STAT 3675Q Homework 2

Due date: **Thursday, September 11, at noon**

Zeshi Feng

Note:

- Ensure that your code is fully visible in the PDF and not cropped. If needed, break the code into multiple lines to fit.
- It is recommended to write descriptive answers outside of R code chunks (i.e., as text in the main body), while comments within the code chunks can be reserved for brief code annotations.
- In all homework questions, include a written explanation of any output to earn full credit.

Question 1 [25 points]

Consider the dataset `county_2019` in the R package `usdata`. To read the documentation of the dataset, run `?county_2019`.

a. Load the dataset into the working environment as an object. Use an R function to verify that it is a data frame.

Answer:

```
library(usdata)
data("county_2019")
class(county_2019)
```

```
## [1] "data.frame"
```

“ “ “library() loads the package; data() loads the dataset into your working environment; class() loads the class” “ ”

b. Use an R function to check the numbers of observations and variables.

Answer:

```
dim(county_2019)
```

```
## [1] 3142   95
```

“ “ “use dim() function which returns a vector of two numbers: rows, columns.” “ ”

c. Visually check the data frame. What data types does it include? Are there missing values?

Answer:

```
str(county_2019)
```

```
## 'data.frame':    3142 obs. of  95 variables:
##  $ state                                    : chr  "Alabama" "Alabama" "Alabama" "Ala
##  $ name                                     : chr  "Autauga County" "Baldwin County"
##  $ fips                                     : int  1001 1003 1005 1007 1009 1011 1013
##  $ age_over_18                              : num  76.2 78.3 79.1 79.4 76.8 79.2 77.5
##  $ age_over_18_moe                          : num  0.1 NA 0.1 0.2 0.1 NA 0.5 0.1 0.2
##  $ age_over_65                              : num  15 20 18.6 15.9 17.9 16 19.7 17.2
##  $ age_over_65_moe                          : num  0.2 0.1 0.1 0.3 0.2 NA 0.2 0.1 0.2
##  $ age_over_85                              : num  1.6 1.9 1.6 2 1.8 1.7 2.6 1.8 1.9
##  $ age_over_85_moe                          : num  0.3 0.3 0.4 0.7 0.3 0.7 0.7 0.2 0.
##  $ age_under_5                              : num  5.8 5.5 5.3 5.8 5.9 5.3 5.8 5.8 6.
##  $ age_under_5_moe                          : num  0.2 0.1 0.1 0.8 0.1 0.1 0.2 0.1 0.
##  $ asian                                    : num  1 0.9 0.5 0.1 0.4 0.5 0.3 0.9 1.1
##  $ asian_moe                                : num  0.3 0.1 0.1 0.2 0.1 0.6 0.3 0.1 0.
##  $ avg_family_size                          : num  3.09 3.24 3.01 3.74 3.33 3.3 3.78
##  $ avg_family_size_moe                      : num  0.07 0.05 0.12 0.24 0.08 0.37 0.21
##  $ bachelors                                : num  26.6 31.9 11.6 10.4 13.1 12.1 16.1
##  $ bachelors_moe                            : num  2 1.2 1.4 2.2 1.3 2.8 2.1 1.1 1.7
##  $ black                                    : num  19 9.3 47.6 22.3 1.6 74.8 45.4 20.
##  $ black_moe                                : num  0.5 0.2 0.6 0.5 0.3 3.2 0.8 0.4 0.
##  $ hispanic                                 : num  2.8 4.6 4.4 2.6 9.3 2.6 1.4 3.8 2.
##  $ hispanic_moe                             : num  NA NA NA NA NA 3.1 NA NA NA NA ...
