STAT 1293 Assignment 1

Gordon Lu

6/27/2020

Problem 1: R as a Calculator (10 points)

Calculate the following using R:

1a) $log_{10}(0.28)$

Solution:

We can use either the `log()` function or the `log10()` function. I will use the `log()` function in this case. The first argument being the number we wish to evaluate the `log` of, and the second argument being the base.

```
log(0.28, 10) #Evaluate log_10(0.28)
```

```
## [1] -0.552842
```

```
log10(0.28) #Yields the same answer
```

```
## [1] -0.552842
```

1b) $\frac{1}{\sqrt{2\pi}}e^{-2}$

Solution:

We need to use the `exp()` function, and the `sqrt()` function here. We can divide the expression into tw subexpressions. Sub-expression 1 can be $\frac{1}{\sqrt{2\pi}}$, while sub-expression 2 can be e^{-2}.

```
sub_expr_1 <- 1/(sqrt(2*pi)) #1/sqrt(2*pi)
sub_expr_2 <- exp(-2) #e^-2
full_expr = sub_expr_1 * sub_expr_2 #compute 1/sqrt(2*pi) * e^-2
full_expr #print out to console
```

```
## [1] 0.05399097
```

1c) C_9^{20}

Solution:

We need to use the `choose()` function. In this case, we can imagine we are attempting to form a committee of 9 from a group of 20 people. In such a case, we can think of it like `n choose k` where `n` is the total number, and `k` is the subset we are trying to pull.

```
choose(20, 9) #compute 20 choose 9
```

```
## [1] 167960
```

1d) Round the result of 1b) to 3 decimal places.

Solution:

We need to use the `round()` function. Since, we have a variable storing the result of 1b), `full_expr`, we can call the `round()` function with `full_expr` as the first argument, and the number of places to round to as the second argument. In this case, `3` would be the second argument.

```
round(full_expr, 3) #Round the result of 1b) to 3 decimal places.
```

```
## [1] 0.054
```

1e) $e^{-3} + e^3$

Solution:

We need to use the `exp()` function.

```
exp(-3) + exp(3) #exp(3) will yield e^3, and exp(-3) will yield e^-3, so just sum them together.
```

```
## [1] 20.13532
```

Problem 2: Vectors (10 points)

Part I: Create the following vectors (3 points)

1) `v1` is a vector of integers from `-5` to `5`.

Solution:

We can either use the `seq()` function, incrementing by `1` starting from `-5` and ending at `5`, or we can use the `:` operator like so: `-5:5`.

```
v1 <- seq(-5, 5, 1) #Creates a sequence of integers from -5 to 5 by increments of 1.
v1
```

```
##  [1] -5 -4 -3 -2 -1  0  1  2  3  4  5
```

```
#The following should be equivalent:
-5:5 #Should generate a sequence of consecutive integers from -5 to 5.
```

```
##  [1] -5 -4 -3 -2 -1  0  1  2  3  4  5
```

2) `v2` is a character vector which has two elements: `M` and `F`.

Solution:

We can use the `c()` function to create a vector of two values. Them being, `M` and `F`.

```
v2 <- c("M", "F") #Create a vector of two characters, "M" and "F".
v2
```

```
## [1] "M" "F"
```

3) `v3` is a logical vector which is the result of comparing `v1` with `0`, i.e. checking if elements of v_1 is greater than `0`.

Solution:

We can use the relational operator `>` and evaluate each element of `v1` using the expression, `v1 > 0` and producing a vector of logical values, and storing it into `v3`.

```
v3 <- v1 > 0
v3
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

Part II: Convert vectors (1 point)

1) Convert `v1` to a factor-type vector and assign character levels ("A", "B", "C", etc) to it. Make sure you create a new vector. You may name it `v1.f`

Solution:

We need to use the `factor()` function to convert `v1` into a facotr-type vector first. We must also define the levels as the possible values that `v1` can take on. In this case, it is the consecutive integers from -5 to 5, as defined by the `v1` vector. Once the factor-type vector is successfully formed, we need to make the levels more interpretable by redefining the levels with character vectors. In this case, I have decided to use the first 11 capital letters from the built-in `LETTERS` vector as the levels.

