Assignment 4 STA 445

Analea Poppen

2023-10-26

Question 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:

```r
Survey <- read.csv('https://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`.*

```r
Survey1 <- Survey %>%
  mutate(Year= factor(Year, levels= c('FirstYear', 'Sophomore', 'Junior', 'Senior'))) %>%
  select(Year, Sex) %>%
  group_by(Year, Sex) %>%
  summarise(n=n(), .groups='drop') %>%
  drop_na()
print(Survey1)
```

```
## # A tibble: 8 x 3
##   Year      Sex       n
##   <fct>     <chr> <int>
## 1 FirstYear F        43
## 2 FirstYear M        51
## 3 Sophomore F        96
## 4 Sophomore M        99
## 5 Junior    F        18
## 6 Junior    M        17
## 7 Senior    F        10
## 8 Senior    M        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				

```
Survey2 <- Survey1 %>%
  pivot_wider(names_from='Year', values_from='n') %>%
  mutate(Sex= if_else(Sex == 'M', 'Male', 'Female'))
print(Survey2)
```

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

Question 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.*

```
FlagAir <- read.csv('https://raw.githubusercontent.com/dereksonderegger/444/master/data-raw/FlagMaxTemp

FlagAir2 <- FlagAir %>%
  filter(Year== 2005) %>%
  pivot_longer(X1:X31, names_to= 'Days', values_to= 'Temp')

FlagAir2$Days <- str_replace_all(FlagAir2$Days, "^X", "")

FlagAir3 <- FlagAir2 %>%
