Homework 4

[Ra-Zakee Muhammad]

Due 9/28/2021

Classmates/other resources consulted: [type answer here]

```
library(tidyverse)
```

Question 1 (8 points)

Suppose you want to import data about police misconduct settlements in Los Angeles, as collected and detailed at https://github.com/fivethirtyeight/police-settlements/tree/main/los_angeles_ca.

a. **Import this data into a tibble directly using a URL (The data is in the folder called "final", and be sure you are providing the URL of the raw data)**

```
read_csv("https://raw.githubusercontent.com/fivethirtyeight/police-settlements/main/los_angeles_ca/final
```

```
## Rows: 997 Columns: 23

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (9): fiscal_year, city, state, docket_number, claim_number, matter_name...
## dbl  (4): incident_year, filed_year, amount_awarded, calendar_year
## lgl  (7): other_expenses, collection, total_incurred, case_outcome, court, p...
## date (3): incident_date, filed_date, closed_date

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## # A tibble: 997 x 23
##    fiscal_year city        state incident_date incident_year filed_date filed_year
##    <chr>       <chr>       <chr> <date>                <dbl> <date>          <dbl>
##  1 FY2009/10   Los Angeles CA    NA                       NA NA                 NA
##  2 FY2009/10   Los Angeles CA    NA                       NA NA                 NA
##  3 FY2009/10   Los Angeles CA    NA                       NA NA                 NA
##  4 FY2009/10   Los Angeles CA    NA                       NA NA                 NA
##  5 FY2009/10   Los Angeles CA    NA                       NA NA                 NA
##  6 FY2009/10   Los Angeles CA    NA                       NA NA                 NA
##  7 FY2009/10   Los Angeles CA    NA                       NA NA                 NA
##  8 FY2009/10   Los Angeles CA    NA                       NA NA                 NA
```

```
##  9 FY2009/10   Los Angeles CA    NA                        NA NA                  NA
## 10 FY2009/10   Los Angeles CA    NA                        NA NA                  NA
## # ... with 987 more rows, and 16 more variables: closed_date <date>,
## #   amount_awarded <dbl>, other_expenses <lgl>, collection <lgl>,
## #   total_incurred <lgl>, case_outcome <lgl>, docket_number <chr>,
## #   claim_number <chr>, court <lgl>, plaintiff_name <lgl>, matter_name <chr>,
## #   plaintiff_attorney <lgl>, location <chr>, summary_allegations <chr>,
## #   claim_or_lawsuit <chr>, calendar_year <dbl>
```

b. **In 2-4 sentences, explain when importing data directly using a URL is a good idea, and when importing data directly using a URL is a bad idea.**

Its a good idea when the url send you to the raw data. If this isn't the case then importing using a url will give you an html filw which wont actually have the data you expect it to have.

Question 2 (24 points)

This question concerns the pets_info.csv files that was attached to the homework assignment.

a. **Import this csv file, without specifying any additional options. Explain why the tibble you get is not what you want.**

```
read_csv("pets_info.csv")
```

```
## New names:
## * `` -> ...2

## Rows: 10 Columns: 2

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): This file contains some information about pets, ...2

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## Warning: One or more parsing issues, see `problems()` for details

## # A tibble: 10 x 2
##    `This file contains some information about pets` ...2
##    <chr>                                            <chr>
##  1 <NA>                                             ,,
##  2 Name                                             Age,Species,DateAdopted
##  3 Sparky                                           10,Dog,3/4/2021
##  4 Fluffy                                           4,Cat,5/16/2020
##  5 Spot                                             3,Dog,11/23/2020
##  6 Buddy                                            7,Dog,12/3/2015
##  7 Fido                                             12,Dog,4/5/2016
##  8 Patches                                          3,Cat,1/14/2019
##  9 Socks                                            6,Cat,4/19/2018
## 10 Lassie                                           2,Dog,7/17/2021
```

The first 2 lines of the data frame contain descriptions of the entire csv file which means that when we try to import the file to a tibble, the first line of the frame is how the factors of the frame are created and this isn't how the data is intended to be broken up if we look at the third line of the frames which has "Name Age,Species,DateAdopted" which is probably how the frame was intended to be broken up.

