CMDA-3654

Homework 2

Eduardo Salvador

Due as a .pdf upload

Instructions:

Delete this Instructions section from your write-up!!

I have given you this assignment as an .Rmd (R Markdown) file.

- Change the name of the file to: `Lastname_Firstname_CMDA_3654_HW3.Rmd` , and your output should therefore match but with a `.pdf` extension.
- You need to edit the R Markdown file by adding chunks and filling them in appropriately with your code. Output will be generated automatically when you compile the document.
- You also need to add your own text before and after the chunks to explain what you are doing or to interpret the output.

Required: The final product that you turn in must be a .pdf file.

- You can Knit this document directly to a PDF if you have LaTeX installed (which is preferred).
- If you absolutely can't get LaTeX installed and/or working, then you can compile to a .html first, by clicking on the arrow button next to knit and selecting Knit to HTML.
- You must then print you .html file to a .pdf by using first opening it in a web browser and then printing to a .pdf

---

Problem 1: (40 pts) Learning to write functions in `R`.

Part 1:

Recall the Pythagorean theorem:

$$a^2 + b^2 = c^2$$

- Write a function called `pythagorean()` that, given the lengths of two sides of the triangle, calculates the length of the third side.
- This function should be flexible - that is, the function works if I give it values for `a` and `b` , or `b` and `c` , or `a` and `c` .

- If the user only provides the length of one side, the function should throw an error with `stop()` (read about the `stop()` function). Likewise, if the user provides the lengths of all three sides, the function should throw an error.
- If the user provides any values other than numeric values, the function should throw an error.
- Negative values don't make sense, so you should also throw an error.

```
#Create pythagorean function
pythagorean = function(a,b,c){
#Look for is there are NA values
ValuesM = is.na(c(a,b,c));
if(sum(ValuesM)==0){
stop("There are none values missing");
}
if(sum(ValuesM)>1){
stop("Values are missing");
}
values<-c(a,b,c)[!ValuesM];
#Look if there is any negative
if (any(values<=0)){
stop("Value cannot be negative");
}
#Make sure values are numbers
if (any(is.na(as.numeric(values)))){
stop("Values must be numbers");
}
#Calculate the pythagorean theorem whether you have a and b, c and b or c and a
if(is.na(c)){
c=sqrt(a^2 + b^2);
}
if(is.na(a))
a = sqrt(c^2 - b^2)

if(is.na(b))
b = sqrt(c^2 - a^2);

c(a,b,c)
}

pythagorean(2,3,NA)
```

```
[1] 2.000000 3.000000 3.605551
```

Part 2:

Now write a new function called `pythagoreans_revenge()` that can take in the following data frame (or similar data frames):

```
prob4df <- data.frame(
  a = c(2.5,  4.3,   NA, NA, -2, 4),
  b = c(8.7,   NA,  5.9, NA, 5,  4),
  c = c( NA, 12.4,  7.7,  5, NA, 4)
)

prob4df
```

a <dbl>	b <dbl>	c <dbl>
2.5	8.7	*NA*
4.3	*NA*	12.4
NA	5.9	7.7
NA	*NA*	5.0
-2.0	5.0	*NA*
4.0	4.0	4.0

6 rows

- This time we want to fill in the missing values with the appropriate value that satisfies the Pythagorean theorem.
- This time don't have the function stop when there is a problem. Instead create a new character array or factor to add onto the data frame that has the following categories:
    - `"okay"` - it took the two values and computed the missing value without any problem;
    - `"has negative"` - one or more of the values was negative;
    - `"too many missing"` - two values were missing, hence we cannot find them;
    - `"incorrect math"` - all the values were provided, but they don't satisfy the theorem.

```
#Create function pythagoreans_revenge

pythagoreans_revenge <- function(df){

  #Make NA = 0
  dataff=df
  dataff[is.na(df)]=0
  #Loop by rows
  for (i in 1:nrow(df)){

  NAV<-is.na(df[i,])
  RandomV<-dataff[i,]
  RandomV<-c(-3,2,NA)
  #Look for any negatives
  NegativeN<-sum(RandomV<0,na.rm = T)
  #Tell if too many NA values
  if (sum(NAV)>1){dataff[i,"Output"]<-factor("Too many NA Values")}

  #Tell if too many negativees
  else if (NegativeN>=1){dataff[i,"Output"]<-factor("Cannot have negative numbers")}

  #Get value if NA
  else if (dataff[i,3]==0) {dataff[i,3]<-sqrt(dataff[i,1]^2+dataff[i,2]^2)
    dataff[i,"Output"]<-factor("Okay", levels=c("Okay", "Has negative", "Too many missin
g", "Incorrect math"))      }

  #Getting missing value of a
  else if (dataff[i,1]==0) {dataff[i,1]<-sqrt(dataff[i,3]^2+dataff[i,2]^2)
   dataff[i,"Output"]=factor("Okay")
    }
  #Getting missing value of b
  else if (dataff[i,2]==0) {dataff[i,2]<-sqrt(dataff[i,3]^2+dataff[i,1]^2)
    dataff[i,"Output"]=factor("Okay")
      }
  #Print incorrect if numbers don't match
  else if (dataff[i,3]!=sqrt(dataff[i,3]^2+dataff[i,1]^2))
           dataff[i,"Output"]=factor("Incorrect math")
  }
print(dataff)
}
```

Problem 2: (40 pts) Riemann Sums with `R`

In this problem, we will write our own function to perform numerical integration via Riemann sums - a refresher of this concept can be found here (https://en.wikipedia.org/wiki/Riemann_sum).

