Lab 1

Frank Palma Gomez

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) # exclusive this is 10
pi
```

```
## [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 $1/3, 1/6, 1/9 \ldots$

```
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)) # underflow
```

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

Is this answer *exactly* correct?

Not exactly correct because we reached the smallest possible number. We experienced numerical underflow.

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

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

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

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

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

```
##  [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))
x = log(x, base=3)
```

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

Comment on the appropriateness of the non-numeric values.

There is a -Inf because log(0) is -Inf. The proceeding are NaN because log is not defined for negative answers

- 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
```

- 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 == FALSE)
```

```
##  [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 52
```

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

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

```
## [1] 52
```

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

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

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

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

```
## [1] 53
```

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

```
as.factor(x)
```

```
##  [1] Inf               4.19180654857877  4.1734172518943   4.15464876785729
##  [5] 4.13548512895119  4.11590933734319  4.09590327428938  4.07544759935851
##  [9] 4.05452163806914  4.03310325630434  4.01116871959141  3.98869253500376
## [13] 3.96564727304425  3.94200336638929  3.91772888178973  3.89278926071437
## [17] 3.86714702345081  3.84076143030548  3.81358809221559  3.78557852142874
## [21] 3.75667961082847  3.72683302786084  3.69597450568212  3.66403300987579
## [25] 3.63092975357146  3.59657702661571  3.56087679500731  3.52371901428583
## [29] 3.48497958377173  3.44451784578705  3.40217350273288  3.3577627814323
## [33] 3.31107361281783  3.26185950714291  3.20983167673402  3.15464876785729
## [37] 3.09590327428938  3.03310325630434  2.96564727304425  2.89278926071437
## [41] 2.8135880922156   2.72683302786084  2.63092975357146  2.52371901428583
## [45] 2.40217350273288  2.26185950714291  2.09590327428938  1.89278926071437
## [49] 1.63092975357146  1.26185950714291  0.630929753571457 -Inf
## [53] NaN               NaN               NaN               NaN
## [57] NaN               NaN               NaN               NaN
## [61] NaN               NaN
```

```
## 53 Levels: -Inf 0.630929753571457 1.26185950714291 ... NaN
```

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

NaN got converted to NA

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

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

```
##  [1] NA  4  4  4  4  4  4  4  4  4  4  3  3  3  3  3  3  3  3  3  3  3  3  3
## [26]  3  3  3  3  3  3  3  3  3  3  3  3  2  2  2  2  2  2  2  2  2  1  1  1
## [51]  0 NA NA NA NA NA NA NA NA NA NA NA
```

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

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

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

- 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.

```
sample(c(0, 1), size=100, replace=TRUE)
```

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

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

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

```
##   [1] 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1
##  [75] 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1
## [112] 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 0
## [149] 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0
## [223] 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [260] 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [297] 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [334] 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 1 1 1 1 1 1 1 1
## [371] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1
## [408] 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 1 1 1
```

```
## [445] 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
## [482] 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

- 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] 0.909
```

- 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. unordered) factor.

```
x_3 = as.factor(sample(c("none", "infraction", "misdemeanor", "felony"), size=100, replace=TRUE))
```

- 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
```

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

- 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"), order=TRUE)
x_3_ord
```

```
##   [1] felony      felony      none        none        felony      misdemeanor
##   [7] felony      felony      misdemeanor misdemeanor felony      misdemeanor
##  [13] felony      misdemeanor felony      felony      felony      infraction
##  [19] infraction  misdemeanor infraction  misdemeanor none        felony
##  [25] misdemeanor misdemeanor infraction  felony      misdemeanor infraction
##  [31] misdemeanor infraction  felony      infraction  none        felony
##  [37] infraction  infraction  none        infraction  none        felony
##  [43] infraction  felony      infraction  felony      misdemeanor infraction
##  [49] felony      felony      none        infraction  infraction  infraction
##  [55] none        felony      infraction  infraction  infraction  misdemeanor
##  [61] misdemeanor misdemeanor misdemeanor felony      felony      none
##  [67] infraction  felony      infraction  felony      felony      felony
##  [73] infraction  misdemeanor none        infraction  infraction  none
##  [79] none        misdemeanor infraction  none        misdemeanor none
##  [85] none        none        misdemeanor misdemeanor none        misdemeanor
##  [91] misdemeanor infraction  infraction  infraction  none        infraction
##  [97] none        none        infraction  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 == "infraction")
x_3_misdimeanor = as.integer(x_3 == "misdimeanor")
```

```
x_3_felony = as.integer(x_3 == "felony")

X = cbind(x_3_infraction, x_3_misdimeanor, x_3_felony) # concatenate column wise

cols = levels(x_3)[1:3] # get the first three columns
X = matrix(X, nrow=length(x_3), ncol=length(cols)) # convert into matrix
colnames(X) = cols # set column names

head(X) # print
```

```
##      felony infraction misdemeanor
## [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 sum crimes an observation has. It should result in a vector.

Verify that.