b. **Add a parameter to your import command to correctly import the data in this file into a tible, with the correct column names and format.**

```
read_csv("pets_info.csv", skip = 2)
```

```
## Rows: 8 Columns: 4

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): Name, Species, DateAdopted
## dbl (1): Age

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## # A tibble: 8 x 4
##   Name      Age Species DateAdopted
##   <chr>   <dbl> <chr>   <chr>
## 1 Sparky     10 Dog     3/4/2021
## 2 Fluffy      4 Cat     5/16/2020
## 3 Spot        3 Dog     11/23/2020
## 4 Buddy       7 Dog     12/3/2015
## 5 Fido       12 Dog     4/5/2016
## 6 Patches     3 Cat     1/14/2019
## 7 Socks       6 Cat     4/19/2018
## 8 Lassie      2 Dog     7/17/2021
```

c. **Run an additional command to confirm that there were no parsing arrors that occurred when importing this file**

```
problems(read_csv("pets_info.csv", skip = 2))
```

```
## Rows: 8 Columns: 4

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): Name, Species, DateAdopted
## dbl (1): Age

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## # A tibble: 0 x 5
## # ... with 5 variables: row <int>, col <int>, expected <chr>, actual <chr>,
## #   file <chr>
```

d. **(In most systems), the column age was imported as a double, even though all the ages given were integers. Correct this by adjusting your import command from the previous part.**

```
read_csv("pets_info.csv",
                      col_types = cols(Age = col_integer()),
         skip= 2)
```

```
## # A tibble: 8 x 4
##   Name      Age Species DateAdopted
##   <chr>   <int> <chr>   <chr>
## 1 Sparky     10 Dog     3/4/2021
## 2 Fluffy      4 Cat     5/16/2020
## 3 Spot        3 Dog     11/23/2020
## 4 Buddy       7 Dog     12/3/2015
## 5 Fido       12 Dog     4/5/2016
## 6 Patches     3 Cat     1/14/2019
## 7 Socks       6 Cat     4/19/2018
## 8 Lassie      2 Dog     7/17/2021
```

e. **(In most systems), the Date Adopted column was imported as a character string instead of as a date. Correct this by adjusting your import command from the previous part.**

```
read_csv("pets_info.csv",
                      col_types = cols(Age = col_integer(),
                                       DateAdopted = col_date(format = "%m/%d/%Y")), skip= 2)
```

```
## # A tibble: 8 x 4
##   Name      Age Species DateAdopted
##   <chr>   <int> <chr>   <date>
## 1 Sparky     10 Dog     2021-03-04
## 2 Fluffy      4 Cat     2020-05-16
## 3 Spot        3 Dog     2020-11-23
## 4 Buddy       7 Dog     2015-12-03
## 5 Fido       12 Dog     2016-04-05
## 6 Patches     3 Cat     2019-01-14
## 7 Socks       6 Cat     2018-04-19
## 8 Lassie      2 Dog     2021-07-17
```

f. **Suppose you want to enforce that only the species "Dog" and "Cat" are allowed in your tibble. How would you ensure that any entries other than "Dog" and "Cat" in your tibble become NA? Adjust your import command from the previous part to ensure this.**

```
read_csv("pets_info.csv",
         col_types = cols(Age = col_integer(),
                          DateAdopted = col_date(format = "%m/%d/%Y"),
                          Species = col_factor(levels = c("Cat", "Dog"))),
                       skip= 2)
```

```
## # A tibble: 8 x 4
##   Name      Age Species DateAdopted
##   <chr>   <int> <fct>   <date>
## 1 Sparky     10 Dog     2021-03-04
## 2 Fluffy      4 Cat     2020-05-16
## 3 Spot        3 Dog     2020-11-23
## 4 Buddy       7 Dog     2015-12-03
## 5 Fido       12 Dog     2016-04-05
## 6 Patches     3 Cat     2019-01-14
## 7 Socks       6 Cat     2018-04-19
## 8 Lassie      2 Dog     2021-07-17
```

Question 3 (4 points)

