STA 445 - Assignment 6

Richard McCormick

2023-10-23

Exercise 1 – 3 pts. 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.Small <- Survey %>% count( Sex, Year ) %>% drop_na()

Survey.Small
```

```
##   Sex      Year  n
## 1   F FirstYear 43
## 2   F    Junior 18
## 3   F    Senior 10
## 4   F Sophomore 96
## 5   M FirstYear 51
## 6   M    Junior 17
## 7   M    Senior 26
## 8   M Sophomore 99
```

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

```
Survey.Small %>% pivot_wider( names_from=Year, values_from=n )
```

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

Exercise 2 – 2 pts. 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.

```
Flag.Temp <- read_csv( "https://raw.githubusercontent.com/dereksonderegger/444/master/data-raw/FlagMaxT
```

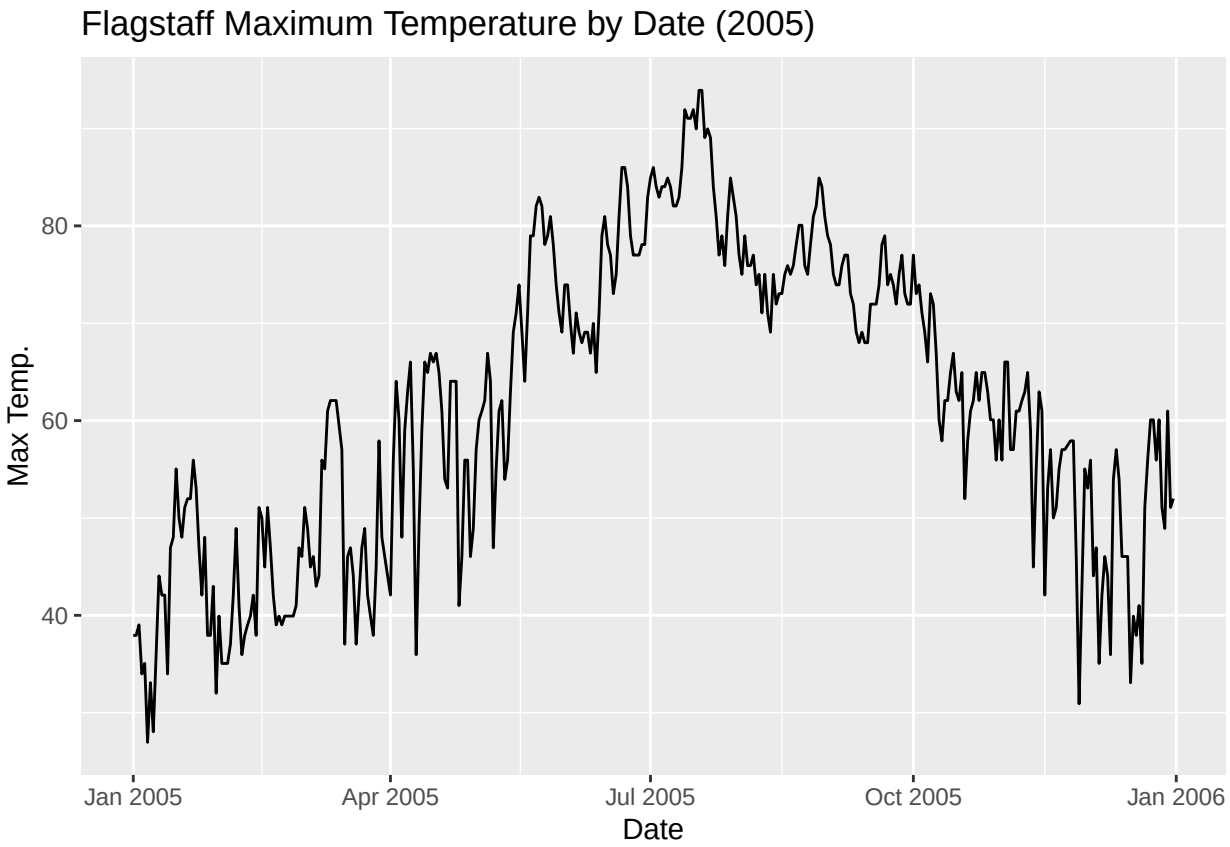
```
## New names:
## * `` -> `...1`
```

```
Flag.Temp <- Flag.Temp %>% filter( Year == 2005 ) %>%
    pivot_longer(
    4:34,                # which columns to apply this to
    names_to  = 'Day',   # What should I call the column of old column names
    values_to = 'MaxTemp') %>%
  mutate( date = mdy( paste(Month, Day, Year ) ) ) %>%
  drop_na()
```

```
## Warning: There was 1 warning in `mutate()`.
## i In argument: `date = mdy(paste(Month, Day, Year))`.
## Caused by warning:
## !  7 failed to parse.
```

```
Temp.Plot <- ggplot( data=Flag.Temp ) +
  geom_line( aes( x=date, y=MaxTemp) ) +
  labs( title="Flagstaff Maximum Temperature by Date (2005)",
        x="Date", y="Max Temp." )

Temp.Plot
```



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.

