Lab 1

Elizabeth McHugh

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`.

```
options(digits = 11)
x <- pi
x
```

```
## [1] 3.1415926536
```

- Sum up the first 103 terms of the series $1 + 1/2 + 1/4 + 1/8 + \ldots$

```
sum(1 / (2^(0:102)) )
```

```
## [1] 2
```

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

```
prod(1 / (3 * (1:37)) )
```

```
## [1] 1.613528728e-61
```

```
prod(1/seq(from = 3, by = 3, length.out = 37))
```

```
## [1] 1.613528728e-61
```

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

```
prod(1/(2^(0:386)))
```

```
## [1] 0
```

Is this answer *exactly* correct?

No. Since the sequence 1/2^(n) approaches 0 as n approaches infinity, the product would not be exactly 0 were we to stop before reaching an infinite number of terms. However, once we choose a large enough 'n', that is "enough" terms, the resulting fraction is so small that, when added in the product, the computer stops computing decimal places.

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

```
sum(log(1/(2^(0:386))))
```

```
## [1] -51771.856063
```

```
-log(2)*sum(0:386)
```

```
## [1] -51771.856063
```

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

```
c(Inf, seq(from = 20, to = -20, by = -2))
```

```
##  [1] Inf  20  18  16  14  12  10   8   6   4   2   0  -2  -4  -6  -8 -10 -12 -14
## [20] -16 -18 -20
```

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))

log(x, base = 3)
```

```
## Warning: NaNs produced
```

```
##  [1]          Inf 4.19180654858 4.17341725189 4.15464876786 4.13548512895
##  [6] 4.11590933734 4.09590327429 4.07544759936 4.05452163807 4.03310325630
## [11] 4.01116871959 3.98869253500 3.96564727304 3.94200336639 3.91772888179
## [16] 3.89278926071 3.86714702345 3.84076143031 3.81358809222 3.78557852143
## [21] 3.75667961083 3.72683302786 3.69597450568 3.66403300988 3.63092975357
## [26] 3.59657702662 3.56087679501 3.52371901429 3.48497958377 3.44451784579
## [31] 3.40217350273 3.35776278143 3.31107361282 3.26185950714 3.20983167673
## [36] 3.15464876786 3.09590327429 3.03310325630 2.96564727304 2.89278926071
## [41] 2.81358809222 2.72683302786 2.63092975357 2.52371901429 2.40217350273
## [46] 2.26185950714 2.09590327429 1.89278926071 1.63092975357 1.26185950714
## [51] 0.63092975357         -Inf          NaN          NaN          NaN
## [56]          NaN          NaN          NaN          NaN          NaN
## [61]          NaN          NaN
```

Comment on the appropriateness of the non-numeric values.

x[1]: Inf is appropriate since the limit approaching infinity of log is infinity.

x[52]: -Inf is appropriate since the limit approaching 0 of log is negative infinity.

x[53: 62]: NaN is appropriate, as log is undefined for all non-negative integers.

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

```
y = c(!is.nan(x) & !is.infinite(x) & x > 0)
y
```

```
##  [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [37]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [49]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [61] FALSE FALSE
```

- 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(!y)
```

```
##  [1]  1 52 53 54 55 56 57 58 59 60 61 62
```

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

```
which(is.infinite(x))
```

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

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

```
which.min(x)
```

```
## [1] 62
```

```
which.max(x)
```

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

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

```
length(unique(x))
```

```
## [1] 62
```

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

```
factor(x)
```

```
##  [1] Inf 100 98  96  94  92  90  88  86  84  82  80  78  76  74  72  70  68  66
## [20] 64  62  60  58  56  54  52  50  48  46  44  42  40  38  36  34  32  30  28
## [39] 26  24  22  20  18  16  14  12  10  8   6   4   2   0   -2  -4  -6  -8  -10
## [58] -12 -14 -16 -18 -20
## 62 Levels: -20 -18 -16 -14 -12 -10 -8 -6 -4 -2 0 2 4 6 8 10 12 14 16 18 ... Inf
```

