Homework 5

Sarah Cannon

2024-10-01

Classmates/other resources consulted: N/A

> **Be sure to load the tidyverse.**

```r
library(tidyverse)
```

Question 1 (6 points)

> **Parse the following dates and date/time combinations**

> a.

```r
d_a <- "Sep. 25, (2024)"
parse_datetime(d_a, "%b. %d, (%Y)")
```

```
## [1] "2024-09-25 UTC"
```

> b.

```r
d_b <- "2024-februar-12"
parse_datetime(d_b, "%Y-%B-%d", locale = locale(date_names = "de"))
```

```
## [1] "2024-02-12 UTC"
```

(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 (https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes))

> c.

```r
dt_c <- "February 13, 2023 at 7:45 am"
parse_datetime(dt_c, "%B %d, %Y at %I:%M %p")
```

```
## [1] "2023-02-13 07:45:00 UTC"
```

Question 2 (9 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, then parse all three columns so that they have the correct data types. You can either create new columns with the correct data types or replace the existing columns, it's up to you. Make sure the columns with the correct data types are the first three columns in your resulting tibble.**

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

```
## Rows: 40 Columns: 3
## ── Column specification ────────────────────────────────────────────────────────
────────────────────────────────────────────────────
## Delimiter: ","
## chr (3): Date_times, Dates, Times
##
## 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.
```

```
dates_times <- dates_times %>%
  mutate(
    Date_times_parsed = parse_datetime(Date_times, "%m-%d-%y: at %H:%M"),
    Dates_parsed = parse_date(Dates, "%d %B %Y", locale = locale(date_names = "fr")),
    Times_parsed = parse_time(Times, "%M minutes after %H %p")
  ) %>%
  select(Date_times_parsed, Dates_parsed, Times_parsed, everything())
dates_times
```

```
## # A tibble: 40 × 6
##    Date_times_parsed   Dates_parsed Times_parsed Date_times        Dates         Time
s
##    <dttm>              <date>       <time>       <chr>             <chr>         <chr
>
##  1 2021-08-01 16:43:00 2021-04-01   16:34        8-1-21: at 16:43  1 Avril 2021  34 m
inutes after 4pm
##  2 2021-08-02 17:16:00 2021-04-02   13:35        8-2-21: at 17:16  2 Avril 2021  35 m
inutes after 1pm
##  3 2021-08-03 16:32:00 2021-04-03   16:34        8-3-21: at 16:32  3 Avril 2021  34 m
inutes after 4pm
##  4 2021-08-04 16:32:00 2021-04-04   18:54        8-4-21: at 16:32  4 Avril 2021  54 m
inutes after 6pm
##  5 2021-08-05 16:23:00 2021-04-05   16:23        8-5-21: at 16:23  5 Avril 2021  23 m
inutes after 4pm
##  6 2021-08-06 17:12:00 2021-04-06   17:23        8-6-21: at 17:12  6 Avril 2021  23 m
inutes after 5pm
##  7 2021-08-07 16:55:00 2021-04-07   16:25        8-7-21: at 16:55  7 Avril 2021  25 m
inutes after 4pm
##  8 2021-08-08 16:35:00 2021-04-08   21:45        8-8-21: at 16:35  8 Avril 2021  45 m
inutes after 9pm
##  9 2021-08-09 17:01:00 2021-04-09   16:25        8-9-21: at 17:01  9 Avril 2021  25 m
inutes after 4pm
## 10 2021-08-10 17:49:00 2021-04-10   16:12        8-10-21: at 17:49 10 Avril 2021 12 m
inutes after 4pm
## # ℹ 30 more rows
```

Question 3 (4 points)

