STA445_Assignment4

Nicole Sylvester

2023-10-26

Exercises

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('',' '))
```

```
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`.*
```

```
Survey.2 <- Survey %>%
  drop_na(Year, Sex) %>%  #remove empty values
  group_by(Year, Sex) %>%
  count(Sex) %>%
   arrange(Year= fct_relevel(Year, 'FirstYear','Sophomore','Junior', 'Senior'))

Survey.2
```

```
## # A tibble: 8 x 3
## # Groups:   Year, Sex [8]
##   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  |  Sophmore  |  Junior   |  Senior   |
    |:-----------:|:------------:|:----------:|:---------:|:---------:|
    |  **Female** |              |            |           |           |
    |  **Male**   |              |            |           |           |
```

```
Survey.3 <- Survey.2 %>% pivot_longer(Year, names_to = "Year", values_to = )
Survey.3
```

```
## # A tibble: 8 x 4
## # Groups:   Year, Sex [2]
##   Sex       n Year  value
##   <chr> <int> <chr> <chr>
## 1 F        43 Year  FirstYear
## 2 M        51 Year  FirstYear
## 3 F        96 Year  Sophomore
## 4 M        99 Year  Sophomore
## 5 F        18 Year  Junior
## 6 M        17 Year  Junior
## 7 F        10 Year  Senior
## 8 M        26 Year  Senior
```

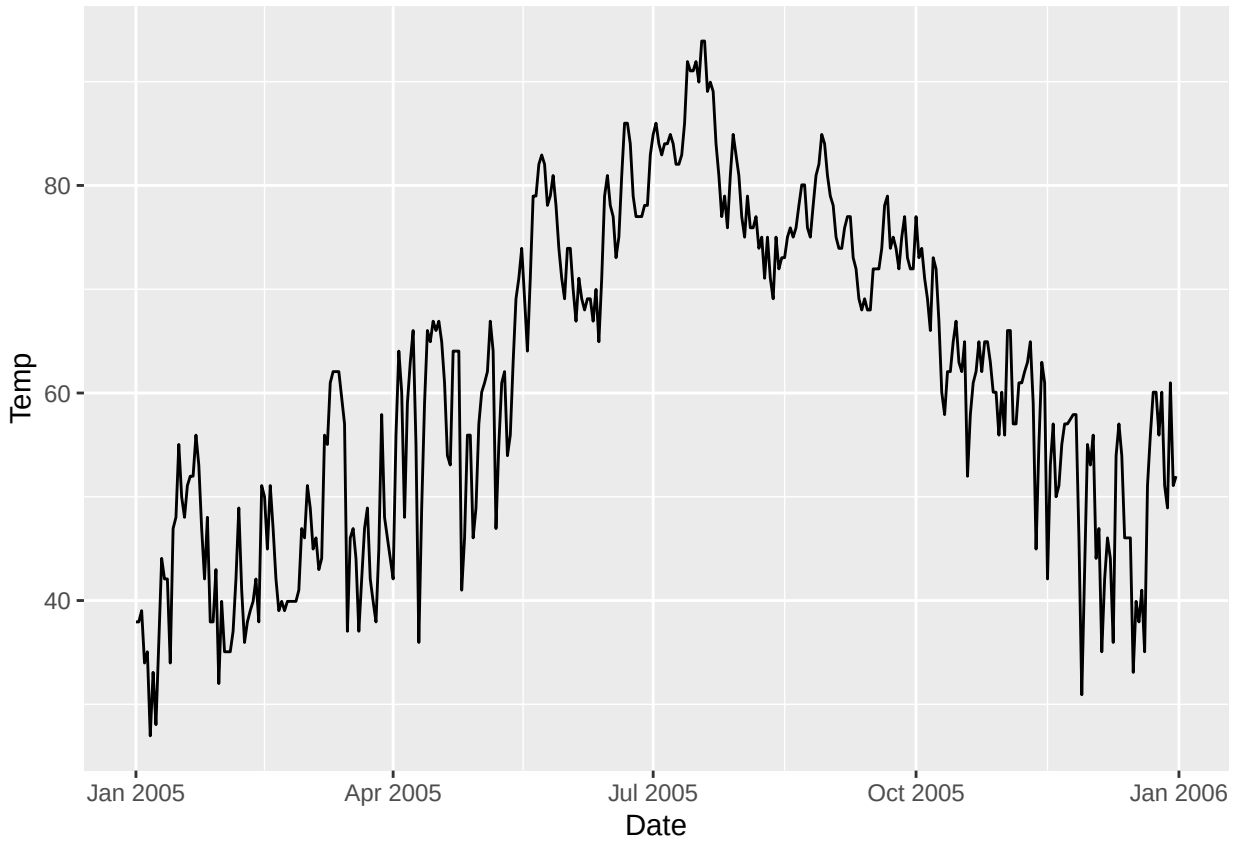
2. From the book website, there is a .csv file of the daily maximum temperature in Flagstaff at the Pulliam Airport. The direction link is at: https://raw.githubusercontent.com/dereksonderegger/444/master/data-raw/FlagMaxTemp.csv
   a) Create a line graph that gives the daily maximum temperature for 2005. *Make sure the x-axis is a date and covers the whole year.*

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

Temp_2005 <- Temp %>%
  filter(Year == 2005) %>%
  pivot_longer(
    cols = X1:X31,
    names_to = 'Day',
    values_to = 'MaxTemp'
  ) %>%
  mutate(MaxTemp = as.numeric(MaxTemp)) %>%
  mutate(Day = as.numeric(str_replace_all(Day, 'X', ""))) %>%
  mutate(Date = as.Date(paste(Year, Month, Day, sep = "-"))) %>%
  select(-X, -Year, -Month, -Day) %>%
  drop_na()
```