Consider the pets_info2.csv file attached to the homework assignment; it does not have column names, but the columns are, in order, Name, Age, Species, and Date Adopted. Import the pets_info2.csv file and specify the column names as you do.

```
read_csv("pets_info2.csv", col_names = c("Name", "Age", "Species", "Date Adopted"))
```

```
## Rows: 8 Columns: 4

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): Name, Species, Date Adopted
## dbl (1): Age

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## # A tibble: 8 x 4
##   Name      Age Species `Date Adopted`
##   <chr>   <dbl> <chr>   <chr>
## 1 Sparky     10 Dog     3/4/2021
## 2 Fluffy      4 Cat     5/16/2020
## 3 Spot        3 Dog     11/23/2020
## 4 Buddy       7 Dog     12/3/2015
## 5 Fido       12 Dog     4/5/2016
## 6 Patches     3 Cat     1/14/2019
## 7 Socks       6 Cat     4/19/2018
## 8 Lassie      2 Dog     7/17/2021
```

Question 4 (6 points)

What happens when one row of a csv file has a different number of commas than other rows? Refer the following two examples in your explanation if you'd like, though make sure your explanation is more general than just these two examples. Be sure to mention any parsing errors that might occur.

```
read_csv("a,b,c,d
         1,2,3,4
         5,6,7
         8,9,10,11")
```

```
## Rows: 3 Columns: 4

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (4): a, b, c, d

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## Warning: One or more parsing issues, see `problems()` for details

## # A tibble: 3 x 4
##       a     b     c     d
##   <dbl> <dbl> <dbl> <dbl>
## 1     1     2     3     4
## 2     5     6     7    NA
## 3     8     9    10    11
```

```
problems(read_csv("a,b,c,d
         1,2,3,4
         5,6,7,8,9
         10,11,12,13"))
```

```
## Rows: 3 Columns: 4

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (3): a, b, c

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## Warning: One or more parsing issues, see `problems()` for details

## # A tibble: 1 x 5
##     row   col expected  actual    file
##   <int> <int> <chr>     <chr>     <chr>
## 1     3     5 4 columns 5 columns /private/var/folders/4s/xc7wfd1n4m74y439jl83z~
```

When a row has fewer number of entries than the others, these are read as missing values when the tibble is created by r and the entry becomes NA, and there is a parsing error saying that expected columns differs from actual. When a row has greater number of entries than the others, those extra entries are read as a final concatenated value on the right most column, and there is a parsing error saying that expected columns differs from actual.

Question 5 (4 points)

Suppose you have a large data file and only want to import the first 1500 rows of it. Take a look at the possible parameters for read_csv, and figure out which one controls how many rows of the data are imported. Change that parameter below to only import the first 1500 rows of the New Zeland Card Transactions data set.

```
read_csv("New_Zeland_Electronic_card_transactions_aug_2021.csv", n_max = 1500)
```

```
## Rows: 1500 Columns: 14

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (8): Series_reference, STATUS, UNITS, Subject, Group, Series_title_1, Se...
## dbl (3): Period, Data_value, Magnitude
## lgl (3): Suppressed, Series_title_4, Series_title_5

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## # A tibble: 1,500 x 14
##    Series_reference Period Data_value Suppressed STATUS UNITS   Magnitude Subject
##    <chr>             <dbl>      <dbl> <lgl>      <chr>  <chr>       <dbl> <chr>
##  1 ECTA.S19A1        2001.      2462. NA         F      Dollars         6 Electr~
##  2 ECTA.S19A1        2002.     17177. NA         F      Dollars         6 Electr~
##  3 ECTA.S19A1        2003.     22530. NA         F      Dollars         6 Electr~
##  4 ECTA.S19A1        2004.     28005. NA         F      Dollars         6 Electr~
##  5 ECTA.S19A1        2005.     30630. NA         F      Dollars         6 Electr~
##  6 ECTA.S19A1        2006.     33317. NA         F      Dollars         6 Electr~
##  7 ECTA.S19A1        2007.     36422  NA         F      Dollars         6 Electr~
##  8 ECTA.S19A1        2008.     39198  NA         F      Dollars         6 Electr~
##  9 ECTA.S19A1        2009.     40629. NA         F      Dollars         6 Electr~
## 10 ECTA.S19A1        2010.     41815. NA         F      Dollars         6 Electr~
## # ... with 1,490 more rows, and 6 more variables: Group <chr>,
## #   Series_title_1 <chr>, Series_title_2 <chr>, Series_title_3 <chr>,
## #   Series_title_4 <lgl>, Series_title_5 <lgl>
```

