Assignment4

Nathan Underwood

2023-11-21

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:

```
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`.*
```

```
SurveyClean <- Survey %>%
  mutate(Year = fct_relevel(Year, "FirstYear", "Sophomore", "Junior", "Senior")) %>%
  drop_na() %>%
  count(Sex, Year)
```

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

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

```
SurveyTable <- SurveyClean %>%
  pivot_wider(names_from = Year, values_from = n)

SurveyTable
```

```
## # A tibble: 2 x 5
##   Sex   FirstYear Sophomore Junior Senior
##   <chr>     <int>     <int>  <int>  <int>
## 1 F            36        90     15     10
## 2 M            43        89     16     26
```

Question 2

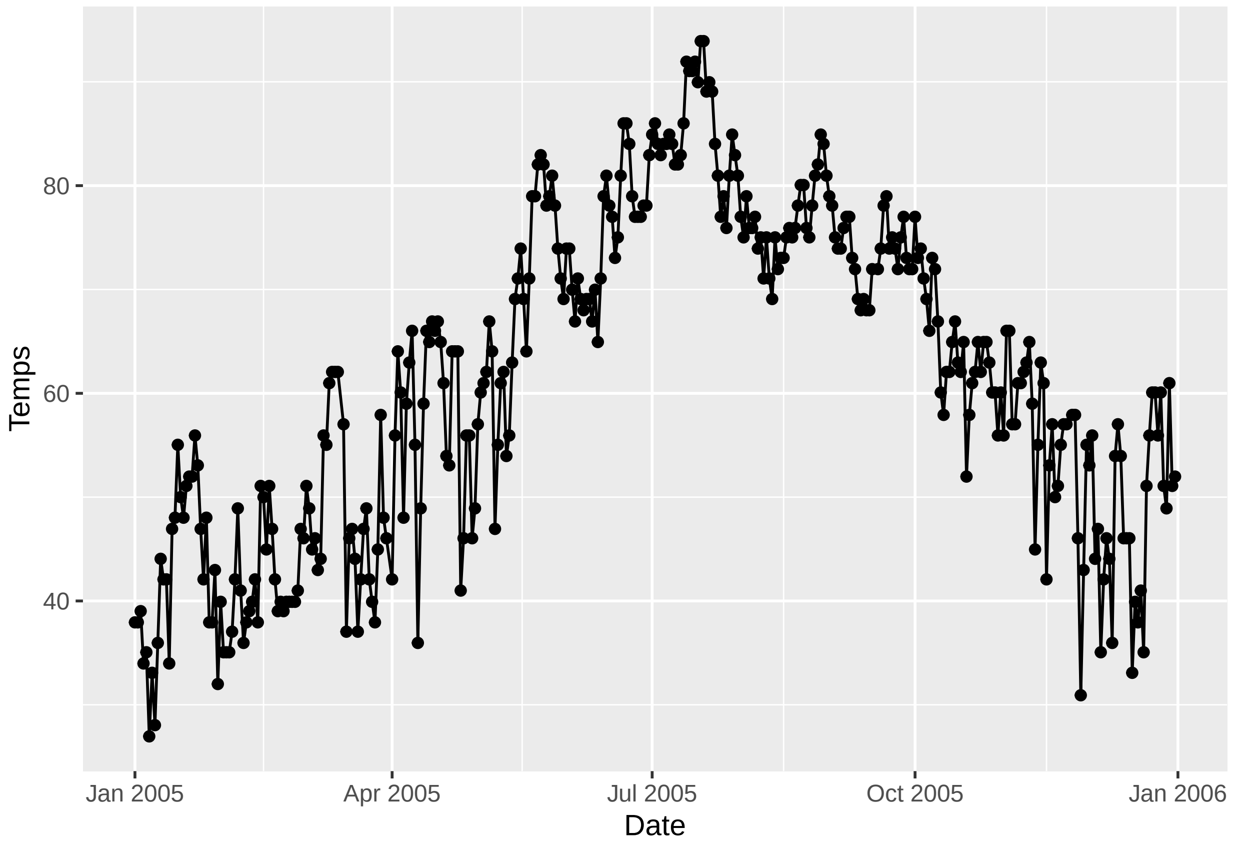
From the book website, there is a .csv file of the daily maximum temperature in Flagstaff at the Pulliam Airport. The direction link is at: https://raw.githubusercontent.com/dereksonderegger/444/master/data-raw/FlagMaxTemp.csv a) Create a line graph that gives the daily maximum temperature for 2005.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