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

```
ggplot(data = Temp_2005, aes(x = Date, y = MaxTemp)) +
  geom_line() +
  labs(x = "Date", y = "Temp")
```



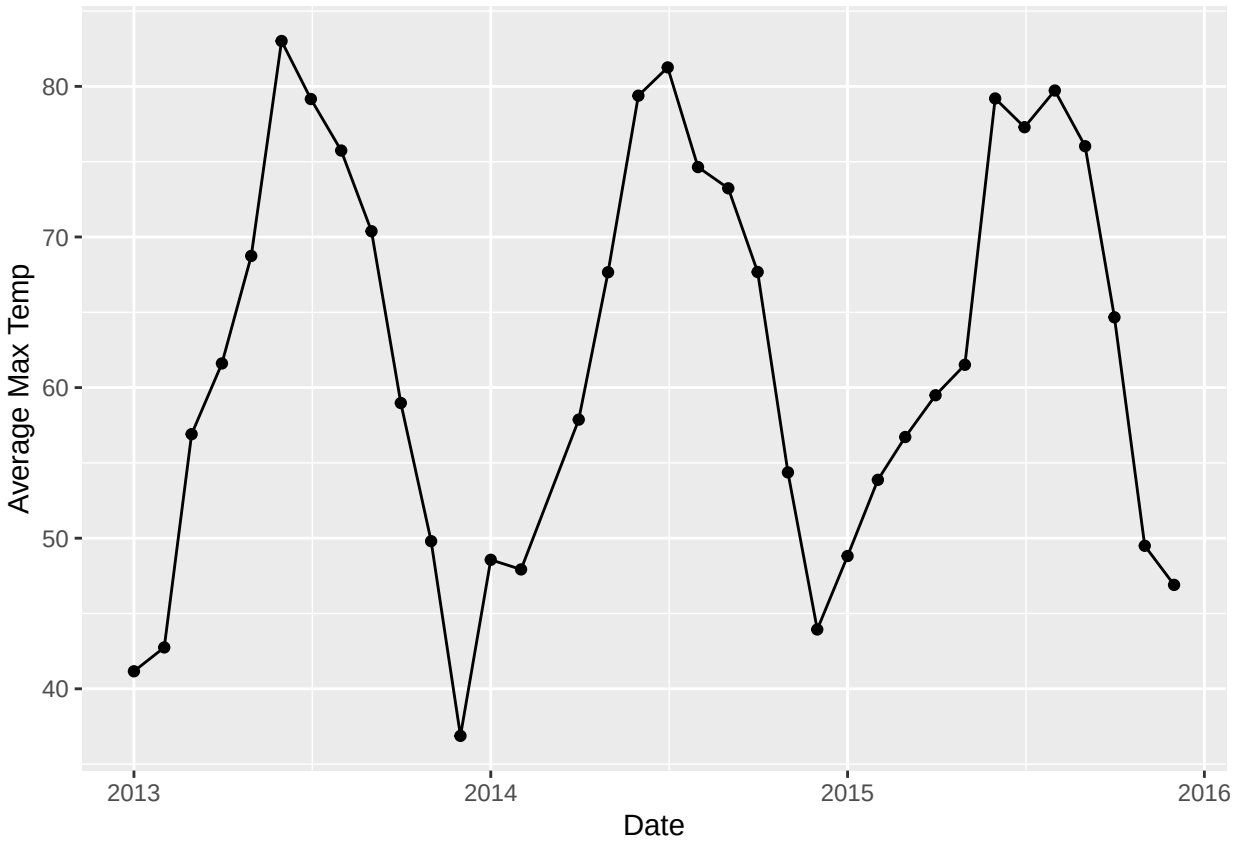
b) Create a line graph that gives the monthly average maximum temperature for 2013 - 2015. *Again the x-axis should be the date and the axis* *spans 3 years.*

```
Temp.2 <- Temp %>%
  filter(Year >= 2013 & Year <= 2015) %>%
  pivot_longer(
    cols = X1:X31,
    names_to = 'Day',
    values_to = 'MaxTemp'
  ) %>%
  mutate(MaxTemp = as.numeric(MaxTemp)) %>%
  drop_na() %>%
  group_by(Year, Month) %>%
  summarize(AvgTemp = mean(MaxTemp, na.rm = TRUE)) %>%
  ggplot(aes(x = as.Date(paste(Year, Month, "01", sep = "-")), y = AvgTemp)) +
  geom_line() + geom_point() +
  labs(x = "Date", y = "Average Max Temp")
```

```
## 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.
```

```
Temp.2
```



