

STA445_Assignment4

Bailey Rosato

2023-11-02

Question 1

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(' ', ''))
#Survey
```

- 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``, ``Sophomore``, ``Junior``, and then ``Senior``. *You might want to look at the following functions: ``dplyr::count`` and ``dplyr::drop_na``.

```
##?dplyr::count
##?dplyr::drop_na
Survey.1 <- Survey %>% group_by(Year) %>% count(Sex) %>% drop_na(Sex,Year) %>%
  arrange( Year = fct_relevel(Year, 'FirstYear', 'Sophomore','Junior','Senior'))
```

Survey.1

```
## # A tibble: 8 x 3
## # Groups:   Year [4]
##   Year      Sex      n
##   <chr>    <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	Sophomore	Junior	Senior
Female					
Male					

```
Survey.2 <- Survey.1 %>% pivot_wider(names_from = 'Year', values_from = 'n') %>%
  mutate(Sex = recode(Sex, 'F' = 'Female', 'M' = 'Male'))
```

Survey.2

```
## # 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

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.*

```
TempSurvey <- read.csv('https://raw.githubusercontent.com/dereksonderegger/444/master/data-raw/FlagMaxTemp.csv',
  na.strings=c('', ' '))
```

```
TempSurvey.2 <- TempSurvey %>% filter(str_detect(Year, "2005")) %>%
  pivot_longer(X1:X31, names_to='Day', values_to='MaxTemp')
```

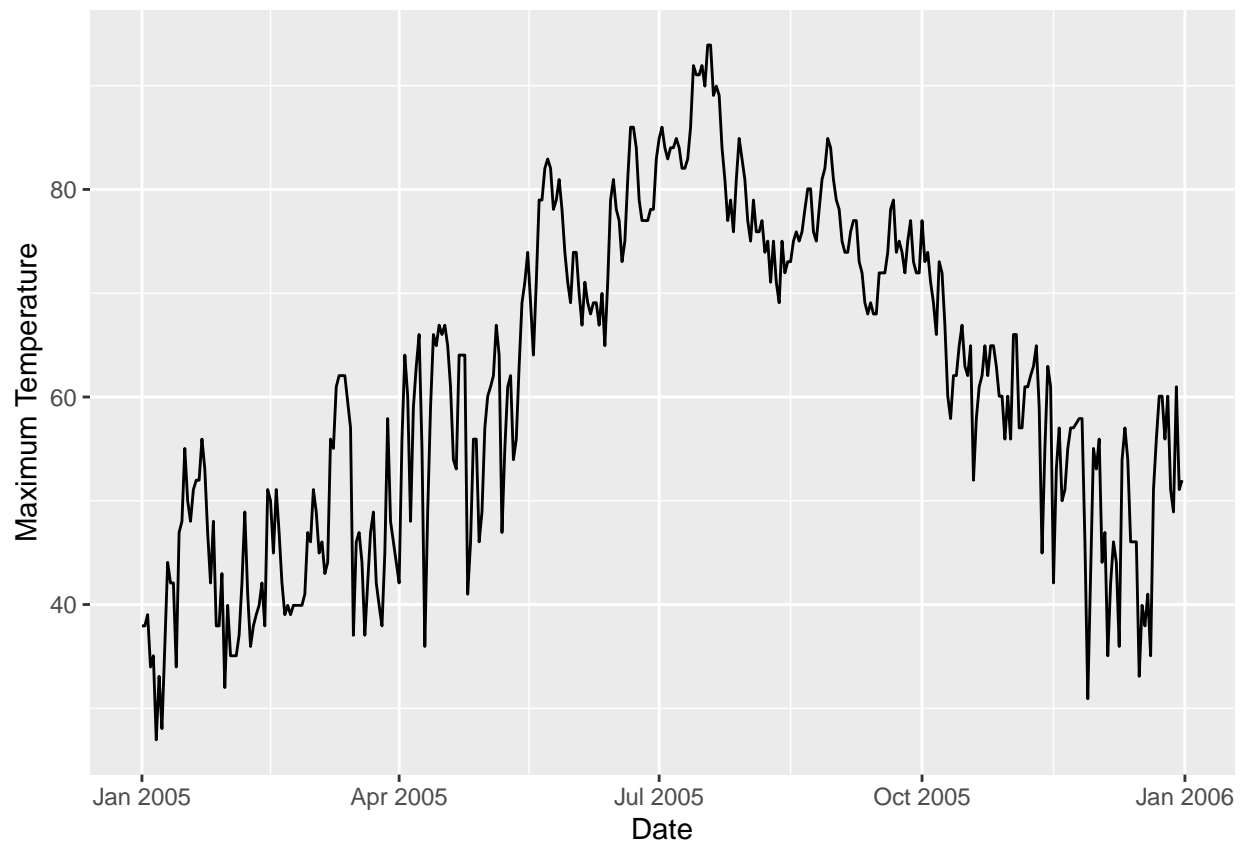
```
TempSurvey.3 <- TempSurvey.2 %>% mutate(Day = str_replace_all(Day, "X", "")) %>%
  mutate(Date = as.Date(paste(Year, Month, Day, sep='-')))
```