Question 6 (12 points)

a. **Suppose you want to parse the string "$1,000" to mean one thousand. How would you do this?**

```
parse_number("$1,000")
```

```
## [1] 1000
```

b. **Suppose you want to parse the string "1.000,00" to mean one thousand. How would you do this?**

```
parse_number("1.000,00",  locale = locale(grouping_mark = "."))
```

```
## [1] 1000
```

c. **Explain the difference between parse_double() and parse_number()**

Parse_double parses into double form any string that is already in a form that r can understand as a number (it just so happens to be a string). Parse_number gets rid of any additional characters attached to a string containing anumber so that that number within the larger string of numeric and non-numeric characters is extracted.

Question 7 (8 points)

a. **Parse the following string, using the ASCII encoding, the Latin1 encoding, and the UTF-8 encoding. Which do you think is correct?**

(note: the outcomes of these parsing steps may not display correctly in the knitted file, and that's fine - look at the outputs when running the code chunk in RMarkdown to make your determination)

```
#{r} #string_a = "\xd8\xa3\xd9\x87\xd9\x84\xd8\xa7"  # parse_character(string_a, locale
= locale(encoding = "UTF-8"))  #
```

b. **Use the guess_encoding function to guess the correct encoding of the following string, and then parse it correctly.**

```
#{r} #string_b = "\uc548\ub155\ud558\uc2ed\ub2c8\uae4c"  #y1 <- charToRaw(string_b)
#guess_encoding(y1) #parse_character(string_b, locale = locale(encoding = "UTF-8")) #
```

Question 8 (12 points)

Parse the following dates and date/time combinations

a.

```
d_a <- "Aug. 19 (2015)"
```

```
parse_date(d_a, "%b. %e (%Y)")
```

```
## [1] "2015-08-19"
```

b.

```
d_b <- "2015-Juni-20"
```

(Hint: the language here is German; a list of the ISO 639-1 language abbreviations that R uses can be found at https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes)

```
parse_date(d_b, "%Y-%B-%d", locale = locale("de"))
```

```
## [1] "2015-06-20"
```

c.

```
dt_c <- "January 15, 2017, at 5:45 am"
```

```
parse_datetime(dt_c,"%B %e, %Y, at %I:%M %p")
```

```
## [1] "2017-01-15 05:45:00 UTC"
```

Question 9 (12 points)

Consider the dates_times.csv file, in which the first column has a date and time, the second column has a date, and the third column has a time. Import this file; your import should correctly parse these columns to the standard R format for dates and/or times.

```
read_csv("dates_times.csv")
```

```
## Rows: 59 Columns: 3

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): Date_time, Date, Time

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## # A tibble: 59 x 3
##    Date_time        Date          Time
##    <chr>            <chr>         <chr>
##  1 1-1-2021; 16:43  1 Enero 2021  4:43pm
##  2 1-2-2021; 17:16  2 Enero 2021  5:16pm
##  3 1-3-2021; 16:32  3 Enero 2021  4:32pm
##  4 1-4-2021; 16:32  4 Enero 2021  4:32pm
##  5 1-5-2021; 16:23  5 Enero 2021  4:23pm
##  6 1-6-2021; 17:12  6 Enero 2021  5:12pm
##  7 1-7-2021; 16:55  7 Enero 2021  4:55pm
##  8 1-8-2021; 16:35  8 Enero 2021  4:35pm
##  9 1-9-2021; 17:01  9 Enero 2021  5:01pm
## 10 1-10-2021; 17:49 10 Enero 2021 5:49pm
## # ... with 49 more rows
```

```
read_csv("dates_times.csv") %>%
  mutate(Date_time = parse_datetime(Date_time, format = "%m-%d-%Y; %H:%M") ,
         Time = parse_time(Time, format = "%I:%M%p"),
         Date = parse_date(Date, format = "%d %B %Y", locale = locale("es")))
```

```
## Rows: 59 Columns: 3
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): Date_time, Date, Time
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## # A tibble: 59 x 3
##    Date_time           Date       Time
##    <dttm>              <date>     <time>
##  1 2021-01-01 16:43:00 2021-01-01 16:43
##  2 2021-01-02 17:16:00 2021-01-02 17:16
##  3 2021-01-03 16:32:00 2021-01-03 16:32
##  4 2021-01-04 16:32:00 2021-01-04 16:32
##  5 2021-01-05 16:23:00 2021-01-05 16:23
##  6 2021-01-06 17:12:00 2021-01-06 17:12
##  7 2021-01-07 16:55:00 2021-01-07 16:55
##  8 2021-01-08 16:35:00 2021-01-08 16:35
##  9 2021-01-09 17:01:00 2021-01-09 17:01
## 10 2021-01-10 17:49:00 2021-01-10 17:49
## # ... with 49 more rows
```

