title: "Lab 1"

author: "Asher Katz"

output: pdf_document

date: "11:59PM February 13, 2021"

You should have RStudio installed to edit this file. You will write code in places marked "TO-DO" to complete the problems. Most of this will be a pure programming assignment but there are some questions that instead ask you to "write a few sentences". This is a W class! The tools for the solutions to these problems can be found in the class practice lectures. I prefer you to use the methods I taught you. If you google and find esoteric code you don't understand, this doesn't do you too much good.

To "hand in" the homework, you should first download this file. The best way to do this is by cloning the class repository then copying this file from the folder of that clone into the folder that is your personal class repository. Then do the assignment by filling in the TO-DO's. After you're done, compile this file into a PDF (use the "knit to PDF" button on the submenu above). This PDF will include output of your code. Then push the PDF and this Rmd file by the deadline to your github repository in a directory called "labs".

Basic R Skills

* Print out the numerical constant pi with ten digits after the decimal point using the internal constant `pi`.

```{r}
#TO-DO
print(format(pi, nsmall= 10))
```

* Sum up the first 103 terms of the series 1 + 1/2 + 1/4 + 1/8 + ...

```{r}
#TO-DO
sum(1/(2^(0:103)))
```

* Find the product of the first 37 terms in the sequence 1/3, 1/6, 1/9 ...

```{r}
#TO-DO
1/(3*(1:37))
prod(1/(3*(1:37)))
```

* Find the product of the first 387 terms of `1 * 1/2 * 1/4 * 1/8 *` ...

```{r}
#TO-DO
1/(2^(0:386))
prod(1/(2^(0:386)))
# the answer is 0 because of Underflow
```

Is this answer *exactly* correct?

No, its rounded because of underflow

```
#TO-DO
```

  * Figure out a means to express the answer more exactly. Not compute exactly, but express more exactly.

```{r}
#TO-DO
-log(2)*sum((0:386))
#this partially eliminates underflow
```

* Create the sequence `x = [Inf, 20, 18, ..., -20]`.

```{r}
#TO-DO
x = c(Inf, seq(from = 20, to = -20, by = -2))
print(x)
```

Create the sequence `x = [log_3(Inf), log_3(100), log_3(98), ... log_3(-20)]`.

```{r}
#TO-DO
x = log( c(Inf, seq(from = 100, to = - 20, by = -2)), base = 3)
print(x)
```

Comment on the appropriateness of the non-numeric values.

NAN occurs because you cannot take the log of a negative number.

-Inf occurs when you take the log of 0.

* Create a vector of booleans where the entry is true if `x[i]` is positive and finite.

```{r}
#TO-DO
x = log( c(Inf, seq(from = 100, to = - 20, by = -2)), base = 3)
is.finite(x) & x > 0
# if x is finite & positive
```

* Locate the indices of the non-real numbers in this vector. Hint: use the `which` function. Don't hesitate to use the documentation via `?which`.

```{r}
#TO-DO
#so that we can see the documentation
#there are two types of non-real numbers here, inf and NAND

which(is.nan(x) |is.infinite(x) )

```

* Locate the indices of the infinite quantities in this vector.

```{r}
```

#TO-DO

which(is.infinite(x))

```

* Locate the indices of the min and max in this vector. Hint: use the `which.min` and `which.max` functions.

```{r}

#TO-DO

which.min(x)

which.max(x)

```

* Count the number of unique values in `x`.

```{r}

#TO-DO

length(unique(x))

```

* Cast `x` to a factor. Do the number of levels make sense?

yes, because Nan is counted a level

```{r}

#TO-DO

#factors are data objects, categorizes the data according to "levels".

factor(x)

# the number of levels is 53

```

* Cast `x` to integers. What do we learn about R's infinity representation in the integer data type?

That R doesn't recognize infinity as an integer

```{r}

#TO-DO

as.integer(x)

```

* Use `x` to create a new vector `y` containing only the real numbers in x.

