3 Functions and Macros Homework

06/15/2020

General instructions.

There are five exercises below. You are required to provide five solutions, with the same options for choosing languages as with the last exericse. The first three exercises refer back to Homework 2 and you should produce the same values for this set of exercises. We will likely use the first four functions in later exercises.

Exercise 1

Implement Cohen's d as a function of

$$d = f(m_1, s_1, m_2, s_2) = \frac{|m_1 - m_2|}{s_{pooled}}$$

where s_{pooled} is a pooled standard deviation. Use the formula $s_{pooled} = \sqrt{(s_1^2 + s_2^2)/2}$. You may implement pooled standard deviation as a function as well.

Call this function to calculate the effect size d for the differences among calories per serving, 1936 versus 2006, 1936 vs 1997 and 1997 vs 2006, as in the previous homework. Name this function `cohen.d` (or similar if using SAS).

Answer

Define your function(s) in the code chunk below, the call the function with appropriate parameters in the following sections

```
cohen.d <- function(m_1,m_2,s_1,s_2) {
  s.pooled <- sqrt((s_1^2 + s_2^2)/2)
  return(abs(m_1-m_2)/s.pooled)
}
```

1936 versus 2006

```
m_1 <- 268.1
s_1 <- 124.8
m_2 <- 384.4
s_2 <- 168.3

d <- cohen.d(m_1,m_2,s_1,s_2)
d

## [1] 0.784987603959
```

1936 versus 1997

```
m_1 <- 268.1
s_1 <- 124.8
m_2 <- 288.6
s_2 <- 122.0

d <- cohen.d(m_1,m_2,s_1,s_2)
d

## [1] 0.166115727787
```

1997 versus 2006

```
m_1 <- 288.6
s_1 <- 122.0
m_2 <- 384.4
s_2 <- 168.3

d <- cohen.d(m_1,m_2,s_1,s_2)
d

## [1] 0.651769377713
```

Exercise 2.

Define a function to calculate required replicates. Define m_1, s_1, m_2 and s_2 as required parameters, and α and β as optional parameters. Let `alpha=0.05` and `beta=0.2`.

Your function should return an integer n, such that

$$n \geq 2 \times \left(\frac{CV}{\%Diff}\right)^2 \times \left(z_{\alpha/2} + z_{\beta}\right)^2$$

where $\%Diff = \frac{m_1 - m_2}{(m_1+m_2)/2}$ and $CV = \frac{sd_{pooled}}{(m_1+m_2)/2}$.

You may use the pooled standarad deviation function from Ex. 1 (if you defined such a function).

Name this function `required.replicates` (or similar if using SAS)

Answer

Define your function(s) in the code chunk below, the call the function with appropriate parameters in the following sections

```
required.replicates <- function(m_1,s_1,m_2,s_2,alpha = 0.05, beta = 0.20) {
  s.pooled <- sqrt((s_1^2 + s_2^2)/2)
  z_alpha <- qnorm(1-alpha/2)
  z_beta <- qnorm(1-beta)
  cv <- s.pooled/((m_1 + m_2)/2)
```

```
  dif <- (m_1 - m_2)/((m_1 + m_2)/2)
  return(round(2*((cv/dif)^2)*((z_alpha + z_beta)^2),0))
}
```

1936 versus 2006

```
m_1 <- 268.1
s_1 <- 124.8
m_2 <- 384.4
s_2 <- 168.3

rrep <- required.replicates(m_1,s_1,m_2,s_2)
rrep
```

```
## [1] 25
```

1936 versus 1997

```
m_1 <- 268.1
s_1 <- 124.8
m_2 <- 288.6
s_2 <- 122.0

rrep <- required.replicates(m_1,s_1,m_2,s_2)
rrep
```

```
## [1] 569
```

1997 versus 2006

```
m_1 <- 288.6
s_1 <- 122.0
m_2 <- 384.4
s_2 <- 168.3

rrep <- required.replicates(m_1,s_1,m_2,s_2)
rrep
```

```
## [1] 37
```