```
Flag.Temp <- read_csv( "https://raw.githubusercontent.com/dereksonderegger/444/master/data-raw/FlagMaxT
```

```
## New names:
## * `` -> `...1`
```

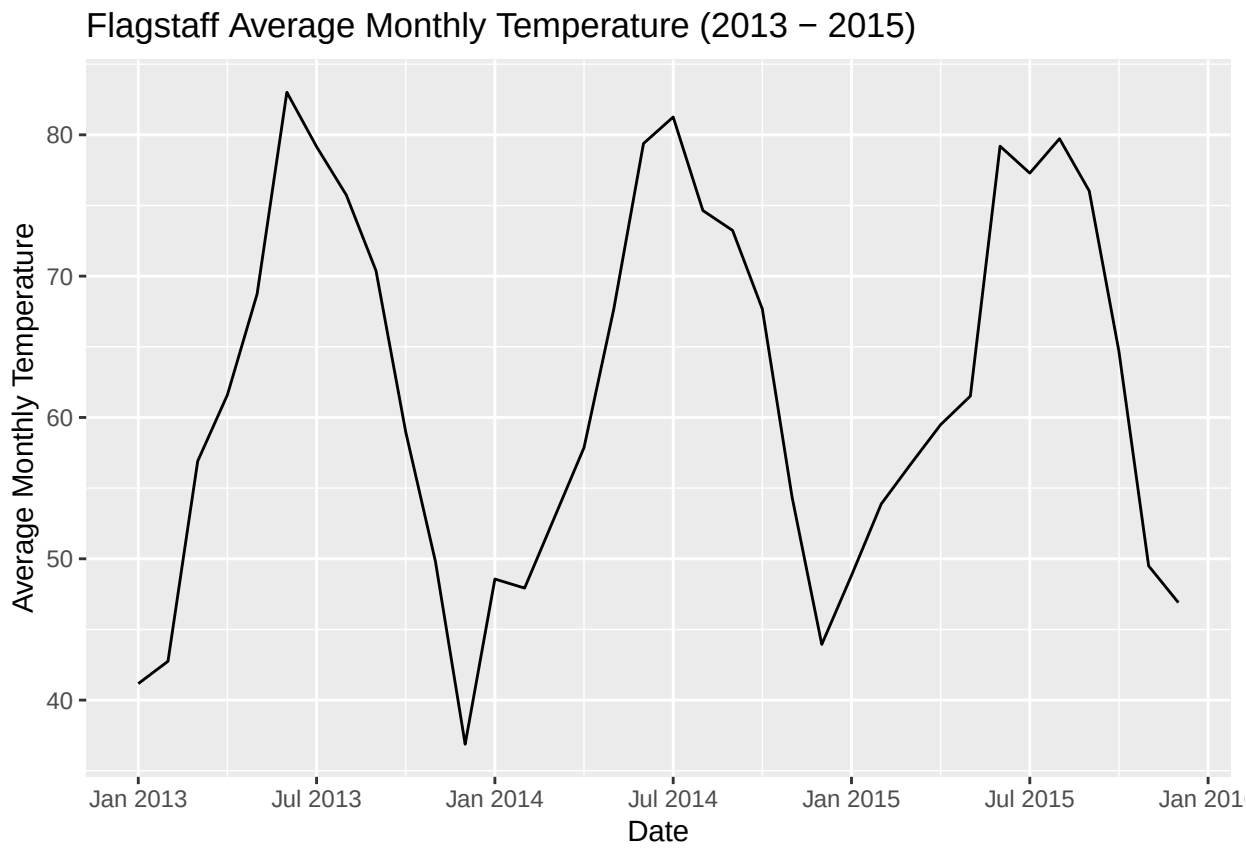
```
Flag.Temp <- Flag.Temp %>% filter( Year == 2013 | Year == 2014 | Year == 2015 ) %>%
    pivot_longer(
    4:34,                # which columns to apply this to
    names_to  = 'Day',   # What should I call the column of old column names
    values_to = 'MaxTemp') %>%
  mutate( date = mdy( paste(Month, Day, Year ) ) ) %>%
  drop_na() %>%
  group_by( Year, Month ) %>%
  summarise_at( vars( MaxTemp ), list( name = mean ) )
```

