Homework 4

GSND 5345Q, Fundamentals of Data Science

Due Wednesday, February 19th, 2025

Now its time to practice what we have learned in class and learn even more! Note that from now on your homework should be written in R Markdown. Turn in your html file and Rmd files, and any other relevant files in a tarball. Then turn it in by uploading to canvas.

Data Structures (20 points)

1. (10 points) Learn more about the **scan**, **readLines**, **read_html**, **readr**, and **readxl** functions for getting data into R. Use these functions to read data from a files into a tibble using R. You can use your own data example or a dataset used in class. How are these different from the functions we learned in class? Report what you find and give some examples!

2. (10 points) Learn about the **S3**, **S4**, and **R6** classes in R. When do you think you would use these? Describe what you learned and give some examples!

R Markdown (30 points; +20 points extra credit)

1. (30 points) R markdown is a powerful tool for literate programming. To gain more practice, as part recreate the `.Rmd` file for the example file in the homework folder: "Rmarkdown_example.html."

2. (20 points EXTRA CREDIT) Complete the markdown tutorial at https://www.markdowntutorial.com/. Confirm here that you completed it (on your honor!).

Tidyverse (50 points; 2 points each)

These exercises will give you some introductory experience with the `tidyverse`. Please complete the following:

1. Examine the built-in dataset `co2`. Which of the following is true:

   a. `co2` is tidy data: it has one year for each row.
   b. `co2` is not tidy: we need at least one column with a character vector.
   c. `co2` is not tidy: it is a matrix instead of a data frame.
   d. `co2` is not tidy: to be tidy we would have to wrangle it to have three columns (year, month and value), then each co2 observation would have a row.

2. Examine the built-in dataset `ChickWeight`. Which of the following is true:

   a. `ChickWeight` is not tidy: each chick has more than one row.
   b. `ChickWeight` is tidy: each observation (a weight) is represented by one row. The chick from which this measurement came is one of the variables.
   c. `ChickWeight` is not tidy: we are missing the year column.
   d. `ChickWeight` is tidy: it is stored in a data frame.

3. Examine the built-in dataset `BOD`. Which of the following is true:

   a. `BOD` is not tidy: it only has six rows.
   b. `BOD` is not tidy: the first column is just an index.
   c. `BOD` is tidy: each row is an observation with two values (time and demand)

d. `BOD` is tidy: all small datasets are tidy by definition.

4. Which of the following built-in datasets is tidy (you can pick more than one):

a. `BJsales`
b. `EuStockMarkets`
c. `DNase`
d. `Formaldehyde`
e. `Orange`
f. `UCBAdmissions`

5. Load the **dplyr** package and the murders dataset.

```
library(dplyr)
library(dslabs)
data(murders)
```

You can add columns using the **dplyr** function `mutate`. This function is aware of the column names and inside the function you can call them unquoted:

```
murders <- mutate(murders, population_in_millions = population / 10^6)
```

We can write `population` rather than `murders$population`. The function `mutate` knows we are grabbing columns from `murders`.

Use the function `mutate` to add a murders column named `rate` with the per 100,000 murder rate as in the example code above. Make sure you redefine `murders` as done in the example code above (murders <- [your code]) so we can keep using this variable.

6. If `rank(x)` gives you the ranks of `x` from lowest to highest, `rank(-x)` gives you the ranks from highest to lowest. Use the function `mutate` to add a column `rank` containing the rank, from highest to lowest murder rate. Make sure you redefine `murders` so we can keep using this variable.

7. With **dplyr**, we can use `select` to show only certain columns. For example, with this code we would only show the states and population sizes:

```
select(murders, state, population) %>% head()
```

Use `select` to show the state names and abbreviations in `murders`. Do not redefine `murders`, just show the results.

8. The **dplyr** function `filter` is used to choose specific rows of the data frame to keep. Unlike `select` which is for columns, `filter` is for rows. For example, you can show just the New York row like this:

```
filter(murders, state == "New York")
```

You can use other logical vectors to filter rows.

Use `filter` to show the top 5 states with the highest murder rates. After we add murder rate and rank, do not change the murders dataset, just show the result. Remember that you can filter based on the `rank` column.

9. We can remove rows using the `!=` operator. For example, to remove Florida, we would do this:

```
no_florida <- filter(murders, state != "Florida")
```

Create a new data frame called `no_south` that removes states from the South region. How many states are in this category? You can use the function `nrow` for this.

10. We can also use `%in%` to filter with **dplyr**. You can therefore see the data from New York and Texas like this:

```
filter(murders, state %in% c("New York", "Texas"))
```

Create a new data frame called `murders_nw` with only the states from the Northeast and the West. How many states are in this category?

11. Suppose you want to live in the Northeast or West **and** want the murder rate to be less than 1. We want to see the data for the states satisfying these options. Note that you can use logical operators with `filter`. Here is an example in which we filter to keep only small states in the Northeast region.

```
filter(murders, population < 5000000 & region == "Northeast")
```

Make sure `murders` has been defined with `rate` and `rank` and still has all states. Create a table called `my_states` that contains rows for states satisfying both the conditions: it is in the Northeast or West and the murder rate is less than 1. Use `select` to show only the state name, the rate, and the rank.