> **Import the attached data set “Monthly_amounts.txt”. The first column of this data set contains year and month information; parse it so that it is of the correct data type. While the original data has no day information, what happens in your new column?**

```
monthly_amounts <- read_csv("Monthly_Amounts.txt", skip = 1)
```

```
## Warning: One or more parsing issues, call `problems()` on your data frame for detail
s, e.g.:
##   dat <- vroom(...)
##   problems(dat)
```

```
## Rows: 20 Columns: 3
## ── Column specification ──────────────────────────────────────────────────────
────────────────────────────────────────────────────
## Delimiter: ","
## chr (3): Year_month, largest_amount, average_amount
##
## 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.
```

```
mutate(monthly_amounts, Year_month_parsed = parse_date(Year_month, "%Y-%m"))
```

```
## # A tibble: 20 × 4
##    Year_month largest_amount average_amount Year_month_parsed
##    <chr>      <chr>          <chr>          <date>
##  1 2023-1     63             45.7           2023-01-01
##  2 2023-2     56             35.2           2023-02-01
##  3 2023-3     54             34.2           2023-03-01
##  4 2023-4     12             11.6           2023-04-01
##  5 2023-5     <NA>           35.2           2023-05-01
##  6 2023-6     87             56.9           2023-06-01
##  7 2023-7     82             45.7           2023-07-01
##  8 2023-8     36             17.9           2023-08-01
##  9 2023-9     98             54,3           2023-09-01
## 10 2023-10    16             9.8            2023-10-01
## 11 2023-11    78             50.2           2023-11-01
## 12 2023-12    45             43.6           2023-12-01
## 13 2024-1     35             31.2           2024-01-01
## 14 2024-2     91             -              2024-02-01
## 15 2024-3     45             34.5           2024-03-01
## 16 2024-4     67             34.6           2024-04-01
## 17 2024-5     34             21.9           2024-05-01
## 18 2024-6     *              49.8           2024-06-01
## 19 2024-7     35             23.8           2024-07-01
## 20 2024-8     45             43             2024-08-01
```

The new column will have the first day of the month as the default day. Assuming we can ignore the warning coming from the extra column in one row.

Question 4 (6 points)

> a. **Create a string in R containing the following sentence, including its punctuation: It’s sunny today, but he said, “It’ll be rainy tomorrow.” To be sure you’ve made the correct string, print it out using the writeLines() function.**

```
sentence <- "It's sunny today, but he said, \"It'll be rainy tomorrow.\""
writeLines(sentence)
```

```
## It's sunny today, but he said, "It'll be rainy tomorrow."
```

> b. **Explain the difference between the strings: "a b" and "a n b". The answer is not just that one has an extra space and one doesn't. Your explanation should mention escape characters. Note: to make a backslash show up in your explanation in your knitted file, you must use two backslashes, like \. Be sure to check your knitted file to make sure all parts of your explanation are appearing correctly.**

The string "a \n b" contains an escape character `\n`, which represents a newline. Therefore, when printed, it will display as:

a

b

On the other hand, "a \ n b" does not contain a valid escape character due to the space after the backslash, so it will be printed as is, with a space and a backslash:

a \ n b

Question 5 (4 points)

> **This question references the following strings**

```
s1 <- "the cat, gracie, is sleepy"
```

```
s2 <- "The Dog Is Sleepy Too!"
```

> a. **Make s1 uppercase**

```
s1_upper <- str_to_upper(s1)
s1_upper
```

```
## [1] "THE CAT, GRACIE, IS SLEEPY"
```

> b. **Make s2 lowercase**

```
s2_lower <- str_to_lower(s2)
s2_lower
```

```
## [1] "the dog is sleepy too!"
```

> c. **Make the first letter of every word in s1 capitalized, while all other letters are lowercase.**

```
s1_title <- str_to_title(s1)
s1_title
```

```
## [1] "The Cat, Gracie, Is Sleepy"
```

> d. **Write a command that will output the number of characters in string s2.**

```
s2_length <- str_length(s2)
s2_length
```

```
## [1] 22
```

Question 6 (6 points)

> **In the U.S., mailing addresses have zipcodes consisting of five digits, then a dash, then four digits. An example might be 91711-4285. Suppose you have a tibble, like the following example, where the first five digits are in a different column than the last four digits.**

```
zip_codes <- tibble(Zip = c("91711-3452", "20322-3009", "93782-8473", "78392-8762", "876
39-2563", "47628-5416", "20874-5726"))
```

> a. **Use the str_sub() function to split the Zip column into two columns, one with the first five digits of the zip code and one with the last four digits of the zip code.**

```
zip_codes <- zip_codes %>%
  mutate(
    first_five = str_sub(Zip, 1, 5),
    last_four = str_sub(Zip, 7, 10)
  )
zip_codes
```

```
## # A tibble: 7 × 3
##   Zip        first_five last_four
##   <chr>      <chr>      <chr>
## 1 91711-3452 91711      3452
## 2 20322-3009 20322      3009
## 3 93782-8473 93782      8473
## 4 78392-8762 78392      8762
## 5 87639-2563 87639      2563
## 6 47628-5416 47628      5416
## 7 20874-5726 20874      5726
```