```
## Warning: There was 1 warning in `mutate()`.
## i In argument: `date = mdy(paste(Month, Day, Year))`.
## Caused by warning:
## !  21 failed to parse.
```

```
Flag.Temp$date = as.yearmon(paste( Flag.Temp$Year, Flag.Temp$Month), "%Y %m" )

Temp.Avg <- ggplot( data=Flag.Temp ) +
  geom_line( aes( x=date, y=name ) ) +
  labs( title="Flagstaff Average Monthly Temperature (2013 - 2015)",
        x="Date", y="Average Monthly Temperature" )

Temp.Avg
```



Exercise 3 – Skip: Come back to it later if you are interested. ## Challenging! We often are given data in a table format that is easy for a human to parse, but annoying a program. In the following example we have data of US government expenditures from 1962 to 2015. I downloaded this data from https://obamawhitehouse.archives.gov/omb/budget/Historicals (Table 3.2) on Sept 22, 2019. I separated the Function/Subfunction from a single column into two and removed the sub-sub functions for the military. (Look at the non-gentle version of the file to see what the original file looked like.) Our goal is to end up with a data frame with columns for Function, Subfunction, Year, and Amount. We'll ignore the "On-budget" and "Off-budget" distinction.

a. Download the data file, inspect it, and read in the data using the readxl package. Hint, your column names should be the the years.

```
history <- read_excel( "hist03z2.xls", range="A3:BJ144" )

history
```

```
## # A tibble: 141 x 62
##    `Function and Subfunction`   `1962` `1963` `1964` `1965` `1966` `1967` `1968`
##    <chr>                        <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>
##  1 050 National Defense:        <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>
##  2 051 Department of Defense-M~ <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>
##  3 Military Personnel           16331  16256  17422  17913  20009  22952  25118
##  4 Operation and Maintenance    11594  11874  11932  12349  14710  19000  20578
##  5 Procurement                  14532  16632  15351  11839  14339  19012  23283
##  6 Research, Development, Test~ 6319   6376   7021   6236   6259   7160   7747
##  7 Military Construction        1347   1144   1026   1007   1334   1536   1281
##  8 Family Housing               259    563    550    563    569    485    495
##  9 Other                        -271   -1696  -717   -1127  -590   -76    1853
## 10 051 Subtotal, Department of~ 50111  51147  52585  48780  56629  70069  80355
## # i 131 more rows
## # i 54 more variables: `1969` <chr>, `1970` <chr>, `1971` <chr>, `1972` <chr>,
## #   `1973` <chr>, `1974` <chr>, `1975` <chr>, `1976` <chr>, TQ <chr>,
## #   `1977` <chr>, `1978` <chr>, `1979` <chr>, `1980` <chr>, `1981` <chr>,
## #   `1982` <chr>, `1983` <chr>, `1984` <chr>, `1985` <chr>, `1986` <chr>,
## #   `1987` <chr>, `1988` <chr>, `1989` <chr>, `1990` <chr>, `1991` <chr>,
## #   `1992` <chr>, `1993` <chr>, `1994` <chr>, `1995` <chr>, `1996` <chr>, ...
```

b. Remove any row with Total, Subtotal, On-budget or Off-budget. Also remove the row at the bottom that defines what NA means.

```
history <- history %>%  filter(!`Function and Subfunction` %in%
                                 c( "(Off-budget)",
                                    "(On-budget)" ) ) %>%
  filter( !str_detect( `Function and Subfunction`, 'Total' ) ) %>%
  filter( !str_detect( `Function and Subfunction`, 'Subtotal' ) )

