ST 516: Foundations of Data Analytics

Simulation Case Study

Let's work through a coding example from scratch.

Similar, but different, to your project assignment:

Examine the spread of the sampling distribution for the sample mean for samples of size 10, 100 and 1000 from a Normal distribution. Does the standard deviation of the sampling distribution agree with the theoretical form, $\sigma/\sqrt{(n)}$?

Break the problem down, start simple

Let's start with one sample size, $n = 10$. For this sample size, we need to:

Repeat many times:

- Take a sample of size 10 from a Normal distrbution
- Find the sample mean of the sample

Find the standard deviation of the many sample means and compare to $\sigma/\sqrt{n}$.

In R code:

```
means <- replicate(1000, sd(rnorm(10, sd = 2)))
sd(means)
2/sqrt(10) # true_se
```

Now repeat for other sample sizes...

What's wrong with this?

```
means <- replicate(1000, mean(rnorm(10, sd = 2)))
sd(means)
2/sqrt(10) # true_se

means <- replicate(1000, mean(rnorm(100, sd = 2)))
sd(means)
2/sqrt(100) # true_se

means <- replicate(1000, mean(rnorm(1000, sd = 2)))
sd(means)
2/sqrt(1000) # true_se
```

What's wrong with this?

No comments

Lot's of repetition

Hard to change number of simulations or population standard deviations

`means` keeps getting over written

Remove hard coded numbers, add better names

```r
n_sim <- 1000
pop_sd <- 2

# Sample size = 10 ==================
n <- 10
means_10 <- replicate(n_sim, mean(rnorm(n, sd = pop_sd)))
sd(means_10)
pop_sd/sqrt(n) # true_se
```

Remove hard coded numbers, add better names

```r
n_sim <- 1000
pop_sd <- 2

# Sample size = 10 =================
n <- 10
means_10 <- replicate(n_sim, mean(rnorm(n, sd = pop_sd)))
sd(means_10)
pop_sd/sqrt(n) # true_se
```

```r
# Sample size = 100 ================
n <- 100
means_100 <- replicate(n_sim, mean(rnorm(n, sd = pop_sd)))
sd(means_100)
pop_sd/sqrt(n) # true_se

# Sample size = 1000 ================
n <- 1000
means_1000 <- replicate(n_sim, mean(rnorm(n, sd = pop_sd)))
sd(means_1000)
pop_sd/sqrt(n) # true_se
```

Perhaps even clearer

Rearrange

```
n_sim <- 1000
pop_sd <- 2

means_10 <- replicate(n_sim, mean(rnorm(10, sd = pop_sd)))
means_100 <- replicate(n_sim, mean(rnorm(100, sd = pop_sd)))
means_1000 <- replicate(n_sim, mean(rnorm(1000, sd = pop_sd)))

c(sd(means_10), sd(means_100), sd(means_1000))
pop_sd/c(sqrt(10), sqrt(100), sqrt(1000))
```

For the writeup

Add `set.seed` and get R to round results.

```
set.seed(101910)
n_sim <- 1000
pop_sd <- 2

means_10 <- replicate(n_sim, mean(rnorm(10, sd = pop_sd)))
means_100 <- replicate(n_sim, mean(rnorm(100, sd = pop_sd)))
means_1000 <- replicate(n_sim, mean(rnorm(1000, sd = pop_sd)))

spread_sampdist <- c(sd(means_10), sd(means_100), sd(means_1000))
true_se <- pop_sd/c(sqrt(10), sqrt(100), sqrt(1000))

rbind(round(spread_sampdist, 3),
  round(true_se, 3))
```

```
##       [,1]  [,2]  [,3]
## [1,] 0.625 0.202 0.063
## [2,] 0.632 0.200 0.063
```

DRY principle

Don't **R**epeat **Y**ourself

Captures the idea that each unique peice of information should be only represented once in your code.

Saves time in the long run:

- less silly copy & paste mistakes
- changes only need to made in one place
- more understandable/expandable structure

Functionalize

Still lots of repetition, e.g:

```
means_10 <- replicate(n_sim, mean(rnorm(10, sd = pop_sd)))
means_100 <- replicate(n_sim, mean(rnorm(100, sd = pop_sd)))
means_1000 <- replicate(n_sim, mean(rnorm(1000, sd = pop_sd)))

spread_sampdist <- c(sd(means_10), sd(means_100), sd(means_1000))
```

You might imagine wrapping up the simulation of sample means into a function

```
get_means <- function(n, n_sim = 1000, pop_sd = 2) {
  replicate(n_sim, mean(rnorm(n, sd = pop_sd)))
}
```

So, you could do

```
means_10 <- get_means(n = 10)
means_100 <- get_means(n = 100)
means_1000 <- get_means(n = 1000)
```

Lists and `lapply`

This is such a common structure, that there are a whole family of functions to help out:

```
means <- lapply(c(10, 100, 1000), get_means)
str(means)
```

`lapply` takes each element of the first argument, in this case each sample size, and passes it to the function in the second argument, in this case `get_means`.

Each result is stored in a separate element of the returned list.

```
means[[1]] # the 1000 simluated sample means for n = 10
sd(means[[1]]) # the sd of the 1000 simluated sample means for n = 10

lapply(means, sd) # the sd of the mean for each sample size
```

Putting it together

```
set.seed(101910)
n_sim <- 1000
pop_sd <- 2

get_means <- function(n, n_sim, pop_sd) {
  replicate(n_sim, mean(rnorm(n, sd = pop_sd)))
}

ns <- c(10, 100, 1000)
means <- lapply(ns, get_means, n_sim = n_sim, pop_sd = pop_sd)

spread_sampdist <- sapply(means, sd)
true_se <- pop_sd/ns

rbind(round(spread_sampdist, 3),
  round(true_se, 3))
```

If we want to change the number of simulations, set of sample sizes, population, or the statistic, we only have to make the change in one place.