# STA 445: Assignment 4

Kaylin McLiverty

2023-11-03

## Chapter 13

### Exercise 1

A common task is to take a set of data that has multiple categorical variables and create a table of the number of cases for each combination. An introductory statistics textbook contains a dataset summarizing student surveys from several sections of an intro class. The two variables of interest for us are `Gender` and `Year` which are the students gender and year in college.

```
  a)  Download the dataset and correctly order the `Year` variable using the
    following: Survey <- read.csv('https://www.lock5stat.com/datasets3e/StudentSurvey.csv',
    na.strings=c('',' '))
```

```
# Exercise 1 part a
#Downloading the dataset
Survey <- read.csv('https://www.lock5stat.com/datasets3e/StudentSurvey.csv',
        na.strings=c('',' '))
head(Survey)
```

```
##        Year Sex Smoke   Award HigherSAT Exercise TV Height Weight Siblings
## 1    Senior   M    No Olympic      Math       10  1     71    180        4
## 2 Sophomore   F   Yes Academy      Math        4  7     66    120        2
## 3 FirstYear   M    No   Nobel      Math       14  5     72    208        2
## 4    Junior   M    No   Nobel      Math        3  1     63    110        1
## 5 Sophomore   F    No   Nobel    Verbal        3  3     65    150        1
## 6 Sophomore   F    No   Nobel    Verbal        5  4     65    114        2
##   BirthOrder VerbalSAT MathSAT  SAT  GPA Pulse Piercings
## 1          4       540     670 1210 3.13    54         0
## 2          2       520     630 1150 2.50    66         3
## 3          1       550     560 1110 2.55   130         0
## 4          1       490     630 1120 3.10    78         0
## 5          1       720     450 1170 2.70    40         6
## 6          2       600     550 1150 3.20    80         4
```

```
#don't have to worry about ordering
#order(Survey$Year, decreasing = FALSE, method=c('FirstYear', 'Sophomore', 'Junior', 'Senior'))
```

```
  b)  Using some combination of `dplyr` functions, produce a data set with
    eight rows that contains the number of responses for each gender:year
    combination. Make sure your table orders the `Year` variable in the
    correct order of `First Year`, `Sophmore`, `Junior`, and then `Senior`.
    *You might want to look at the following functions: `dplyr::count` and *
    *`dplyr::drop_na`.*
```

```
# Exercise 1 part b
survey1 <- Survey %>%
  count(Sex,Year) %>%
  drop_na()
survey1
```

```
##   Sex      Year  n
## 1   F FirstYear 43
## 2   F    Junior 18
## 3   F    Senior 10
## 4   F Sophomore 96
## 5   M FirstYear 51
## 6   M    Junior 17
## 7   M    Senior 26
## 8   M Sophomore 99
```

```
  c)  Using `tidyr` commands, produce a table of the number of responses in
    the following form:

    |   Gender    |  First Year  |  Sophmore  |  Junior   |  Senior   |
    |:-----------:|:------------:|:----------:|:---------:|:---------:|
    |  **Female** |              |            |           |           |
    |  **Male**   |              |            |           |           |
```

```
# Exercise 1 part c
survey2 <- survey1 %>% pivot_wider(
  names_from=Year,
  values_from=n)
survey2
```

```
## # A tibble: 2 x 5
##   Sex   FirstYear Junior Senior Sophomore
##   <chr>     <int>  <int>  <int>     <int>
## 1 F            43     18     10        96
## 2 M            51     17     26        99
```

## Exercise 2

From the book website, there is a .csv file of the daily maximum temperature in Flagstaff at the Pulliam Airport. The direction link is at: https://raw.githubusercontent.com/dereksonderegger/444/master/data-raw/FlagMaxTemp.csv

```
a)  Create a line graph that gives the daily maximum temperature for 2005.
    *Make sure the x-axis is a date and covers the whole year.*
```

```
# Exercise 2 part a
#loading the .csv file
maxtemp <- read.csv('https://raw.githubusercontent.com/dereksonderegger/444/master/data-raw/FlagMaxTemp
maxtemp_05 <- filter(maxtemp, Year==2005)

maxtemp_05 <- pivot_longer(
  maxtemp_05,
  X1:X31,
  names_to='Day',
  values_to='Temperature')
```