```
v1.f <- factor(v1, levels = -5:5) #Create a factor-type vector

levels(v1.f) <- LETTERS[1:11] #Make levels interpretable
v1.f
```

```
##  [1] A B C D E F G H I J K
## Levels: A B C D E F G H I J K
```

Part III: Calculation and expression involving vectors (6 points)

What are the values returned by the following commands?

```
c(v1, v2) #concatenate v1 and v2, will convert down to the least strict type: character!
```

```
##  [1] "-5" "-4" "-3" "-2" "-1" "0"  "1"  "2"  "3"  "4"  "5"  "M"  "F"
```

```
c(v1, v3) #concatenate v1 and v3, will convert down to the least strict type: numeric!
```

```
##  [1] -5 -4 -3 -2 -1  0  1  2  3  4  5  0  0  0  0  0  0  1  1  1  1  1
```

```
c(v2, v3) #concatenate v2 and v3, will convert down to the least strict type: character!
```

```
##  [1] "M"     "F"     "FALSE" "FALSE" "FALSE" "FALSE" "FALSE" "FALSE" "TRUE"
## [10] "TRUE"  "TRUE"  "TRUE"  "TRUE"
```

```
v1 + v3 #add v1 and v3, convert v3 into numeric, so TRUE = 1, so we add 1 for all numbers above 0.
```

```
##  [1] -5 -4 -3 -2 -1  0  2  3  4  5  6
```

```
v1 * v3 #multiply v1 and v3, all numbers <= 0 in v1 will be 0, and everything above 0 in v1 is same.
```

```
##  [1] 0 0 0 0 0 0 1 2 3 4 5
```

```r
v1 + v2 #Error, since v2 is a character vector, and we cannot add characters using "+"
```

```
## Error in v1 + v2: non-numeric argument to binary operator
```

Problem 3: Matrices (10 points)

1) Create the following matrix (2 points)

Solution:

We can form a matrix by taking advantage of the built-in `letters` matrix in R. By taking the first `16` characters, we can form a new vector, containing said characters. Then, we can assign the dimensionality as `4x4` to form the matrix.

```r
matA <- letters[1:16]
dim(matA) <- c(4,4)
matA
```

```
##      [,1] [,2] [,3] [,4]
## [1,] "a"  "e"  "i"  "m"
## [2,] "b"  "f"  "j"  "n"
## [3,] "c"  "g"  "k"  "o"
## [4,] "d"  "h"  "l"  "p"
```

2) Create the following matrix (2 points)

Solution:

We can form this matrix by forming individual row vectors, then binding said vectors using the `cbind()` function. Noting that if we used the `rbind()` function, we would need to take the transpose, `t()` to glue the vectors successfully.

```r
A <- seq(5, 80, 25)
B <- seq(10, 85, 25)
C <- seq(15, 90, 25)
D <- seq(20, 95, 25)
E <- seq(25, 100, 25)

matB <- cbind(A, B, C, D, E)
matB
```

```
##       A  B  C  D   E
## [1,]  5 10 15 20  25
## [2,] 30 35 40 45  50
## [3,] 55 60 65 70  75
## [4,] 80 85 90 95 100
```

3) Add a row to M2 in a logical way, i.e., add `105, 110, 115, 120, 125` to it. Name the new matrix as `M3`. (3 points)

Solution:

We can add a new row into `M2` by first forming a vector from 105 to 125 in increments of 5. However, if we were to call `cbind()` with this new vector, it would take the transpose of this vector, and create a new column vector from 105 to 125, but we want a new row. So, instead, by taking the transpose of the initial row vector, we form a column from 105 to 125 in increments of 5, we can do so by using the `t()` function. Then, we can call `rbind()` to form a new matrix, `M3` with `M2` and the new row added.