##  $ household_has_broadband                  : num  80.6 81.8 60.5 69.2 73 60.1 64.6 7
##  $ household_has_broadband_moe              : num  1.9 1.2 2.6 4.3 2.5 6.8 3.4 1.5 2.
##  $ household_has_computer                   : num  73 76.3 51.9 54.7 63.5 52.5 60.5 6
##  $ household_has_computer_moe               : num  2.1 1.5 3 5 2.3 8.1 3.6 1.6 2.8 3.
##  $ household_has_smartphone                 : num  78.4 81.7 64.2 66.6 70.1 70.6 57.1
##  $ household_has_smartphone_moe             : num  1.8 1.1 2.8 4.2 2.5 5.2 3.8 1.4 2.
##  $ households                               : num  21397 80930 9345 6891 20847 ...
##  $ households_moe                           : num  325 1127 313 333 394 ...
##  $ households_speak_asian_or_pac_isl        : num  1.8 0.6 0.6 0 0.1 0.4 0.2 0.8 1 0.
##  $ households_speak_asian_or_pac_isl_moe    : num  0.5 0.2 0.4 0.5 0.1 0.6 0.2 0.3 0.
##  $ households_speak_limited_english         : num  0.7 1.2 1.6 0.6 1.8 1.4 0.3 0.9 0.
##  $ households_speak_limited_english_moe     : num  0.5 0.4 0.8 0.7 0.7 1.6 0.6 0.4 0.
##  $ households_speak_other                   : num  0.2 0 0 0 0.2 0 0 0.1 0 0.1 ...
##  $ households_speak_other_moe               : num  0.3 0.1 0.1 0.5 0.2 0.9 0.5 0.1 0.
##  $ households_speak_other_indo_euro_lang    : num  0.3 1.8 1.1 0.5 0.9 1.6 0.6 0.8 0.
```

```
##  $ households_speak_other_indo_euro_lang_moe: num  0.2 0.3 0.5 0.5 0.5 1.3 0.4 0.2 0.
##  $ households_speak_spanish                 : num  2.9 4.6 5.2 1.9 6.6 3.6 1.4 3.1 1.
##  $ households_speak_spanish_moe             : num  0.8 0.6 1.3 1.3 0.7 3 0.9 0.5 0.4
##  $ housing_mobile_homes                     : num  26.7 24.8 39.1 25.6 21.2 28.9 30 2
##  $ housing_mobile_homes_moe                 : num  2.2 1.1 2.6 3.7 2 6.9 2.9 1.3 2.3
##  $ housing_one_unit_structures              : num  17.3 11.5 26.1 29.7 24 39.3 24.9 1
##  $ housing_one_unit_structures_moe          : num  1.9 0.8 2.4 3.9 2 6 2.6 1 1.8 2.9
##  $ housing_two_unit_structures              : num  73.3 75.2 60.9 74.4 78.8 71.1 70 7
##  $ housing_two_unit_structures_moe          : num  2.2 1.1 2.6 3.7 2 6.9 2.9 1.3 2.3
##  $ hs_grad                                  : num  88.5 90.8 73.2 79.1 80.5 74.7 85 8
##  $ hs_grad_moe                              : num  1.4 0.7 1.8 3.2 1.6 4.9 1.9 1 1.8
##  $ mean_household_income                    : num  75326 80986 47068 60182 65639 ...
##  $ mean_household_income_moe                : num  6004 1930 2424 5709 3912 ...
##  $ mean_work_travel                         : num  24.4 NA NA NA NA 27.2 23.7 NA 23.7
##  $ mean_work_travel_moe                     : num  1.3 NA NA NA NA 3.5 1.5 NA 1.3 NA
##  $ median_age                               : num  38.2 43 40.4 40.9 40.7 40.2 40.8 3
##  $ median_age_moe                           : num  0.6 0.3 0.5 1.3 0.3 2.3 0.7 0.3 0.
##  $ median_household_income                  : num  58731 58320 32525 47542 49358 ...
##  $ median_household_income_moe              : num  4410 1564 2291 5504 2136 ...
##  $ median_individual_income                 : num  29725 29802 17963 21958 26976 ...
##  $ median_individual_income_moe             : num  1643 905 1198 1400 969 ...
##  $ median_individual_income_age_25plus      : num  40778 37897 27434 28789 39004 ...
##  $ median_individual_income_age_25plus_moe  : num  1343 1119 2812 3605 1955 ...