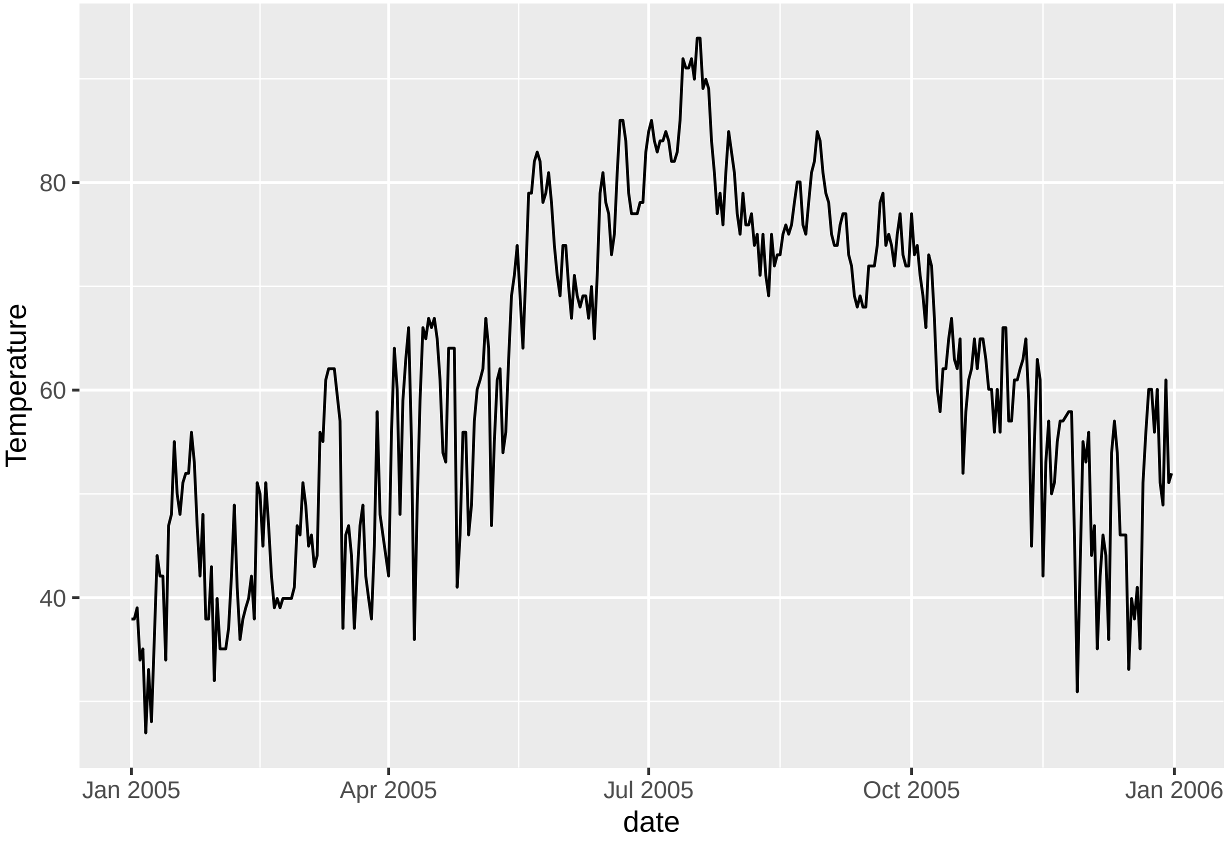
```
maxtemp_05 <- unite(maxtemp_05, col='Date', c('Year','Month','Day'))

maxtemp_05 <- maxtemp_05 %>%
  mutate(date=lubridate::ymd(Date)) %>% drop_na()
```

```
## Warning: There was 1 warning in `mutate()`.
## i In argument: `date = lubridate::ymd(Date)`.
## Caused by warning:
## !  7 failed to parse.
```

```
head(maxtemp_05)
```

```
## # A tibble: 6 x 4
##         X Date       Temperature date
##     <int> <chr>            <dbl> <date>
## 1     235 2005_1_X1         37.9 2005-01-01
## 2     235 2005_1_X2         37.9 2005-01-02
## 3     235 2005_1_X3         39.0 2005-01-03
## 4     235 2005_1_X4         34.0 2005-01-04
## 5     235 2005_1_X5         35.1 2005-01-05
## 6     235 2005_1_X6         27.0 2005-01-06
```

```
#Creating a line graph
ggplot(maxtemp_05, aes(x=date, y=Temperature )) +
  geom_line()
```



```
b)  Create a line graph that gives the monthly average maximum temperature
    for 2013 - 2015. *Again the x-axis should be the date and the axis*
    *spans 3 years.*
```

```
# Exercise 2 part b
maxtemp_35 <- filter(maxtemp, Year >= 2013)

maxtemp_35 <- maxtemp_35 %>% group_by(Month, Year)

maxtemp_35 <- pivot_longer(
  maxtemp_35,
  X1:X31,
  names_to='day',
  values_to='Temperature') %>% drop_na()
maxtemp_35
```

```
## # A tibble: 1,001 x 5
## # Groups:   Month, Year [35]
##        X  Year Month day   Temperature
##    <int> <int> <int> <chr>       <dbl>
##  1   330  2013     1 X1            30.0
##  2   330  2013     1 X2            28.9
##  3   330  2013     1 X3            36.0
##  4   330  2013     1 X4            48.0
##  5   330  2013     1 X5            43.0
##  6   330  2013     1 X6            41
##  7   330  2013     1 X7            46.0
##  8   330  2013     1 X8            46.9
##  9   330  2013     1 X9            46.9
## 10   330  2013     1 X10           46.9
## # i 991 more rows
```

```
maxtemp_35 <- maxtemp_35 %>%
  group_by(Month, Year) %>%
  summarise(MeanTemp=mean(Temperature)) %>%
  mutate(Day=1)
```