```{r}

#TO-DO

y = x[ !( is.nan(x) | is.infinite(x) ) ]

y

```

* Use the left rectangle method to numerically integrate x^2 from 0 to 1 with rectangle width size 1e-6.

```{r}

#TO-DO

d = 1E-6

# the rate of change in x values

x = seq(0, 1 - d, by = d) # the domain

y = x^2 # the function, given the height of each rectangle

sum(y)*d # factored the d out of the sequence

```

* Calculate the average of 100 realizations of standard Bernoullis in one line using the `sample` function.

```{r}

#TO-DO

# a standard Bernouli is a yes or no random variable

#sample(Sample from where?, size(=) of sample?, replace(=) while sampling?, display prob(=)abilities? )

#sample( c(0,1), 100, replace = TRUE )

# replace puts the element back so that the every time samples the full vector, not a vec - an element

mean( sample( c(0,1), 100, replace = TRUE ) )

```

* Calculate the average of 500 realizations of Bernoullis with p = 0.9 in one line using the `sample` and `mean` functions.

```{r}

#TO-DO

mean( sample( c(0,1),100, replace = TRUE, prob = c(0.1, 0.9) ) )

# prob of no is 0.1, prob of yes is 0.9

```
```

* Calculate the average of 1000 realizations of Bernoullis with p = 0.9 in one line using `rbinom`.

```{r}
#TO-DO
mean( rbinom( n = 1000, size = 1, prob = 0.9) )
```

* In class we considered a variable `x_3` which measured "criminality". We imagined L = 4 levels "none", "infraction", "misdemeanor" and "felony". Create a variable `x_3` here with 100 random elements (equally probable). Create it as a nominal (i.e. un-ordered) factor.

```{r}
#TO-DO
x_3 = factor(sample( c("none", "infraction", "misdemeanor", "felony"), size = 100, replace = TRUE))
x_3
```

* Use `x_3` to create `x_3_bin`, a binary feature where 0 is no crime and 1 is any crime.

```{r}
#TO-DO
x_3_bin = x_3 != "none"
x_3_bin = as.numeric( x_3_bin )
x_3_bin
```

* Use `x_3` to create `x_3_ord`, an ordered factor variable. Ensure the proper ordinal ordering.

```{r}

#TO-DO

x_3_ord = factor(x_3, levels = c("none", "infraction", "misdemeanor", "felony"), ordered = TRUE)

x_3_ord

```

* Convert this variable into three binary variables without any information loss and put them into a data matrix.

```{r}

#TO-DO

X = matrix( nrow = length(x_3), ncol = 3 )

# 100 rows, 3 columns

X[,1] = as.numeric(x_3 =="infraction")

#X[,1] the first column will be true wherever x_3 is "infraction, this is boolean, so the numeric representation would be 0 or 1

X[,2] = as.numeric(x_3 =="misdemeanor")

X[,3] = as.numeric(x_3 =="felony")

X

```

* What should the sum of each row be (in English)?

The sum of each row should be 1 or 0 because there is only a single value in each of the 100 entries in x_3, 0 if no crime

#TO-DO

Verify that.

```{r}

#TO-DO

rowSums(X)

table(rowSums(X))

#table of how many rowSum of 1, rowSum of 0

```

* How should the column sum look (in English)?

#TO-DO

It should look like 3 columns, each representing the sum of a particular crime type and each with a value of approximately 25. There should be approximately 25 missing

Verify that.

```{r}

#TO-DO

colSums(X)

table(x_3)

