Potts Data Science Assessment

2024-08-07

Question 1

Q1 (a) Using R or Python scrape the wikipedia page on natural disasters: https://en.wikipedia.org/wiki/List_of_natural_disasters_by_death_toll for the tables of the 20th and 21st century all cause disasters into a data frame, tibble or pandas data frame.

```
url = "https://en.wikipedia.org/wiki/List_of_natural_disasters_by_death_toll"
disaster_tab = read_html(url) %>% # Parse an html table into df
  html_table()

# saving the 20th and 21st tables from the webpage
tw_cent = disaster_tab[[3]] %>%
  janitor::clean_names()
head(tw_cent)
```

```
## # A tibble: 6 x 6
##    year death_toll   event                          countries_affected type  date
##   <int> <chr>        <chr>                          <chr>              <chr> <chr>
## 1  1900 6,000-12,000 1900 Galveston hurricane     United States      Trop~ Sept~
## 2  1901 9,500        1901 eastern United States ~ United States      Heat~ June~
## 3  1902 29,000       1902 eruption of Mount Pelée Martinique         Volc~ Apri~
## 4  1903 3,500        1903 Manzikert earthquake    Turkey             Eart~ Apri~
## 5  1904 400          1904 Sichuan earthquake      China              Eart~ Augu~
## 6  1905 20,000+      1905 Kangra earthquake       India              Eart~ Apri~
```

```
twfirst_cent = disaster_tab[[4]] %>%
  janitor::clean_names()
head(twfirst_cent)
```

```
## # A tibble: 6 x 6
##    year death_toll    event                          countries_affected type  date
##   <int> <chr>         <chr>                          <chr>              <chr> <chr>
## 1  2001 13,805-20,023 2001 Gujarat earthquake      India              Eart~ Janu~
## 2  2002 1,200         2002 Hindu Kush earthquakes  Afghanistan        Eart~ Marc~
## 3  2003 72,000        2003 European heat wave      Europe             Heat~ July~
## 4  2004 227,898       2004 Indian Ocean earthqua~  Indonesia, Sri La~ Eart~ Dece~
## 5  2005 86,000-87,351 2005 Kashmir earthquake      India, Pakistan    Eart~ Octo~
## 6  2006 5,749-5,778   2006 Yogyakarta earthquake   Indonesia          Eart~ May ~
```

Q1 (b) Convert the death toll to numbers using the midpoints when a range is given and the bound when an upper or lower bound is given (example 20,000+ converts to 20000).

To clean the death_toll variable, we remove any extraneous elements, such as commas, a citation [#], +, or (estimate), as well as extracting the midpoint from any ranges.

```r
# cleaning process for 20th century df
tw_cent2 = tw_cent %>%
  mutate(orig_death_toll = death_toll) %>% # create copy of death toll to check against
  mutate(death_toll = str_replace_all(death_toll, "[,+]", "")) %>% # remove , and +
  mutate(death_toll = gsub("\\[[^][]*]", "", death_toll)) %>% # remove [#]
  mutate(lower_bound = stri_extract_first_regex(death_toll, "[0-9]+")) %>%
  mutate(upper_bound = stri_extract_last_regex(death_toll, "[0-9]+")) %>% # separate ranges
  mutate(midpoint = (as.numeric(upper_bound) + as.numeric(lower_bound))/2) %>% # calculate midpoint
  mutate(death_toll = ifelse(lower_bound == upper_bound, lower_bound, midpoint)) %>%
  select(-c("upper_bound", "lower_bound", "midpoint")) # condense clean death_toll

head(tw_cent2)
```

```
## # A tibble: 6 x 7
##    year death_toll event          countries_affected type  date  orig_death_toll
##   <int> <chr>      <chr>          <chr>              <chr> <chr> <chr>
## 1  1900 9000       1900 Galvesto~ United States      Trop~ Sept~ 6,000-12,000
## 2  1901 9500       1901 eastern ~ United States      Heat~ June~ 9,500
## 3  1902 29000      1902 eruption~ Martinique         Volc~ Apri~ 29,000
## 4  1903 3500       1903 Manziker~ Turkey             Eart~ Apri~ 3,500
## 5  1904 400        1904 Sichuan ~ China              Eart~ Augu~ 400
## 6  1905 20000      1905 Kangra e~ India              Eart~ Apri~ 20,000+
```

```r
# cleaning process for 21th century df
twfirst_cent2 = twfirst_cent %>%
  mutate(orig_death_toll = death_toll) %>% # create copy of death toll to check against
  mutate(death_toll = str_replace_all(death_toll, "[,+]", "")) %>% # remove , and +
  mutate(death_toll = gsub("\\s*\\([^\\)]+\\)", "", death_toll)) %>% # remove (estimate)
  mutate(death_toll = gsub("\\[[^][]*]", "", death_toll)) %>% # remove [#]
  mutate(lower_bound = stri_extract_first_regex(death_toll, "[0-9]+")) %>% # separate ranges
  mutate(upper_bound = stri_extract_last_regex(death_toll, "[0-9]+")) %>%
  mutate(midpoint = (as.numeric(upper_bound) + as.numeric(lower_bound))/2) %>% # calculate midpoint
  mutate(death_toll = ifelse(lower_bound == upper_bound, lower_bound, midpoint)) %>%
  select(-c("upper_bound", "lower_bound", "midpoint")) # condense clean death_toll

head(twfirst_cent2)
```

