STA 445 Assignment #6

Chip Haskins

2024-11-04

Exercise 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 data set summarizing student surveys from several sections of an intro class. The two variables of interest are `Gender` and `Year` which are the students gender and year in college. *Note: you will need to refer to Chapter 4 and Chapter 7 for some of the operations needed below - this is a great time to review chapter 4!*

a) Download the data set using the following:

```
Survey <- read.csv('https://www.lock5stat.com/datasets2e/StudentSurvey.csv', na.strings=c('',' '))
```

b) Select the specific columns of interest **Year** and **Gender**

```
Survey <- Survey %>% select(Year, Gender)
head(Survey)
```

```
##        Year Gender
## 1    Senior      M
## 2 Sophomore      F
## 3 FirstYear      M
## 4    Junior      M
## 5 Sophomore      F
## 6 Sophomore      F
```

c) Convert the **Year** column to factors and properly order the factors based on common US progression (FirstYear - Sophomore - Junior - Senior)

```
Survey <- Survey %>% mutate(
  Year = fct_relevel(Year, 'FirstYear', 'Sophomore', 'Junior', 'Senior')
)
head(Survey)
```

```
##        Year Gender
## 1    Senior      M
## 2 Sophomore      F
## 3 FirstYear      M
## 4    Junior      M
## 5 Sophomore      F
## 6 Sophomore      F
```

d) Convert the **Gender** column to factors and rename them Male/Female.

```
Survey <- Survey %>% mutate(
  Gender = fct_relabel(Gender, ~ ifelse(. == "M", "Male", "Female"))
)
head(Survey)
```

```
##        Year Gender
## 1    Senior   Male
## 2 Sophomore Female
## 3 FirstYear   Male
## 4    Junior   Male
## 5 Sophomore Female
## 6 Sophomore Female
```

e) Produce a data set with eight rows and three columns that contains the number of responses for each gender:year combination. *You might want to look at the following functions:* ***dplyr::count*** *and* ***dplyr::drop_na****.*

```
Survey <- Survey %>%
  group_by(Year, Gender) %>%
  drop_na() %>%
  count(name = "Count")
Survey
```

```
## # A tibble: 8 x 3
## # Groups:   Year, Gender [8]
##   Year      Gender Count
##   <fct>     <fct>  <int>
## 1 FirstYear Female    43
## 2 FirstYear Male      51
## 3 Sophomore Female    96
## 4 Sophomore Male      99
## 5 Junior    Female    18
## 6 Junior    Male      17
## 7 Senior    Female    10
## 8 Senior    Male      26
```

f) Pivot the table in part (e) to produce a table of the number of responses in the following form:

Gender	First Year	Sophomore	Junior	Senior
Female				
Male				

```
Survey <- Survey %>% pivot_wider(
  names_from = "Year",
  values_from = "Count"
)
Survey
```

