STA 032 Homework 1

CHANGE YOUR NAME HERE

DUE: April 14 2023, 12PM

Instructions

- Upload a PDF file, named with your UC Davis email ID and homework number (e.g., `xjw18_hw1.pdf`), to Gradescope (accessible through Canvas). You will give the commands to answer each question in its own code block, which will also produce output that will be automatically embedded in the output file. When asked, answer must be supported by written statements as well as any code used.
- All code used to produce your results must be shown in your PDF file (e.g., do not use `echo = FALSE` or `include = FALSE` as options anywhere). `Rmd` files do not need to be submitted, but may be requested by the TA and must be available when the assignment is submitted.
- Students may choose to collaborate with each other on the homework, but must clearly indicate with whom they collaborated. Every student must upload their own submission.
- Start to work on it as early as possible

Level 1 Parts

Problem 1: R basics

1. What is the sum of the first 100 positive integers? The formula for the sum of integers 1 through n is $n(n+1)/2$. Define $n = 100$ and then use R to compute the sum of 1 through 100 using the formula.

2. Now use the same formula to compute the sum of the integers from 1 through 1,000.

3. Look at the result of typing the following code into R:

```
n <- 1000
x <- seq(1, n)
sum(x)
```

Based on the result, what do you think the functions `seq` and `sum` do? You can use `help`.

a. `sum` creates a list of numbers and `seq` adds them up.

b. `seq` creates a list of numbers and `sum` adds them up.

c. `seq` creates a random list and `sum` computes the sum of 1 through 1,000.

d. `sum` always returns the same number.

Solution: Type your solution here

4. Which of the following will always return the numeric value stored in `x`? You can try out examples and use the help system if you want.

a. `log(10^x)`
b. `log10(x^10)`
c. `log(exp(x))`
d. `exp(log(x, base = 2))`

5.Another way to create vectors is with the `rep()` function. The `rep()` function creates a vector by replicating a value or vector of values. Write comments inside those code chunks to explain what happens by running these code.

The first argument of `rep()` is the thing to replicate. The argument `times` is the number of times to replicate. The argument `each` means each element of input is repeated each times.

```
rep(10, times = 5)
```

```
[1] 10 10 10 10 10
```

```
#
rep(c("apple","banana","cream"), times = 3)
```

```
[1] "apple"  "banana" "cream"  "apple"  "banana" "cream"  "apple"  "banana"
[9] "cream"
```

```
#
rep(c("apple","banana","cream"), each = 3)
```

```
[1] "apple"  "apple"  "apple"  "banana" "banana" "banana" "cream"  "cream"
[9] "cream"
```

```
#
rep(c(1,2,3),times =  c(1,2,3))
```

```
[1] 1 2 2 3 3 3
```

```
#
```

Problem 2: Vectors

1. Use the function `c` to create a vector with the average high temperatures in January for Beijing, Lagos, Paris, Rio de Janeiro, San Juan, and Toronto, which are 35, 88, 42, 84, 81, and 30 degrees Fahrenheit. Call the object `temp`.

2. Now create a vector with the city names and call the object `city`.

3. Use the `names` function and the objects defined in the previous exercises to associate the temperature data with its corresponding city. (Hint: `names(vector) = name`)

4. Use the `[` and `:` operators to access the temperature of the first three cities on the vector

5. Use the `[` operator to access the temperature of Paris and San Juan. (Hint: subset by name)

6. Use the `:` operator to create a sequence of numbers $12, 13, 14, \ldots, 73$.

7. Create a vector containing all the positive odd numbers smaller than 100.

8. Create a vector of numbers that starts at 6, does not pass 55, and adds numbers in increments of 4/7: 6, 6 + 4/7, 6 + 8/7, and so on. How many numbers does the list have? Hint: use `seq` and `length`.

Problem 3: Dataframes, vectors, lists

1. Load the US murders dataset.

```
# If you have not install the package, use the commented command to install it.
#install.packages(dslabs)
library(dslabs)
data(murders)
```

Use the function `str` to examine the structure of the `murders` object. Which of the following best describes the variables represented in this data frame?

a. The 51 states.
b. The murder rates for all 50 states and DC.
c. The state name, the abbreviation of the state name, the state's region, and the state's population and total number of murders for 2010.
d. `str` shows no relevant information.

Solution: Type your solution here

2. What are the column names used by the data frame for these five variables?

3. Use the accessor `$` to extract the state abbreviations and assign them to the object `a`. What is the class of this object?

4. Now use the square brackets to extract the state abbreviations and assign them to the object `b`. (Hint: `dataset["Column name"]`)

Use the `identical(a, b)` function to determine if `a` and `b` are the same.

5. We saw that the `region` column stores a factor. You can corroborate this by typing:

```
class(murders$region)
```

With one line of code, use the function `levels` and `length` to determine the number of regions defined by this dataset.

6. The function `table` takes a vector and returns the frequency of each element. You can quickly see how many states are in each region by applying this function. Use this function in one line of code to create a table of states per region.

Problem 4: vector arithmetics and functions

1. Previously we created this data frame:

```
temp <- c(35, 88, 42, 84, 81, 30)
city <- c("Beijing", "Lagos", "Paris", "Rio de Janeiro",
          "San Juan", "Toronto")
city_temps <- data.frame(name = city, temperature = temp)
```

Remake the data frame using the code above, but add a line that converts the temperature from Fahrenheit to Celsius. The conversion formula is $C = \frac{5}{9} \times (F - 32)$.

