BMI 713 Problem Set #1, Fall 2023 (80 points total)

Cheryl Gu

2023-09-20

Overview

Instructions

You will be submitting this assignment via GitHub Classroom. Your submission will be recorded when you push a commit to the repository on GitHub Classroom (instructions below), and you can make as many commits as you want before the deadline. We will grade assignments based on the latest commit.

How to commit code to GitHub

1. Open either GitHub Desktop or a command line terminal and navigate to the repository that contains this assignment (via the GUI or cd command)
2. Add the new or modified files to the staging area by either checking the box next to the files or using the command "git add <filename(s)>"
3. Commit the changes. You are required to include a commit message. This is found in the small text box near the commit button on GitHub Desktop or using the command "git commit -m 'commit message goes here' "
4. Push your changes and files to the remote repository. This is done by pressing the push to origin button on GitHub Classroom or with the command "git push"

What to Submit

You will need to submit **both** an R Markdown file and a knitted HTML or PDF for this assignment. When you finish the assignment, make sure to press the arrow next to the "Knit" button near the top panel of RStudio and select the option to knit to HTML or PDF. Make sure the new HTML or PDF appears in your repository and then follow the steps to commit and push to GitHub classroom. If you are having trouble knitting with the `{txt}` chunks, you can remove those chunks and type your answers outside of the chunks with clear labels.

Problem 1 (24 pts)

1.1 (2 pts)

For this problem set, we will use the `tidyverse` and `bmi713neiss` packages to work with a dataset of consumer-product related injuries collected by the National Electronic Injury Surveillance System (NEISS) from 2013 to 2018. Write code to load the packages into your R session in the chunk below.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(bmi713neiss)
library(dplyr)
library(tidyr)
```

1.2 (4 pts)

The `Age_Group` column of the NEISS dataset classifies each person into an age group: "Infants," "Children," "Adolescents," or several different adult age ranges.

First, let's simplify this categorization scheme. Add a new column to `neiss_2013_2018` named `Age_Group_injury`. If an injury's `Age_Group` is "Infants," "Children," or "Adolescents," `Age_Group_injury` should equal the value in `Age_Group` (i.e., keep the same age group for that injury). Otherwise, `Age_Group_injury` should equal "Adults".

```
# Assuming neiss_2013_2018 is your data frame
neiss_2013_2018 <-as.data.frame(neiss_2013_2018)
neiss_2013_2018_new <- neiss_2013_2018 %>%
  mutate(Age_Group_injury = case_when(
    Age_Group == "Infants" ~ Age_Group,
    Age_Group == "Children" ~ Age_Group,
    Age_Group == "Adolescents" ~ Age_Group,
    TRUE ~ "Adults"))
```

1.3 (4 pts)

What is the range of ages in each of the categories in `Age_Group_injury`? Use `dplyr` functions and pipes to make a summary table that contains the minimum and maximum ages for each value of `Age_Group_injury`.

Hint: You may want to convert the class of `neiss_2013_2018$Age` first and save this to the data frame. This will come in handy for later questions too.

```
neiss_2013_2018_new <- neiss_2013_2018_new %>%
  mutate(Age = as.numeric(Age))

age_range <- neiss_2013_2018_new %>%
  group_by(Age_Group_injury) %>%
  summarize(
    min_age = min(Age, na.rm = TRUE),
    max_age = max(Age, na.rm = TRUE)
```

```
  )

print(age_range)
```

```
## # A tibble: 4 x 3
##   Age_Group_injury min_age max_age
##   <chr>               <dbl>   <dbl>
## 1 Adolescents            10  19
## 2 Adults                 20 113
## 3 Children                1   9
## 4 Infants                 0   0.917
```

Keep these ranges in mind: later in this problem set, we will use these NEISS age ranges to define 'Adolescents' and 'Adults.'

1.4 (3 pts)

Suppose we are part of a team studying adolescent head injuries. Any person who sustained a head injury as an adolescent but is now an adult is eligible for a hypothetical follow-up study.

The ages listed in `neiss_2013_2018$Age` were recorded at the time of treatment, but we need to know how old each person is now. Write a new function that takes two parameters:

1. age at the time of treatment
2. treatment year

Your function should return the age the person would be in 2023 (you do not need to consider the specific month or day of the injury).

```
cal_curr_age <- function(Age, Year) {
  years_after_trt <- 2023 - Year
  curr_age <- Age + years_after_trt
  return(curr_age)
}
```

1.5 (4 pts)

Use your function to calculate the current age for each injury in the `neiss_2013_2018` data. The relevant information is stored in the `Age` and `Year` columns. Modify your function if needed to make it compatible with the control structure you choose.

Save your results in a new column of `neiss_2013_2018` named `Age_current`.