history
```

```
## # A tibble: 107 x 62
```

```
##    `Function and Subfunction`   `1962` `1963` `1964` `1965` `1966` `1967` `1968`
##    <chr>                        <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>
##  1 050 National Defense:        <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>
##  2 051 Department of Defense-M~ <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>
##  3 Military Personnel           16331  16256  17422  17913  20009  22952  25118
##  4 Operation and Maintenance    11594  11874  11932  12349  14710  19000  20578
##  5 Procurement                  14532  16632  15351  11839  14339  19012  23283
##  6 Research, Development, Test~ 6319   6376   7021   6236   6259   7160   7747
##  7 Military Construction        1347   1144   1026   1007   1334   1536   1281
##  8 Family Housing               259    563    550    563    569    485    495
##  9 Other                        -271   -1696  -717   -1127  -590   -76    1853
## 10 053 Atomic energy defense a~ 2074   2041   1902   1620   1466   1277   1336
## # i 97 more rows
## # i 54 more variables: `1969` <chr>, `1970` <chr>, `1971` <chr>, `1972` <chr>,
## #   `1973` <chr>, `1974` <chr>, `1975` <chr>, `1976` <chr>, TQ <chr>,
## #   `1977` <chr>, `1978` <chr>, `1979` <chr>, `1980` <chr>, `1981` <chr>,
## #   `1982` <chr>, `1983` <chr>, `1984` <chr>, `1985` <chr>, `1986` <chr>,
## #   `1987` <chr>, `1988` <chr>, `1989` <chr>, `1990` <chr>, `1991` <chr>,
## #   `1992` <chr>, `1993` <chr>, `1994` <chr>, `1995` <chr>, `1996` <chr>, ...
```

c. For all of the NA values in the Department column, fill them in with the value above. Hint: the function tidyr::fill() will be helpful.

```
history <- history %>%
  fill( -`Function and Subfunction` )

history
```

```
## # A tibble: 107 x 62
##    `Function and Subfunction`   `1962` `1963` `1964` `1965` `1966` `1967` `1968`
##    <chr>                        <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>
##  1 050 National Defense:        <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>
##  2 051 Department of Defense-M~ <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>
##  3 Military Personnel           16331  16256  17422  17913  20009  22952  25118
##  4 Operation and Maintenance    11594  11874  11932  12349  14710  19000  20578
##  5 Procurement                  14532  16632  15351  11839  14339  19012  23283
##  6 Research, Development, Test~ 6319   6376   7021   6236   6259   7160   7747
##  7 Military Construction        1347   1144   1026   1007   1334   1536   1281
##  8 Family Housing               259    563    550    563    569    485    495
##  9 Other                        -271   -1696  -717   -1127  -590   -76    1853
## 10 053 Atomic energy defense a~ 2074   2041   1902   1620   1466   1277   1336
## # i 97 more rows
## # i 54 more variables: `1969` <chr>, `1970` <chr>, `1971` <chr>, `1972` <chr>,
## #   `1973` <chr>, `1974` <chr>, `1975` <chr>, `1976` <chr>, TQ <chr>,
## #   `1977` <chr>, `1978` <chr>, `1979` <chr>, `1980` <chr>, `1981` <chr>,
## #   `1982` <chr>, `1983` <chr>, `1984` <chr>, `1985` <chr>, `1986` <chr>,
## #   `1987` <chr>, `1988` <chr>, `1989` <chr>, `1990` <chr>, `1991` <chr>,
## #   `1992` <chr>, `1993` <chr>, `1994` <chr>, `1995` <chr>, `1996` <chr>, ...
```

d. Remove rows that corresponded to the Function name that have no data. Hint, you can just check if the 2015 column is NA.

```
history <- history %>% drop_na()

history
```

```
## # A tibble: 105 x 62
##    `Function and Subfunction`   `1962` `1963` `1964` `1965` `1966` `1967` `1968`
##    <chr>                        <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>
##  1 Military Personnel           16331  16256  17422  17913  20009  22952  25118
##  2 Operation and Maintenance    11594  11874  11932  12349  14710  19000  20578
##  3 Procurement                  14532  16632  15351  11839  14339  19012  23283
##  4 Research, Development, Test~ 6319   6376   7021   6236   6259   7160   7747
##  5 Military Construction        1347   1144   1026   1007   1334   1536   1281
##  6 Family Housing               259    563    550    563    569    485    495
##  7 Other                        -271   -1696  -717   -1127  -590   -76    1853
##  8 053 Atomic energy defense a~ 2074   2041   1902   1620   1466   1277   1336
##  9 054 Defense-related activit~ 160    212    270    220    16     71     235
## 10 150 International Affairs:   160    212    270    220    16     71     235
## # i 95 more rows
## # i 54 more variables: `1969` <chr>, `1970` <chr>, `1971` <chr>, `1972` <chr>,
## #   `1973` <chr>, `1974` <chr>, `1975` <chr>, `1976` <chr>, TQ <chr>,
## #   `1977` <chr>, `1978` <chr>, `1979` <chr>, `1980` <chr>, `1981` <chr>,
## #   `1982` <chr>, `1983` <chr>, `1984` <chr>, `1985` <chr>, `1986` <chr>,
## #   `1987` <chr>, `1988` <chr>, `1989` <chr>, `1990` <chr>, `1991` <chr>,
## #   `1992` <chr>, `1993` <chr>, `1994` <chr>, `1995` <chr>, `1996` <chr>, ...
```

e. Reshape the data into four columns for Function, Subfunction, Year, and Amount.