Question 10 (5 points)

The following command imports the nz_cards data set, which was attached to this assignment.

```
nz_cards <- read_csv("New_Zeland_Electronic_card_transactions_aug_2021.csv")
```

```
## Rows: 18024 Columns: 14
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (9): Series_reference, Suppressed, STATUS, UNITS, Subject, Group, Series...
## dbl (3): Period, Data_value, Magnitude
## lgl (2): Series_title_4, Series_title_5
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

As we saw in the in-class activities, it has a column called Period, which gives a year and month in YYYY.MM format. While you can parse this into a standard R date (see activity solutions), another approach is to make separate columns for month and date. Here's some code that does that using the floor function, which rounds a number down to the nearest integer. (This code also only keeps a few relevant columns, for simplicity)

```
nz_cards2 <- nz_cards %>%
  mutate(year = floor(Period), month = (Period - floor(Period)) * 100) %>%
  select(Period, year, month, Data_value, UNITS)

nz_cards2
```

```
## # A tibble: 18,024 x 5
##    Period  year month Data_value UNITS
##     <dbl> <dbl> <dbl>      <dbl> <chr>
##  1  2001.  2001  3.00      2462. Dollars
##  2  2002.  2002  3.00     17177. Dollars
##  3  2003.  2003  3.00     22530. Dollars
##  4  2004.  2004  3.00     28005. Dollars
##  5  2005.  2005  3.00     30630. Dollars
##  6  2006.  2006  3.00     33317. Dollars
##  7  2007.  2007  3.00     36422  Dollars
##  8  2008.  2008  3.00     39198  Dollars
##  9  2009.  2009  3.00     40629. Dollars
## 10  2010.  2010  3.00     41815. Dollars
## # ... with 18,014 more rows
```

We can see that the tibble nz_cards2 has several entries where the month is 3. However, when we filter to only keep those rows where the month is 3, we get an empty tibble!

What is the problem here? Why are none of the rows where the month is 3 showing up? Explain how you would fix this problem.

Warning The realities of computer arithmetic can cause unexpected results, especially with floor and ceiling. For example, we 'know' that floor(log(x, base = 8)) for x = 8 is 1, but 0 has been seen on an R platform. It is normally necessary to use a tolerance.

```
ttt <- nz_cards2 %>% arrange(month)
ttt[2179:4954,]
```

```
## # A tibble: 2,776 x 5
##    Period  year month Data_value UNITS
##     <dbl> <dbl> <dbl>      <dbl> <chr>
##  1  2001.  2001  3.00      2462. Dollars
##  2  2002.  2002  3.00     17177. Dollars
##  3  2003.  2003  3.00     22530. Dollars
##  4  2004.  2004  3.00     28005. Dollars
##  5  2005.  2005  3.00     30630. Dollars
##  6  2006.  2006  3.00     33317. Dollars
##  7  2007.  2007  3.00     36422  Dollars
```

```
##  8  2008.  2008  3.00     39198  Dollars
##  9  2009.  2009  3.00     40629. Dollars
## 10  2010.  2010  3.00     41815. Dollars
## # ... with 2,766 more rows
```

Question 11 (5 points)

Consider the flights data set (in the nycflights13 library). Use an ifelse statement to make a new column in the data set called pos_dep_delay; this column should be 0 if dep_delay is negative or 0, and equal to the dep_delay if the dep_delay is positive.

```
nycflights13::flights %>% mutate(pos_dep_delay = ifelse(dep_delay <= 0, 0, dep_delay))
```

```
## # A tibble: 336,776 x 20
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # ... with 336,766 more rows, and 12 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>,
## #   pos_dep_delay <dbl>
```