Castellano_CS636_Lab01

January 27, 2020

This is an R Markdown Notebook. When you execute code within the notebook, the results appear beneath the code.

1.1 Use R as you would a calculator to find numeric answers to the following:

```
1+2*(3+4)
```

```
## [1] 15
```

```
4**3+3**(2+1)
```

```
## [1] 91
```

```
sqrt((4+3)*(2+1))
```

```
## [1] 4.582576
```

```
((1+2)/(3+4))**2
```

```
## [1] 0.1836735
```

1.2 Rewrite these R expressions as math expressions, using parentheses to show the order in which R performs the computations.

```
(2 + 3) -4
```

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

```
2+(3*4)
```

```
## [1] 14
```

```
(2/3)/4
```

```
## [1] 0.1666667
```

```
(2*3*4)
```

```
## [1] 24
```

```
# Apparently, R uses proper Order of Operations
```

1.3 Use R to compute the following

```
(1+2*3**4)/(5/6-7)
```

```
## [1] -26.43243
```

Getting Started With R

1.4 Use R to Compute the Following

```
(0.25-0.2)/sqrt(0.2*(1-0.3)/100)
```

```
## [1] 1.336306
```

1.5 Assign the numbers 2 through 5 to different variables, then use the variables to multiply all the values.

```
a <- 1
b <- 2
c <- 3
d <- 4
e <- 5

prod(a,b,c,d,e)
```

```
## [1] 120
```

```
factorial(5)
```

```
## [1] 120
```

1.6 The rivers data set is loaded when R is. View the data by typing its name and then the return key. What is the value listed?

```
rivers
```

```
##   [1]  735  320  325  392  524  450 1459  135  465  600  330  336  280  315  870
##  [16]  906  202  329  290 1000  600  505 1450  840 1243  890  350  407  286  280
##  [31]  525  720  390  250  327  230  265  850  210  630  260  230  360  730  600
##  [46]  306  390  420  291  710  340  217  281  352  259  250  470  680  570  350
##  [61]  300  560  900  625  332 2348 1171 3710 2315 2533  780  280  410  460  260
##  [76]  255  431  350  760  618  338  981 1306  500  696  605  250  411 1054  735
##  [91]  233  435  490  310  460  383  375 1270  545  445 1885  380  300  380  377
## [106]  425  276  210  800  420  350  360  538 1100 1205  314  237  610  360  540
## [121] 1038  424  310  300  444  301  268  620  215  652  900  525  246  360  529
## [136]  500  720  270  430  671 1770
```

The last value listed is 1770.

1.7 The exec.pay (UsingR) data set is available from the command line after loading the package UsingR. Load the package, and inspect the data set. Scan the values to find the largest one.

```
library(UsingR)
```

```
## Loading required package: MASS
```

```
## Loading required package: HistData
```

```
## Loading required package: Hmisc
```

```
## Loading required package: lattice
```

```
## Loading required package: survival

## Loading required package: Formula

## Loading required package: ggplot2

##
## Attaching package: 'Hmisc'

## The following objects are masked from 'package:base':
##
##     format.pval, units

##
## Attaching package: 'UsingR'

## The following object is masked from 'package:survival':
##
##     cancer
```

```
exec.pay
```

```
##   [1]  136   74    8   38   46   43    9    9   12   11   20    9   95   34    7
##  [16]   14   39   12   29   21   60   35   17   36   29  162   88   31    6  135
##  [31]   13   20    9   14   28   42   10   35    2   16   28   42  142   33  134
##  [46]   23   34   16   13  167    9   22   39   28   30   22   14    9   25  106
##  [61]   32   30   89   89   47   17   26 1231    6  103   48   24   11   19   13
##  [76]   29   20   45    3   33   41    7   11   10   22   36    7   19   41   40
##  [91]   10   15   93   67   29   25   91   38 2510    5   32   65    0   13   27
## [106]   16   21    6    0   28    8   13   71   36   11  106   37   41   13  900
## [121]   38   24   15   27   12   12   22   40   49   22  118   48   10    1   36
## [136]  155    9   34   29   12    0   28   21   32   18   52   29   13  199   40
## [151]   11   51   45   43   31    5   18   15   25    9   18   13   58   22   40
## [166]   34   16   31   27   15   23   49   60   28   74   42   24   17    9   61
## [181]   20   23   26   31  167   19   14   13  146  283   12   53   26   16   29
## [196]   51   15   22   27
```

```
max(exec.pay)
```

```
## [1] 2510
```