> b. **Use the separate() function to split the Zip column into two columns, one with the first five digits of the zip code and one with the last four digits of the zip code.**

```
zip_codes <- zip_codes %>%
  separate(Zip, into = c("first_five", "last_four"), sep = "-")
zip_codes
```

```
## # A tibble: 7 × 2
##   first_five last_four
##   <chr>      <chr>
## 1 91711      3452
## 2 20322      3009
## 3 93782      8473
## 4 78392      8762
## 5 87639      2563
## 6 47628      5416
## 7 20874      5726
```

Question 7 (8 points)

> **At a particular company, an employee's email address consists of their first initial, their middle initial (if they have one), their last name, and the last two digits of their Employee ID number, followed by "@company.com". For example, for an employee Alice A. Smith with employee ID number 45398545, her email address would be AASmith45@company.com (mailto:AASmith45@company.com). For an employee Bob Jones (who does not have a middle initial) with employee ID number 345582, his email address would be BJones82@company.com (mailto:BJones82@company.com). For the table below, write code to add a new column consisting of each employee's email address, computed from the values in the other columns.**

```
employees <- tibble(
  FirstName = c("Alice", "Bob", "Simba", "Nala", "Timon", "Pumbaa", "Rafiki", "Scar"),
  MiddleInitial = c("A", NA, "E", "Q", "P", NA, "P", "L"),
  LastName = c("Smith", "Jones", "Clark", "Davis", "Evans", "Frank", "Ghosh", "Hills"),
  EmployeeID = c(45398545, 345582, 2354463, 345346, 2346377022, 20345423, 20223454, 2042
54))
employees
```

```
## # A tibble: 8 × 4
##   FirstName MiddleInitial LastName EmployeeID
##   <chr>     <chr>         <chr>         <dbl>
## 1 Alice     A             Smith      45398545
## 2 Bob       <NA>          Jones        345582
## 3 Simba     E             Clark       2354463
## 4 Nala      Q             Davis        345346
## 5 Timon     P             Evans    2346377022
## 6 Pumbaa    <NA>          Frank      20345423
## 7 Rafiki    P             Ghosh      20223454
## 8 Scar      L             Hills        204254
```

```
employees <- employees %>%
  mutate(
    Email = str_c(
      str_sub(FirstName, 1, 1),
      ifelse(is.na(MiddleInitial), "", MiddleInitial),
      LastName,
      str_sub(as.character(EmployeeID), -2, -1),
      "@company.com"
    )
  )
employees
```

```
## # A tibble: 8 × 5
##   FirstName MiddleInitial LastName EmployeeID Email
##   <chr>     <chr>         <chr>         <dbl> <chr>
## 1 Alice     A             Smith      45398545 AASmith45@company.com
## 2 Bob       <NA>          Jones        345582 BJones82@company.com
## 3 Simba     E             Clark       2354463 SEClark63@company.com
## 4 Nala      Q             Davis        345346 NQDavis46@company.com
## 5 Timon     P             Evans    2346377022 TPEvans22@company.com
## 6 Pumbaa    <NA>          Frank      20345423 PFrank23@company.com
## 7 Rafiki    P             Ghosh      20223454 RPGhosh54@company.com
## 8 Scar      L             Hills        204254 SLHills54@company.com
```

Question 8 (12 points)

> **In each part, say whether the data is tidy or not, and explain why.**

> a.

```
## # A tibble: 10 × 4
##    Team_Abbreviation Team_Name          Division `Wins-Losses`
##    <chr>             <chr>              <chr>    <chr>
##  1 TB                Tampa Bay Rays     East     100-62
##  2 BOS               Boston Red Sox     East     92-70
##  3 NYY               New York Yankees   East     92-70
##  4 TOR               Toronto Blue Jays  East     91-71
##  5 BAL               Baltimore Orioles  East     52-110
##  6 CHW               Chicago White Sox  Central  93-69
##  7 CLE               Cleveland Indians  Central  80-82
##  8 DET               Detroit Tigers     Central  77-85
##  9 KC                Kansas City Royals Central  74-88
## 10 MIN               Minnesota Twins    Central  73-89
```

This data is not tidy because the `Wins-Losses` column contains two values (wins and losses) in a single entry. Each column should contain only one value.

> b.

```
## # A tibble: 10 × 4
##    Name      College Info       Value
##    <chr>     <chr>   <chr>      <dbl>
##  1 Student A CMC     GPA         3.8
##  2 Student B CMC     GPA         3.7
##  3 Student C Pitzer  GPA         3.72
##  4 Student D CMC     GPA         3.66
##  5 Student E Scripps GPA         3.72
##  6 Student A CMC     Graduation 2022
##  7 Student B CMC     Graduation 2024
##  8 Student C Pitzer  Graduation 2023
##  9 Student D CMC     Graduation 2023
## 10 Student E Scripps Graduation 2023
```

This data is not tidy because each student has multiple rows for different types of information (GPA, Graduation). Each row should represent a single observation (a student), and each column should represent a single variable.