```

* Generate a matrix with 100 rows where the first column is realization from a normal with mean 17 and variance 38, the second column is uniform between -10 and 10, the third column is poisson with mean 6, the fourth column in exponential with lambda of 9, the fifth column is binomial with n = 20 and p = 0.12 and the sixth column is a binary variable with exactly 24% 1's dispersed randomly. Name the rows the entries of the `fake_first_names` vector.

```{r}
```

```
fake_first_names = c(
  "Sophia", "Emma", "Olivia", "Ava", "Mia", "Isabella", "Riley",
  "Aria", "Zoe", "Charlotte", "Lily", "Layla", "Amelia", "Emily",
  "Madelyn", "Aubrey", "Adalyn", "Madison", "Chloe", "Harper",
  "Abigail", "Aaliyah", "Avery", "Evelyn", "Kaylee", "Ella", "Ellie",
  "Scarlett", "Arianna", "Hailey", "Nora", "Addison", "Brooklyn",
  "Hannah", "Mila", "Leah", "Elizabeth", "Sarah", "Eliana", "Mackenzie",
  "Peyton", "Maria", "Grace", "Adeline", "Elena", "Anna", "Victoria",
  "Camilla", "Lillian", "Natalie", "Jackson", "Aiden", "Lucas",
  "Liam", "Noah", "Ethan", "Mason", "Caden", "Oliver", "Elijah",
  "Grayson", "Jacob", "Michael", "Benjamin", "Carter", "James",
  "Jayden", "Logan", "Alexander", "Caleb", "Ryan", "Luke", "Daniel",
  "Jack", "William", "Owen", "Gabriel", "Matthew", "Connor", "Jayce",
  "Isaac", "Sebastian", "Henry", "Muhammad", "Cameron", "Wyatt",
  "Dylan", "Nathan", "Nicholas", "Julian", "Eli", "Levi", "Isaiah",
  "Landon", "David", "Christian", "Andrew", "Brayden", "John",
  "Lincoln"
)
#TO-DO
n = 100
x = matrix(NA, nrow = n, ncol = 6)
rownames(x) = fake_first_names
x[,1] = rnorm(n, mean = 17, sd = sqrt(38))
#the first column is realization from a normal with mean 17 and variance 38
# individual variance is (indi val - mean)^2, (general) variance is the avereage of all the individual var,
x[,2] = runif(n, -10, 10)
# the second column is uniform between -10 and 10
x[,3] = rpois(n, lambda = 6)
# the third column is poisson with mean 6
```

#lambda is is the expected number of event occurrences in a given amount of time.

we think 6 will happen, how many actually happen

x[,4] = rexp(n, rate = 1/9)

#fourth column in exponential with lambda of 9

#

x[,5] = rbinom(n, size =20, prob =.12)

#the fifth column is binomial with n = 20 and p = 0.12

x[,6] = sample(c(rep(1,24), rep(0,76)))

#the sixth column is a binary variable with exactly 24% 1's dispersed randomly

x

```

* Create a data frame of the same data as above except make the binary variable a factor "DOMESTIC" vs "FOREIGN" for 0 and 1 respectively. Use RStudio's `View` function to ensure this worked as desired.

```{r}

#TO-DO

df = data.frame(x)

df$X6 = factor(df$X6, levels = c(0, 1), labels = c("DOMESTIC", "FOREIGN"))

#col is $, we're giving the binary col two levels: domestic=0 and foreign=1

View(df)

```
```

* Print out a table of the binary variable. Then print out the proportions of "DOMESTIC" vs "FOREIGN".

```{r}
#TO-DO
table(df$X6)
table(df$X6)/n
# value /100
```

Print out a summary of the whole dataframe.

```{r}
#TO-DO
summary(df)
```

* Let `n = 50`. Create a n x n matrix `R` of exactly 50% entries 0's, 25% 1's 25% 2's. These values should be in random locations.

```{r}
#TO-DO
n = 50
# 2500 entries, 1250 of them 0's, 625 of them 1's, 625 of them 2's
# rep is repeat(a value, x many times)
R = matrix( sample( c( rep(0, 1250), rep(1, 625), rep(2, 625) ) ), nrow = n, ncol = n )
View(R)
```