```
neiss_2013_2018_new <- neiss_2013_2018_new %>%
  mutate(Age_current = cal_curr_age(Age, Year))
```

1.6 (4 pts)

Now, assign each person to an age group *based on their current age (Age_current)*. Each person should be classified as either "Infants," "Children," "Adolescents," or "Adults" based on the age ranges that you observed in 1.3.

Save your results to a new column in `neiss_2013_2018` called `Age_Group_current`.

```
neiss_2013_2018_new <- neiss_2013_2018_new %>%
  mutate(Age_Group_current = case_when(
    Age_current <  0.9166667     ~ "Infants",
    Age_current >= 1 & Age_current < 9.0000000 ~ "Children",
    Age_current >= 10 & Age_current < 19.0000000 ~ "Adolescents",
    TRUE ~ "Adults"
  ))
```

1.7 (3 pts)

Use `dplyr` functions and pipes to create a summary table that lists the number of people in each combination of `Age_Group_injury` and `Age_Group_current` values. Your table should have three columns: `Age_Group_injury`, `Age_Group_current`, and `count`.

```
summary_table <- neiss_2013_2018_new %>%
  group_by(Age_Group_injury, Age_Group_current) %>%
  summarise(count = n())
```

```
## `summarise()` has grouped output by 'Age_Group_injury'. You can override using
## the `.groups` argument.
```

```
print(summary_table)
```

```
## # A tibble: 9 x 3
## # Groups:   Age_Group_injury [4]
##   Age_Group_injury Age_Group_current   count
##   <chr>            <chr>               <int>
## 1 Adolescents      Adolescents         85749
## 2 Adolescents      Adults             388151
## 3 Adults           Adults            1205373
## 4 Children         Adolescents        370184
## 5 Children         Adults              53014
## 6 Children         Children            77987
## 7 Infants          Adolescents          7437
## 8 Infants          Adults               7443
## 9 Infants          Children            31980
```

Problem 2 (22 pts)

2.1 (3 pts)

Recreate the summary table from 1.7, with one modification: only include injuries for which the `Body_Part` column equals *" HEAD"* (make sure to include the extra whitespace before the word HEAD). Save the results as a data frame called `head_counts`.

```
head_counts <- neiss_2013_2018_new %>%
  filter(Body_Part == " HEAD") %>%
  group_by(Age_Group_injury, Age_Group_current) %>%
  summarise(count = n())
```

```
## `summarise()` has grouped output by 'Age_Group_injury'. You can override using
## the `.groups` argument.
```

```
print(head_counts)
```

```
## # A tibble: 9 x 3
## # Groups:   Age_Group_injury [4]
##   Age_Group_injury Age_Group_current  count
##   <chr>            <chr>              <int>
## 1 Adolescents      Adolescents        12175
## 2 Adolescents      Adults             53594
## 3 Adults           Adults            166602
## 4 Children         Adolescents        74856
## 5 Children         Adults             12250
## 6 Children         Children           20561
## 7 Infants          Adolescents         4026
## 8 Infants          Adults              4100
## 9 Infants          Children           17603
```

2.2 (4 pts)

Use `tidyr` function(s) to convert `head_counts` to a table called `head_counts_byagegrp`. `head_counts_byagegrp` should contain one row per `Age_Group_current` category, and one column per `Age_Group_injury` category. Each entry in `head_counts_byagegrp` should list the number of head injuries for the respective `Age_Group_current` (row) and `Age_Group_injury` (column) combination.

```
head_counts_byagegrp <- head_counts %>%
  pivot_wider(names_from = Age_Group_injury, values_from = count)
```

2.3 (4 pts)

Does `head_counts_byagegrp` contain NA values? If so, replace the NAs with a sensible value of your choice. With one sentence, justify why you picked this value.

```
is.na(head_counts_byagegrp)
```

```
##      Age_Group_current Adolescents Adults Children Infants
## [1,]             FALSE       FALSE   TRUE    FALSE   FALSE
## [2,]             FALSE       FALSE  FALSE    FALSE   FALSE
## [3,]             FALSE        TRUE   TRUE    FALSE   FALSE
```

```
head_counts_byagegrp <- head_counts_byagegrp %>%
  replace_na(list(Infants = 0, Children = 0, Adolescents = 0, Adults = 0))
head(head_counts_byagegrp)
```

```
## # A tibble: 3 x 5
##   Age_Group_current Adolescents Adults Children Infants
##   <chr>                   <int>  <int>    <int>   <int>
## 1 Adolescents             12175      0    74856    4026
## 2 Adults                  53594 166602    12250    4100
## 3 Children                    0      0    20561   17603
```

```
# Justify the value you chose to replace NAs
Yes,"head_counts_byagegrp" contain NA values. Since the NA values in each
column refers to zero cases, for example there are no adolescents with injuries
in head during the treatment and now turned into adults. Thus, we need to
compute these cases as 0.
```

2.4 (4 pts)

Consider `head_counts` and `head_counts_byagegrp`. For each table, indicate whether it is wide or long and if it is tidy or not tidy. Briefly justify (max 2-3 sentences for each table).

