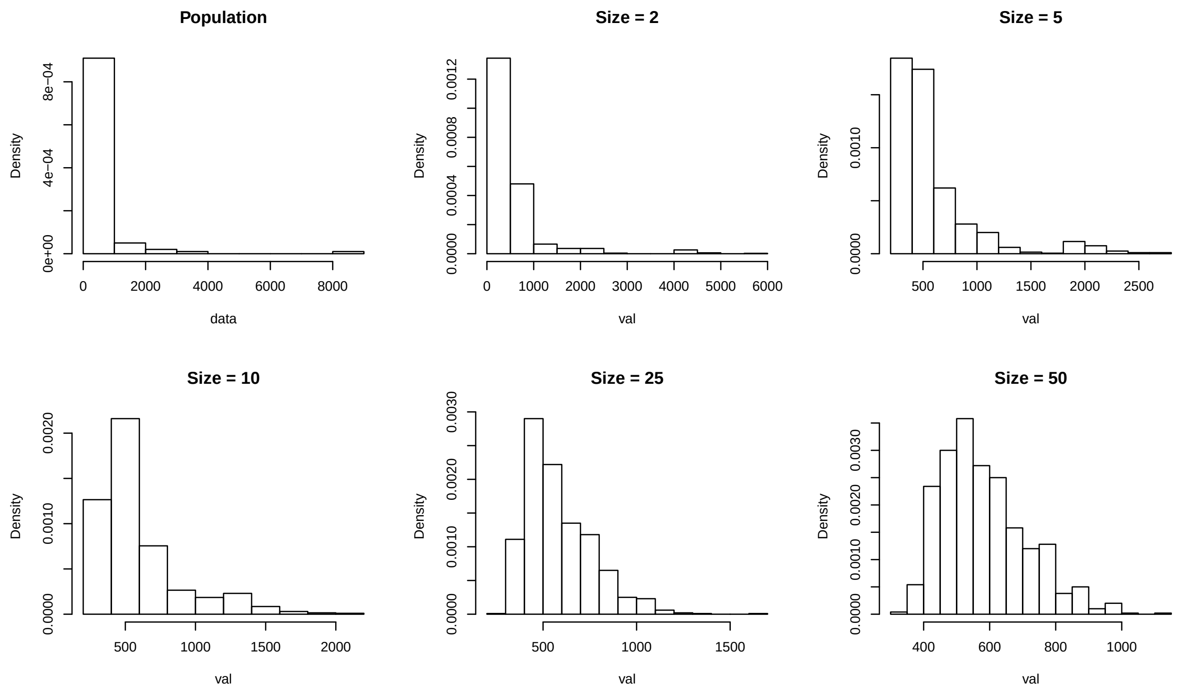
```
> mean.simulation(data=Data$Population/1000)
```

Try sampling with replacement.



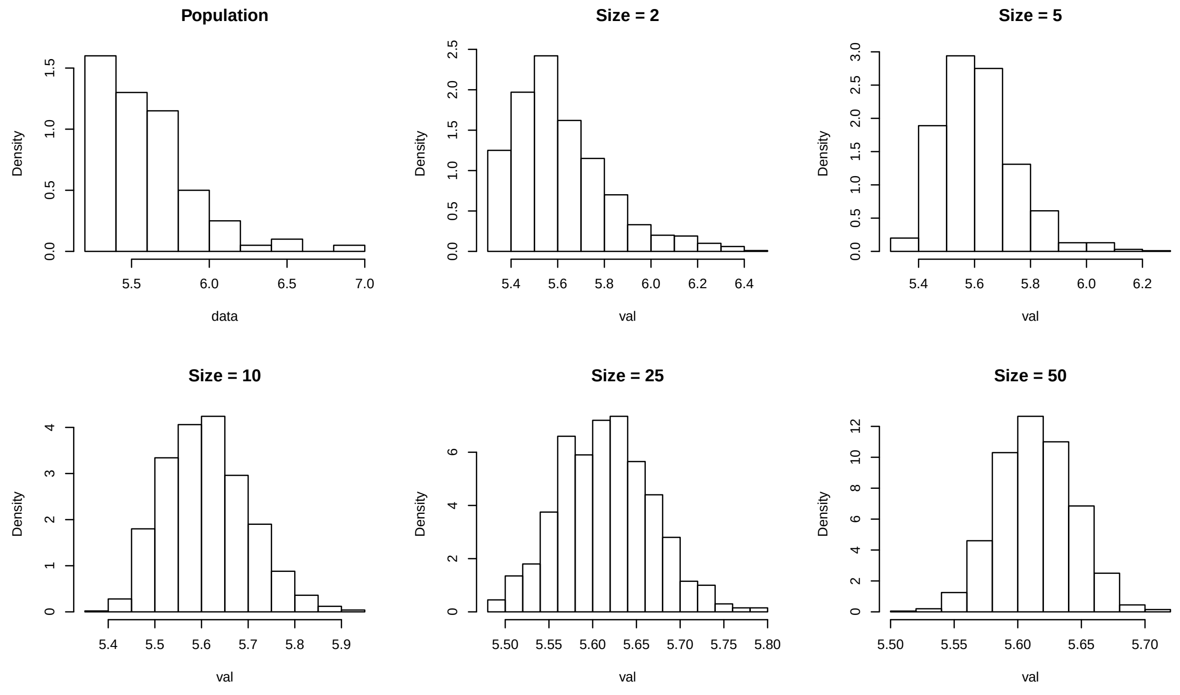
Try sampling with replacement.

```
> mean.simulation(data=Data$Population/1000, replace=T)
```



Consider log transformed populations.

```
> mean.simulation(data=log10(Data$Population))
```


Population
Size = 2
Size = 5
Size = 10
Size = 25
Size = 50
Density
data
val