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

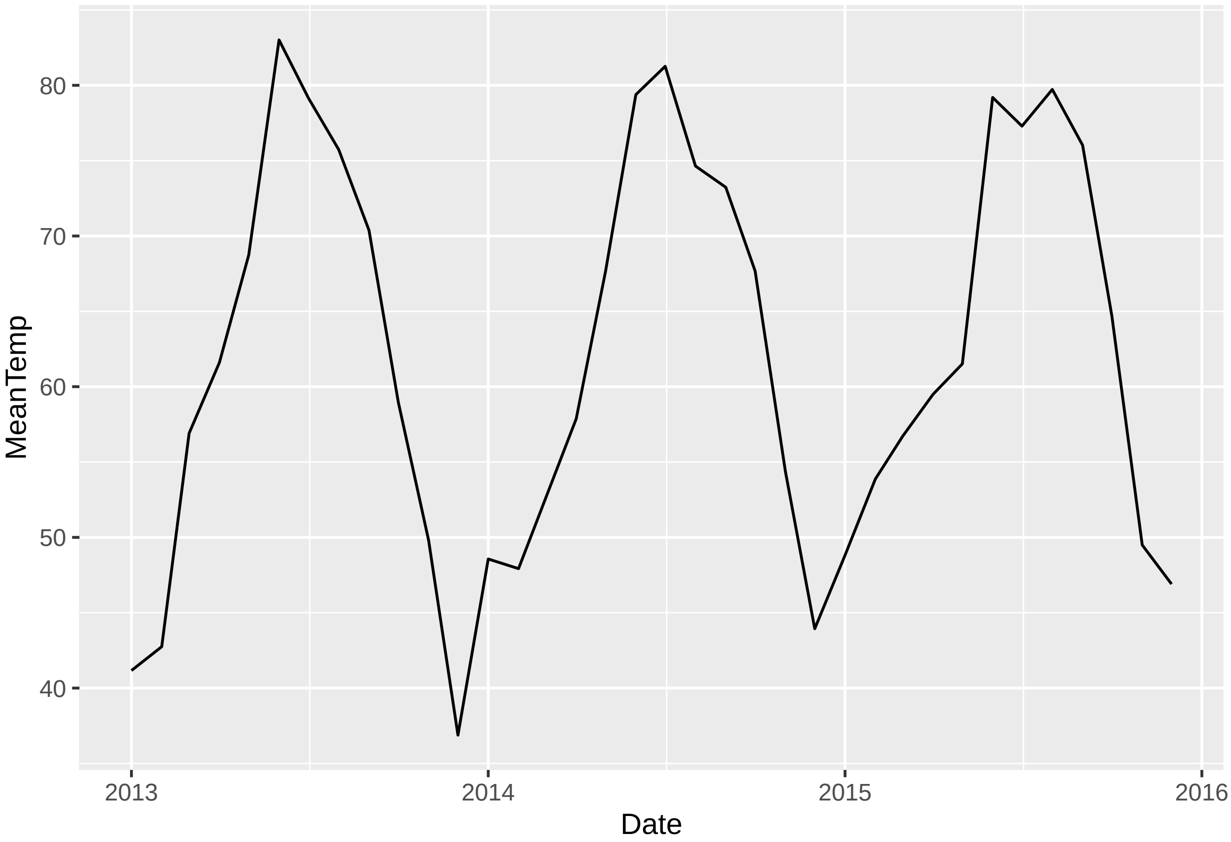
```
maxtemp_35
```

```
## # A tibble: 35 x 4
## # Groups:   Month [12]
##    Month  Year MeanTemp   Day
##    <int> <int>    <dbl> <dbl>
##  1     1  2013     41.2     1
##  2     1  2014     48.6     1
##  3     1  2015     48.8     1
##  4     2  2013     42.7     1
##  5     2  2014     47.9     1
##  6     2  2015     53.9     1
##  7     3  2013     56.9     1
##  8     3  2015     56.7     1
##  9     4  2013     61.6     1
## 10     4  2014     57.9     1
## # i 25 more rows
```

```
maxtemp_35 <- unite(maxtemp_35, col='Date', c('Year','Month','Day')) %>%
  mutate(Date=lubridate::ymd(Date))

maxtemp_35
```

```
## # A tibble: 35 x 2
##    Date       MeanTemp
##    <date>        <dbl>
##  1 2013-01-01     41.2
##  2 2014-01-01     48.6
##  3 2015-01-01     48.8
##  4 2013-02-01     42.7
##  5 2014-02-01     47.9
##  6 2015-02-01     53.9
##  7 2013-03-01     56.9
##  8 2015-03-01     56.7
##  9 2013-04-01     61.6
## 10 2014-04-01     57.9
## # i 25 more rows
```

```
#Creating a line graph
ggplot(maxtemp_35, aes(x=Date, y=MeanTemp )) +
  geom_line()
```



### Exercise 4

For this problem we will consider two simple data sets.

```
#Code given for Exercise 4:
 A <- tribble(
      ~Name, ~Car,
      'Alice', 'Ford F150',
      'Bob',   'Tesla Model III',
      'Charlie', 'VW Bug')

    B <- tribble(
      ~First.Name, ~Pet,
      'Bob',  'Cat',
      'Charlie', 'Dog',
      'Alice', 'Rabbit')
```