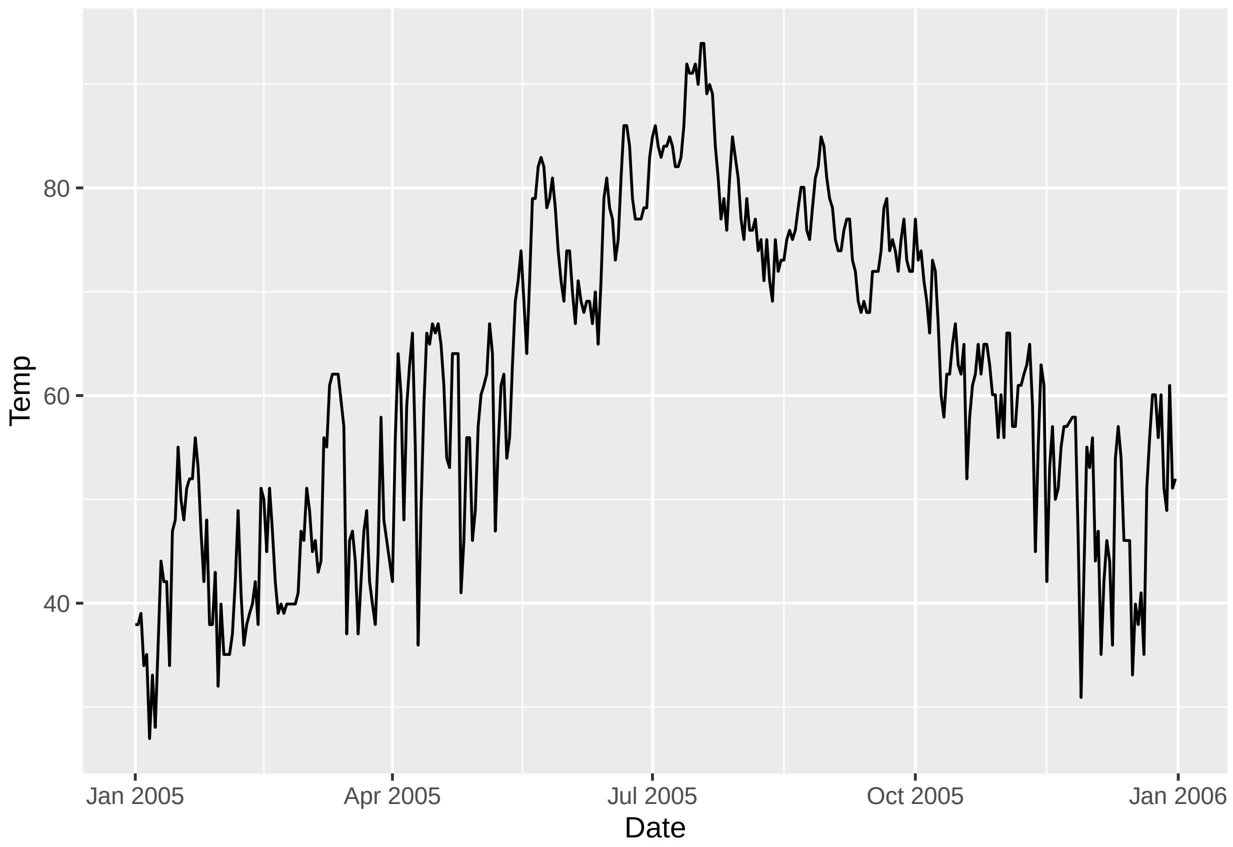
```
## # A tibble: 2 x 5
## # Groups:   Gender [2]
##   Gender FirstYear Sophomore Junior Senior
##   <fct>      <int>     <int>  <int>  <int>
## 1 Female        43        96     18     10
## 2 Male          51        99     17     26
```

Exercise 2

From this book's GitHub there is a .csv file of the daily maximum temperature in Flagstaff at the Pulliam Airport. The link is: https://raw.githubusercontent.com/BuscagliaR/STA_444_v2/master/data-raw/FlagMaxTemp.csv

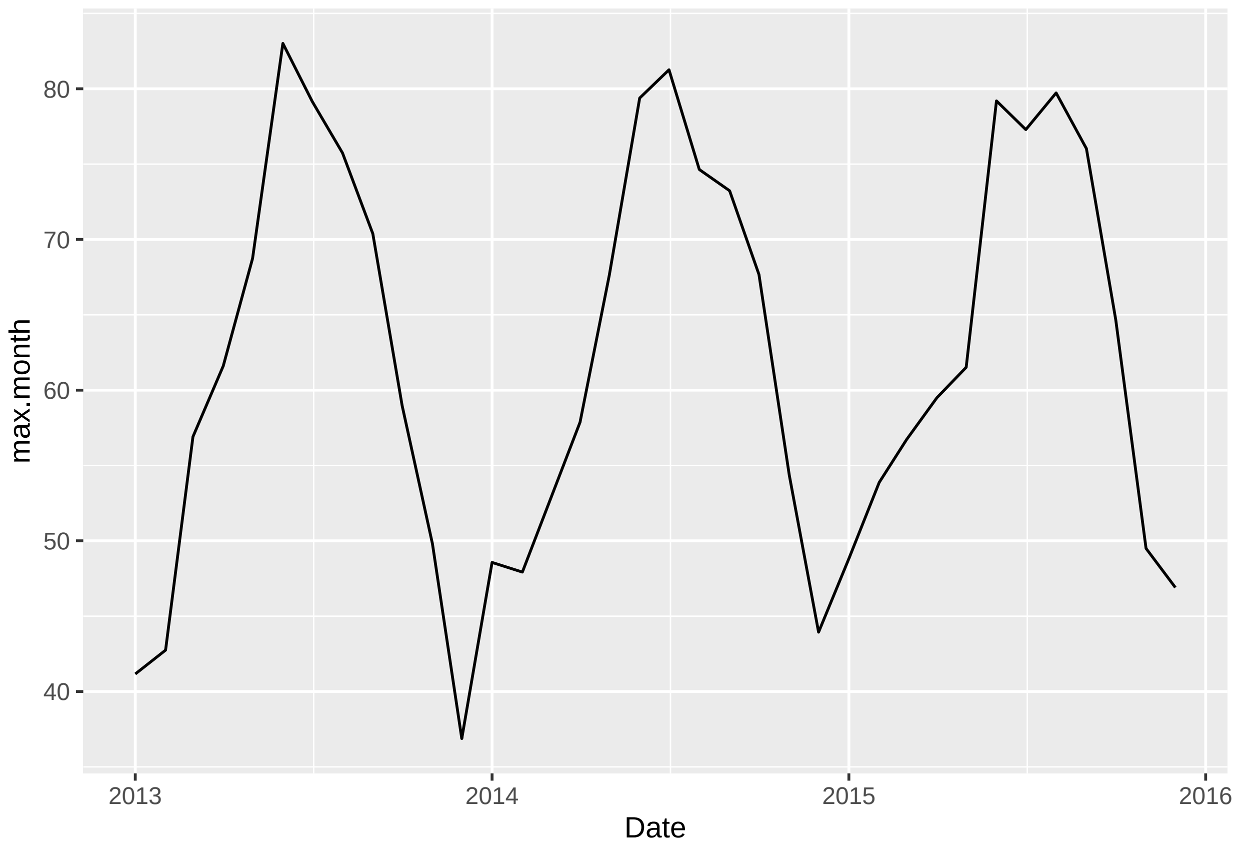
a) Create a line graph that gives the daily maximum temperature for 2005. *Make sure the x-axis is a date and covers the whole year.*

```
FlagTemp <-
  read_csv(
    "https://raw.githubusercontent.com/BuscagliaR/STA_444_v2/master/data-raw/FlagMaxTemp.csv",
    col_select = !...1,
    show_col_types = FALSE,
    name_repair = "unique_quiet"
  ) %>%
  pivot_longer(
    !(Year | Month),
    names_to = "Day",
    values_to = "Temp"
  ) %>%
  drop_na()

FlagTemp.daily <- FlagTemp %>%
  filter(Year == 2005) %>%
  mutate(
    Date = paste(Year, Month, Day) %>% ymd()
  )

ggplot(FlagTemp.daily, aes(x = Date,y = Temp)) +
  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 span 3 years.*