##  $ native                                   : num  0.3 0.8 0.3 0.1 0.1 0 0.1 0.3 0.3
##  $ native_moe                               : num  0.1 0.2 0.2 0.3 0.1 0.3 0.1 0.2 0.
##  $ other_single_race                        : num  0.7 1.1 3.6 0 0.9 2 0.2 1.9 0.3 0.
##  $ other_single_race_moe                    : num  0.6 0.3 0.7 0.1 0.6 3 0.3 0.4 0.3
##  $ pac_isl                                  : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ pac_isl_moe                              : num  0.1 0.1 0.1 0.1 0.1 0.3 0.2 0.1 0.
##  $ per_capita_income                        : num  29819 32626 18473 20778 24747 ...
##  $ per_capita_income_moe                    : num  2345 758 942 1703 1318 ...
##  $ persons_per_household                    : num  2.56 2.59 2.41 2.99 2.74 2.79 3 2.
##  $ persons_per_household_moe                : num  0.04 0.04 0.07 0.14 0.05 0.2 0.13
##  $ pop                                      : num  55380 212830 25361 22493 57681 ...
##  $ pop_moe                                  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ poverty                                  : num  15.2 10.4 30.7 NA 13.6 NA NA 17.9
##  $ poverty_moe                              : num  1.8 0.9 2.4 NA 1.7 NA NA 1.3 2.2 N
##  $ poverty_65_and_over                      : num  8.7 7.4 16.8 NA 10.9 NA NA 9.7 11.
##  $ poverty_65_and_over_moe                  : num  2.1 1.4 3.1 NA 2.8 NA NA 1.5 2.4 N
##  $ poverty_under_18                         : num  23.2 13.4 50.1 NA 18.4 NA NA 25.6
##  $ poverty_under_18_moe                     : num  4 2 4.9 NA 3.8 NA NA 2.5 5.6 NA ..
##  $ two_plus_races                           : num  2.2 1.7 1.2 0.6 1.6 0.8 2.1 2.4 1
##  $ two_plus_races_moe                       : num  0.6 0.3 0.5 0.4 0.3 1 0.7 0.4 0.6
##  $ unemployment_rate                        : num  3.5 4 9.4 7 3.1 4.1 7 7.2 4 4.4 ..
##  $ unemployment_rate_moe                    : num  1 0.6 1.9 2.9 0.9 3.4 1.9 0.8 1.2
```

```
##  $ uninsured                                : num  7.1 8.9 11.3 10.7 10.8 11.4 8.7 9.
##  $ uninsured_moe                            : num  1 0.7 1.5 2.2 1.4 4.2 1.5 0.8 1.7
##  $ uninsured_65_and_older                   : num  0 0.3 0.3 0 0.2 0 0 0.2 0.4 0.2 ..
##  $ uninsured_65_and_older_moe               : num  0.4 0.3 0.3 0.9 0.2 2.1 0.9 0.2 0.
##  $ uninsured_under_19                       : num  1.7 3.8 3.3 2 5.9 1 2.3 2.4 5.4 0.
##  $ uninsured_under_19_moe                   : num  0.9 1.1 2 1.7 2.5 1.2 1.4 0.8 3.4
##  $ uninsured_under_6                        : num  1.7 2.2 3.4 4.5 6.1 2.1 0.5 1 8.1
##  $ uninsured_under_6_moe                    : num  2.3 1.3 2.4 4.4 3.2 3.6 0.8 0.8 9.
##  $ veterans                                 : num  12.6 11.8 6.6 8 7.7 3.3 6.6 10.8 7
##  $ veterans_moe                             : num  1.3 0.6 0.8 1.4 0.8 1.4 1.1 0.7 1.
##  $ white                                    : num  76.8 86.2 46.8 76.8 95.5 21.9 51.8
##  $ white_moe                                : num  0.5 0.4 0.6 0.3 0.6 0.5 0.3 0.5 0.
##  $ white_not_hispanic                       : num  74.6 83.1 45.8 74.5 86.9 21.4 51.6
##  $ white_not_hispanic_moe                   : num  0.1 0.1 0.1 0.1 0.5 0.3 0.2 0.2 0.
```

```r
anyNA(county_2019)
```