```
row_counts = rowSums(X, dims=1)
row_counts
```

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

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

It should be a vector with the size of the number of columns

Verify that.

```
col_counts = colSums(X, dims=1)
col_counts
```

```
##      felony  infraction misdemeanor
##          30           0          27
```

- 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"
)
norm = rnorm(n=100, mean=17, sd=sqrt(38))
unif = runif(n=100, min=-10, max=10)
pois = rpois(n=100, lambda=6)
exp = rexp(n=100, rate=9)
binom = rbinom(n=20, size=1, p=0.12)
rand = rbinom(n=100, size=1, p=0.24)
cols = c('norm', 'unif', 'pois', 'exp', 'binom', 'binary')

X = cbind(norm, unif, pois, exp, binom, rand)
X = matrix(X, nrow=100, ncol=6)
rownames(X) = fake_first_names
colnames(X) = cols
head(X)
```

```
##                  norm           unif pois             exp binom binary
## Sophia   20.580742010  4.42339935806    8 0.0224850507867     0      0
## Emma     15.673824956 -0.32417375594    8 0.0618617391317     0      0
## Olivia   20.272108289 -6.54996076133    4 0.0369593734439     0      1
## Ava      14.714160775  2.68836348318    3 0.0919946807700     0      0
## Mia      21.003815417 -8.07490918785    8 0.1427429807702     0      0
## Isabella 22.800037966  8.09769294690    9 0.0040534077659     0      0
```

- 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.

```
X = data.frame(
  norm = X[, "norm"],
  unif = X[, "unif"],
  pois = X[, "pois"],
  exp = X[, "exp"],
  binom = X[, "binom"],
  binary = X[, "binary"]
)
X$binary = factor(X$binary, labels=c("DOMESTIC", "FOREIGN"))
```

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

```
table(X$binary)
```

```
##
## DOMESTIC  FOREIGN
##       76       24
```

Print out a summary of the whole dataframe.

```
summary(X)
```

```
##       norm                unif                 pois
##  Min.   : 6.5042103   Min.   :-9.79503158   Min.   : 1.00
```

```
##  1st Qu.:12.9480580   1st Qu.:-5.23529199   1st Qu.: 4.00
##  Median :16.6386898   Median :-0.24723901   Median : 6.00
##  Mean   :17.2579218   Mean   :-0.54746954   Mean   : 5.81
##  3rd Qu.:21.4884514   3rd Qu.: 3.96955856   3rd Qu.: 7.00
##  Max.   :33.6345558   Max.   : 9.56398258   Max.   :13.00
##       exp                    binom        binary
##  Min.   :0.00004730085   Min.   :0   DOMESTIC:76
##  1st Qu.:0.03038306949   1st Qu.:0   FOREIGN :24
##  Median :0.06856002064   Median :0
##  Mean   :0.11191047990   Mean   :0
##  3rd Qu.:0.14840337885   3rd Qu.:0
##  Max.   :0.59275266711   Max.   :0
```

- 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(0:2, size=n * n, prob=c(0.5, 0.25, 0.25), replace=TRUE), nrow=n, ncol=n)
head(R)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,]    1    1    1    0    2    0    0    0    1     1     0     0     0     0
## [2,]    2    0    1    1    1    0    2    0    0     1     1     0     0     0
## [3,]    1    1    0    0    2    0    0    0    1     2     0     1     1     0
## [4,]    0    1    1    0    2    1    1    0    1     1     0     2     0     2
## [5,]    0    0    2    1    0    0    0    0    2     0     2     0     1     1
## [6,]    0    1    2    0    0    1    0    0    0     2     0     0     0     1
##      [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
## [1,]     1     0     0     0     0     0     0     1     0     1     2     0
## [2,]     1     2     0     0     2     1     0     2     0     2     0     0
## [3,]     1     0     2     0     0     1     0     1     2     0     0     0
## [4,]     0     1     1     2     1     0     0     2     0     0     0     0
## [5,]     2     0     1     0     0     0     2     1     1     0     2     2
## [6,]     1     0     0     0     2     1     2     0     0     0     1     2
##      [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38]
## [1,]     0     0     0     0     0     1     2     2     2     0     0     2
## [2,]     0     1     2     2     0     1     1     0     1     0     0     2
## [3,]     2     0     0     0     1     0     2     0     1     0     1     2
## [4,]     0     0     2     0     1     0     0     2     1     0     0     0
## [5,]     1     0     0     0     1     2     0     0     1     0     0     0
## [6,]     2     2     2     0     0     1     1     0     2     2     2     0
##      [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49] [,50]
## [1,]     1     1     0     2     0     0     0     2     0     1     0     1
## [2,]     1     0     2     1     1     0     0     1     1     0     2     0
## [3,]     1     1     1     1     0     0     2     2     0     1     0     0
## [4,]     0     0     2     2     0     1     2     0     0     1     2     1
## [5,]     0     0     1     0     0     0     0     0     2     0     0     0
## [6,]     1     0     1     2     0     0     0     0     0     0     1     1
```

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