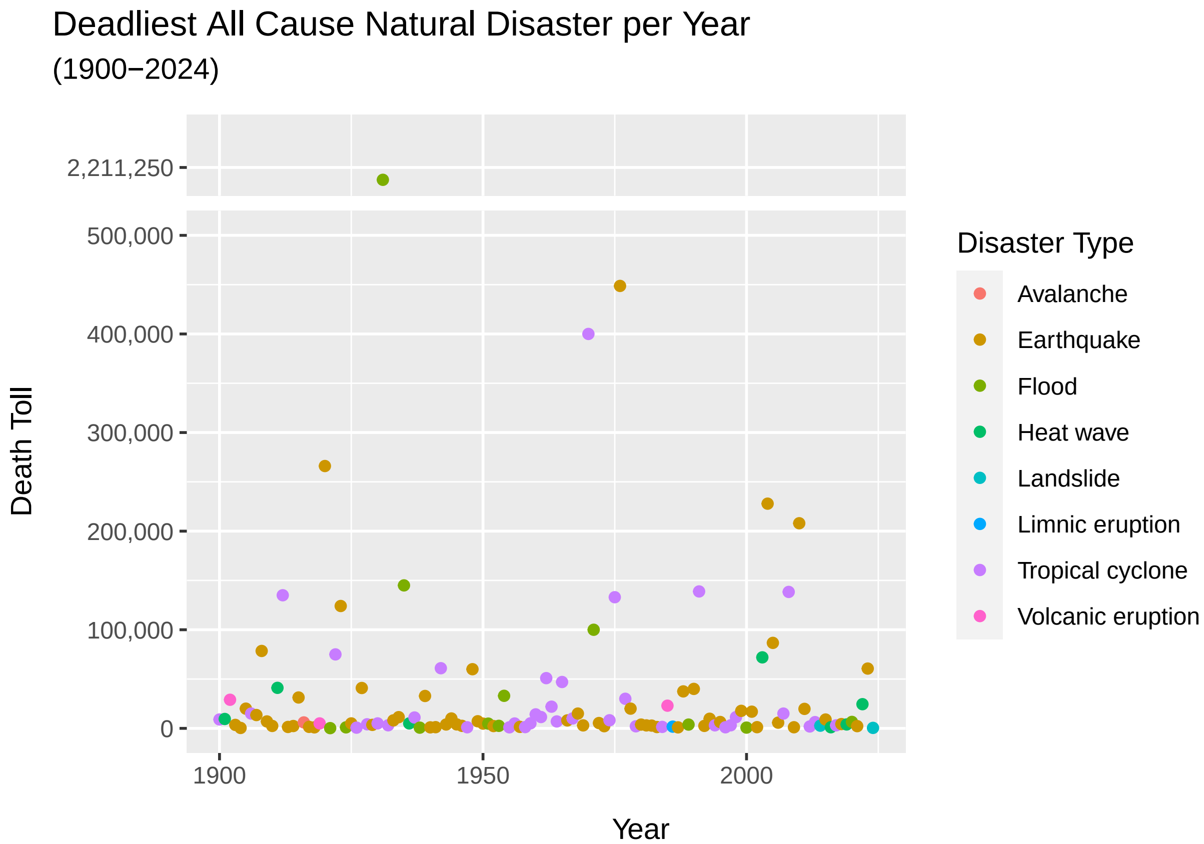
```
## # A tibble: 6 x 7
##    year death_toll event          countries_affected type  date  orig_death_toll
##   <int> <chr>      <chr>          <chr>              <chr> <chr> <chr>
## 1  2001 16914      2001 Gujarat ~ India              Eart~ Janu~ 13,805-20,023
## 2  2002 1200       2002 Hindu Ku~ Afghanistan        Eart~ Marc~ 1,200
## 3  2003 72000      2003 European~ Europe             Heat~ July~ 72,000
## 4  2004 227898     2004 Indian O~ Indonesia, Sri La~ Eart~ Dece~ 227,898
## 5  2005 86675.5    2005 Kashmir ~ India, Pakistan    Eart~ Octo~ 86,000-87,351
## 6  2006 5763.5     2006 Yogyakar~ Indonesia          Eart~ May ~ 5,749-5,778
```

Q1 (c) Merge the 20th and 21st century data frames and plot the death toll (vertical / y axis) by year (horizontal / x axis) color coded by kind of disaster.