```
"head_counts" is long and tidy. It is long because it has repeated values
in the first column and it has more rows than columns. It is tidy because
it satifies all three criteria of tidy data, including each variable
corresponds to one column, each observation has a unique row and each cell
in the table has a value.

"head_counts_byagegrp" is wide and not tidy. It is wide since its first column
has unique values and names for each column are unique as well. It is not tidy
because it fails the tidy data criteria. Even after we compute the NA values
into 0, all of the columns refer to different values from the Age_group_injury variable, meaning it fai
```

2.5 (2 pts)

We want to conduct a follow-up study of people who sustained a head injury as an adolescent but are now adults. Assuming each injury in NEISS is a unique person, how many people are eligible for our hypothetical follow-up study?

```
eligible_counts <- head_counts %>%
  filter(Age_Group_injury == "Adolescents", Age_Group_current == "Adults") %>%
  select(Age_Group_injury, Age_Group_current, count)

print(eligible_counts)
```

```
## # A tibble: 1 x 3
## # Groups:   Age_Group_injury [1]
##   Age_Group_injury Age_Group_current count
##   <chr>            <chr>             <int>
## 1 Adolescents      Adults            53594
```

```
There are 53594 people are eligible for the hypothetical follow-up study.
```

2.6 (2 pts)

What proportion of people who sustained a head injury as an adolescent are now adults? You can calculate this manually from the numbers you observe in `head_counts_byagegrp`.

```r
adolescents_adults <- head_counts_byagegrp %>%
  filter(Age_Group_current == "Adults") %>%
  select(Adolescents) %>%
  pull()
proportion <- (adolescents_adults/sum(head_counts_byagegrp$Adolescents))*100
print(proportion)
```

```
## [1] 81.48824
```

```r
# What proportion of people who sustained a head injury as an adolescent are now adults?
81.49% people sustained a head injury as an adolescent are now adults.
```

2.7 (3 pts)

Subset `neiss_2013_2018` to cases eligible for follow-up. Remember, the criteria are: 1) injury to the head, 2) adolescent at the time of injury, and 3) adult at current time. Save this subset in a new dataframe called `neiss_follow_up`.

```r
neiss_follow_up <- neiss_2013_2018_new %>%
  filter(
    Body_Part == " HEAD",
    Age_Group_injury == "Adolescents",
    Age_Group_current == "Adults"
  )
```

Reminder: Make an intermediate commit!

You're halfway through! Try committing your work to this point and pushing to GitHub Classroom. It's good to get into a habit of committing your work early and saving consistent checkpoints of your progress.

Problem 3 (24 points)

3.1 (3 pts)

Provided in the P1-data_files folder in the homework repository is a file called `neiss_product_codes_700_sport.csv` which links product codes to product descriptions and indicates whether or not the products are sports-related. Open this file using an application of your choice (e.g. Numbers, Excel) and take a look.

Use `read.csv` to (1) read in this CSV, (2) force the first column to be of class "numeric", the second column of class "character" and the third column of class "logical", (3) save the table in a variable called `products`. *Hint: Use `?read.csv` to get help on which function parameters to use to change the column variable classes.*

```r
github_path <- "~/Documents/GitHub/problem-set-1-Akitakeiko/"
products <- read.csv(paste0(github_path,
                     "neiss_product_codes_700_sport.csv"),
                     colClasses = c("numeric", "character", "logical"))
```

3.2 (4 pts)

Use a function from the `dplyr _join` family to add the new information from `products` to `neiss_follow_up`. `neiss_follow_up` should gain new columns, but maintain the same number of rows. Save the output back to `neiss_follow_up`.

For the rest of this problem, use ***`neiss_follow_up`****, not the full NEISS dataset.*

```
neiss_follow_up <- neiss_follow_up %>%
  left_join(products, by=c("Product_code" = "Code"))
```

3.3 (5 pts)

What column(s) of `neiss_follow_up` has the largest proportion of NA values? Write a function that takes a vector as input and calculates the proportion of NA values; then use the function alongside a command from the R `apply` family to calculate this proportion for each column.