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

“ “ “We use str() function to tells you the data types of each variable; Function anyNA() gives out wheather or not the data set contains NaN” “ ”

Question 2 [25 points]

a. Generate the following sequence using `seq()` with the `by` argument: 2 5 8 11 14 17. Create the vector and print it out using two separate commands.

Answer:

```r
sequence_in_partA <- seq(from = 2, to = 17, by = 3)
sequence_in_partA
```

```
## [1]  2  5  8 11 14 17
```

“ “ “from, to :the starting and (maximal) end values of the sequence. Of length 1 unless just from is supplied as an unnamed argument; by:number: increment of the sequence.”

b. Generate a sequence of 15 numbers that are equally spaced between -1 and 1 without defining the increment. Hint: Read the documentation of the `seq` function. Create the vector and print it out using a single-line command.

Answer:

```r
sequence_in_partB <- seq(from = -1, to = 1, length = 15)
sequence_in_partB
```

```
##  [1] -1.0000000 -0.8571429 -0.7142857 -0.5714286 -0.4285714 -0.2857143
##  [7] -0.1428571  0.0000000  0.1428571  0.2857143  0.4285714  0.5714286
## [13]  0.7142857  0.8571429  1.0000000
```

“ “ “same as last one, length.out: desired length of the sequence. A non-negative number, which for seq and seq.int will be rounded up if fractional.” “ ”

c. Print out the following sequences using `rep()` and `c()`, without creating any object in the environment:

i). 5 12 13 5 12 13 5 12 13

ii). 5 5 5 5 5 12 12 12

Answer:

```
#i)
rep(c(5, 12, 13), times = 3)
```

```
## [1]  5 12 13  5 12 13  5 12 13
```

```
#ii)
c(rep(5, 5), rep(12, 3))
```

```
## [1]  5  5  5  5  5 12 12 12
```

“ “ “nothing new, just tricks of function rep() and c(); BTW c() is a generic function which combines its arguments.” “ ”

d. Create a vector containing the 1st, 3rd, 5th, 7th, 9th and 11th elements of the sequence in part b. Print it out.

Answer:

```
sequence_in_partB[c(1, 3, 5, 7, 9, 11)]
```

```
## [1] -1.0000000 -0.7142857 -0.4285714 -0.1428571  0.1428571  0.4285714
```

“ “ “Using the combined value as position of the array.” “ ”

e. What happens when a vector is multiplied by a number, a vector of the same length, or a vector of a different length? Use the vector from part a as an example. Try multiplying it by 2, `rep(2,6)`, and `c(1,2)`, respectively, using the operator `*`. Explain the results you get.

Answer:

```
#1)
sequence_in_partA * 2
```

```
## [1]  4 10 16 22 28 34
```

```
#2)
sequence_in_partA * rep(2,6)
```

```
## [1]  4 10 16 22 28 34
```

```
#3)
sequence_in_partA * c(1, 2)
```

```
## [1]  2 10  8 22 14 34
```

“ “ “a) scalar multiplication; b) element-wise multiplication; c) repeats the shorter vector until lengths match” “ ”

Question 3 [25 points]

a. Reconsider the vector in Question 2b. Convert it to a 3×5 matrix, filling the entries by column.

Answer:

```
my_matrix_a <- matrix(sequence_in_partB, nrow = 3, ncol = 5)
my_matrix_a
```

```
##            [,1]       [,2]       [,3]      [,4]      [,5]
## [1,] -1.0000000 -0.5714286 -0.1428571 0.2857143 0.7142857
## [2,] -0.8571429 -0.4285714  0.0000000 0.4285714 0.8571429
## [3,] -0.7142857 -0.2857143  0.1428571 0.5714286 1.0000000
```

“ “ “matrix(x, nrow, ncol) reshapes a vector into a matrix. By default, R fills by column unless you add byrow = TRUE” ”

b. Redo part a, filling the entries by row.

