Assignment 4

Rebecca Chavez

2023-10-26

Chapter Thirteen

Question One

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('http://www.lock5stat.com/datasets3e/StudentSurvey.csv', na.strings=c('',' '))
```

```
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‘.*
```

```
# use fct_relevel to order year names properly
Survey$Year <- fct_relevel(Survey$Year, "FirstYear", "Sophomore", "Junior", "Senior")
Survey <- Survey %>%
  drop_na() %>% # get rid of rows with NA valeus
  mutate(Gender = ifelse(Sex == 'M', 'Male', 'Female')) %>% # make gender col
  group_by(Gender, Year) %>% # get gender/year combos
  count(name='numResponses') # count number of responses for each combo
Survey
```

```
## # A tibble: 8 x 3
## # Groups:   Gender, Year [8]
##   Gender Year      numResponses
##   <chr>  <fct>            <int>
## 1 Female FirstYear           36
## 2 Female Sophomore           90
## 3 Female Junior              15
## 4 Female Senior               10
## 5 Male   FirstYear            43
## 6 Male   Sophomore            89
## 7 Male   Junior               16
## 8 Male   Senior               26
```

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

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

```r
# use pivot wider to get five columns and two rows
Survey.wider <- Survey %>%
  pivot_wider(names_from = Year, values_from = numResponses)
Survey.wider
```

```
## # A tibble: 2 x 5
## # Groups:   Gender [2]
##   Gender FirstYear Sophomore Junior Senior
##   <chr>      <int>     <int>  <int>  <int>
## 1 Female        36        90     15     10
## 2 Male          43        89     16     26
```

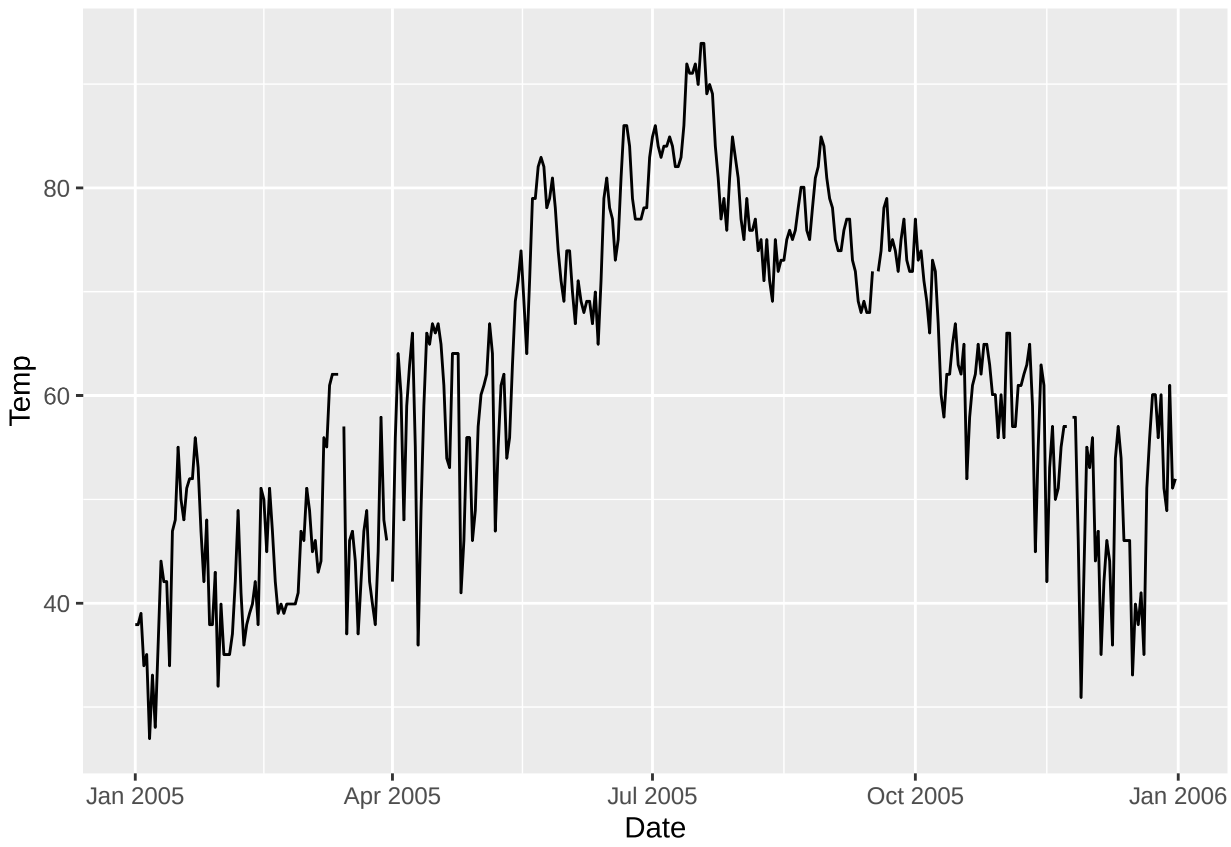
Question Two

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.*
```