> c.

```
## # A tibble: 7 × 5
##   Day       Temperature_F Wind_mph UV_index ChanceOfRain_percent
##   <chr>             <dbl>    <dbl>    <dbl>                <dbl>
## 1 Thursday             71        7        2                   70
## 2 Friday               63       12        4                   80
## 3 Saturday             71       10        7                    4
## 4 Sunday               78       11        7                    0
## 5 Monday               71       13        6                    7
## 6 Tuesday              70       11        6                    0
## 7 Wednesday            74       10        6                    0
```

This data is tidy because each row represents an observation, each column represents a variable, and each entry contains only one value.

> d.

```
## # A tibble: 5 × 6
##   cut       extra_small small medium large extra_large
##   <chr>           <dbl> <dbl>  <dbl> <dbl>       <dbl>
## 1 Fair               32    23     34    23          34
## 2 Good               45    45     45    21          56
## 3 Very Good          67    26     63    43          23
## 4 Premium            32    78     78    47          14
## 5 Ideal              14    23     99    21          21
```

This data is tidy because each row represents an observation (a diamond cut), and each column represents a variable.

Question 9 (12 points)

For each of these tibbles, perform the necessary operation to make it tidy.

a.

```
## # A tibble: 12 × 3
##    month     metric           average
##    <chr>     <chr>              <dbl>
##  1 September high_temperature   89
##  2 September low_temperature    60
##  3 September rain_inches         0.15
##  4 September daylight_hours     12.5
##  5 October   high_temperature   80
##  6 October   low_temperature    55
##  7 October   rain_inches         1.05
##  8 October   daylight_hours     11.5
##  9 November  high_temperature   74
## 10 November  low_temperature    47
## 11 November  rain_inches         1.62
## 12 November  daylight_hours     10.5
```

```
avg_weather_tidy <- avg_weather %>%
  pivot_wider(names_from = metric, values_from = average)
avg_weather_tidy
```

```
## # A tibble: 3 × 5
##   month     high_temperature low_temperature rain_inches daylight_hours
##   <chr>                <dbl>           <dbl>       <dbl>          <dbl>
## 1 September               89              60        0.15           12.5
## 2 October                 80              55        1.05           11.5
## 3 November                74              47        1.62           10.5
```

b.

```
## # A tibble: 6 × 2
##   Chemical_Name Safe_Temperature_Range
##   <chr>         <chr>
## 1 Chemical 1    32-212
## 2 Chemical 2    50-100
## 3 Chemical 3    45-48
## 4 Chemical 4    40-345
## 5 Chemical 5    100-250
## 6 Chemical 6    112-140
```

```
chemicals_tidy <- chemicals %>%
  separate(Safe_Temperature_Range, into = c("Min_Temperature", "Max_Temperature"), sep =
"-")
chemicals_tidy
```

```
## # A tibble: 6 × 3
##   Chemical_Name Min_Temperature Max_Temperature
##   <chr>         <chr>           <chr>
## 1 Chemical 1    32              212
## 2 Chemical 2    50              100
## 3 Chemical 3    45              48
## 4 Chemical 4    40              345
## 5 Chemical 5    100             250
## 6 Chemical 6    112             140
```