  mutate(Date= make_date(year= Year, month= Month, day= Days))

ggplot(data=FlagAir3, aes(x=Date, y=Temp)) +
  geom_line()
```
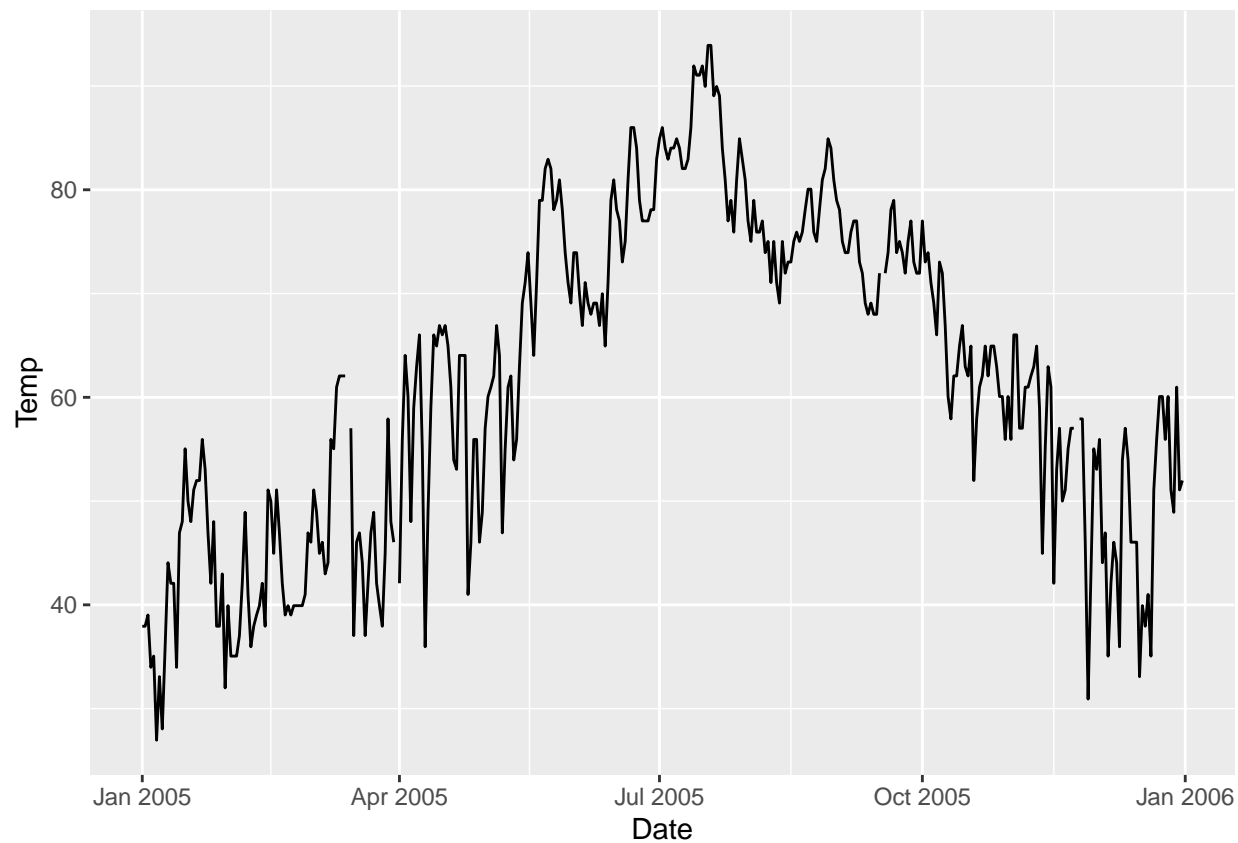
```
## Warning: Removed 7 rows containing missing values (`geom_line()`).
```

```
print(FlagAir2)
```

```
## # A tibble: 372 x 5
##        X  Year Month Days   Temp
##    <int> <int> <int> <chr> <dbl>
## 1    235  2005     1 1      37.9
## 2    235  2005     1 2      37.9
## 3    235  2005     1 3      39.0
## 4    235  2005     1 4      34.0
## 5    235  2005     1 5      35.1
## 6    235  2005     1 6      27.0
## 7    235  2005     1 7      33.1
## 8    235  2005     1 8      28.0
## 9    235  2005     1 9      36.0
## 10   235  2005     1 10     44.1
## # i 362 more rows
```

```
print(FlagAir3)
```

```
## # A tibble: 372 x 6
##        X  Year Month Days   Temp Date
##    <int> <int> <int> <chr> <dbl> <date>
## 1    235  2005     1 1      37.9 2005-01-01
## 2    235  2005     1 2      37.9 2005-01-02
```

```
##  3   235  2005     1 3       39.0 2005-01-03
##  4   235  2005     1 4       34.0 2005-01-04
##  5   235  2005     1 5       35.1 2005-01-05
##  6   235  2005     1 6       27.0 2005-01-06
##  7   235  2005     1 7       33.1 2005-01-07
##  8   235  2005     1 8       28.0 2005-01-08
##  9   235  2005     1 9       36.0 2005-01-09
## 10   235  2005     1 10      44.1 2005-01-10
## # i 362 more rows
```