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

```
as.integer(x)
```

```
## Warning: NAs introduced by coercion to integer range
```

```
##  [1]  NA 100  98  96  94  92  90  88  86  84  82  80  78  76  74  72  70  68  66
## [20]  64  62  60  58  56  54  52  50  48  46  44  42  40  38  36  34  32  30  28
## [39]  26  24  22  20  18  16  14  12  10   8   6   4   2   0  -2  -4  -6  -8 -10
## [58] -12 -14 -16 -18 -20
```

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

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

```
##  [1] 100  98  96  94  92  90  88  86  84  82  80  78  76  74  72  70  68  66  64
## [20]  62  60  58  56  54  52  50  48  46  44  42  40  38  36  34  32  30  28  26
## [39]  24  22  20  18  16  14  12  10   8   6   4   2   0  -2  -4  -6  -8 -10 -12
## [58] -14 -16 -18 -20
```

- 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
```

```
## [1] 0.33333283333
```

- 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
```

```
## [1] 0.55
```

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

```
mean(sample(c(0, 1), size = 500, replace = TRUE, prob = c(0.1, 0.9)))
```

```
## [1] 0.904
```

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

```
mean(rbinom(n = 1000, size = 1, prob = 0.9))
```

```
## [1] 0.905
```

- 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", "misdemeanor", "felony"), size = 100, replace = TRUE))

x_3
```

```
##   [1] felony      felony      none        none        felony      none
##   [7] felony      felony      none        none        infraction  infraction
##  [13] none        infraction  infraction  misdemeanor none        none
##  [19] misdemeanor none        misdemeanor felony      felony      felony
##  [25] infraction  misdemeanor infraction  none        none        none
##  [31] misdemeanor felony      felony      misdemeanor infraction  none
##  [37] misdemeanor none        misdemeanor none        misdemeanor felony
##  [43] infraction  felony      none        misdemeanor infraction  misdemeanor
##  [49] misdemeanor infraction  felony      felony      infraction  infraction
##  [55] misdemeanor infraction  infraction  none        infraction  none
##  [61] infraction  none        none        infraction  none        infraction
##  [67] none        infraction  felony      infraction  misdemeanor felony
##  [73] felony      misdemeanor infraction  misdemeanor felony      none
##  [79] misdemeanor none        none        misdemeanor infraction  none
##  [85] felony      infraction  none        infraction  misdemeanor none
##  [91] infraction  misdemeanor felony      misdemeanor infraction  misdemeanor
##  [97] none        felony      misdemeanor felony
## Levels: felony infraction misdemeanor none
```

- 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"

as.numeric(x_3_bin)
```

```
##   [1] 1 1 0 0 1 0 1 1 0 0 1 1 0 1 1 1 0 0 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 0 1
##  [38] 0 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 0 1 0 1 0 1 1 1 1 1 1 1
##  [75] 1 1 1 0 1 0 0 1 1 0 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1
```

- 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", "misdemeanor", "felony"), ordered = TRUE)

x_3_ord
```

```
##   [1] felony      felony      none        none        felony      none
##   [7] felony      felony      none        none        infraction  infraction
##  [13] none        infraction  infraction  misdemeanor none        none
```

```
##  [19] misdemeanor none        misdemeanor felony      felony      felony
##  [25] infraction  misdemeanor infraction  none        none        none
##  [31] misdemeanor felony      felony      misdemeanor infraction  none
##  [37] misdemeanor none        misdemeanor none        misdemeanor felony
##  [43] infraction  felony      none        misdemeanor infraction  misdemeanor
##  [49] misdemeanor infraction  felony      felony      infraction  infraction
##  [55] misdemeanor infraction  infraction  none        infraction  none
##  [61] infraction  none        none        infraction  none        infraction
##  [67] none        infraction  felony      infraction  misdemeanor felony
##  [73] felony      misdemeanor infraction  misdemeanor felony      none
##  [79] misdemeanor none        none        misdemeanor infraction  none
##  [85] felony      infraction  none        infraction  misdemeanor none
##  [91] infraction  misdemeanor felony      misdemeanor infraction  misdemeanor
##  [97] none        felony      misdemeanor felony
## Levels: none < infraction < misdemeanor < felony
```

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