c. **Information about the following data is available at https://github.com/rfordatascience/tidytuesday/blob/master/data/2023/2023-07-11/readme.md (https://github.com/rfordatascience/tidytuesday/blob/master/data/2023/2023-07-11/readme.md)**

```
## Rows: 144 Columns: 19
## ── Column specification ────────────────────────────────────────────────────────
────────────────────────────────────────────────────
## Delimiter: ","
## dbl (19): Year, Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec, J-D, D-N,
DJF, MAM, JJA, SON
##
## 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: 144 × 13
##     Year   Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1  1880 -0.19 -0.25 -0.09 -0.17 -0.1  -0.21 -0.18 -0.11 -0.15 -0.24 -0.22 -0.18
##  2  1881 -0.2  -0.15  0.03  0.05  0.05 -0.19  0    -0.04 -0.16 -0.22 -0.19 -0.08
##  3  1882  0.16  0.13  0.04 -0.16 -0.14 -0.22 -0.17 -0.08 -0.15 -0.24 -0.17 -0.36
##  4  1883 -0.3  -0.37 -0.13 -0.19 -0.18 -0.08 -0.08 -0.14 -0.23 -0.12 -0.24 -0.11
##  5  1884 -0.13 -0.09 -0.37 -0.4  -0.34 -0.35 -0.31 -0.28 -0.28 -0.25 -0.34 -0.31
##  6  1885 -0.59 -0.34 -0.27 -0.42 -0.45 -0.44 -0.34 -0.32 -0.29 -0.24 -0.24 -0.11
##  7  1886 -0.44 -0.51 -0.43 -0.28 -0.24 -0.35 -0.18 -0.31 -0.24 -0.28 -0.28 -0.26
##  8  1887 -0.72 -0.57 -0.36 -0.35 -0.31 -0.25 -0.26 -0.36 -0.26 -0.36 -0.27 -0.33
##  9  1888 -0.34 -0.36 -0.41 -0.2  -0.22 -0.17 -0.11 -0.16 -0.12  0.01  0.03 -0.04
## 10  1889 -0.09  0.16  0.06  0.1  -0.01 -0.1  -0.08 -0.2  -0.24 -0.25 -0.33 -0.29
## # ℹ 134 more rows
```

```
global_temps_tidy <- global_temps %>%
  pivot_longer(cols = Jan:Dec, names_to = "Month", values_to = "Temperature")
global_temps_tidy
```

```
## # A tibble: 1,728 × 3
##     Year Month Temperature
##    <dbl> <chr>       <dbl>
##  1  1880 Jan         -0.19
##  2  1880 Feb         -0.25
##  3  1880 Mar         -0.09
##  4  1880 Apr         -0.17
##  5  1880 May         -0.1
##  6  1880 Jun         -0.21
##  7  1880 Jul         -0.18
##  8  1880 Aug         -0.11
##  9  1880 Sep         -0.15
## 10  1880 Oct         -0.24
## # ℹ 1,718 more rows
```

Question 10 (9 points)

> a. **Consider the following example table.**

```
diamonds_counts <- tibble(cut = c("Fair", "Good", "Very Good", "Premium", "Ideal"),
                          extra_small = c(32,45,67,32,14),
                          small = c(23,45,26,78,23),
                          medium = c(34,45,63,78,99),
                          large = c(23,21,43,47,21),
                          extra_large = c(34,56,23,14,21))
diamonds_counts
```

```
## # A tibble: 5 × 6
##   cut       extra_small small medium large extra_large
##   <chr>           <dbl> <dbl>  <dbl> <dbl>       <dbl>
## 1 Fair               32    23     34    23          34
## 2 Good               45    45     45    21          56
## 3 Very Good          67    26     63    43          23
## 4 Premium            32    78     78    47          14
## 5 Ideal              14    23     99    21          21
```

> **Explain in your own words why the following two code chunks produce the same tibble.**

```r
diamonds_counts %>%
  pivot_longer(extra_small:extra_large, names_to = "Size", values_to = "Count") %>%
  filter(Size %in% c("extra_small", "extra_large"))
```

```
## # A tibble: 10 × 3
##    cut       Size        Count
##    <chr>     <chr>       <dbl>
##  1 Fair      extra_small    32
##  2 Fair      extra_large    34
##  3 Good      extra_small    45
##  4 Good      extra_large    56
##  5 Very Good extra_small    67
##  6 Very Good extra_large    23
##  7 Premium   extra_small    32
##  8 Premium   extra_large    14
##  9 Ideal     extra_small    14
## 10 Ideal     extra_large    21
```

```r
diamonds_counts %>%
  select(cut, extra_small, extra_large) %>%
  pivot_longer(extra_small:extra_large, names_to = "Size", values_to = "Count")
```

```
## # A tibble: 10 × 3
##    cut       Size        Count
##    <chr>     <chr>       <dbl>
##  1 Fair      extra_small    32
##  2 Fair      extra_large    34
##  3 Good      extra_small    45
##  4 Good      extra_large    56
##  5 Very Good extra_small    67
##  6 Very Good extra_large    23
##  7 Premium   extra_small    32
##  8 Premium   extra_large    14
##  9 Ideal     extra_small    14
## 10 Ideal     extra_large    21
```

Both code chunks produce the same tibble because they both transform the data from wide to long format, focusing only on the `extra_small` and `extra_large` columns. They just do it in a different order with the first code chunk filtering the data after it's pivoted and the second code chunk filtering/selecting the data before it's pivoted.

