title: "Lab 1" author: "Amir ElTabakh" output: pdf_document

date: "11:59PM February 18, 2021"

You should have RStudio installed to edit this file. You will write code in places marked "TO-DO" to complete the problems. Some of this will be a pure programming assignment. The tools for the solutions to these problems can be found in the class practice lectures. I want you to use the methods I taught you, not for you to google and come up with whatever works. You won't learn that way.

To "hand in" the homework, you should compile or publish this file into a PDF that includes output of your code. Once it's done, push by the deadline to your repository in a directory called "labs".

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

```{r} options(digits=11) x <- pi x
```

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

```{r}
sum(1/(2^(0:102)))
```

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

```{r} prod(1/(3*(1:37))) prod(1/seq(from=3, by=3, length.out=37))
```

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

```{r}
prod(1/(2^(0:386)))
```

Is this answer *exactly* correct?

This answer is not exactly correct, the program is rounding to zero.

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

```{r} sum(log(1/(2^(0:386)))) -log(2)*sum(0:386)
```

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

```{r}
x <- c(Inf, seq(from=20, to=-20, by=-2))
x
```

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

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

```
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}
y = !is.nan(x) & is.finite(x) & x > 0
y
```

- 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} ?which which(!y) which(y == FALSE)
```

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

```{r}
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} which.min(x) which.max(x)

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

```{r}
length(unique(x))
```

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

```{r} as.factor(x)

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

```{r}
as.integer(x)
```

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

```{r} y = x[!is.nan(x) & is.finite(x)] y

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

```{r}
sum(seq(from=0, to=1-(1e-6), by=1e-6)^2)*1e-6
```

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

```{r} sum(sample(c(0,1), size=100, replace=TRUE))/100

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

```{r}
sum(sample(c(0,1), size=500, replace=TRUE, prob=c(0.1, 0.9)))/500
```

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

```{r} ?rbinom rbinom(n=1000, size=1, p=0.9)

* In class we considered a variable `x_3` which measured "criminality". We imagined L = 4 levels "none", "infraction", "misdimeanor"

```{r}
x_3 = as.factor(sample(c("none",  "infraction", "misdimeanor", "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} x_3_bin = x_3 != "none" x_3_bin

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

```{r}
x_3_ord = factor(x_3, levels = c("none",  "infraction", "misdimeanor", "felony"), order=TRUE)
x_3_ord
```

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

```{r} x_3_matrix = matrix(nrow = length(x_3), ncol = 3) x_3_matrix[ ,1] = as.numeric(x_3 == "infraction") x_3_matrix[ ,2] = as.numeric(x_3 == "felony") x_3_matrix[ ,3] = as.numeric(x_3 == "misdimeanor") colnames(x_3_matrix) = c("infraction", "felony", "is_misdimeanor") x_3_matrix

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

The sum of each row should be 1 or 0. If the individual has a record of 'none', that will be captured by a row sum of zero.

  Verify that.

```{r}
rowSums(x_3_matrix)
```

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

We should expect for there to be about 25 values per column. This is assuming the sample() function uniformly distributes values.

Verify that.

```{r} colSums(x_3_matrix)

* Generate a matrix with 100 rows where the first column is realization from a normal with mean 17 and variance 38, the second colum

```{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"
)

n <- 100
X <- matrix(nrow=n, ncol=6)
X[,1] <- rnorm(n=n, mean=17, sd=sqrt(38))
X[,2] <- runif(n=n, min=-10, max=10)
X[,3] <- rpois(n=n, lambda=6)
X[,4] <- rexp(n=n, rate=9)
X[,5] <- rbinom(n=n, size=20, p=0.12)
X[,6] <- sample(c(rep(1, n * 0.24), rep(0, n*0.76)))

rownames(X) = fake_first_names

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} df = data.frame(X) df$X6 = factor(df$X6, levels = c(0, 1), labels = c("DOMESTIC", "FOREIGN")) View(df, "Lab 1 DF")

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

```{r}
table(df$X6)
```

Print out a summary of the whole dataframe.

```{r} 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}
n <- 50
R <- matrix(nrow=n, ncol=n, sample(c(rep(0, n*n*0.5), rep(1, n*n*0.25), rep(2, n*n*0.25))))
df <- data.frame(R)
df
```

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

```{r} n <- 50 R <- matrix(nrow=n, ncol=n, sample(c(rep(0, n*0.5), rep(1, n*0.25), rep(2, n*0.25))))

holes = matrix(nrow=n, ncol=n, sample(c(rep(0, n*0.7), rep(3, n*0.3))))

for(i in 1:n){ for(j in 1:n){ if(holes[i,j] == 3){ R[i, j] = NA } } } R

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

```{r}
order(rowSums(R, na.rm=TRUE), decreasing=TRUE)
```

- 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} row <- apply(R, MARGIN = 1, sd, na.rm=TRUE) col <- apply(R, MARGIN = 2, 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

```{r}
apply(R>0, MARGIN = 2, sum, na.rm=TRUE)
```

- 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} split(R, col(R))

```
* In one statement, use the `lapply` function to create a list whose keys are the column number and values are themselves a list wit

```{r}
lapply(split(R, col(R)), function(R){list(min = min(R, na.rm = T), max = max(R, na.rm = T), pct_missing = (sum(is.na(R)) / n), first
```

- 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.

sd = sqrt(var) var = sd^2 ```{r} set.seed(5) n <- 1000 v <- rnorm(n, mean=-10, sd = sqrt(100))

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

```{r}
set.seed(5)
n <- 1000
v <- rnorm(n, mean=-10, sd = sqrt(100))
v
```

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

```{r} avg_v <- mean(v) avg_v se_v <- sd(v)/n se_v

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

```{r}
fifth_percentile <- quantile(v, probs = 0.05)
fifth_percentile

qnorm(0.05, mean = -10, sd = sqrt(100))
```

- What is the percentile of `v` that corresponds to the value 0? What should it be theoretically? Is the estimate about what is expected by theory?

```{r} ecdf(v)(0) pnorm(0, mean = -10, sd = sqrt(100))
```
```