```
history <- pivot_longer( history,
                         cols=2:62,
                         names_to = "Year",
                         values_to = "Amount" ) %>% na.omit()

history <- history %>%
  select(`Function and Subfunction`) %>%
  mutate(`Subfunction` = str_extract(`Function and Subfunction`, "[A-Z]"))

history
```

```
## # A tibble: 6,405 x 2
##    `Function and Subfunction` Subfunction
##    <chr>                      <chr>
##  1 Military Personnel         M
##  2 Military Personnel         M
##  3 Military Personnel         M
##  4 Military Personnel         M
##  5 Military Personnel         M
##  6 Military Personnel         M
```

```
##  7 Military Personnel         M
##  8 Military Personnel         M
##  9 Military Personnel         M
## 10 Military Personnel         M
## # i 6,395 more rows
```

f. Remove rows that have Amount value of ……. Alternatively, we could have used this as one of the NA strings during the import stage.

g. Make sure that Year and Amount are numeric. Hint: it is OK to get rid of the estimate rows for 2016+. Alternatively you could transform those by transforming 2016 estimate to just 2016.*

h. Make a line graph that compares spending for National Defense, Health, Medicare, Income Security, and Social Security for each of the years 2001 through 2015. Notice you'll have to sum up the sub-functions within each function.

Exercise 4 – 3 pts. 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.

```
# Method 1 - cbind
B.two <- B %>% rename( Name = First.Name )
B.two$Name <- as.factor( B.two$Name )

B.two <- arrange( B.two, Name )

squished <- cbind( A, B.two$Pet )
squished
```

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

```
# Method 2 - dplyr
squished.dplyr <- inner_join( A, B.two )
```

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

```
squished.dplyr
```

```
## # 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.pet <- tibble( First.Name='Alice', Pet='Guinea Pig' )
B <- rbind( B, new.pet )
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.

```
B.two <- B %>% rename( Name = First.Name )
B.two$Name <- as.factor( B.two$Name )

B.two <- arrange( B.two, Name )

squished <- merge( A, B.two )
squished
```

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

```
# Method 2 - dplyr
squished.dplyr <- inner_join( A, B.two )
```

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

```
squished.dplyr
```

```
## # 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 is significantly easier to program this function using the dplyr package than using cbind.

Exercise 5 – 5 pts. Warning: This one will take a while. 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.info <- Customers %>% filter( Name == 'Derek Sonderegger' )
derek.card <- Cards %>% filter( PersonID == derek.info$PersonID )
derek.transactions <- Transactions %>% filter( CardID == derek.card$CardID )
derek.retailers <- Retailers %>%
  filter( RetailID == derek.transactions$RetailID ) %>%
  mutate( Retailer = Name )

derek.statement = cbind( derek.info, derek.card, derek.transactions, derek.retailers )
select( derek.statement, c( 'DateTime', 'Amount', 'Retailer', 'Street' ) )
```

```
##              DateTime Amount       Retailer        Street
## 1 2019-10-01 08:31:23   5.68 Kickstand Kafe 231 River Run
## 2 2019-10-02 08:26:31   5.68 Kickstand Kafe 231 River Run
## 3 2019-10-02 08:30:09   9.23 Kickstand Kafe 231 River Run
```

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.

```
personID <- Customers %>%
  filter( Name == 'Aubrey Sonderegger' ) %>%
  select( PersonID )

cardID <- Cards %>%
  filter( PersonID == personID$PersonID ) %>%
  select( CardID )

Cards[Cards$CardID == cardID$CardID,]$Exp_DateTime <-
  mdy_hms( "Oct 15, 2019 4:28:21 PM")

new.entry = tribble( ~CardID, ~PersonID, ~Issue_DateTime, ~Exp_DateTime,
  '1234567891234567', personID$PersonID,
  mdy_hms( "Oct 15, 2019 4:28:21 PM"), mdy_hms( "Oct 15, 2019 4:28:21 PM") + dyears( 3 ) )
Cards <- rbind( Cards, new.entry )

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 2019-10-15 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 1234567891234567        2 2019-10-15 16:28:21 2022-10-15 10:28:21
```