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.

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

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

```
People2 <- inner_join(A, B, by = c("Name" = "First.Name"))
People2
```

```
## # 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`.

```
# create a new row for Alice's guinea pig
newRow <- tribble(
  ~First.Name, ~Pet,
  'Alice', 'Guinea Pig'
)

B2 <- bind_rows(B, newRow)
B2
```

```
## # 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.

```
# PeopleCBind <- cbind(A, B2)
PeopleJoin <- inner_join(A, B2, by = c("Name" = "First.Name"))

# PeopleCBind
PeopleJoin
```

```
## # 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
```

cbind gave me an error so I commented it out. The Join was less likely to have errors in comparison to cbind.

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.*

```
derekCustomer <- Customers %>%
  filter(Name == 'Derek Sonderegger')

derekCards <- Cards %>%
  inner_join(derekCustomer, by = c("PersonID" = "PersonID"))

derekTransactions <- Transactions %>%
  inner_join(derekCards, by = "CardID")

derekStatement <- derekTransactions %>%
  inner_join(Retailers, by = c("RetailID" = "RetailID"))

derekStatement <- derekStatement %>%
  select(DateTime, Amount, Name.y)

derekStatement
```

```
## # A tibble: 3 x 3
```

```
##   DateTime            Amount Name.y
##   <dttm>               <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.

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

# get id's
personID <- aubreyInfo$PersonID
cardID <- aubreyInfo$CardID

#close card by updating expire date
closeTime <- "2019-10-15 16:28:21"

Cards <- Cards %>%
  mutate(Exp_DateTime = ifelse(CardID == cardID, closeTime, Exp_DateTime))

newCard <- rbind(data.frame(
  CardID = 8,
  PersonID = personID,
  Issue_DateTime = closeTime,
  Exp_DateTime = closeTime
))

Cards <- rbind(Cards, newCard)

Cards <- Cards %>% distinct()

Cards
```

```
## # A tibble: 5 x 4
##   CardID           PersonID Issue_DateTime      Exp_DateTime
##   <chr>               <dbl> <dttm>              <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 8                       2 2019-10-15 23:28:21 2019-10-15 16:28:21
```

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.

```
cardID <- 8
retailid <- 1
dateTime <- ymd_hms("2019-10-16 14:30:21")
amount <- 4.98

newTransaction <- data.frame(CardID = cardID, RetailID = retailid, DateTime = dateTime, Amount = amount)
transactions <- rbind(Transactions, newTransaction)

transactions
```

```
## # A tibble: 8 x 4
##   CardID           RetailID DateTime            Amount
##   <chr>               <dbl> <dttm>               <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 8                       1 2019-10-16 14:30:21   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

# 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){
  newTransaction <- data.frame(CardID = cardID, RetailID = retailid, DateTime = dateTime, Amount
  transactions <- rbind(Transactions, newTransaction)

}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.

```
AubreyCustomer <- Customers %>%
  filter(Name == 'Aubrey Sonderegger')

AubreyCards <- Cards %>%
  inner_join(AubreyCustomer, by = c("PersonID" = "PersonID"))

AubreyTransactions <- Transactions %>%
  inner_join(AubreyCards, by = "CardID")

AubreyStatement <- AubreyTransactions %>%
  inner_join(Retailers, by = c("RetailID" = "RetailID"))

AubreyStatement <- AubreyStatement %>%
  select(CardID, DateTime, Amount, Name.y)

AubreyStatement
```

```
## # A tibble: 1 x 4
##   CardID           DateTime            Amount Name.y
##   <chr>            <dttm>               <dbl> <chr>
## 1 5628927579821287 2019-10-05 18:58:57   68.5 REI
```