Answer:

```
my_matrix_b <- matrix(sequence_in_partB, nrow = 3, ncol = 5, byrow = TRUE)
my_matrix_b
```

```
##            [,1]       [,2]       [,3]       [,4]       [,5]
## [1,] -1.0000000 -0.8571429 -0.7142857 -0.5714286 -0.4285714
## [2,] -0.2857143 -0.1428571  0.0000000  0.1428571  0.2857143
## [3,]  0.4285714  0.5714286  0.7142857  0.8571429  1.0000000
```

c. Explain why it is unnecessary to specify both `nrow` and `ncol` for parts a and b. Then, try the following command: `matrix(c(1,2,3), nrow=5, ncol=6)`, and explain why in this case both `nrow` and `ncol` are needed.

Answer: “ “ “If vector length is a perfect multiple of rows or columns, you only need one of nrow/ncol. If recycling is required, R cannot infer the shape automatically, so you must provide both.” “ ”

d. Extract the odd-numbered columns from the matrix created in part a.

Answer:

```
my_matrix_a[ , c(1, 3, 5)]
```

```
##            [,1]       [,2]      [,3]
## [1,] -1.0000000 -0.1428571 0.7142857
## [2,] -0.8571429  0.0000000 0.8571429
## [3,] -0.7142857  0.1428571 1.0000000
```

“ “ “ “ “The first blank means “all rows, c(1,3,5) means column” ”

Question 4 [25 points]

a. Explore the `set.seed()` function in R. Explain the difference between the command

```
rnorm(10)
```

```
##  [1]  0.19220431 -0.06815762  0.18301112 -0.46762846  1.25786434 -0.31454252
##  [7] -1.10967133 -0.03926590  0.41498141 -2.06525339
```

and the commands

```
set.seed(1)
rnorm(10)
```

```
##  [1] -0.6264538  0.1836433 -0.8356286  1.5952808  0.3295078 -0.8204684
##  [7]  0.4874291  0.7383247  0.5757814 -0.3053884
```

in terms of reproducibility.

Answer: “ “ “set.seed(k) ensures reproducibility: same seed means same “random” results; no seed means different results” “ ”

b. Set the random seed to 1. Create a 5×3 matrix, with the ith column filled by randomly generated numbers from the student t distribution with i degrees of freedom, for $i = 1, 2, 3$. The row names of the matrix are “1”, . . . , “5”, and the column names are “t1”, “t2”, “t3”. Print out the matrix.

Answer:

```
set.seed(1)
col1 <- rt(5, df = 1)
col2 <- rt(5, df = 2)
col3 <- rt(5, df = 3)
my_mat <- cbind(col1, col2, col3)
rownames(my_mat) <- as.character(1:5)
colnames(my_mat) <- c("t1", "t2", "t3")
my_mat
```

```
##           t1          t2         t3
## 1 -0.5947235  0.25139991 -0.3962365
## 2 -0.3536704  1.05070872  0.4879252
```

```
## 3 11.4386179 -1.95606023  0.9429963
## 4 -0.4521079 -0.08920524 -0.3248886
## 5  0.8089471 -1.75859322 -0.1820845
```

“ “ “rt(n, df) generates n random numbers from a Student-t distribution with df degrees of freedom. cbind() binds columns together.” “ ”

c. Convert the matrix to a data frame.

Answer:

```
my_df <- as.data.frame(my_mat)
my_df
```

```
##           t1          t2         t3
## 1 -0.5947235  0.25139991 -0.3962365
## 2 -0.3536704  1.05070872  0.4879252
## 3 11.4386179 -1.95606023  0.9429963
## 4 -0.4521079 -0.08920524 -0.3248886
## 5  0.8089471 -1.75859322 -0.1820845
```

“ “ “as.data.frame() takes the matrix and converts it into a data frame” “ ”