    a)  Squish the data frames together to generate a data set with three rows
    and three columns. Do two ways: first using `cbind` and then using one
    of the `dplyr` `join` commands.

```
# Exercise 4 part a
#Using cbind
B2 <- B[order(B$First.Name), ]
B2
```

```
## # A tibble: 3 x 2
##   First.Name Pet
##   <chr>      <chr>
## 1 Alice      Rabbit
## 2 Bob        Cat
## 3 Charlie    Dog
```

```
cbind(A, Pet=B2$Pet)
```

```
##      Name             Car    Pet
## 1   Alice       Ford F150 Rabbit
## 2     Bob Tesla Model III    Cat
## 3 Charlie          VW Bug    Dog
```

```
#using 'dplyr' 'join' commands
inner_join(A,B2, by=c('Name'='First.Name'))
```

```
## # A tibble: 3 x 3
##   Name    Car             Pet
##   <chr>   <chr>           <chr>
## 1 Alice   Ford F150       Rabbit
## 2 Bob     Tesla Model III Cat
## 3 Charlie VW Bug          Dog
```

    b)  It turns out that Alice also has a pet guinea pig. Add another row to
    the `B` data set. Do this using either the base function `rbind`, or
    either of the `dplyr` functions `add_row` or `bind_rows`.

```
# Exercise 4 part b
#Used cbind
pig <- c('Alice','Guinea Pig')

B3 <- rbind(B2,pig)
B3
```

```
## # A tibble: 4 x 2
##   First.Name Pet
##   <chr>      <chr>
## 1 Alice      Rabbit
## 2 Bob        Cat
## 3 Charlie    Dog
## 4 Alice      Guinea Pig
```

```
    c)  Squish the `A` and `B` data sets together to generate a data set with
    four rows and three columns. Do this two ways: first using `cbind` and
    then using one of the `dplyr` `join` commands. Which was easier to
    program? Which is more likely to have an error.
```

```
# Exercise 4 part c
#Using cbind
Q <- cbind(A, Pet=B2$Pet)
Q <- rbind(Q,pig)
```

```
## Warning in rbind(deparse.level, ...): number of columns of result, 3, is not a
## multiple of vector length 2 of arg 2
```

```
Q
```

```
##      Name             Car    Pet
## 1   Alice       Ford F150 Rabbit
## 2     Bob Tesla Model III    Cat
## 3 Charlie          VW Bug    Dog
## 4   Alice      Guinea Pig  Alice
```

```
#using 'dplyr' 'join' commands
R <- full_join(A,B3, by=c('Name'='First.Name'))
R
```

```
## # A tibble: 4 x 3
##   Name    Car             Pet
##   <chr>   <chr>           <chr>
## 1 Alice   Ford F150       Rabbit
## 2 Alice   Ford F150       Guinea Pig
## 3 Bob     Tesla Model III Cat
## 4 Charlie VW Bug          Dog
```

*The second way of using one of the 'dplyr' 'join' commands was easier to program and is less likely to have error. Using cbind is bound for more error and the function itself is a bit more finicky to work with. I found it easier to combine the columns first and then add the additional row*

*Note for Exercise 5: Respectfully, I have looked over Exercise 5 and will review it in more detail. I know that this exercise is difficult and will not be graded. Thus, I have not made a productive attempt yet.*