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.*

```
FlagAir4 <- FlagAir %>%
  filter(Year > 2012 & Year < 2016) %>%
  pivot_longer(X1:X31, names_to= 'Days', values_to= 'Temp') %>%
  group_by(Month,Year) %>%
  drop_na() %>%
  summarize(AvgMonthTemp= mean(Temp)) %>%
  mutate(Date= make_date(year= Year, month= Month))
```

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

```
ggplot(data=FlagAir4, aes(x= Date, y=AvgMonthTemp)) +
  geom_line()
```
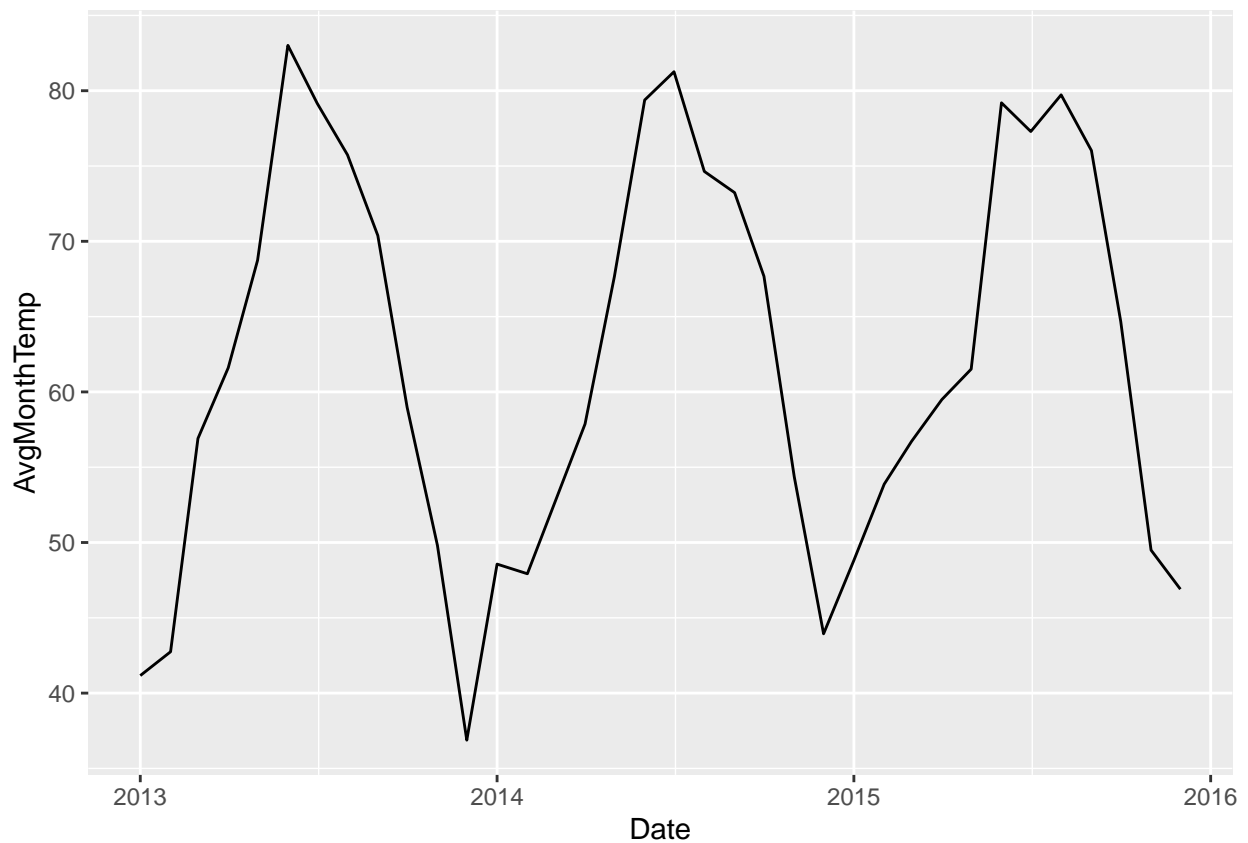
```
print(FlagAir4)
```

```
## # A tibble: 35 x 4
## # Groups:   Month [12]
##     Month  Year AvgMonthTemp Date
##     <int> <int>        <dbl> <date>
## 1      1  2013         41.2 2013-01-01
## 2      1  2014         48.6 2014-01-01
## 3      1  2015         48.8 2015-01-01
## 4      2  2013         42.7 2013-02-01
## 5      2  2014         47.9 2014-02-01
## 6      2  2015         53.9 2015-02-01
## 7      3  2013         56.9 2013-03-01
## 8      3  2015         56.7 2015-03-01
## 9      4  2013         61.6 2013-04-01
## 10     4  2014         57.9 2014-04-01
## # i 25 more rows
```

Question 4

For this problem we will consider two simple data sets.

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.

```
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')

df1 <- cbind(A, B)
print(df1)
```

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

```
df2 <- right_join(A, B, by=c("Name"="First.Name"))
print(df2)
```

```
## # A tibble: 3 x 3
##    Name    Car            Pet
```

5

```
##   <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`.

```
C <- tibble(First.Name= 'Alice', Pet= 'Guinea Pig')
df3 <- bind_rows(B, C)
print(df3)
```

```
## # 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.

```
df4 <- right_join(A, df3, by=c("Name"="First.Name"))
print(df4)
```

```
## # 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 join command is so much easier than the cbind command. Especially when column names are not written the same way

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

```
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.*

```
Derek <- Customers %>%
  filter(Name== 'Derek Sonderegger')
Derek1 <- inner_join(Derek, Cards)
```

```
## Joining with 'by = join_by(PersonID)'
```

```
Derek2 <- inner_join(Derek1, Transactions)
```

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

```
Derek3 <- left_join(Derek2, Retailers)
```

```
## Joining with 'by = join_by(Name, Street, City, State, RetailID)'
```

```
print(Derek3)
```

```
## # A tibble: 3 x 11
##   PersonID Name               Street       City  State CardID Issue_DateTime
##      <dbl> <chr>              <chr>        <chr> <chr> <chr>  <dttm>
## 1        1 Derek Sonderegger 231 River R~ Flag~ AZ    98767~ 2019-09-20 00:00:00
## 2        1 Derek Sonderegger 231 River R~ Flag~ AZ    98767~ 2019-09-20 00:00:00
## 3        1 Derek Sonderegger 231 River R~ Flag~ AZ    98767~ 2019-09-20 00:00:00
## # i 4 more variables: Exp_DateTime <dttm>, RetailID <dbl>, DateTime <dttm>,
## #   Amount <dbl>
```