* Randomly punch holes (i.e. `NA`) values in this matrix so that an each entry is missing with probability 30%.

```{r}
for ( i in 1:n ){
  for ( j in 1:n ){
    if ( runif(1) < 0.3 ){
      # runif(1) creates a random probability between 1 and 0. 0.3 is 30%
      R[i,j] = NA
      # overall, we have a nested for loop to look at every entry and 30% of the time, punch an NA
    }
  }
}
View(R)
```

* Sort the rows in matrix `R` by the largest row sum to lowest. Be careful about the NA's!

```{r}
#TO-DO
SR = R[ order( rowSums( R, na.rm = TRUE), decreasing = TRUE), ]

rowSums(R, na.rm = TRUE)
rowSums( SR, na.rm = TRUE)
View(SR)

```

* We will now learn the `apply` function. This is a handy function that saves writing for loops which should be eschewed in R. Use the apply function to compute a vector whose entries are the standard deviation of each row. Use the apply function to compute a vector whose entries are the standard deviation of each column. Be careful about the NA's! This should be one line.

```{r}
#TO-DO

#apply( array or mat, apply_a_function_to_where?, apply_what_function? , extra arguments, simplify = TRUE)

apply(R, MARGIN = c(1), FUN = sd, na.rm = TRUE )

apply(R, MARGIN = c(2), FUN = sd, na.rm = TRUE )
```

* Use the `apply` function to compute a vector whose entries are the count of entries that are 1 or 2 in each column. This should be one line.

```{r}
#TO-DO

R = matrix( sample( c( rep(0, 1250), rep(1, 625), rep(2, 625) ) ), nrow = n, ncol = n )

apply(R, MARGIN = c(2), FUN = count())
```

* Use the `split` function to create a list whose keys are the column number and values are the vector of the columns. Look at the last example in the documentation `?split`.

```{r}
#TO-DO

lst = split(R, col(R))

lst

# take every column of R and make it an element of a list
```

```

* In one statement, use the `lapply` function to create a list whose keys are the column number and values are themselves a list with keys: "min" whose value is the minimum of the column, "max" whose value is the maximum of the column, "pct_missing" is the proportion of missingness in the column and "first_NA" whose value is the row number of the first time the NA appears.

```{r}
#TO-DO
lapply(split(R, col(R)), function(c) { list("min" = min(c, na.rm = TRUE), "max" = max(c, na.rm = TRUE), "pct_missing" = sum(is.na(c)) / length(c), "first_NA" = which(is.na(c)) [1] )})
```

* Set a seed and then create a vector `v` consisting of a sample of 1,000 iid normal realizations with mean -10 and variance 100.

```{r}
#TO-DO
set.seed(37)
v = rnorm(1000, mean = -10, sd = sqrt(100))
v
```

* Repeat this exercise by resetting the seed to ensure you obtain the same results.

```{r}
#TO-DO
set.seed(37)
v = rnorm(1000, mean = -10, sd = 10)
```
```

v

```

* Find the average of `v` and the standard error of `v`.

```{r}
#TO-DO
mean(v)
SE = sd(v)/sqrt(length(v))
SE
```

* Find the 5%ile of `v` and use the `qnorm` function to compute what it theoretically should be. Is the estimate about what is expected by theory?

Yes

```{r}
#TO-DO
quantile(v, .05)
# this simply divides the sample into 20 quantiles and returns one in the bottom 5%
qnorm(.05, mean = -10, sd = 10)
# qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
# qnorm finds the 5th percentile of a normal distribution with mean = -10 and standard deviation = 10.
# 5% of the values in this normally distributed population will lie below our solution. 95% of val are above
```

* What is the percentile of `v` that corresponds to the value 0? something like 84.0055%
```

What should it be theoretically?

Is the estimate about what is expected by theory? yes

```{r}
#TO-DO
quantile(v, 0.840055)
# 0 is around 84.0055 percentile of the sample
pnorm(0, mean = -10, sd = 10)
```