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

MaxTemps2005 <- DailyMaxTemps %>%
  filter(Year == "2005") %>%
  pivot_longer(names_to = "Day", values_to = "Temps", X1:X31) %>%
  mutate(Day = str_replace(Day, pattern = 'X', replacement = '')) %>%
  mutate(Date = make_date(year = Year, month = Month, day = Day)) %>%
  mutate(Temps = as.numeric(Temps)) %>%
  drop_na
```

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

```
ggplot(MaxTemps2005, aes(x = Date, y = Temps)) +
  geom_point() +
  geom_line()
```



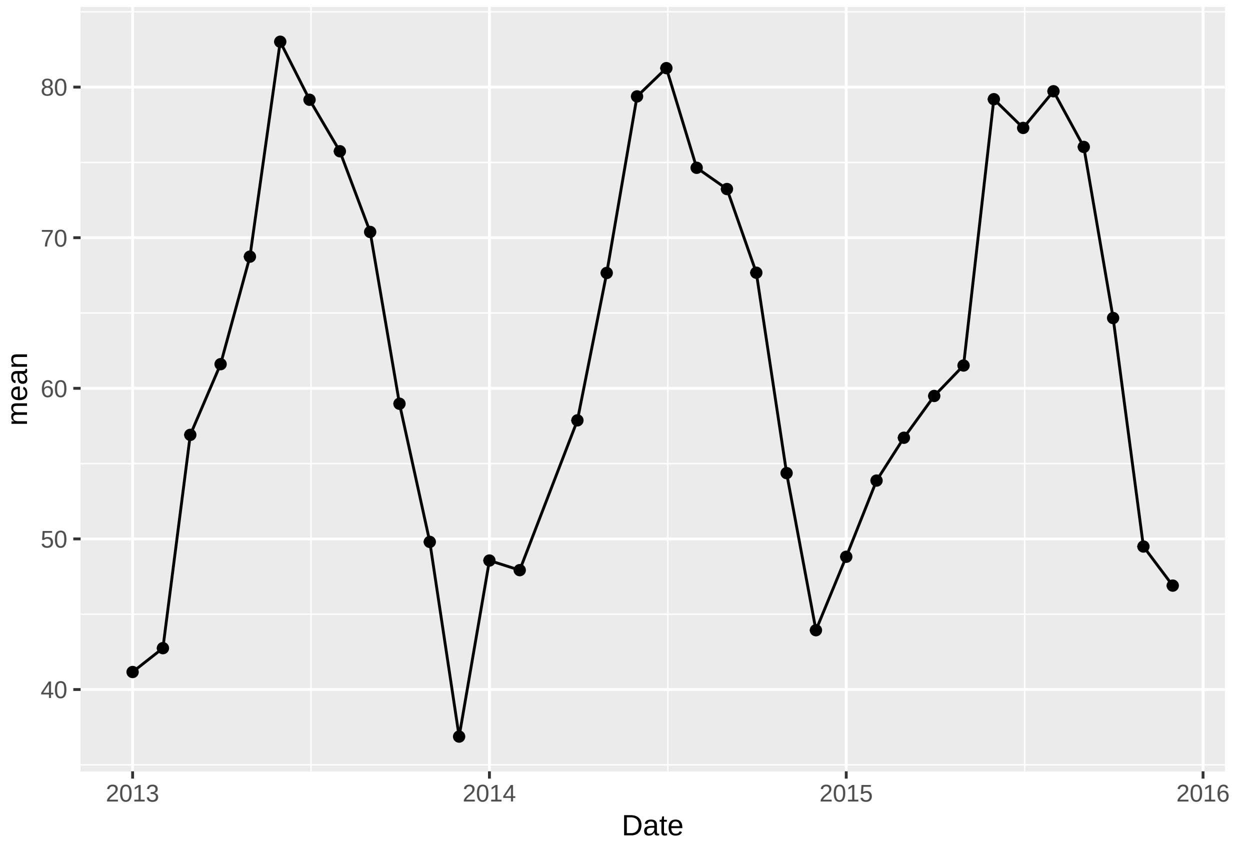
b) Create a line graph that gives the monthly average maximum temperature for 2013 - 2015. *Again the x-axis should be the date and the axis spans 3 years.*

```r
MaxTemps2013_15 <- DailyMaxTemps %>%
  filter(Year >= 2013, Year <= 2015) %>%
  pivot_longer(names_to = "Day", values_to = "Temps", X1:X31) %>%
  mutate(Day = str_replace(Day, pattern = 'X', replacement = '')) %>%
  mutate(Temps = as.numeric(Temps)) %>%
  group_by(Year, Month) %>%
  drop_na %>%
  summarise(mean = mean(Temps)) %>%
  mutate(Date = make_date(year = Year, month = Month))
```

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

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