```
cal_na_prop <- function(vec) {
  na_count <- sum(is.na(vec))
  total_count <- length(vec)
  prop_na <- na_count / total_count * 100
  return(prop_na)
}

na_proportions <- apply(neiss_follow_up, 2, cal_na_prop)
print(na_proportions)
```

```
##  CPSC_Case_Number    Treatment_Date               Age               Sex
##       0.000000000       0.000000000       0.000000000       0.000000000
##              Race        Other_Race         Body_Part         Diagnosis
##       0.000000000      92.185692428       0.000000000       0.000000000
##   Other_Diagnosis       Disposition          Location  Fire_Involvement
##      97.509049520       0.000000000       0.000000000       0.000000000
##         Product_1         Product_2       Narrative_1       Narrative_2
##       0.000000000      87.082509236       0.000000000      36.491025115
##           Stratum               PSU            Weight      Product_code
##       0.001865881       0.001865881       0.001865881       0.000000000
##              Year             Month               Day           Age_Cat
##       0.000000000       0.000000000       0.000000000       0.000000000
##         Age_Group  Age_Group_injury       Age_current Age_Group_current
##       0.000000000       0.000000000       0.000000000       0.000000000
##     Product.Title    Sports_related
##      10.503041385      10.503041385
```

```
max_na_column <- names(which.max(na_proportions))
print(max_na_column)
```

```
## [1] "Other_Diagnosis"
```

```
# Which column(s) has the largest proportion of NA values?
Other_Diagnosis has the largest proportion of NA values.
```

3.4 (3 pts)

We're interested in digging deeper into sports-related injuries. Use a `tidyr` or `dplyr` function to remove rows of `neiss_follow_up` where the `Sports_related` column equals NA. Save the output back to `neiss_follow_up`.

```
neiss_follow_up <- neiss_follow_up %>%
  filter(!is.na(Sports_related))
```

3.5 (6 pts)

Use tidyverse pipes to create a summary table from `neiss_follow_up`. Your summary table should have three columns: month, number of injuries in that month, and the number of sports-related injuries in that month (i.e., injuries for which `Sports_related` equals TRUE). Then, add a fourth column called "proportion" that contains the proportion of sports-related injuries out of all injuries in each month. Assign this table to a variable (you can choose any variable name).

```
summary_follow_up <- neiss_follow_up %>%
  group_by(Month) %>%
  summarise(
    total_injuries = n(),
    sports_related_injuries = sum(Sports_related, na.rm = TRUE)
  ) %>%
  mutate(proportion = sports_related_injuries / total_injuries * 100)
```

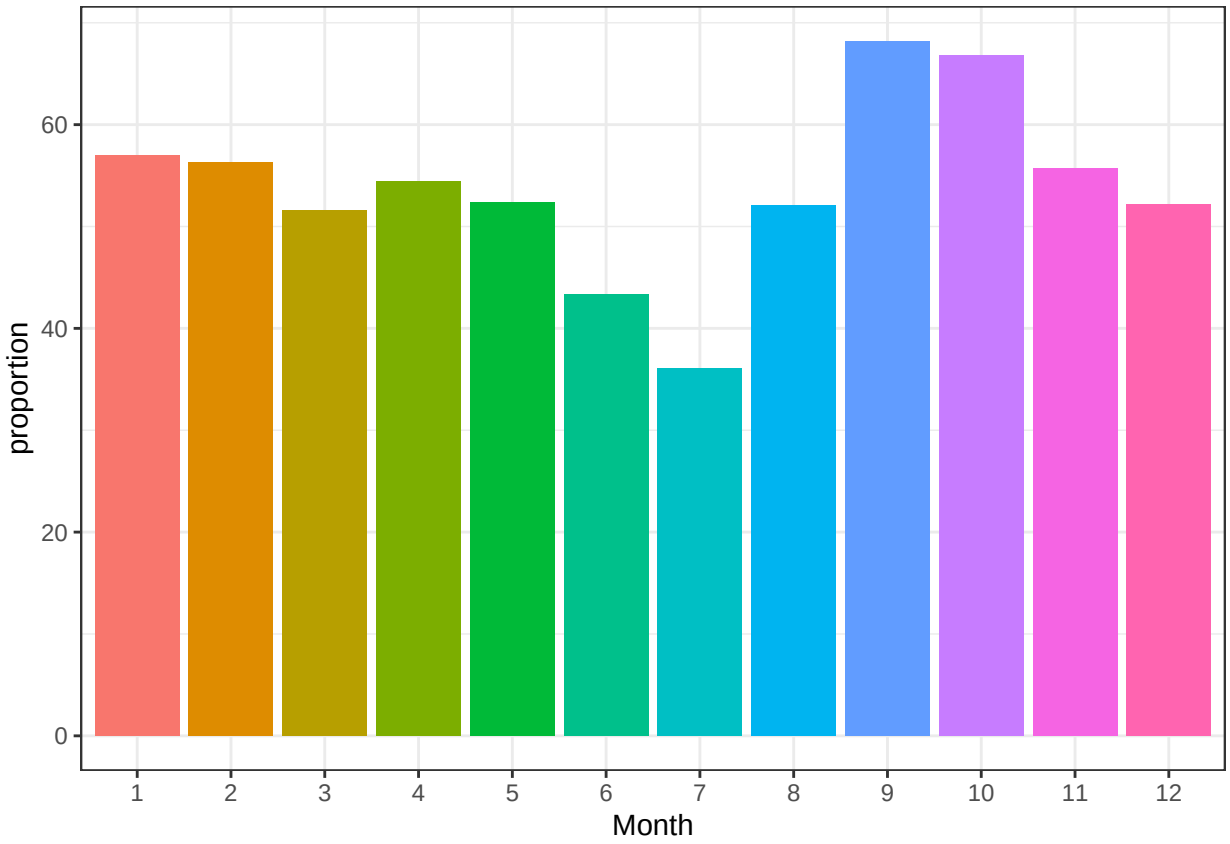
3.6 (3 pts)

We have provided this function that plots a bar plot of a table with a `Month` column and a `proportion` column:

```
plot_proportion <- function(table){
  library(ggplot2)
  g = ggplot(table, aes(factor(Month), proportion, fill=factor(Month)))
  g = g + geom_bar(stat="identity") + theme_bw() + xlab("Month") + theme(legend.position = "none")
  return(g)
}
```

Input your summary table from 3.5 to the R function above to plot the proportion of sports injuries in each month. Make sure your month column is named `Month` and your proportion of sports injuries column is named `proportion`.

```
plot_proportion(summary_follow_up)
```



Make one observation about how the proportion of sports injuries changes throughout the calendar year.

```
The proportion of sports injuries slowly decreased at the beginning half of
the year, until July, it hit the lowest. However, starts from July, it then
increased rapidly till Septemeber, and it hit the maximum of the entire year
in September, and decreased again until the end of the year.
```

Problem 4 (10 points)

Exploring Git

4.1 (4 pts)

You have a repository with a set of data files and R scripts to analyze them. You want to share this repository with a collaborator with git. Does your collaborator need a GitHub account to use git? What is the relationship between git and GitHub?

```
The collaborator does not necessarily need a github account to use git.
Git is the version control system, and GitHub is the platform that uses
git for version control and it also hosts repositories to allow users do
collaborative work.
```

4.2 (4 pts)

Here is how your repository is organized:

my_repo/ - scripts/ - data_cleaning.R - data_tidying.R - seasonal_analysis.R - data/ - injuries.txt - locations.txt - products.txt

The `data_cleaning.R` script includes the following line of code to read in a file:

```
read.table("~/Documents/BMI713/GitHub/my_repo/data/injuries.txt")
```

Is this an absolute path or a relative path? Which is more appropriate for code that you are sharing with someone else and why?

```
This is an abosolute path since the location it described starts from the
root directory. It is more appropriate for code sharing with the others to
use a relative path. Since everyone has distinct root directory, the entire
directory structure would have to modified if we use absolute path for sharing,
otherwise it would not run appropriately on other collaborator's computer.
A relative path specifies the location of a file relative to the current
(working) directory, so as long as the collaborator has the same working
directory, the line of code reading table would run safely.
```

4.3 (2 pts)

You start working on a new script that you save in `my_repo`, but you don't want the collaborator to see it yet. How can you store a file in a folder without tracking its changes or committing it to the git repository? **Hint:** Check out GitHub's Getting Started guide: https://docs.github.com/en/get-started/getting-started-with-git>

```
We can create a .gitignore file in my repository's root directory to tell
git which files and directories to ignore when you make a commit, so that
git would not track these files and others would not see them yet.
```

Don't Forget to Commit

Please remember to submit your assignment by adding all relevant files to the staging area (in this case the R Markdown file and corresponding knitted PDF), committing them to your local repository, and pushing them to the remote repo/GitHub Classroom.

Be courteous with knitting

As a final reminder, please be aware of the length of your knitted HTML or PDF file. If you have used code to print or examine something with a very long output, that should not be included in your knitted HTML or PDF. Please double check that there are no overtly long print-outs or lines getting cut off on the edge of the page in your HTML or PDF before submitting.