2. What is the following sum to $n = 100$ terms of the series? Calculate using vector arithmetics.

$$\frac{2n-1}{n(n+1)(n+2)}$$

Hint: The theoretical result should be:

$$\sum_{k=1}^{n} \frac{2k-1}{k(k+1)(k+2)} = \frac{1}{4}(3 - \frac{10}{n+2} + \frac{2}{n+1})$$

Problem 5: Dataframe and vector manipulation

Start by loading the library and data.

```
library(dslabs)
data(murders)
# This will give you the column names of the murders.
colnames(murders)
```

```
[1] "state"      "abb"        "region"     "population" "total"
```

The total murder for each state is `murders$total`, and the total population for each state is `murders$population`. per 100,000 murder rate is defined as `total/population * 100000`.

1. Compute the per 100,000 murder rate for each state and store it in an object called `murder_rate`. Then use logical operators to create a logical vector named `low` that tells us which entries of `murder_rate` are lower than 1.

2. Now use the results from the previous exercise and the function `which` to determine the indices of `murder_rate` associated with values lower than 1.

3. Use the results from the previous exercise to report the names of the states with murder rates lower than 1. (Hint: subset based on the index in 2)

Level 2 parts

Problem 5: Dataframe and vector manipulation continue

4. Now extend the code from exercises 2 and 3 to report the states in the Northeast with murder rates lower than 1. Hint: 1. use the previously defined logical vector `low` and the logical operator `&`. 2. Define a new logical vector to determine the indices of `murder_rate` that `region` is exactly `"Northeast"` using `==` operator.

5. In a previous exercise we computed the murder rate for each state and the average of these numbers. How many states are below the average?

6. Use the `%in%` operator to create a logical vector that answers the question: which of the following are actual abbreviations: MA, ME, MI, MO, MU?

7. Extend the code you used in exercise 6 to report the one entry that is **not** an actual abbreviation. Hint: use the `!` operator, which turns `FALSE` into `TRUE` and vice versa.

Problem 6

1. Create an empty list with 4 elements. Fill in the first element with the numbers 1 through 10, the second element with the letters “a” through “j”, the third element with a data frame with two columns and two rows, and the fourth element with a list containing a numeric vector and a character vector.

2. Here is an example of `paste()` function in R:

```
paste("I love", "Data science", sep = " ")
```

```
[1] "I love Data science"
```

Where the first 2 arguments are strings (If one argument is not string, it will be first converted into string automatically), the `sep = " "` indicates we concatenate the two strings using a space.

Now create vector: `names` with student names “Andy”, “Bob”, “Catherine”, vector: `scores` with score 80, 90, 85. Use `paste()` function to output: “Andy STA32 score is 80”, “Bob STA32 score is 90”, “Catherine STA32 score is 85”.

(Hint: What should the `sep=?` be?)

Problem 7

For these exercises we will use the US murders dataset. Make sure you load it prior to starting.

```
library(dslabs)
data("murders")
```

For this problem you may need to use `?sort`, `?order`, `?ranks` or Google the usage of these functions. They are not covered in lecture note. But it’s common to use new functions, so start learning how to know using new functions by your own is crucial.

1. Use the `$` operator to access the population size data and store it as the object `pop`. Then use the `sort` function to redefine `pop` so that it is sorted. Finally, use the `[` operator to report the smallest population size.

2. Now instead of the smallest population size, find the index of the entry with the smallest population size. Hint: use `order` instead of `sort`.

3. We can actually perform the same operation as in the previous exercise using the function `which.min`. Write one line of code that does this.

4. Now we know how small the smallest state is and we know which row represents it. Which state is it? Define a variable `states` to be the state names from the `murders` data frame. Report the name of the state with the smallest population.

5. You can create a data frame using the `data.frame` function. Here is a quick example:

```
temp <- c(35, 88, 42, 84, 81, 30)
city <- c("Beijing", "Lagos", "Paris", "Rio de Janeiro",
          "San Juan", "Toronto")
city_temps <- data.frame(name = city, temperature = temp)
```

Use the `rank` function to determine the population rank of each state from smallest population size to biggest. Save these ranks in an object called `ranks`, then create a data frame with the state name and its rank. Call the data frame `my_df`.

6. Repeat the previous exercise, but this time order `my_df` so that the states are ordered from least populous to most populous. Hint: create an object `ind` that stores the indexes needed to order the population values. Then use the bracket operator `[` to re-order each column in the data frame.

7. The `na_example` vector represents a series of counts. You can quickly examine the object using:

```
data("na_example")
str(na_example)
```

```
 int [1:1000] 2 1 3 2 1 3 1 4 3 2 ...
```

However, when we compute the average with the function `mean`, we obtain an `NA`:

```
mean(na_example)
```

```
[1] NA
```

The `is.na` function returns a logical vector that tells us which entries are `NA`. Assign this logical vector to an object called `ind` and determine how many `NA`s does `na_example` have. (Hint: use `sum` on the logical object)

8. Now compute the average again, but only for the entries that are not `NA`. Hint: the `!` operator, try `!(c(TRUE, FALSE, TRUE))`.

Include the person you coop with:

Names:

Appendix

```
sessionInfo()
```

```
R version 4.2.3 (2023-03-15 ucrt)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 10 x64 (build 22621)

Matrix products: default

locale:
[1] LC_COLLATE=English_United States.utf8
[2] LC_CTYPE=English_United States.utf8
[3] LC_MONETARY=English_United States.utf8
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.utf8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] dslabs_0.7.4

loaded via a namespace (and not attached):
 [1] digest_0.6.31   R6_2.5.1        jsonlite_1.8.4  evaluate_0.20
 [5] cachem_1.0.7    rlang_1.1.0     cli_3.6.0       rstudioapi_0.14
 [9] jquerylib_0.1.4 bslib_0.4.2     rmarkdown_2.20  tools_4.2.3
[13] xfun_0.37       yaml_2.3.7      fastmap_1.1.1   compiler_4.2.3
[17] htmltools_0.5.4 knitr_1.42      sass_0.4.5
```