```r
ggplot(MaxTemps2013_15, aes(x = Date, y = mean)) +
  geom_point() +
  geom_line()
```



## Question 4

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.
```

```
B <- rename(B, Name = First.Name)

C <- cbind(A, B)

C <- full_join(A, B)
```

```
## Joining with `by = join_by(Name)`
```

```
C
```

```
## # 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`.
```

```
B <- add_row(B, Name = "Alice", Pet = "Guinea Pig")

B
```

```
## # A tibble: 4 x 2
##   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.
```

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

D <- full_join(A, B)
```

```
## Joining with `by = join_by(Name)`
```

```
D
```

```
## # 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 `cbind` implementation is more likely to have errors because it struggles when there are tables of different sizes, unlike join.

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.*
```

```
DerekStatement <- Customers %>%
  filter(Name == "Derek Sonderegger") %>%
  inner_join(Cards) %>%
  inner_join(Transactions) %>%
  inner_join(Retailers, by = "RetailID")
```

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

```
DerekStatement
```

```
## # 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        1 Derek Sonderegger 231 Rive~ Flags~ AZ      98767~ 2019-09-20 00:00:00
## 2        1 Derek Sonderegger 231 Rive~ Flags~ AZ      98767~ 2019-09-20 00:00:00
## 3        1 Derek Sonderegger 231 Rive~ Flags~ AZ      98767~ 2019-09-20 00:00:00
## # 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>
```

```
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 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.*
```

```
findPersonInfo <- function(name) {
  if(name %in% Customers$Name)
  {
      dataRow <- which(Customers$Name == name, arr.ind = TRUE)
      personID <- Customers$PersonID[dataRow]
  }
}

findCardInfo <- function(name) {
  if(name %in% Customers$Name)
  {
    dataRow <- which(Cards$PersonID == personID, arr.ind = TRUE)
    cardID <- Cards$CardID[dataRow]
  }
}

personID <- findPersonInfo("Aubrey Sonderegger")
```

```
cardID <- findCardInfo("Aubrey Sonderegger")

Cards$Exp_DateTime[2] <- ymd_hms("2019-9-15 4:28:21 PM")

Cards <- Cards %>%
  add_row(CardID = "7742900840178223", PersonID = personID,
          Issue_DateTime = ymd("2019-9-15"), Exp_DateTime = ymd("2022-9-15"))

Cards
```

```
## # A tibble: 5 x 4
##   CardID           PersonID Issue_DateTime      Exp_DateTime
##   <chr>               <dbl> <dttm>              <dttm>
## 1 9876768717278723        1 2019-09-20 00:00:00 2022-09-20 00:00:00
## 2 5628927579821287        2 2019-09-20 00:00:00 2019-09-15 16:28:21
## 3 7295825498122734        3 2019-09-28 00:00:00 2022-09-28 00:00:00
## 4 8723768965231926        4 2019-09-30 00:00:00 2022-09-30 00:00:00
## 5 7742900840178223        2 2019-09-15 00:00:00 2022-09-15 00:00:00
```

The `findCardInfo` function needs an additional check to see if the card it is accessing is a valid card, rather than an expired one.
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 <- "7742900840178223"
retailID <- 1
dateTime <- ymd_hms("2019-10-16 2:30:21 PM")
amount <- 4.98

ValidCards <- Cards %>%
  filter(CardID == card, Issue_DateTime <= dateTime, dateTime <= Exp_DateTime)

if( card %in% ValidCards$CardID) {

  Transactions <- Transactions %>%
    add_row(CardID = card, RetailID = retailID, DateTime = dateTime, Amount = amount)

} else {

  print('Card Denied')

}

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 7742900840178223        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:*
```

```r
card <- "5628927579821287"
retailID <- 3
dateTime <- ymd_hms("2019-10-17 3:23:19 PM")
amount <- 374.63

if( card %in% ValidCards$CardID) {

  Transactions <- Transactions %>%
    add_row(CardID = card, RetailID = retailID, DateTime = dateTime, Amount = amount)

} 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
AubreyStatement <- Customers %>%
  filter(Name == "Aubrey Sonderegger") %>%
  inner_join(Cards) %>%
  inner_join(Transactions) %>%
  inner_join(Retailers, by = "RetailID")
```

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

```r
AubreyStatement
```

```
## # A tibble: 2 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      77429~ 2019-09-15 00:00:00
## # 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>
```