```
x_3_infraction = as.integer( x_3_ord == "infraction" )
x_3_misdemeanor = as.integer( x_3_ord == "misdemeanor")
x_3_felony = as.integer( x_3_ord =="felony" )

X = cbind(x_3_infraction, x_3_misdemeanor, x_3_felony)
colnames(X) = list("Infraction", "Misdemeanor", "Felony")

head(X)
```

```
##      Infraction Misdemeanor Felony
## [1,]          0           0      1
## [2,]          0           0      1
## [3,]          0           0      0
## [4,]          0           0      0
## [5,]          0           0      1
## [6,]          0           0      0
```

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

The sum of each row should be tell us whether or not each subject had been charged with a crime (0 for no crime, 1 for crime).

Verify that.

```
rowSums(X, na.rm = FALSE, dims = 1)
```

```
##   [1] 1 1 0 0 1 0 1 1 0 0 1 1 0 1 1 1 0 0 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 0 1
##  [38] 0 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 0 1 0 1 0 1 1 1 1 1 1 1 1
##  [75] 1 1 1 0 1 0 0 1 1 0 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1
```

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

The sum of each column should give the number of occurrences of each type of crime.

Verify that.

```
colSums (X, na.rm = FALSE, dims = 1)
```

```
##  Infraction Misdemeanor      Felony
##          26          23          22
```

- 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.

```
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 = length(fake_first_names)

Names = data.frame(
  normal = rnorm(n, mean = 17, sd = 38),
  uniform = runif(n, min = -10, max = 10),
  poisson = rpois(n, 6),
  exponential = rexp(n, rate = 9),
  binomial = rbinom(n, size = 20, p = 0.12),
  binary =  rbinom(n, size = 1, p = 0.24)
)


head(Names)
```

```
##           normal        uniform poisson     exponential binomial binary
## 1  44.4262606250  6.82487245649       7 0.0022978653821        2      0
## 2 -20.8778522233  5.29107909650       8 0.1059441962750        5      0
## 3  14.6731021914  0.56393908337       4 0.2136410660948        2      0
## 4  -8.8015875095 -3.95408915821       6 0.0935704602474        2      0
## 5 101.5876168495  7.38516540732      10 0.1000190622423        4      1
## 6  78.6099448229 -5.44198126066       4 0.0418845606434        2      0
```

```
tail(Names)
```

```
##             normal        uniform poisson    exponential binomial binary
## 95   44.0875809002  9.84488975257       8 0.198613306912        3      1
## 96  -26.4211811775  2.81448271591       9 0.387978228200        3      0
## 97   42.9739797066 -1.09606390353       7 0.224611596298        3      1
## 98   -7.0798920563 -2.85976238083       4 0.081473677207        1      0
## 99  -49.2611043785 -0.98462041933       6 0.101016430143        1      0
## 100  33.6949035723 -0.66754406784       7 0.240644676196        4      1
```

- 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.

```
n = length(fake_first_names)

Names = data.frame(
  normal = rnorm(n, mean = 17, sd = 38),
  uniform = runif(n, min = -10, max = 10),
  poisson = rpois(n, 6),
  exponential = rexp(n, rate = 9),
  binomial = rbinom(n, size = 20, p = 0.12),
  binary =  rbinom(n, size = 1, p = 0.24)
)