```
TempSurvey.4 <- TempSurvey.3 %>% mutate(MaxTemp = as.numeric(MaxTemp)) %>% drop_na()
```

```
## Warning: There was 1 warning in `mutate()`.
## i In argument: `MaxTemp = as.numeric(MaxTemp)`.
## Caused by warning:
## ! NAs introduced by coercion
```

```
TempSurvey.5 <- TempSurvey.4 %>%
  ggplot(aes(x=Date, y=MaxTemp)) + geom_line() +
  labs(x="Date", y="Maximum Temperature")
```

TempSurvey.5



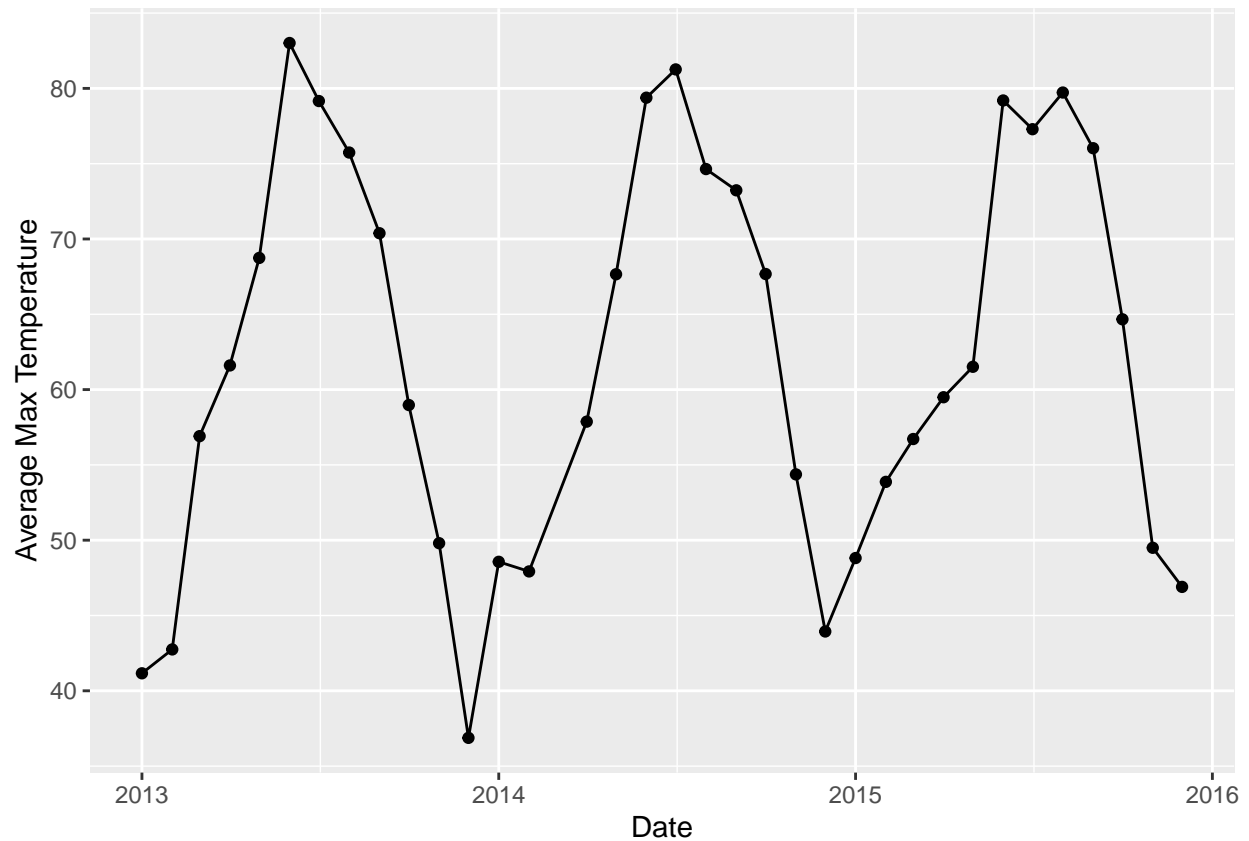
- b) Create a line graph that gives the monthly average maximum temperature for 2013 - 2015. *Again the x-axis should be the date and the axis*
spans 3 years.