```
new_row <- t(seq(105, 125, 5))
M3 <- rbind(matB, new_row)
M3
```

```
##        A   B   C   D   E
## [1,]   5  10  15  20  25
## [2,]  30  35  40  45  50
## [3,]  55  60  65  70  75
## [4,]  80  85  90  95 100
## [5,] 105 110 115 120 125
```

4) Subsetting (3 points)

a) Pull out the value at the 2nd row and 3rd column of `M3` using R (1 point)

Solution:

We can retrieve the value by specifying the row, then the column, separated by a column. In this case, `M3[2,3]` will suffice.

```
M3[2,3] #Retrieve value at 2nd row, 3rd column
```

```
##  C
## 40
```

b) Pull out the value at the 3rd row and 2nd column of M3 using R.

Solution:

We can retrieve the value by specifying the row, then the column, separated by a column. In this case, `M3[3,2]` will suffice.

```
M3[3,2] #Retrieve value at 3rd row, 2nd column
```

```
##  B
## 60
```

c) Pull out the 4th column of `M3` using R (1 point)

Solution:

We can retrieve a column by leaving the row portion blank, and specifying the column. In this case, `M3[, 4]` will suffice.

```
M3[,4] #Retrieve the 4th column
```

```
## [1]  20  45  70  95 120
```

Problem 4: Data Frame (10 points)

1) Create three vectors: `name`, `age`, and `hair` with the following elements, respectively.

(2 points)

Solution:

First, we can create the `name` vector by using the `c()` function, the same can be done with the `age` and `hair` functions.

```
name <- c("Ariel", "Bella", "Cindy", "Dora", "Ella") #create name vector
age <- c(19, 20, 18, 17, 20) #create age vector
hair <- c("red", "brown", "blond", "black", "blond") #create hair vector

#print out name, age and hair vectors to be sure:
name;age;hair
```

```
## [1] "Ariel" "Bella" "Cindy" "Dora"  "Ella"
```

```
## [1] 19 20 18 17 20
```

```
## [1] "red"   "brown" "blond" "black" "blond"
```

2) Create a data frame (just call it df) by binding the threee vectors (2 points)

Solution:

We can use the `data.frame()` function to create a data frame, and pass in the three vectors as arguments.

```
df <- data.frame(name, age, hair) #create data frame
df #display df contents
```

```
##    name age  hair
## 1 Ariel  19   red
## 2 Bella  20 brown
## 3 Cindy  18 blond
## 4  Dora  17 black
## 5  Ella  20 blond
```

3) Sort the data frame you created in Part 2) by `age` (3 points)

Solution:

First, we need to create an ordering based on `age`. Then, we need to apply this ordering to all elements of the data frame.

```
order_age <- order(df$age) #create ordering by age

df[order_age,] #now apply it to the data frame
```

```
##    name age  hair
## 4  Dora  17 black
## 3 Cindy  18 blond
## 1 Ariel  19   red
## 2 Bella  20 brown
## 5  Ella  20 blond
```

3) Sort the data frame by `hair color` and `age`

Solution:

First, we need to create an ordering based on `hair_color` and then `age`. Then, we need to apply this ordering to all elements of the data frame.

```
order_color_age <- order(df$hair, df$age) #create ordering by hair color, then age

df[order_color_age,] #now apply it to the data frame
```

```
##    name age  hair
## 4  Dora  17 black
## 3 Cindy  18 blond
## 5  Ella  20 blond
## 2 Bella  20 brown
## 1 Ariel  19   red
```

Problem 5: Set Operations (12 points)

Let `A` be the set of positive integers from `20` to `40`.

Let `B` be the set of even integers from `10` to `50`.

Let `C` be a single value `0`.

Find the following:

1) $A \cup B$

Solution:

We can use the `union()` function in R. First, we must create the sets `A`, `B`, and `C`. This should be the even integers from `10` to `20`, all integers from `20` to `40`, and all even integers from `40` to `50`.

```
setA <- 20:40 #Create sequence of consecutive integers from 20 to 40
setB <- seq(10, 50, 2) #Create a sequence of integers from 10 to 50 in increments of 2
setC <- 0 #Set C to be 0

A_union_B <- union(setA, setB) #Compute the union of A and B
A_union_B
```

```
##  [1] 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 10 12 14 16
## [26] 18 42 44 46 48 50
```

2) $A \cap B$

Solution:

We can use the `intersect()` function in R. This should be the set of even integers from `20` to `40`.