- Consider the function $f(x) = 2e^{-2x}$ (notice, this is the probability density function for an exponential distribution with $\lambda = 2$, but that's not important here). Use midpoint Riemann sums to approximate

$$P(0 \leq x \leq 1) = \int_0^1 f(x)dx$$

  - For a specified number of segments n, your function should compute the approximate probability and compare with the exact answer by displaying the exact answer, and showing the error (you can determine the exact answer however you like, it can be hardcoded).
  - It should have an option called: `make_plot = FALSE` where `FALSE` is it's default value. When the option is `TRUE`, then your function should plot $f(x) = 2e^{-2x}$ on the specified interval `[0, 1]`, as well as the rectangles used to approximate the area. **Hint: look into using the `rect()` and `lines()` functions.**
  - Finally, it should return the run-time (how long it took to do everything) in seconds.

a. For `n = 250`, what is your approximate answer and how does it compare to the exact answer? How long did your function take to run? Go ahead provide plot (`make_plot = TRUE`) for this situation.

b. For `n = 500`, what is your approximate answer and how does it compare to the exact answer? How long did your function take to run?

c. Use the function you just wrote to estimate the probability for `n <- c( seq(10, 90, by = 10), seq(100, 1000, by = 100))`, make sure you save all of the answers.

d. Create a scatterplot with connecting lines for the estimate of the probability versus `n`. Use the function `abline()` to overlay a red line that shows the exact answer.

```
# sequence function
rieman_sum<-function(n=5, make_plot=F){
#find the subintervals boudaries
subinterval<-seq(0,1,1/n)
#find the midpoint and y values
midpoint<-seq(subinterval[2]/2, subinterval[length(subinterval)],subinterval[2])
y<-2*exp(-2*midpoint)
#find deltax
deltax<-(subinterval[n+1]-subinterval[1])/n

#evaluate f(x) at the midpoint
areas<-matrix(nrow=n,ncol=1)
for(i in 1:nrow(areas)){
  area<-deltax*y[i]
  areas[i,1]<-area
}
#Add the areas:
aproxarea<-sum(areas[,1])
AreaReal<-0.864664716767633873
cat("The real area is:", AreaReal,"\n", "the approx area is:",aproxarea,"\n","Error is:"
,AreaReal-aproxarea)

#Plotting graph
if(make_plot==T){
  plot(midpoint,y,tittle="Rieman Sum",ylab="f(x)=2e^(-2x)",xlab="x",ylim=c(-0.1,2.2),xli
m=c(-0,1,2.2))

  #Adds x and y limits
  abline(v=c(0,1),lty=c(1,2))
  abline(h=c(0,2),lty=c(1,2))
  curve(2*exp(-2*x),add=T)

  #Creating Rectangles
  for(i in 1:n){
    rect(midpoint[x]-0.5*deltax,0,midpoint[i]+0.5*deltax,y[i])
  }
}
#a
rieman_sum(250,T)
#b
rieman_sum(500,F)
#c
rieman_sum(n=c(seq(10,90,by=10)),T)
}
```

Problem 3: (20 pts) More function writing with `R`

A common concept in machine learning is one of cross-validation (https://en.wikipedia.org/wiki/Cross-validation_(statistics)). Don't worry about what it is or why it's important just yet, this topic will be covered in more detail in the future. The basic idea of the setup is to take a dataset with n observations, and randomly partition it into k equally sized subsets. Here, you will implement your own function to split a dataset into k folds.

- Write a function called `k.fold.partition` that takes in any data set (but demonstrate by passing in `mtcars`), shuffles the rows, and splits it into $k = 4$ folds, and returns a list (with components named `fold1`, `fold2`, etc) of the folds. Essentially, return an `R` object containing these smaller dataframes. °

```
#Create k.fold.partition function
k.fold.partition <- function(mtcars){
#Fill by row random values
random <-nrow(mtcars);
fold <- sample(1:10, 1)
rows <- sample(random);
#split rows
fold1 <- split(x = rows, f = 1:fold)
print(paste0('Splitting dataset into ',fold,' parts'));
#Empty list being defined and setting counter to 1
datasets <- list()
i=1
#Looping to make each side equal size and making dataframes into datasets
for (fold2 in fold1){
datasets[i] <- mtcars[fold2,]
i <- i+1;
}
datasets
}
#Creating dataframe and defining it
anydataf <- data.frame(cy1 = c(1:30),
disp = c(41:70)
)

k.fold.partition(anydataf)
```

```
[1] "Splitting dataset into 9 parts"
```

```
[[1]]
[1] 11 13  1 21

[[2]]
[1]  8 26  2 25

[[3]]
[1] 14  5 22  6

[[4]]
[1] 15 27  9

[[5]]
[1] 20 28  3

[[6]]
[1] 19  4 12

[[7]]
[1] 24 17 23

[[8]]
[1] 18  7 10

[[9]]
[1] 16 29 30
```