> b. **Why doesn't the following code work as expected? Explain what went wrong here, and why the pivot_wider function doesn't work for this data set in the same way we learned in class.**

```
cats <- tribble(
  ~name,             ~names,  ~values,
  "Gracie the Cat",   "age",       6.5,
  "Gracie the Cat",   "height_in",   14,
  "Gracie the Cat",   "age",       5,
  "Patches the Cat", "age",       2,
  "Patches the Cat", "height_in",   11
)

newtibble <- cats %>% pivot_wider(names_from = names, values_from = values)
```

```
## Warning: Values from `values` are not uniquely identified; output will contain list-c
ols.
## • Use `values_fn = list` to suppress this warning.
## • Use `values_fn = {summary_fun}` to summarise duplicates.
## • Use the following dplyr code to identify duplicates.
##   {data} |>
##   dplyr::summarise(n = dplyr::n(), .by = c(name, names)) |>
##   dplyr::filter(n > 1L)
```

```
newtibble
```

```
## # A tibble: 2 × 3
##   name            age       height_in
##   <chr>           <list>    <list>
## 1 Gracie the Cat  <dbl [2]> <dbl [1]>
## 2 Patches the Cat <dbl [1]> <dbl [1]>
```

The code doesn't work because there are 2 `age` values for Gracie the Cat but the `pivot_wider` function requires unique combinations of the `name` and `names` columns.

> c. **Look up what the tidyverse's spread() and gather() functions do and explain them below. These functions are no longer under active development, but exist in a lot of previously written code. Which functions we've learned recently are the updated versions of spread and gather?**

The `spread()` function in the tidyverse is used to turn key-value pairs into columns, effectively making the data wider. The `gather()` function is used to turn columns into key-value pairs, making the data longer. They functions have been obsoleted by `pivot_wider()` and `pivot_longer()`.

Question 11 (12 points)

> **Consider the following data set; more information is available at https://github.com/rfordatascience/tidytuesday/blob/master/data/2020/2020-03-24/readme.md (https://github.com/rfordatascience/tidytuesday/blob/master/data/2020/2020-03-24/readme.md)**

```
brain_injuries <- read_csv("https://raw.githubusercontent.com/rfordatascience/tidytuesda
y/master/data/2020/2020-03-24/tbi_age.csv") %>% select(age_group:number_est) %>% filter
(age_group != "Total")
```

```
## Rows: 231 Columns: 5
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr (3): age_group, type, injury_mechanism
## dbl (2): number_est, rate_est
##
## 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.
```

```
brain_injuries
```

```
## # A tibble: 210 × 4
##    age_group type                          injury_mechanism
number_est
##    <chr>     <chr>                         <chr>
<dbl>
##  1 0-17      Emergency Department Visit Motor Vehicle Crashes
47138
##  2 0-17      Emergency Department Visit Unintentional Falls
397190
##  3 0-17      Emergency Department Visit Unintentionally struck by or against an objec
t        229236
##  4 0-17      Emergency Department Visit Other unintentional injury, mechanism unspeci
fied      55785
##  5 0-17      Emergency Department Visit Intentional self-harm
NA
##  6 0-17      Emergency Department Visit Assault
24360
##  7 0-17      Emergency Department Visit Other or no mechanism specified
57983
##  8 0-4       Emergency Department Visit Motor Vehicle Crashes
5464
##  9 0-4       Emergency Department Visit Unintentional Falls
230776
## 10 0-4       Emergency Department Visit Unintentionally struck by or against an objec
t          53436
## # ℹ 200 more rows
```

> a. (3 points) **Make a table that displays all the different values in the age_group category and how many times each appears in the data set.**

```
brain_injuries %>%
  group_by(age_group) %>%
  summarize(count = n())
```

```
## # A tibble: 10 × 2
##    age_group count
##    <chr>     <int>
##  1 0-17         21
##  2 0-4          21
##  3 15-24        21
##  4 25-34        21
##  5 35-44        21
##  6 45-54        21
##  7 5-14         21
##  8 55-64        21
##  9 65-74        21
## 10 75+          21
```