```r
FlagTemp.monthly <- FlagTemp %>%
  filter(between(Year, 2013, 2015)) %>%
  summarize(max.month = mean(Temp), .by = c(Year, Month)) %>%
  mutate(
    Date = paste(Year, Month) %>% parse_date_time("ym")
  )

ggplot(FlagTemp.monthly, aes(x = Date, y = max.month)) +
  geom_line()
```



Exercise 3

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) Combine the data frames together to generate a data set with three rows and three columns using `join` commands.

```
full_join(A, B, by = c("Name" = "First.Name"))
```

```
## # 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, First.Name = "Alice", Pet = "Guinea Pig")
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) Combine again the `A` and `B` data sets together to generate a data set with four rows and three columns using `join` commands.

```
full_join(A, B, by = c("Name" = "First.Name"))
```

```
## # 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
```

Note: You may want to also try using `cbind` to address questions (a) and (c). Leave this as a challenge question and focus on the easier to use `join` functions introduced in this chapter.

Exercise 4

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`.

```
library(nycflights13)
data(flights)
data(airports)
data(airlines)
```

```
flights <- flights %>%
  filter(month == 2, day == 14, carrier == "VX") %>%
  select(carrier, dest, dep_time, air_time)

left_join(flights, airports, by = c("dest"="faa"))
```

```
## # A tibble: 10 x 11
##    carrier dest  dep_time air_time name          lat   lon   alt    tz dst   tzone
##    <chr>   <chr>    <int>    <dbl> <chr>     <dbl> <dbl> <dbl> <dbl> <chr> <chr>
##  1 VX      LAX        706      347 Los Ange~  33.9 -118.   126    -8 A     Amer~
##  2 VX      SFO        732      344 San Fran~  37.6 -122.    13    -8 A     Amer~
##  3 VX      LAX        909      341 Los Ange~  33.9 -118.   126    -8 A     Amer~
##  4 VX      LAS        934      307 Mc Carra~  36.1 -115.  2141    -8 A     Amer~
##  5 VX      SFO       1029      351 San Fran~  37.6 -122.    13    -8 A     Amer~
##  6 VX      LAX       1317      349 Los Ange~  33.9 -118.   126    -8 A     Amer~
##  7 VX      LAX       1706      335 Los Ange~  33.9 -118.   126    -8 A     Amer~
##  8 VX      SFO       1746      358 San Fran~  37.6 -122.    13    -8 A     Amer~
##  9 VX      SFO       1852      355 San Fran~  37.6 -122.    13    -8 A     Amer~
## 10 VX      LAX       2017      337 Los Ange~  33.9 -118.   126    -8 A     Amer~
```

