Section Assignment 4

Liam Mueller

01/23/2022

Please Read!

This week, your assignment is to work with the tidyverse package to rearrange and manipulate data tables. Just like last week the first thing we need to do is install tidyverse:

```
install.packages("tidyverse")
```

But that line only needs to be run once! Now load the tidyverse functions into your working library:

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.3     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

This line of code above will need to be run each time you start a new R session.

Just like last week, one of the traits of a great programmer is their ability to solve a problem they haven't seen before. One of the best ways to solve a problem you have not seen before is to see if anyone else has. Great programmers are experts of web searching. As we are working on becoming expert programmers, this week I have included the link to a set of Rstudio cheat sheets that have high information density all about the different functionalities in the tidyverse. The two most helpful for this week's assignment are the sheets on dplyr and tidyr. https://www.rstudio.com/resources/cheatsheets/

Copying from the internet is one of the foundations of learning how to program, but it only works as learning if you reflect on why the code you used works. For that reason, this week and in future weeks, you will need to annotate your code. Use the lines that look like this at the end of each question to input your explanation of the code:

```
note<-'your note here'
print(note)
```

You will be graded on not just your tables, but your explanation. Remember, in this class, it is okay to copy code, but you still need to demonstrate independent thought.

Questions:

Question 1. (1 point) The following data table (BadTable) is an untidy display of the population estimate for two different years in three large Texas cities. Use the `pivot_longer()` function to assign to the GoodTable object a table with 3 columns and 6 rows. Hint, `names_to = "Year"` and `values_to = "population"`. For more hints look at the tidyr cheat sheet under "Reshape Data".

```
BadTable<-read_csv(file= "City,2014,2019
                         San Antonio,1436697,1547253
                         Houston,2239558,2320268
                         Austin,912791,978908")

GoodTable<-pivot_longer(BadTable,cols = 2:3,names_to = "Year",values_to = "Population")

head(GoodTable)
```

```
## # A tibble: 6 x 3
##   City        Year  Population
##   <chr>       <chr>      <dbl>
## 1 San Antonio 2014     1436697
## 2 San Antonio 2019     1547253
## 3 Houston     2014     2239558
## 4 Houston     2019     2320268
## 5 Austin      2014      912791
## 6 Austin      2019      978908
```

Add your notes for Q1 below, remember the text you type has to be in the quotation marks!: [1] " "

Question 2.(1 point): Extract the rows from "GoodTable" where population is less than 1500000. Hint: Page 1 of the dplyr cheat sheet, "Manipulate Cases"

```
SmallerCities<-filter(GoodTable,Population<1500000)
head(SmallerCities)
```

```
## # A tibble: 3 x 3
##   City        Year  Population
##   <chr>       <chr>      <dbl>
## 1 San Antonio 2014     1436697
## 2 Austin      2014      912791
## 3 Austin      2019      978908
```

Add your notes for Q2 below, remember the text you type has to be in the quotation marks!: [1] " "

Question 3. (1 point): Rearrange "GoodTable" by high to low population. Hint: Page 1 of the dplyr cheat sheet, "Manipulate Cases"

```
OrderedGoodTable<-arrange(GoodTable,desc(Population))
head(OrderedGoodTable)
```

```
## # A tibble: 6 x 3
##   City        Year  Population
##   <chr>       <chr>      <dbl>
## 1 Houston     2019     2320268
## 2 Houston     2014     2239558
## 3 San Antonio 2019     1547253
## 4 San Antonio 2014     1436697
## 5 Austin      2019      978908
## 6 Austin      2014      912791
```

Add your notes for Q3 below, remember the text you type has to be in the quotation marks!: [1] " "

Question 4 (1 point): To really get a feel for the tidyverse, we need to start working with bigger data sets. The following two data tables are subsets from the flights and weather data in the "nycflights2013" package. Download The "flights.csv" and "JFKWeather.csv" files from this week's section activity on Canvas. "flights.csv" contains flight information for the 28 Hawaiian Airlines flights that departed JFK airport in February 2013. The "JFKWeather.csv" contains hourly weather data from JFK airports for the month of February 2013. Read these .csv files into R below:

```
flights<-read.csv("flights.csv")

weather<-read.csv("JFKWeather.csv")
```