b. (3 points) **Some of these age groups are overlapping, which is not ideal for data analysis. For example, its currently very challenging to look in-depth at the age groups 15-17 and 18-24. To simplify the age groups, it's actually easiest if they're columns! Use a function we learned this week to turn the age_group values into columns, where the values in those columns come from the number_est column.**

```
brain_injuries %>%
  pivot_wider(names_from = age_group, values_from = number_est)
```

```
## # A tibble: 21 × 12
##    type                         injury_mechanism              `0-17`  `0-4` `5-14` `15
-24` `25-34` `35-44` `45-54` `55-64` `65-74`  `75+`
##    <chr>                        <chr>                          <dbl>  <dbl>  <dbl>   <
dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>  <dbl>
##  1 Emergency Department Visit Motor Vehicle Crashes           47138   5464  19785  10
3892   71641   44108   40020   27193   13829   8176
##  2 Emergency Department Visit Unintentional Falls            397190 230776 133084   9
6568   70210   68830   95127  112460  120327 286031
##  3 Emergency Department Visit Unintentionally struck by or … 229236  53436 120839  10
6679   44404   32479   30495   20408   11937  13270
##  4 Emergency Department Visit Other unintentional injury, m…  55785  12007  30656   3
7118   22360   17541   17808   12928    7077   7440
##  5 Emergency Department Visit Intentional self-harm              NA     NA     NA
870     650     421     247     105      NA     NA
##  6 Emergency Department Visit Assault                         24360    674   9690   6
5399   57213   34100   27682   11538    2893   1260
##  7 Emergency Department Visit Other or no mechanism specifi…  57983  19360  26022   3
3395   20974   16503   15962   13387   10051  17318
##  8 Hospitalizations           Motor Vehicle Crashes            5830    870   2395   1
2925   11050    7305    8490    7280    4485   3965
##  9 Hospitalizations           Unintentional Falls              7935   4700   2270
3910    4470    5640   12010   18490   25235  74005
## 10 Hospitalizations           Unintentionally struck by or …   1985    510    980
1070     635     610     685     765     790   1045
## # ℹ 11 more rows
```

c. (3 points) **By adding and subtracting columns from the table you produced in the previous part, make new columns for ages 15-17 and 18-24. Hint: the number_est for the 15-17 age group is the estimate for 0-17 minus the estimate for 0-4 and the estimate for 5-14. Keep only the type column, injury_mechanism column, and the columns for the age groups 0-4, 5-14, 15-17, and 18-24.**

```
brain_injuries_wide <- brain_injuries %>%
  pivot_wider(names_from = age_group, values_from = number_est) %>%
  mutate(
    `15-17` = `0-17` - `0-4` - `5-14`,
    `18-24` = `0-4` + `5-14` + `15-24` - `0-17`
  ) %>%
  select(type, injury_mechanism, `0-4`, `5-14`, `15-17`, `18-24`)
brain_injuries_wide
```

```
## # A tibble: 21 × 6
##    type                       injury_mechanism                                 `0-4
` `5-14` `15-17` `18-24`
##    <chr>                      <chr>                                            <dbl
>  <dbl>   <dbl>   <dbl>
##  1 Emergency Department Visit Motor Vehicle Crashes                             546
4  19785   21889   82003
##  2 Emergency Department Visit Unintentional Falls                             23077
6 133084   33330   63238
##  3 Emergency Department Visit Unintentionally struck by or against an object    5343
6 120839   54961   51718
##  4 Emergency Department Visit Other unintentional injury, mechanism unspecified  1200
7  30656   13122   23996
##  5 Emergency Department Visit Intentional self-harm                                N
A     NA      NA      NA
##  6 Emergency Department Visit Assault                                            67
4   9690   13996   51403
##  7 Emergency Department Visit Other or no mechanism specified                   1936
0  26022   12601   20794
##  8 Hospitalizations           Motor Vehicle Crashes                              87
0   2395    2565   10360
##  9 Hospitalizations           Unintentional Falls                               470
0   2270     965    2945
## 10 Hospitalizations           Unintentionally struck by or against an object      51
0    980     495     575
## # ℹ 11 more rows
```

> d. (3 points) **This data is not tidy! Use an appropriate function we learned this week to make the data tidy.**

```
brain_injuries_tidy <- brain_injuries_wide %>%
  pivot_longer(cols = `0-4`: `18-24`, names_to = "age_group", values_to = "number_est")
brain_injuries_tidy
```

```
## # A tibble: 84 × 4
##    type                       injury_mechanism                              age_grou
p number_est
##    <chr>                      <chr>                                         <chr>
<dbl>
##  1 Emergency Department Visit Motor Vehicle Crashes                         0-4
5464
##  2 Emergency Department Visit Motor Vehicle Crashes                         5-14
19785
##  3 Emergency Department Visit Motor Vehicle Crashes                         15-17
21889
##  4 Emergency Department Visit Motor Vehicle Crashes                         18-24
82003
##  5 Emergency Department Visit Unintentional Falls                           0-4
230776
##  6 Emergency Department Visit Unintentional Falls                           5-14
133084
##  7 Emergency Department Visit Unintentional Falls                           15-17
33330
##  8 Emergency Department Visit Unintentional Falls                           18-24
63238
##  9 Emergency Department Visit Unintentionally struck by or against an object 0-4
53436
## 10 Emergency Department Visit Unintentionally struck by or against an object 5-14
120839
## # ℹ 74 more rows
```