Names$binary = factor(Names$binary, labels = c("DOMESTIC", "FOREIGN"))

head(Names)
```

```
##           normal       uniform poisson     exponential binomial   binary
## 1 -37.2238849742 -5.0924811233       2 0.0077729449194        0  FOREIGN
## 2  80.4407855929  7.3889320018      10 0.1352508912437        3 DOMESTIC
## 3 -67.0291078425 -7.8260522103       6 0.0268093114719        2  FOREIGN
## 4  35.8995379722  1.5665974980       7 0.2302332837041        2 DOMESTIC
## 5   7.6535603762 -6.8702971097       6 0.1917664199702        2 DOMESTIC
## 6  29.7769584798  9.7977159079       3 0.0571239266234        2 DOMESTIC
```

```
tail(Names)
```

```
##            normal        uniform poisson    exponential binomial   binary
## 95  -23.070866298  2.75441625621       6 0.209370134389        1 DOMESTIC
## 96   35.242704518  6.59892231692       5 0.221047964766        1 DOMESTIC
## 97   59.314804573  7.60341668036       7 0.071307935818        3 DOMESTIC
## 98  -18.328901666  0.98477563821       7 0.237778169424        3 DOMESTIC
## 99   65.676548563 -3.76007916871       9 0.079137579947        1 DOMESTIC
## 100 -33.235445453  6.63217966910       5 0.040544025827        1 DOMESTIC
```

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

```
table(Names$binary)
```

```
##
## DOMESTIC  FOREIGN
##       79       21
```

Print out a summary of the whole dataframe.

```
summary(Names)
```

```
##      normal               uniform              poisson
##  Min.   :-112.5894962   Min.   :-9.96076734   Min.   : 2.00
##  1st Qu.:  -3.6110413   1st Qu.:-4.98292787   1st Qu.: 4.00
##  Median :  19.4925990   Median : 0.97103927   Median : 6.00
##  Mean   :  19.4319426   Mean   : 0.48197690   Mean   : 6.10
##  3rd Qu.:  47.9829261   3rd Qu.: 6.06769884   3rd Qu.: 7.25
##  Max.   : 113.9931124   Max.   : 9.81914621   Max.   :11.00
##   exponential             binomial         binary
##  Min.   :0.0016458972   Min.   :0.00   DOMESTIC:79
##  1st Qu.:0.0331628152   1st Qu.:1.00   FOREIGN :21
##  Median :0.0767961491   Median :2.00
##  Mean   :0.1156426667   Mean   :2.34
##  3rd Qu.:0.1703339541   3rd Qu.:3.00
##  Max.   :0.6247964683   Max.   :6.00
```

- 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 = matrix(
  sample(c(0,1,2), size = n*n, replace = TRUE, prob = c(0.5, .25, .25)),
  nrow = n,
  ncol = n)

table(R)/(n*n)  #This doesn't give EXACTLY the proportions desired, so I'm not doing something correctl
```

```
## R
##      0      1      2
## 0.5008 0.2472 0.2520
```

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

```
table(R)
```

```
## R
##    0    1    2
## 1252  618  630
```

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

```
R_row_sums = rowSums(R)
R_row_sums
```

```
##  [1] 30 42 37 45 44 34 27 40 40 40 35 34 27 34 44 31 35 41 34 41 42 35 42 52 48
## [26] 33 37 33 31 29 33 34 49 34 44 33 41 38 34 40 40 30 34 41 41 40 53 36 35 31
```

```
order(R_row_sums, decreasing = FALSE)
```

```
##  [1]  7 13 30  1 42 16 29 50 26 28 31 36  6 12 14 19 32 34 39 43 11 17 22 49 48
## [26]  3 27 38  8  9 10 40 41 46 18 20 37 44 45  2 21 23  5 15 35  4 25 33 24 47
```

- 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.

```
#TO-DO
```

- 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.

```
#TO-DO
```

- 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`.

```
?split
```

```
## starting httpd help server ... done
```

```
#TO-DO
```

- 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.

```
#TO-DO
```

- 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.

```
#TO-DO
```

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

```
#TO-DO
```

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

```
#TO-DO
```

- 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?

```
#TO-DO
```

- 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?

```
#TO-DO
```