We make note that some observations have more than one type, and make the decision to color based on the first type listed. While not ideal to add an axis break for the death toll, the graph was difficult to see major trends without doing so for the extreme value of over 2,000,000. Full layman description of the plot is in the README file.

```r
df = rbind(tw_cent2, twfirst_cent2) %>% # stack vertically
  mutate(type = str_to_sentence(type)) %>%
  mutate(type = gsub("(.*),.*", "\\1", type)) %>% # isolate type
  mutate(type = as.factor(type)) %>% # fix data types for plotting
  mutate(death_toll = as.numeric(death_toll))

ggplot(df, aes(x=year, y=death_toll, color=type)) + geom_point() + # scatterplot
  labs(title = "Deadliest All Cause Natural Disaster per Year", subtitle = "(1900-2024)") +
  xlab("Year") + ylab("Death Toll") + # aesthetics
  scale_y_continuous(labels = scales::comma,
                     limits = c(0, 2211252),
                     breaks = c(0, 100000, 200000, 300000,
                                400000, 500000, 2211250)) +
  scale_y_break(c(500000, 2211249), scales = 0.15) +
  theme(axis.text.y.right = element_blank(),
        axis.ticks.y.right = element_blank(),
        axis.title.y.right = element_blank()) +
  scale_x_continuous(limits = c(1900, 2024),
                     breaks = seq(1900, 2024, by = 50)) +
  guides(color = guide_legend(title = "Disaster Type"))
```



Question 2

Let x and y be vectors of length n. Consider minimizing the loss L(b) = ||y - b x||^2 over b where b is a scalar. (The solution is b = <x, y> / ||x||^2 .)

First, we have the loss function

$$L(b) = ||y - b \cdot x||^2$$

which is equivalently $\sum_{i=1}^{n}(y_i - b \cdot x_i)^2$

The gradient is the derivative of $L(b)$ with respect to b:

$$\frac{dL(b)}{db} = \frac{d}{db}\left(\sum_{i=1}^{n}(y_i - b \cdot x_i)^2\right) = \frac{d}{db}\left(\sum_{i=1}^{n} y_i^2 - 2b\sum_{i=1}^{n} y_i \cdot x_i + b^2\sum_{i=1}^{n} x_i^2\right)$$

We can simplify to yield the gradient to be:

$$\frac{dL(b)}{db} = -2\sum_{i=1}^{n} y_i \cdot x_i + 2b\sum_{i=1}^{n} x_i^2$$

We can check that this is correct since we are given the proper solution $b = \frac{<x,y>}{||x||^2}$. Setting $\frac{dL(b)}{db} = 0$ and solving for b gives: $\frac{\sum_{i=1}^{n} y_i \cdot x_i}{\sum_{i=1}^{n} x_i^2} = b$

Q2 (a) Write a function in R or python that takes two vectors or numpy vectors and iterates to solve for b using gradient descent. That is, the update is: Update(b) = Current value of b - e * Derivative of L with respect to b evaluated at the current value of b. Where e is a user-supplied real number usually called the learning rate or step size.

```
gradient_descent <- function(x, y, e, iterations) {
  n <- length(x) # length of vector
  b <- 0 # starting value
  for (i in 1:iterations) {
    gradient <- -2*(sum(x*y) - b*sum(x^2)) # derived above
    b <- b - e*gradient
  }
  return(b)
}

# a random set of standard normal vectors of length 7
x = rnorm(7)
y = rnorm(7)

# check that b_est matches true solution (reasonable e chosen)
b_est = gradient_descent(x, y, e = 0.005, iterations = 2000)
b_est
```

```
## [1] -0.1456682
```

```
# true solution
b = dot(x, y) / sum(x^2)
b
```

```
## [1] -0.1456682
```

Q2 (b) Test your function out on some randomly generated normal vectors where you know the value of b. How does the performance of the algorithm's depend on e? When does the algorithm fail and why?

Note that we set the number of iterations to 2000 (as this specification was sufficient for the algorithm to converge when an appropriate e was chosen) and that we are looking at standard normal vectors for simplicity.

The performance of the algorithm depends on the learning rate (e) such that the minimum loss is achieved in a range of e values, before which the loss is minimally higher and after which the loss exponentially increases to infinity. For example 1, where we generated x and y from a standard normal distribution of length 3, the well-behaved range of e is between 0.00068 and 0.17. When our learning rate is too large (>0.17 in this example), the algorithm is overshooting the minima of the loss function and we never converge to the true optimal value of b. The exact range of appropriate learning rate (and the minimal loss value achieved when the algorithm converges) is dependent on the details of the random vectors which were chosen. However, the general trend described previously holds for examples 2 and 3, where e>0.24 and e>0.076 are too large, respectively.