---

title: “Lab 1” author: “Nasif Khan” output: pdf_document date: “11:59PM February 18, 2021”

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

#Sum up the first 103 terms of the series 1 + 1/2 + 1/4 + 1/8 + ... sum(1/(2^(0:102)))

#Find the product of the first 37 terms in the sequence 1/3, 1/6, 1/9 ... 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 *` ... prod(1/(2^(0:386)))

#Is this answer *exactly* correct?

#TO-DO

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

#Create the sequence `x = [Inf, 20, 18, ..., -20]`. 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)]`. 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 the log of a negative number doesn't exist.

#Create a vector of booleans where the entry is true if `x[i]` is positive and finite. 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`. ?which which(!y) which(y == FALSE)

#Locate the indices of the infinite quantities in this vector. which(is.infinite(x))

#Locate the indices of the min and max in this vector. Hint: use the `which.min` and `which.max` functions. which.min(x) which.max(x)

#Count the number of unique values in `x`. length(unique(x))

#Cast `x` to a factor. Do the number of levels make sense? as.factor(x)

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

#Use `x` to create a new vector `y` containing only the real numbers in x. 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. 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. 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. 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`. ?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” and “felony”. Create a variable `x_3` here with 100 random elements (equally probable). Create it as a nominal (i.e. unordered) factor. 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. 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. 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. levels <- c('none', 'infraction', 'misdimeanor','felony') value <- c('0','1','2','3') df <- data.frame(levels, value) data.matrix(df)

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

#TO-DO #The sums of each row from 1-4 is: 5,4,6,5 #Verify that.

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

#TO-DO

#Verify that.

#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. #fmat=fakenamematrix fmat = matrix(data=NA, nrow=100, ncol=6) fmat[,1] = rnorm(n = 100, mean=17, sd=sqrt(38)) fmat[,2] = runif(100, min=-10, max=10) fmat[,3] = rpois(100, mean=6) fmat[,4] = rexp(100, lambda=9) fmat[,5] = rbinom(100, n=20, p=.12) #fmat[,6] = idk

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")

row.names(fmat) = first_fake_names

#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. levels = c(0,1) factor = c('DOMESTIC', 'FOREIGN') name_frame = data.frame(fake_first_names, factor, levels) View(name_frame)

#Print out a table of the binary variable. Then print out the proportions of "DOMESTIC" vs "FOREIGN". print.table(name_frame)

#Print out a summary of the whole dataframe. summary(name_frame)

#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. n=50 R = data.matrix(sample(c(rep(0, n^2**.5), rep(1, n^2**.25), rep(2, n^2*.25))), nrow=50, ncol=50)

#Randomly punch holes (i.e. `NA`) values in this matrix so that an each entry is missing with probability 30%. for(val in n) R[v] = rand.int(runif(v) < 0.3)

#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. B apply(B, nrow=B, sd, na.rm = TRUE) apply(B, ncol=B, 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. apply(B>0, MARGIN=2, na.rm=TRUE, sum)

#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`. splt(B, col(B)) ?split

#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. lapply(split(row(B), col(B)), min=min(B,na.rm=row),max=max(B,na.rm=col),pct_missing=sum(is.na(B),first_NA=min(which(is.na(B)))))

#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. set.seed(3) v = rnorm(n = 1000, mean=-10, sd=10)

#Repeat this exercise by resetting the seed to ensure you obtain the same results. numbers=(1:10) for(num in numbers) set.seed(i) v = rnorm(n=1000, mean=-10, sd=10) print(v)

#Find the average of `v` and the standard error of `v`. mean_v = mean(v, sqrt(1000)

#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? qnorm(v, mean=mean_v, sd=.05, lower.tail = TRUE, log.v = TRUE)

#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?