```
R = matrix(lapply(R, function(x) x[sample(c(TRUE, NA), size=length(x), prob=c(0.70, 0.30), replace=TRUE
head(R)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,] NA    1    1    0    2    0    0    0    1    1     NA    0     NA    0
```

```
## [2,] 2    0    1    1    NA   NA   2    0    0    NA    NA    NA    0     0
## [3,] 1    NA   NA   0    2    0    0    0    NA   NA    0     1     1     NA
## [4,] NA   1    1    0    2    1    1    NA   1    NA    0     NA    NA    2
## [5,] NA   0    2    1    NA   0    0    0    2    0     NA    0     NA    1
## [6,] 0    1    NA   0    0    NA   0    NA   0    NA    0     0     0     1
##      [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
## [1,] 1     0     0     NA    NA    NA    0     1     0     1     2     0
## [2,] NA    2     NA    0     2     1     0     2     0     2     NA    0
## [3,] NA    NA    2     0     0     NA    0     1     NA    0     0     0
## [4,] 0     1     1     2     1     NA    NA    NA    0     NA    0     0
## [5,] NA    NA    1     0     0     0     2     1     NA    NA    NA    2
## [6,] 1     0     0     0     2     1     2     0     NA    0     1     2
##      [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38]
## [1,] 0     NA    0     NA    0     1     2     2     NA    0     0     2
## [2,] 0     NA    NA    2     0     1     1     0     1     0     0     2
## [3,] 2     0     0     0     1     0     2     0     1     0     NA    NA
## [4,] NA    0     NA    0     NA    0     0     NA    1     0     NA    NA
## [5,] 1     0     0     0     1     2     NA    0     1     0     0     0
## [6,] NA    2     NA    0     0     1     NA    0     2     2     2     0
##      [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49] [,50]
## [1,] 1     1     NA    2     0     0     0     NA    0     1     0     1
## [2,] 1     NA    2     1     NA    0     0     NA    1     0     NA    NA
## [3,] 1     1     1     1     0     0     2     2     NA    1     NA    NA
## [4,] NA    0     2     2     NA    NA    NA    0     0     NA    2     1
## [5,] 0     NA    NA    0     0     0     0     NA    2     0     0     NA
## [6,] 1     0     1     2     0     NA    0     0     0     NA    1     1
```

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

```
#sd_fun = function(x) sd(na.exclude(x))
#R_row_sd = apply(R, 2, sd_fun) # sd for rows
#R_col_sd = apply(R, 2, sd_fun)
```

- 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.
- 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
```

- 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.

```
set.seed(2000)
v = rnorm(1000, mean=-10, sd=sqrt(100))
```

```
head(v)
```

```
## [1] -18.5384323568 -13.5288939552  -1.0985047234   7.4128671643  -0.8392165958
## [6] -21.7660142823
```

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

```
set.seed(2000)
v = rnorm(1000, mean=-10, sd=sqrt(100))
head(v)
```

```
## [1] -18.5384323568 -13.5288939552  -1.0985047234   7.4128671643  -0.8392165958
## [6] -21.7660142823
```

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

```
mean(v)
```

```
## [1] -9.4585200415
```

```
sd(v) / sqrt(length(v))
```

```
## [1] 0.31881861326
```

- 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?

```
quantile(v, 0.05)
```

```
##           5%
## -26.780618034
```

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

```
## [1] -26.44853627
```

- 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?

```
inverse_quant_obj = ecdf(v)
inverse_quant_obj(0)
```

```
## [1] 0.821
```