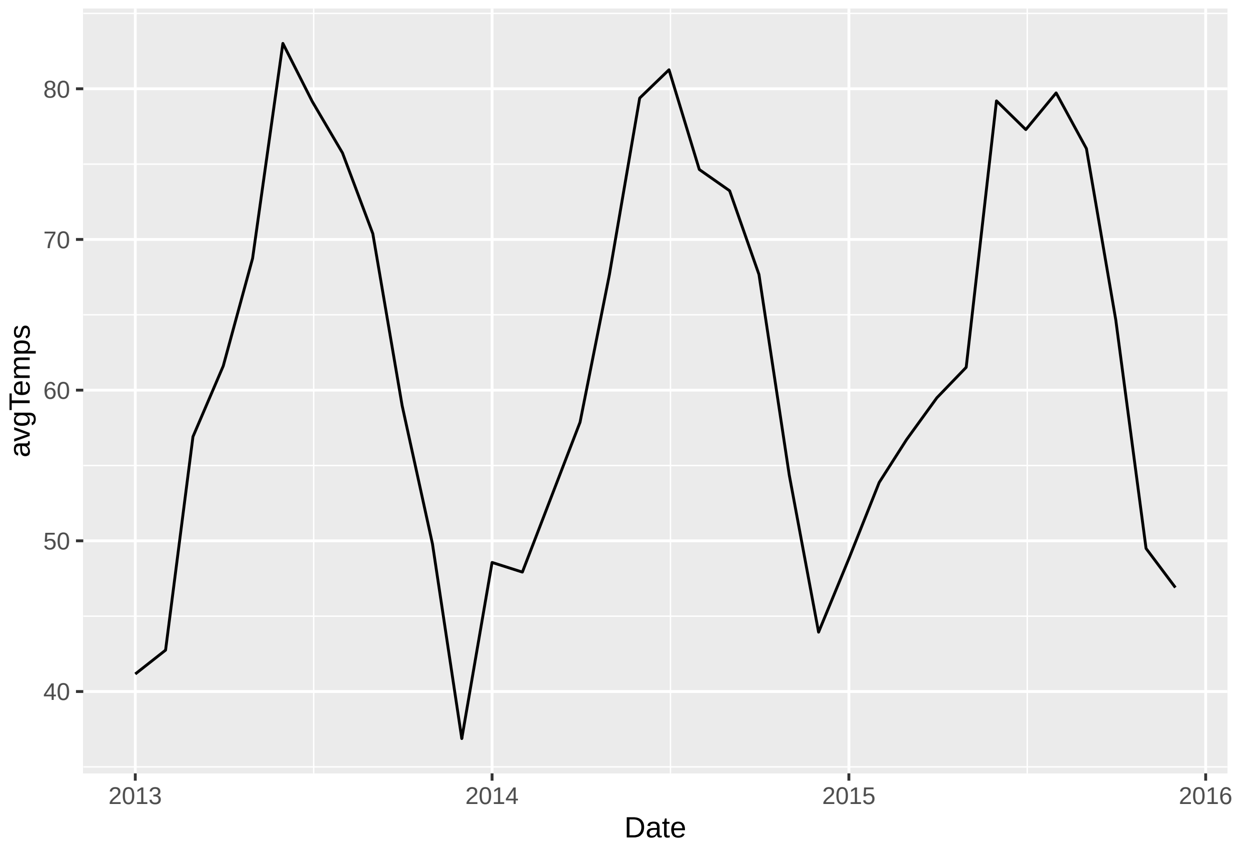
```r
maxTemps <- read.csv('https://raw.githubusercontent.com/dereksonderegger/444/master/data-raw/FlagMaxTemp
maxTemps2005 <- maxTemps %>%
  filter(Year == 2005) %>% # get only year 2005
  pivot_longer(X1:X31, names_to = "Day", values_to = "Temp") %>% # pivot table
  mutate(Day = str_extract(Day, '\\d+')) %>% # get number from X1:X31 cols
  mutate(Date = make_date(year=Year, month=Month, day=Day))
ggplot(maxTemps2005, aes(x=Date, y=Temp)) +
  geom_line()
```

```
## Warning: Removed 7 rows containing missing values (‘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.*