12. The pipe `%>%` can be used to perform operations sequentially without having to define intermediate objects. Start by redefining murder to include rate and rank.

```
murders <- mutate(murders, rate =  total / population * 100000,
                  rank = rank(-rate))
```

In the solution to the previous exercise, we did the following:

```
my_states <- filter(murders, region %in% c("Northeast", "West") &
                      rate < 1)

select(my_states, state, rate, rank)
```

The pipe `%>%` permits us to perform both operations sequentially without having to define an intermediate variable `my_states`. We therefore could have mutated and selected in the same line like this:

```
mutate(murders, rate =  total / population * 100000,
       rank = rank(-rate)) %>%
  select(state, rate, rank)
```

Notice that `select` no longer has a data frame as the first argument. The first argument is assumed to be the result of the operation conducted right before the `%>%`.

Repeat the previous exercise, but now instead of creating a new object, show the result and only include the state, rate, and rank columns. Use a pipe `%>%` to do this in just one line.

13. Reset `murders` to the original table by using `data(murders)`. Use a pipe to create a new data frame called `my_states` that considers only states in the Northeast or West which have a murder rate lower than 1, and contains only the state, rate and rank columns. The pipe should also have four components separated by three `%>%`. The code should look something like this:

```
my_states <- murders %>%
  mutate SOMETHING %>%
  filter SOMETHING %>%
  select SOMETHING
```

For exercises 14-20, we will be using the data from the survey collected by the United States National Center for Health Statistics (NCHS). This center has conducted a series of health and nutrition surveys since the 1960's. Starting in 1999, about 5,000 individuals of all ages have been interviewed every year and they complete the health examination component of the survey. Part of the data is made available via the **NHANES** package. Once you install the **NHANES** package, you can load the data like this:

```
library(NHANES)
data(NHANES)
```

The **NHANES** data has many missing values. The `mean` and `sd` functions in R will return `NA` if any of the entries of the input vector is an `NA`. Here is an example:

```
library(dslabs)
data(na_example)
mean(na_example)
```

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

```
sd(na_example)
```

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

To ignore the `NA`s we can use the `na.rm` argument:

```
mean(na_example, na.rm = TRUE)
```

```
## [1] 2.301754
```

```
sd(na_example, na.rm = TRUE)
```

```
## [1] 1.22338
```

Let's now explore the NHANES data.

14. We will provide some basic facts about blood pressure. First let's select a group to set the standard. We will use 20-to-29-year-old females. `AgeDecade` is a categorical variable with these ages. Note that the category is coded like " 20-29", with a space in front! What is the average and standard deviation of systolic blood pressure as saved in the `BPSysAve` variable? Save it to a variable called `ref`.

Hint: Use `filter` and `summarize` and use the `na.rm = TRUE` argument when computing the average and standard deviation. You can also filter the NA values using `filter`.

15. Using a pipe, assign the average to a numeric variable `ref_avg`. Hint: Use the code similar to above and then `pull`.

16. Now report the min and max values for the same group.

17. Compute the average and standard deviation for females, but for each age group separately rather than a selected decade as in question 1. Note that the age groups are defined by `AgeDecade`. Hint: rather than filtering by age and gender, filter by `Gender` and then use `group_by`.

18. Repeat exercise 4 for males.

19. We can actually combine both summaries for exercises 4 and 5 into one line of code. This is because `group_by` permits us to group by more than one variable. Obtain one big summary table using `group_by(AgeDecade, Gender)`.

20. For males between the ages of 40-49, compare systolic blood pressure across race as reported in the `Race1` variable. Order the resulting table from lowest to highest average systolic blood pressure.

21. Load the `murders` dataset. Which of the following is true?

a. `murders` is in tidy format and is stored in a tibble.
b. `murders` is in tidy format and is stored in a data frame.
c. `murders` is not in tidy format and is stored in a tibble.
d. `murders` is not in tidy format and is stored in a data frame.

22. Use `as_tibble` to convert the `murders` data table into a tibble and save it in an object called `murders_tibble`.

23. Use the `group_by` function to convert `murders` into a tibble that is grouped by region.

24. Write tidyverse code that is equivalent to this code:

```
exp(mean(log(murders$population)))
```

Write it using the pipe so that each function is called without arguments. Use the dot operator to access the population. Hint: The code should start with `murders %>%`.

25. Use the `map_df` to create a data frame with three columns named `n`, `s_n`, and `s_n_2`. The first column should contain the numbers 1 through 100. The second and third columns should each contain the sum of 1 through n with n the row number.

R Packages and Shiny (50 points)

1. (20 points) Walk step by step through the K means shiny app tutorial: "shiny_kmeans_app.html". Run the code to launch the app at each step. Document what changes are made in each step (which functions are used, what they do in the app).

2. (20 points) Now, make a K means R package with at least two functions (e.g., one that calculates the clusters, one that plots the result). Make sure to document your functions and create unit tests. Change the shiny app to rely on your functions, and add the shiny app to your `inst/` directory in your package.

3. (10 points) Post your R package as a repository on GitHub. Please add a hyperlink to your repository to your RMarkdown here.