Add your notes for Q4 below, remember the text you type has to be in the quotation marks!: [1] " "

Question 5 (3 points): Our goal is to combine the flight and weather data tables into one large table containing the flight information and the weather data captured the hour each flight was scheduled to take off. If you do this correctly, the resulting table should have exactly 28 rows and 23 columns. Hint: `inner_join()` on page 2 of the dplyr cheat sheet and the sections titled "RELATIONAL DATA" and "COLUMN MATCHING FOR JOINS". Hint 2: Both of these data tables share the "time_hour" column.

```
CombinedTable<-inner_join(x = flights,y =weather,by="time_hour" )
str(CombinedTable)
```

```
## 'data.frame':    28 obs. of  23 variables:
##  $ month         : int  2 2 2 2 2 2 2 2 2 2 ...
##  $ day.x         : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ dep_time      : int  855 858 857 857 900 853 900 854 1206 857 ...
##  $ dep_delay     : int  -5 -2 -3 -3 0 -7 0 -6 186 -3 ...
##  $ arr_time      : int  1442 1440 1446 1521 1555 1542 1528 1501 1814 1440 ...
##  $ sched_arr_time: int  1540 1540 1540 1540 1540 1540 1540 1540 1540 1540 ...
##  $ arr_delay     : int  -58 -60 -54 -19 15 2 -12 -39 154 -60 ...
##  $ tailnum       : chr  "N388HA" "N388HA" "N382HA" "N380HA" ...
##  $ origin        : chr  "JFK" "JFK" "JFK" "JFK" ...
##  $ dest          : chr  "HNL" "HNL" "HNL" "HNL" ...
##  $ air_time      : int  620 629 614 655 679 691 652 609 645 605 ...
##  $ time_hour     : chr  "2/1/2013 9:00" "2/2/2013 9:00" "2/3/2013 9:00" "2/4/2013 9:00" ...
##  $ day.y         : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ hour          : int  9 9 9 9 9 9 9 9 9 9 ...
##  $ temp          : num  30 24.1 26.6 27 30 ...
##  $ dewp          : num  8.06 6.98 17.96 6.08 24.98 ...
##  $ humid         : num  39 47.5 71 40.5 81.3 ...
##  $ wind_dir      : int  290 250 40 280 50 290 30 60 310 330 ...
##  $ wind_speed    : num  14.96 8.06 9.21 19.56 8.06 ...
##  $ wind_gust     : num  NA NA NA 29.9 NA ...
##  $ precip        : num  0 0 0 0 0 0 0 0.03 0 0 ...
##  $ pressure      : num  1012 1026 NA 1015 1015 ...
##  $ visib         : num  10 10 10 10 7 10 10 6 10 10 ...
```

Add your notes for Q5 below, remember the text you type has to be in the quotation marks!: [1] " "

Question 6 (3 points): Let's say, instead of the weather at the time of the flight, we want to know the average temperature each day in February 2013, at JFK. You will need to combine the `summarise()` and `group_by()` functions with the "weather" data you created back in question 4 to achieve a data table with 28 rows and two columns (day and average temp). Hint: Look at the left column on the first page of the dplyr cheat sheet. Hint 2: Your answer will use "%>%" twice. These are called pipes and allow you to perform sequential operations within a single command. Essentially, "%>%" acts like "and then do". For example: "A %>% B %>% C" really means "take A and then do B to it and then do C to it".

```
MeanDailyTemp<- weather %>%
              group_by(day)%>%
                summarise(avg=mean(temp))
str(MeanDailyTemp)
```

```
## tibble [28 x 2] (S3: tbl_df/tbl/data.frame)
##  $ day: int [1:28] 1 2 3 4 5 6 7 8 9 10 ...
##  $ avg: num [1:28] 28.9 25.2 27.3 28.1 30.1 ...
```

Once you are done, click the “Knit” button above(It looks like a blue ball of yarn). Save the file with your name and the week number in the file name:

(for example: “Liam_Mueller_Section_4”).

Then upload the pdf file to canvas/gradescope under the Week 4 Section Assignment before the deadline.

And that is it for this week!