```
TempSurveyAvg <- TempSurvey %>%
  filter(Year >= 2013 & Year <= 2015) %>%
  pivot_longer(X1:X31, names_to = 'Day', values_to = 'MaxTemp') %>%
  mutate(MaxTemp = as.numeric(MaxTemp)) %>%
  drop_na() %>%
  group_by(Year, Month) %>%
  summarise(AvgMaxTemp = mean(MaxTemp, na.rm = TRUE)) %>%
  ggplot(aes(x = as.Date(paste(Year, Month, "01", sep = "-")),
             y = AvgMaxTemp)) +
  geom_line() + geom_point() +
  labs(x = "Date", y = "Average Max Temperature")
```

```
## Warning: There was 1 warning in `mutate()`.
## i In argument: `MaxTemp = as.numeric(MaxTemp)`.
## Caused by warning:
## ! NAs introduced by coercion

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

```
TempSurveyAvg
```



Question 4

4. For this problem we will consider two simple data sets.

```
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.

```
combinedData <- cbind(A,B)

finalCombinedData <- inner_join(A, B, by = c("Name" = "First.Name"))

finalCombinedData

## # 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`.

```
new_row <- data.frame(First.Name = "Alice", Pet = "Guinea Pig")
```

```
B <- rbind(B, new_row)
```

```
# i accidentally added too many rows when testing! whoops
```

```
B <- B %>% distinct()
```

```
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.

```
#combinedData <- cbind(A,B)
```

```
finalCombinedData <- inner_join(A, B, by = c("Name" = "First.Name"))
```

```
finalCombinedData
```

```
## # 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
```

It was easier to program the `cbind()` function, however it is more prone to error because it does not know how to handle the mismatched size of the table. Using the join function was more functional but took more work.

Question 5

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.*

```

DerekStatement <- Customers %>% filter(Name == 'Derek Sonderegger') %>%
  left_join(Cards, by = 'PersonID') %>%
  left_join(Transactions, by = 'CardID') %>%
  left_join(Retailers, by = 'RetailID') %>%
  select(DateTime, Amount, Name.y)

```

DerekStatement

```

## # A tibble: 3 x 3
##   DateTime          Amount Name.y
##   <dtm>              <dbl> <chr>
## 1 2019-10-01 08:31:23    5.68 Kickstand Kafe
## 2 2019-10-02 08:26:31    5.68 Kickstand Kafe
## 3 2019-10-02 08:30:09    9.23 Kickstand Kafe

```

- 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.