## Exercise 5

Data table joins are extremely common because effective database design almost always involves having multiple tables for different types of objects. To illustrate both the table joins and the usefulness of multiple tables we will develop a set of data frames that will represent a credit card company's customer data base. We will have tables for Customers, Retailers, Cards, and Transactions.

```
# Given code for Exercise 5
Customers <- tribble(
      ~PersonID, ~Name, ~Street, ~City, ~State,
      1, 'Derek Sonderegger',  '231 River Run', 'Flagstaff', 'AZ',
      2, 'Aubrey Sonderegger', '231 River Run', 'Flagstaff', 'AZ',
      3, 'Robert Buscaglia', '754 Forest Heights', 'Flagstaff', 'AZ',
      4, 'Roy St Laurent', '845 Elk View', 'Flagstaff', 'AZ')

    Retailers <- tribble(
      ~RetailID, ~Name, ~Street, ~City, ~State,
      1, 'Kickstand Kafe', '719 N Humphreys St', 'Flagstaff', 'AZ',
      2, 'MartAnnes', '112 E Route 66', 'Flagstaff', 'AZ',
      3, 'REI', '323 S Windsor Ln', 'Flagstaff', 'AZ' )

    Cards <- tribble(
      ~CardID, ~PersonID, ~Issue_DateTime, ~Exp_DateTime,
      '9876768717278723',  1,  '2019-9-20 0:00:00', '2022-9-20 0:00:00',
      '5628927579821287',  2,  '2019-9-20 0:00:00', '2022-9-20 0:00:00',
      '7295825498122734',  3,  '2019-9-28 0:00:00', '2022-9-28 0:00:00',
      '8723768965231926',  4,  '2019-9-30 0:00:00', '2022-9-30 0:00:00' )

    Transactions <- tribble(
      ~CardID, ~RetailID, ~DateTime, ~Amount,
      '9876768717278723', 1, '2019-10-1 8:31:23',    5.68,
      '7295825498122734', 2, '2019-10-1 12:45:45',  25.67,
      '9876768717278723', 1, '2019-10-2 8:26:31',    5.68,
      '9876768717278723', 1, '2019-10-2 8:30:09',    9.23,
      '5628927579821287', 3, '2019-10-5 18:58:57',  68.54,
      '7295825498122734', 2, '2019-10-5 12:39:26',  31.84,
      '8723768965231926', 2, '2019-10-10 19:02:20', 42.83)

    Cards <- Cards %>%
      mutate( Issue_DateTime = lubridate::ymd_hms(Issue_DateTime),
              Exp_DateTime   = lubridate::ymd_hms(Exp_DateTime) )
    Transactions <- Transactions %>%
      mutate( DateTime = lubridate::ymd_hms(DateTime))
```

```
a)  Create a table that gives the credit card statement for Derek. It should
    give all the transactions, the amounts, and the store name. Write your
    code as if the only initial information you have is the customer's name.
    *Hint: Do a bunch of table joins, and then filter for the desired customer*
    *name. To be efficient, do the filtering first and then do the table joins.*
```

```
# Exercise 5 part a
```

```
b)  Aubrey has lost her credit card on Oct 15, 2019. Close her credit card at
    4:28:21 PM and issue her a new credit card in the `Cards` table.
    *Hint: Using the Aubrey's name, get necessary CardID and PersonID and save*
    *those as `cardID` and `personID`. Then update the `Cards` table row that*
    *corresponds to the `cardID` so that the expiration date is set to the time*
    *that the card is closed. Then insert a new row with the `personID` for*
    *Aubrey and a new `CardID` number that you make up.*
```

```
# Exercise 5 part b
```

```
c)  Aubrey is using her new card at Kickstand Kafe on Oct 16, 2019 at
    2:30:21 PM for coffee with a charge of $4.98. Generate a new transaction
    for this action.
    *Hint: create temporary variables `card`,`retailid`,`datetime`, and*
    *`amount` that contain the information for this transaction and then*
    *write your code to use those. This way in the next question you can just*
    *use the same code but modify the temporary variables. Alternatively, you*
    *could write a function that takes in these four values and manipulates the*
    *tables in the GLOBAL environment using the `<<-` command to assign a result*
    *to a variable defined in the global environment. The reason this is OK is*
    *that in a real situation, these data would be stored in a database and we*
    *would expect the function to update that database.*
```

```r
# Exercise 5 part c
```

```
d)  On Oct 17, 2019, some nefarious person is trying to use her OLD credit
    card at REI. Make sure your code in part (c) first checks to see if the
    credit card is active before creating a new transaction. Using the same
    code, verify that the nefarious transaction at REI is denied.
```

```r
# Exercise 5 part d
```

```
 e)  Generate a table that gives the credit card statement for Aubrey. It
    should give all the transactions, amounts, and retailer name for both
    credit cards she had during this period.
```

```r
# Exercise 5 part e
```