Exercise 3

Implement the likelihood formula as a function or macro.

$$L(x;\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Define μ and σ as optional parameters, taking values `mu=0` and `sigma=1`. Name this function `norm.pdf`

Answer

Define your function(s) in the code chunk below, the call the function with appropriate parameters in the following sections

```
norm.pdf <- function(x,mu=0,sigma=1) {
  return((1/(sigma*sqrt(2*pi)))*(exp(1)^((-1*(x-mu)^2)/2*sigma^2)))
}
```

$x = -0.1$

```
x <- -0.1
L <- norm.pdf(x)
L

## [1] 0.396952547477
```

$x = 0.0$

```
x <- 0.0
L <- norm.pdf(x)
L

## [1] 0.398942280401
```

$x = 0.1$

```
x <- 0.1
L <- norm.pdf(x)
L

## [1] 0.396952547477
```

Exercise 4

The probability mass function for value y from Poisson data with a mean and variance λ is given by

$$f(x;\lambda) = \frac{e^{-\lambda}\lambda^{x}}{x!} = exp(-\lambda)(\frac{1}{x!})exp[x \times log(\lambda)]$$

Write a function `pois.pmf` that accepts two parameters, `x` and `lambda`. Use the built in `factorial` function for $x!$. Note that x should be an integer value, so call a rounding function inside your function.

Test your function with $\lambda = 12$ at $x = 4,12,20$

Answer

Define your function(s) in the code chunk below, the call the function with appropriate parameters in the following sections

```
pois.pmf <- function(x,lambda) {
  x <- round(x,0)
  return (exp(-lam)*(1/factorial(x))*exp(x*log(lam)))
}
```

$x = 4$

```
x <- 4
lam <- 12
pois1 <- pois.pmf(x,lam)
pois1

## [1] 0.00530859947328
```

$x = 12$

```
x <- 12
lam <- 12
pois2 <- pois.pmf(x,lam)
pois2

## [1] 0.114367915509
```

$x = 20$

```
x <- 20
lam <- 12
pois3 <- pois.pmf(x,lam)
pois3

## [1] 0.00968203216822
```

You can check your work against the built in Poisson distribution functions.

```
check1 <- dpois(4,12)
check2 <- dpois(12,12)
check3 <- dpois(20,12)

cat('Poisson at x = 4 is',pois1,', check is',check1,'\n')

## Poisson at x = 4 is 0.00530859947328 , check is 0.00530859947328

cat('Poisson at x = 12 is',pois2,', check is',check2,'\n')

## Poisson at x = 12 is 0.114367915509 , check is 0.114367915509

cat('Poisson at x = 20 is',pois3,', check is',check3)

## Poisson at x = 20 is 0.00968203216822 , check is 0.00968203216822
```

Comment - As shown above, my function *pois.pmf* matches the built in function *dpois* for all 3 x values.

Something to ponder. Note that there are two formula given. Can you implement both forms in R/IML/Macro language? Would there be a difference in computational speed or efficiency?

```
alt.pois.pmf <- function(x,lambda) {
  return (exp(1)^(-lam))*(lam^x)/factorial(x)
}
library(microbenchmark)

## Warning: package 'microbenchmark' was built under R version 3.6.3

microbenchmark(pois.pmf(x,lam))

## Unit: microseconds
##              expr min    lq    mean median  uq    max neval
##  pois.pmf(x, lam) 1.1 1.201 1.87298  1.202 1.4 32.201   100

microbenchmark(alt.pois.pmf(x,lam))

## Unit: nanoseconds
##                  expr min  lq     mean median  uq     max neval
##  alt.pois.pmf(x, lam) 300 301 38876.96    401 401 3825401   100
```

Comment - Yes, you can implement both forms into R as seen above (*pois.pmf* vs. *alt.pois.pmf*). I ran a microbenchmark on both functions to see if there was a difference in computational speed. Both functions took nanoseconds to run, so it's a bit of a moot point.