```r
maxTempsMonthly <- maxTemps %>%
  filter(Year > 2012 & Year < 2016) %>% # get years 2012 - 2016
  pivot_longer(X1:X31, names_to = "Day", values_to = "Temp") %>%
  group_by(Year, Month) %>% # get month/year combos
  drop_na() %>% # get rid of rows with NA values
  summarise(avgTemps = mean(Temp), .groups = 'drop') %>% # get means by month
  mutate(Date = make_date(month = Month, year = Year))
ggplot(maxTempsMonthly, aes(x=Date, y=avgTemps)) +
  geom_line()
```



Question Four

For this problem we will consider two simple data sets.

```
‘‘‘r
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.
```

```
# bind A and the second column of B
table <- cbind(A, B[2])
# inner join by the name columns
table2 <- full_join(A, B, by = join_by(Name == First.Name))
table
```

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

```
table2
```

```
## # 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‘.
```

```
newRow <- tibble(First.Name = 'Alice', Pet = 'Guinea pig')
B <- rbind(B, newRow)
B
```

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

```
# must give by since there are no common variable names
table3 <- inner_join(A, B, by = join_by(Name == First.Name))
table3
```

```
## # 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
```

```r
# must add a row to A in order to be able to use cbind after adding to B
A <- A %>% add_row(Name = 'Alice', Car = 'Ford F150')
table4 <- cbind(A, B[2])
table4
```

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

Question Five

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. Below is code that will create and populate these tables.

```
‘‘‘r
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.*
```

```
# find Derek in customer table
derek <- Customers %>%
  filter(Name == 'Derek Sonderegger')

# use person id to find derek's card
derekCard <- Cards %>%
  filter(PersonID == derek$PersonID)

# find all transactions derek made
derekTransactions <- inner_join(Transactions, derekCard)
```

```
## Joining with ‘by = join_by(CardID)‘
```

```
# add store name to statement
statement <- inner_join(derekTransactions, Retailers) %>%
  select(CardID, Name, Amount, DateTime)
```

```
## Joining with ‘by = join_by(RetailID)‘
```

```
statement
```

```
## # A tibble: 3 x 4
##   CardID           Name           Amount DateTime
##   <chr>            <chr>           <dbl> <dttm>
## 1 9876768717278723 Kickstand Kafe   5.68 2019-10-01 08:31:23
## 2 9876768717278723 Kickstand Kafe   5.68 2019-10-02 08:26:31
## 3 9876768717278723 Kickstand Kafe   9.23 2019-10-02 08:30:09
```

```
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.*
```

```
# get person id
personID <- Customers %>%
```

```r
  filter(Name == "Aubrey Sonderegger") %>%
  select(PersonID)

# get card id
cardId <- Cards %>%
  filter(PersonID == personID$PersonID) %>%
  select(CardID)

# set card to expired
Cards <- Cards %>%
  mutate(Exp_DateTime = ifelse(CardID == cardId$CardID,
                               mdy_hms('Nov 02, 2023 4:28:21 PM'), Exp_DateTime))

# add new card
Cards <- Cards %>%
  add_row(CardID = '9823743190671236', PersonID = personID$PersonID,
          Issue_DateTime = mdy(11022023), Exp_DateTime = 11022028)
```

```
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.*

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.
    *Hint: your code ought to look something like this:*

    ‘‘‘r
    card <- ’9876768717278723’
    retailid <- 2
    datetime <- ymd_hms(’2019-10-16 14:30:21’)
    amount <- 4.98

    # If the card is currently valid, this should return exactly 1 row.
    Valid_Cards <- Cards %>%
      filter(CardID == card, Issue_DateTime <= datetime, datetime <= Exp_DateTime)

    # If the transaction is valid, insert the transaction into the table
    if( nrow(Valid_Cards) == 1){
      # Some code to insert the transaction
    }else{
      print(’Card Denied’)
    }
```

‘‘‘

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.