```
AubreyCard <- Customers %>%
  filter(Name == 'Aubrey Sonderegger') %>%
  inner_join(Cards, by = 'PersonID')

AubreyCard

## # A tibble: 1 x 8
##   PersonID Name          Street      City State CardID Issue_DateTime
##   <dbl> <chr>          <chr>      <chr> <chr> <chr> <dtm>
## 1      2 Aubrey Sonderegger 231 River ~ Flag~ AZ    56289~ 2019-09-20 00:00:00
## # i 1 more variable: Exp_DateTime <dtm>
```

```
personID <- AubreyCard$PersonID
cardID <- AubreyCard$CardID

Cards <- Cards %>%
  mutate(Exp_DateTime = ifelse(CardID == cardID,
    "2019-10-15 16:28:21", Exp_DateTime))

new_card_row <- data.frame(PersonID = personID, CardID = '5628927579821288',
  Issue_DateTime = '2019-10-16 12:30:00',
  Exp_DateTime = '2022-9-20 0:00:00')

Cards <- rbind(Cards, new_card_row)

Cards <- Cards %>% distinct()

Cards
```

```
## # A tibble: 5 x 4
##   CardID      PersonID Issue_DateTime      Exp_DateTime
##   <chr>      <dbl> <dtm>          <chr>
## 1 9876768717278723      1 2019-09-20 00:00:00 1663632000
## 2 5628927579821287      2 2019-09-20 00:00:00 2019-10-15 16:28:21
## 3 7295825498122734      3 2019-09-28 00:00:00 1664323200
## 4 8723768965231926      4 2019-09-30 00:00:00 1664496000
## 5 5628927579821288      2 2019-10-16 19:30:00 2022-9-20 0:00:00
```

- 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.

```

card <- '5628927579821288'
retailid <- 1
datetime <- '2019-10-16 14:30:21'
amount <- 4.98

# ~CardID, ~RetailID, ~DateTime, ~Amount,

Valid_Cards <- Cards %>%
  filter(CardID == card,
         Issue_DateTime <= datetime, datetime <= Exp_DateTime)

Valid_Cards

```

```

## # A tibble: 1 x 4
##   CardID      PersonID Issue_DateTime      Exp_DateTime
##   <chr>          <dbl> <dtm>          <chr>
## 1 5628927579821288      2 2019-10-16 19:30:00 2022-9-20 0:00:00

# If the transaction is valid, insert the transaction into the table
if( nrow(Valid_Cards) == 1){
  new_transaction_row <- data.frame(CardID = card, RetailID = retailid,
                                    DateTime = datetime, Amount = amount)

  Transactions <- rbind(Transactions, new_transaction_row)
}else{
  print('Card Denied')
}

Transactions <- Transactions %>% distinct()

Transactions

```

```

## # A tibble: 8 x 4
##   CardID      RetailID DateTime      Amount
##   <chr>          <dbl> <dtm>          <dbl>
## 1 9876768717278723      1 2019-10-01 08:31:23   5.68
## 2 7295825498122734      2 2019-10-01 12:45:45  25.7
## 3 9876768717278723      1 2019-10-02 08:26:31   5.68
## 4 9876768717278723      1 2019-10-02 08:30:09   9.23
## 5 5628927579821287      3 2019-10-05 18:58:57  68.5
## 6 7295825498122734      2 2019-10-05 12:39:26  31.8
## 7 8723768965231926      2 2019-10-10 19:02:20  42.8
## 8 5628927579821288      1 2019-10-16 21:30:21   4.98

```

- 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:*

```

card <- '5628927579821287'
retailid <- 2
datetime <- '2019-10-16 14:30:21'
amount <- 4.98

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){
  new_transaction_row <- data.frame(CardID = card, RetailID = retailid,
                                   DateTime = datetime, Amount = amount)

  Transactions <- rbind(Transactions, new_transaction_row)
}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.

```

AubreyStatement <- Customers %>% filter(Name == 'Aubrey Sonderegger') %>%
  left_join(Cards, by = 'PersonID') %>%
  left_join(Transactions, by = 'CardID') %>%
  left_join(Retailers, by = 'RetailID') %>%
  select(CardID, DateTime, Amount, Name.y)

```

```
AubreyStatement
```

```

## # A tibble: 2 x 4
##   CardID      DateTime      Amount Name.y
##   <chr>      <dtm>      <dbl> <chr>
## 1 5628927579821287 2019-10-05 18:58:57 68.5 REI
## 2 5628927579821288 2019-10-16 21:30:21 4.98 Kickstand Kafe

```