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.*

```
aubrey_info <- Customers %>%
  filter(Name == 'Aubrey Sonderegger') %>%
  left_join(Cards, by = "PersonID")

personID <- aubrey_info$PersonID
old_cardID <- aubrey_info$CardID

closing_time <- ymd_hms('2019-10-15 16:28:21')
Cards <- Cards %>%
  mutate(Exp_DateTime = ifelse(CardID == old_cardID, closing_time, Exp_DateTime))

new_cardID <- 'NEW_CARD_ID'

closing_time <- ymd_hms('2019-10-15 16:28:21')

new_card <- tribble(
  ~CardID, ~PersonID, ~Issue_DateTime, ~Exp_DateTime,
  new_cardID, personID, closing_time, closing_time + years(3)
)

Cards$Exp_DateTime <- lubridate::ymd_hms(Cards$Exp_DateTime)
```

```
## Warning: All formats failed to parse. No formats found.
```

```
Cards <- rbind(Cards, new_card)
```

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
new_transaction <- tribble(
  ~CardID, ~RetailID, ~DateTime, ~Amount,
  new_cardID, 1, as.POSIXct('2019-10-16 14:30:21', tz = 'UTC'), 4.98
)

Transactions <- bind_rows(Transactions, new_transaction)

nefarious_transaction <- tribble(
  ~CardID, ~RetailID, ~DateTime, ~Amount,
  old_cardID, 3, as.POSIXct('2019-10-17 14:30:21', tz = 'UTC'), 100.0
)

Valid_Cards <- Cards %>%
  filter(CardID == nefarious_transaction$CardID,
         Issue_DateTime <= nefarious_transaction$DateTime,
         nefarious_transaction$DateTime <= Exp_DateTime)

if (nrow(Valid_Cards) == 1) {
  Transactions <- bind_rows(Transactions, nefarious_transaction)
} else {
  print('Card Denied')
}
```

```
## [1] "Card Denied"
```

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 check ought to look something like this:*

```r
new_transaction <- tribble(
  ~CardID, ~RetailID, ~DateTime, ~Amount,
  new_cardID, 3, as.POSIXct('2019-10-17 12:00:00', tz = 'UTC'), 100.00
)

Transactions <- bind_rows(Transactions, new_transaction)

nefarious_transaction <- tribble(
  ~CardID, ~RetailID, ~DateTime, ~Amount,
  old_cardID, 3, as.POSIXct('2019-10-17 14:30:21', tz = 'UTC'), 100.0
)

Valid_Cards <- Cards %>%
  filter(CardID == nefarious_transaction$CardID,
         Issue_DateTime <= nefarious_transaction$DateTime,
         nefarious_transaction$DateTime <= Exp_DateTime)

if (nrow(Valid_Cards) == 1) {
  Transactions <- bind_rows(Transactions, nefarious_transaction)
} else {
  print('Card Denied')
}
```

```
## [1] "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.

```r
Aubrey_Statement <- Customers %>%
  filter(Name == 'Aubrey Sonderegger') %>%
  left_join(Cards, by = "PersonID") %>%
  left_join(Transactions, by = "CardID") %>%
  left_join(Retailers, by = "RetailID")

print(Aubrey_Statement)
```

```
## # A tibble: 3 x 15
##   PersonID Name.x             Street.x City.x State.x CardID Issue_DateTime
##      <dbl> <chr>              <chr>    <chr>  <chr>   <chr>  <dttm>
## 1        2 Aubrey Sonderegger 231 Riv~ Flags~ AZ      56289~ 2019-09-20 00:00:00
## 2        2 Aubrey Sonderegger 231 Riv~ Flags~ AZ      NEW_C~ 2019-10-15 16:28:21
## 3        2 Aubrey Sonderegger 231 Riv~ Flags~ AZ      NEW_C~ 2019-10-15 16:28:21
## # i 8 more variables: Exp_DateTime <dttm>, RetailID <dbl>, DateTime <dttm>,
## #   Amount <dbl>, Name.y <chr>, Street.y <chr>, City.y <chr>, State.y <chr>
```