Question 12 (8 points)

> **Consider the file GPAs.csv. Import this data set, and carefully make it both clean and tidy. There will be several steps involved in this process. (the GPAs were randomly generated and do not reflect actual student grades).**

```
gpa <- read_csv("GPAs.csv")
```

```
## Rows: 19 Columns: 5
## ── Column specification ─────────────────────────────────────────────────────────────
────────────────────────────────────────────────────
## Delimiter: ","
## chr (2): College, Class
## dbl (3): Math_GPA, Data Science_GPA, Other_GPA
##
## 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.
```

```
gpa
```

```
## # A tibble: 19 × 5
##    College Class     Math_GPA `Data Science_GPA` Other_GPA
##    <chr>   <chr>        <dbl>              <dbl>     <dbl>
##  1 CMC     Freshman      3.74               3.84      3.78
##  2 CMC     Sophomore     3.09               3.14      3.16
##  3 CMC     Junior        3.74               3.84      3.94
##  4 CMC     Senior        3.63               3.96      3.4
##  5 HMC     freshman      3.26               3.11      3.79
##  6 HMC     sophomore     3.58               3.78      3.64
##  7 HMC     senior        3.93               3.02      3.02
##  8 Pitzer  Freshman      3.33               3.46      3.96
##  9 Pitzer  Sophomore     3.01               3.93      3.9
## 10 Pitzer  Junior        3.87               3.74      3.19
## 11 Pitzer  Senior        3.25               3.41      3.23
## 12 Pomona  freshman      3.7                3.7       3.32
## 13 Pomona  sophomore     3.4                3.31      3.9
## 14 Pomona  junior        3.24               3.82      3.25
## 15 Pomona  senior        3.23               3.72      3.51
## 16 Scripps Freshman      3.6                3.75      3.04
## 17 Scripps Sophomore     3.51               3.34      3.14
## 18 Scripps Junior        3.68               3.45      3.2
## 19 Scripps Senior        3.15               3.53      3.53
```

```
gpa <- read_csv("GPAs.csv")
```

```
## Rows: 19 Columns: 5
## ── Column specification ──────────────────────────────────────────────────────────────
──────────────────────────────────────────────────
## Delimiter: ","
## chr (2): College, Class
## dbl (3): Math_GPA, Data Science_GPA, Other_GPA
##
## 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.
```

```
gpa <- gpa %>%
  rename(Data_Science_GPA = `Data Science_GPA`) %>%
  mutate(Class = str_to_title(Class)) %>%
  complete(College, Class, fill = list(Grade = NA)) %>%
  pivot_longer(cols = Math_GPA:Other_GPA, names_to = "Type", values_to = "GPA") %>%
  mutate(Type = str_replace(Type, "_GPA", ""))
gpa
```

```
## # A tibble: 60 × 4
##    College Class     Type           GPA
##    <chr>   <chr>     <chr>        <dbl>
##  1 CMC     Freshman  Math          3.74
##  2 CMC     Freshman  Data_Science  3.84
##  3 CMC     Freshman  Other         3.78
##  4 CMC     Junior    Math          3.74
##  5 CMC     Junior    Data_Science  3.84
##  6 CMC     Junior    Other         3.94
##  7 CMC     Senior    Math          3.63
##  8 CMC     Senior    Data_Science  3.96
##  9 CMC     Senior    Other         3.4 
## 10 CMC     Sophomore Math          3.09
## # ℹ 50 more rows
```

Question 13 (4 points)

> a. (2 points) **Reflect on how this class is going for you so far - What's working well for you? What are you finding challenging? What changes can you make that might help improve your learning?**

I'm understanding the concepts of the course which is good. Doing the homework quickly is a challenge. I think it's more an issue of study habbits and time managment since I tend to procrastinate.

> b. (2 points) **With the first test coming up, how do you plan to study? What methods of studying do you think will be most effective? Outline a study plan here.**

I plan to study by systemattically reviewing the class notes and activities. I will also colaborate with my homework group to review key concepts and help each other understand the material.