1.8 For the exec.pay (UsingR) data set, apply the functions, mean, min and max. What are the values found?

```
A <- c(exec.pay)
mean(A)
```

```
## [1] 59.88945
```

```
min(A)
```

```
## [1] 0
```

```
#print('The maximum is {}').format(max(A))
max(A)
```

```
## [1] 2510
```

1.9 The basic mean function has an additional arguent, trim. When given, the specified proportion of the data is trimmed from the sorted data before the mean is taken. Compare the difference between mean(exec.pay) and mean(exec.pay, trim=0.10)

```
mean(exec.pay)
```

```
## [1] 59.88945
```

```
mean(exec.pay, trim=0.10)
```

```
## [1] 29.96894
```

1.10 The orange data set is stored as a data frame with three variables. What are the three variables?

```
data("Orange")
ls()
```

```
## [1] "a"      "A"      "b"      "c"      "d"      "e"      "Orange"
```

```
head(Orange)
```

```
##   Tree  age circumference
## 1    1  118            30
## 2    1  484            58
## 3    1  664            87
## 4    1 1004           115
## 5    1 1231           120
## 6    1 1372           142
```

The three variables are tree, age, and circumference.

1.11 Compute the average age of the trees in the Orange data set using mean.

```
mean(Orange[,'age'])
```

```
## [1] 922.1429
```

1.12 Compute the largest circumference of the trees in the Orange dataset.

```
max(Orange[,'circumference'])
```

```
## [1] 214
```

1.18 Define x and y with

```
x = c(1,3,5,7,9)
y = c(2,3,5,7,11,13)
```

Try to guess the result of these R commands

```
x+1 # Element wise addition of 1
```

```
## [1]  2  4  6  8 10
```

```r
y*2 # Element wise multiplication by 2
```

```
## [1]  4  6 10 14 22 26
```

3. lenght(x) and length(y)

```r
length(x) # 5
```

```
## [1] 5
```

```r
length(y) # 6
```

```
## [1] 6
```

4. x+y (recycling)

```r
x+y # Do not know
```

```
## Warning in x + y: longer object length is not a multiple of shorter object
## length
```

```
## [1]  3  6 10 14 20 14
```

5. sum(x>5) and sum(x[x>5])

```r
sum(x>5) # 16
```

```
## [1] 2
```

```r
sum(x[x>5]) # Error
```

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

Interesting. sum(x>5) is a sum of the NUMBER of elements that are greater than 5. sum(x[x>5]) is a sum of the elements themselves which are greater than 5. sum(x>3) should = 3.sum(x[x>3]) should = 21

```r
sum(x>3)
```

```
## [1] 3
```

```r
sum(x[x>3])
```

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

6. sum(x>5|x<3)

```r
sum(x>5|x<3) # sum of number of elements greater than 5 or less than 3.
```

```
## [1] 3
```

```r
# ans = 3
```

7. y[3]

```r
y[3] # ans = 5
```

```
## [1] 5
```

8. y[-3]

```
y[-3] # ans = 11
```

```
## [1]  2  3  7 11 13
```

Interesting. Unlike Python, y[-3] removes the third element from the vector. In Python, y[-3] would select the 3 element from the end of the vector.

9. y[x] (What is NA)

```
y[x]
```

```
## [1]  2  5 11 NA NA
```

NA possibly means 'Not Applicable.' In this case, the computations y[7] and y[9] do 'Not Apply' becasue length(y) < 7 or 9. There are no 7th or 9th elements in y.

10. y[y>=7]

```
y[y>=7] # Selects elements greater or equal to 7 in vector y.
```

```
## [1]  7 11 13
```