However, the *pois.pmf* functions does on average take less time to run than the *alt.pois.pmf* function. Interestly, the *alt.pois.pmf* function has a much larger variance in run time which is pushing the mean time up and causing a large right skew distribution on the run times. This is comfirmed with a low median value compared to the mean.

Exercise 5

Fisher's LSD test is generally used to compare among two treatment means, and two means only. If we use this test to make comparisons among many treatments we risk making a spurious declaration of a signficiant difference. To control for this type of error, we sometimes use Bonferri's method.

Briefly, if we want 95% confidence over several treatment comparisons, we adjust α to account for the number of comparisons. Thus, if we want to compare among four dfferent means (m_1, m_2, m_3, m_4), there are

$$\frac{4 \times 3}{2} = 6$$

possible comparisons (m_1 vs m_2, m_1 vs m_3, etc.), so we use

$$\alpha = \frac{0.05}{6} = 0.01$$

to calculate *LSD*.

Write a function, `corrected.lsd` that has the same parameter list as the `fisher.lsd` given the course outline. Add an optional parameter `g=2`, and let g be the number of means. In this function, calculate the number of possible comparisons among g means as described above, then calculate a corrected *alpha*.

Use the corrected *alpha* to calculate a corrected *LSD*. You can implement the *LSD* formula in your function, or you may copy `fisher.lsd` from the course outline and call `fisher.lsd` with the corrected α. If you choose to copy `fisher.lsd`, be sure to cite your source for the code.

The function `corrected.lsd` should return a list of 5 values:

- Uncorrected α
- Uncorrected *LSD*
- Number of possible comparisons
- Corrected α
- Corrected *LSD*

When the function is called without an argument for g the corrected *LSD* should be the same as the uncorrected *LSD*, so test your function by calling with the same arguments as used in the course outline (`1050.0, 18, 1496.2, 18`), once with the optional argument `g=7` and once without this optional argument.

```
corrected.lsd <- function(s_i,n_i,s_j,n_j,alpha=0.05,g=2) {
  # Code taken from Functions and Macros presentation STAT 600 Summer 2020
Dakota State University
  s2 <- ((n_i-1)*s_i^2 + (n_j-1)*s_j^2) / ((n_i-1)+(n_j-1))
  critical.t <- qt(1 - alpha/2,n_i+n_j-2)
  c.alpha <- alpha/(g*(g-1)/2)
  c.critical.t <- qt(1 - c.alpha/2,n_i+n_j-2)
  return(list('Uncorrected Alpha' = alpha,
              'Uncorrected LSD' = critical.t*sqrt(s2*(1/n_i + 1/n_j)),
              'Number of Possible Comparisons' = (g*(g-1)/2),
              'Corrected Alpha' = c.alpha,
              'Corrected LSD' = c.critical.t*sqrt(s2*(1/n_i + 1/n_j))
              ))
}
s_i <- 1050.0
n_i <- 18
s_j <- 1496.2
n_j <- 18
lsd_7 <- corrected.lsd(s_i,n_i,s_j,n_j,g=7)
cat('Where g=7', '\n', '\n')
```

```
## Where g=7
##

lsd_7

## $`Uncorrected Alpha`
## [1] 0.05
##
## $`Uncorrected LSD`
## [1] 875.558930292
##
## $`Number of Possible Comparisons`
## [1] 21
##
## $`Corrected Alpha`
## [1] 0.00238095238095
##
## $`Corrected LSD`
## [1] 1414.53204173

lsd_2 <- corrected.lsd(s_i,n_i,s_j,n_j)
cat('Where g=2 or default', '\n', '\n')

## Where g=2 or default
##

lsd_2

## $`Uncorrected Alpha`
## [1] 0.05
##
## $`Uncorrected LSD`
## [1] 875.558930292
##
## $`Number of Possible Comparisons`
## [1] 1
##
## $`Corrected Alpha`
## [1] 0.05
##
## $`Corrected LSD`
## [1] 875.558930292
```