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.

```
card <- '1234567891234567'
retailid <- 1
datetime <- ymd_hms('2019-10-16 14:30:21')
amount <- 4.98

new.transaction <- tribble(
  ~CardID, ~RetailID, ~DateTime, ~Amount,
  card, retailid, datetime, amount )
Transactions <- rbind( Transactions, new.transaction )
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 1234567891234567        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 <- 3
datetime <- ymd_hms('2019-10-17 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){
  new.transaction <- tribble(
    ~CardID, ~RetailID, ~DateTime, ~Amount,
```

```
    card, retailid, datetime, amount )
  Transactions <- rbind( Transactions, new.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.

```
# Recycled code - variables say Derek but info is for Aubrey
derek.info <- Customers %>% filter( Name == 'Aubrey Sonderegger' )
derek.card <- Cards %>% filter( PersonID == derek.info$PersonID )
derek.transactions <- Transactions %>% filter( CardID %in% derek.card$CardID )
derek.retailers <- Retailers %>%
  filter( RetailID %in% derek.transactions$RetailID ) %>%
  mutate( Retailer = Name )

derek.statement = merge( derek.retailers, derek.transactions )
select( derek.statement, c( 'DateTime', 'Amount', 'Retailer', 'CardID' ) )
```

```
##              DateTime Amount       Retailer           CardID
## 1 2019-10-16 14:30:21   4.98 Kickstand Kafe 1234567891234567
## 2 2019-10-05 18:58:57  68.54            REI 5628927579821287
```

Exercise 6 – Skip: Come back to it later if you are interested.

The package nycflights13 contains information about all the flights that arrived in or left from New York City in 2013. This package contains five data tables, but there are three data tables we will work with. The data table flights gives information about a particular flight, airports gives information about a particular airport, and airlines gives information about each airline. Create a table of all the flights on February 14th by Virgin America that has columns for the carrier, destination, departure time, and flight duration. Join this table with the airports information for the destination. Notice that because the column for the destination airport code doesn't match up between flights and airports, you'll have to use the by=c("TableA.Col"="TableB.Col") argument where you insert the correct names for TableA.Col and TableB.Col.

```
my.flights <- flights %>% filter( month == 2 ) %>% filter( day == 14 )
my.flights.select <- select( my.flights,
                             c( "carrier", "dest", "dep_time", "air_time" ) )
my.airports <- airports
my.airlines <- airlines

plane.table <- merge( my.flights.select, my.airlines )
plane.table <- merge( plane.table, my.airports,
                      by.x="dest", by.y="faa" ) %>%
  mutate( Destination.Airport = name.y, Airline = name.x ) %>%
  filter( Airline == "Virgin America" )
plane.table <- subset( plane.table, select=-c( name.x, name.y ) )

plane.table
```

```
##    dest carrier dep_time air_time      lat       lon  alt tz dst
## 1   LAS      VX      934      307 36.08006 -115.1522 2141 -8   A
## 2   LAX      VX     1317      349 33.94254 -118.4081  126 -8   A
## 3   LAX      VX      909      341 33.94254 -118.4081  126 -8   A
## 4   LAX      VX      706      347 33.94254 -118.4081  126 -8   A
## 5   LAX      VX     1706      335 33.94254 -118.4081  126 -8   A
## 6   LAX      VX     2017      337 33.94254 -118.4081  126 -8   A
## 7   SFO      VX     1746      358 37.61897 -122.3749   13 -8   A
## 8   SFO      VX     1029      351 37.61897 -122.3749   13 -8   A
## 9   SFO      VX     1852      355 37.61897 -122.3749   13 -8   A
## 10  SFO      VX      732      344 37.61897 -122.3749   13 -8   A
##                  tzone Destination.Airport        Airline
## 1  America/Los_Angeles      Mc Carran Intl Virgin America
## 2  America/Los_Angeles    Los Angeles Intl Virgin America
## 3  America/Los_Angeles    Los Angeles Intl Virgin America
## 4  America/Los_Angeles    Los Angeles Intl Virgin America
## 5  America/Los_Angeles    Los Angeles Intl Virgin America
## 6  America/Los_Angeles    Los Angeles Intl Virgin America
## 7  America/Los_Angeles  San Francisco Intl Virgin America
## 8  America/Los_Angeles  San Francisco Intl Virgin America
## 9  America/Los_Angeles  San Francisco Intl Virgin America
## 10 America/Los_Angeles  San Francisco Intl Virgin America
```