```
A_intersect_B <- intersect(setA, setB)
A_intersect_B
```

```
##  [1] 20 22 24 26 28 30 32 34 36 38 40
```

2) $A \cap B^C$

Solution:

We can use the `setdiff()` function in R. $A \cap B^C$ is equivalent to the set difference of `A` and `B`. This should be all positive numbers from `20` to `40` that are odd integers.

```
A_setdiff_B <- setdiff(setA, setB)
A_setdiff_B
```

```
##  [1] 21 23 25 27 29 31 33 35 37 39
```

3) $A^C \cap B$

Solution:

We can use the `setdiff()` function in R. $A^C \cap B$ is equivalent to the set difference of `B` and `A`. This should be all even integers from 10 to 50, excluding all integers from 20 to 40.

```
B_setdiff_A <- setdiff(setB, setA)
B_setdiff_A
```

```
##  [1] 10 12 14 16 18 42 44 46 48 50
```

4) $A \cup B \cup C$

Solution:

We can use the `union()` function in R. This should be the even integers from `10` to `20`, all the integers from `20` to `40`, all even integers from `40` to `50`, and `0`.

```
A_union_B_union_C <- union(A_union_B, setC)
A_union_B_union_C
```

```
##  [1] 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 10 12 14 16
## [26] 18 42 44 46 48 50  0
```

5) $A \cap B \cap C$

Solution:

We can use the `intersect()` function in R. This should yield the empty set. As neither `A` nor `B` contain the element `0`.

```
A_intersect_B_intersect_C <- intersect(A_intersect_B, setC)
A_intersect_B_intersect_C
```

```
## numeric(0)
```

Problem 6: R built-in functions (8 points)

Calculate the following probabilities using `R` functions.

1) $X \sim N(64.2, 2.8)$**. What is** $P(X > 70)$**? (2 points)**

Solution:

We need to use the `pnorm()` function, since we are asked to find the cumulative probability of an event occurring. Since we are asked to find the right-hand tail of a normal distribution, we must do `1 -` the probability from `pnorm()`. Additionally, since this is not the typical standard normal/Gaussian distribution, we must specify the `mean` and `sd`.

```
1-pnorm(70, 64.2, 2.8) #calculate P(X > 70) for the normal distribution with mean 64.2, and sd = 2.8
```

```
## [1] 0.01915938
```

2) $Y \sim Bin(20, 0.2)$**. What is** $P(Y = 4)$**? (2 points)**

Solution:

We need to use the `dbinom()` function, since we are asked to find the point probability of an event to occur 4 times out of 20 trials.

```
dbinom(4, 20, 0.2) #calculate P(Y = 4) for the binomial distribution with p = 0.2, and 20 trials
```

```
## [1] 0.2181994
```

3) $Y \sim Bin(20, 0.2)$**. What is** $P(Y \geq 4)$**? (2 points)**

Solution:

We need to use the `pbinom()` function. However, the `pbinom()` function computes $P(Y \leq 4)$. By observing that asking $P(Y \geq 4)$ is equivalent to asking $1 - P(Y < 4)$, then we can note that this question is asking for $1 - P(Y \leq 3)$.

```
1 - pbinom(3, 20, 0.2) #calculate P(Y >= 4) for binomial distribution with p = 0.2, and 20 trials
```

```
## [1] 0.5885511
```

4) $W \sim N(100, 15)$. **Plot the density curve of `W` in red. (2 points)**

Solution:

We need to use the `plot()` function to plot the density curve. We first must simulate samples of values for to trace the curve for `W`. We will use a solid line, indicated by `lwd = 2`, and the color as red by specifying `col = 2`. We must specify the type of plot with the `type = "l"` argument.

```
x <- seq(0, 200, 1)
plot(x, dnorm(x, mean = 100, sd = 15), type = "l", ylab = "f(y)", lwd = 2, col = 2)
```