Optional Exercises

Exercise 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 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.*

```
Customers.Derek <- Customers %>% filter(Name == "Derek Sonderegger")

left_join(Customers.Derek, Cards, join_by(PersonID)) %>%
  left_join(Transactions, join_by(CardID)) %>%
  left_join(Retailers, join_by(RetailID), suffix = c(".Customer", ".Retailer")) %>%
  select(DateTime, Amount, ends_with(".Retailer"))
```

```
## # A tibble: 3 x 6
##   DateTime            Amount Name.Retailer  Street.Retailer    City.Retailer
##   <dttm>               <dbl> <chr>          <chr>              <chr>
## 1 2019-10-01 08:31:23   5.68 Kickstand Kafe 719 N Humphreys St Flagstaff
## 2 2019-10-02 08:26:31   5.68 Kickstand Kafe 719 N Humphreys St Flagstaff
## 3 2019-10-02 08:30:09   9.23 Kickstand Kafe 719 N Humphreys St Flagstaff
## # i 1 more variable: State.Retailer <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 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.*

```
Customers.Aubrey <- Customers %>% filter(Name == "Aubrey Sonderegger")

Customers.Aubrey <- left_join(Customers.Aubrey, Cards, join_by(PersonID))

cardID <- Customers.Aubrey %>% pull(CardID)
personID <- Customers.Aubrey %>% pull(PersonID)

Cards %<>% mutate(Exp_DateTime = replace(
  Exp_DateTime, CardID == cardID, mdy_hms("Oct 15, 2019 4:28:21 PM"))) %>%
  add_row(
    CardID = "8257473951384659",
    PersonID = personID,
    Issue_DateTime = ymd("2019-10-16"),
    Exp_DateTime = ymd("2022-10-16")
    )
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 8257473951384659        2 2019-10-16 00:00:00 2022-10-16 00:00:00
```

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.*

```
add_transaction <- function(.data, card, retailid, datetime, amount) {
  # get card info to check transaction validity
  Card_Info <- Cards %>% filter(CardID == card)
  # interval representing when the card was valid
  valid_card <- Card_Info$Issue_DateTime %--% Card_Info$Exp_DateTime

  # If the transaction is not valid, return with error message
  if (!(datetime %within% valid_card)) {
    print('Card Denied')
    return(.data)
  }

  # insert the transaction into the table if transaction valid

  # add new row to input data with supplied args as values
  .data %<>% add_row(
    CardID = card,
    RetailID = retailid,
    DateTime = datetime,
    Amount = amount
  )

  invisible(.data)
}

Transactions %<>%
  add_transaction("8257473951384659", 1, mdy_hms("Oct 16, 2019 2:30:21 PM"), 4.98)
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 8257473951384659        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 <- mdy("Oct 17, 2019")
amount <- 4.98

Transactions %<>% add_transaction(card, retailid, datetime, amount)
```

```
## [1] "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 8257473951384659        1 2019-10-16 14:30:21   4.98
```

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.

```
Customers.Aubrey <- Customers %>% filter(Name == "Aubrey Sonderegger")

left_join(Customers.Aubrey, Cards, join_by(PersonID)) %>%
  left_join(Transactions, join_by(CardID)) %>%
  left_join(Retailers, join_by(RetailID), suffix = c(".Customer", ".Retailer")) %>%
  select(CardID, DateTime, Amount, ends_with(".Retailer"))
```

```
## # A tibble: 2 x 7
##   CardID  DateTime            Amount Name.Retailer Street.Retailer City.Retailer
##   <chr>   <dttm>               <dbl> <chr>         <chr>           <chr>
## 1 562892~ 2019-10-05 18:58:57  68.5  REI           323 S Windsor ~ Flagstaff
## 2 825747~ 2019-10-16 14:30:21   4.98 Kickstand Ka~ 719 N Humphrey~ Flagstaff
## # i 1 more variable: State.Retailer <chr>
```

Exercise 6

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 can download such data from the book's GitHub at this link, which provides US government expenditures from 1962 to 2015. (Data available from ObamaWhiteHouse, Table 3.2, downloaded Sept 22, 2019.) Our goal is to end up with a data frame with columns for `Function`, `Subfunction`, `Year`, and `Amount`. *We will ignore the "On-budget" and "Off-budget" distinction.*

a) Download the data file, inspect it, and read in the data using the `readxl` package.

```
US_Budget <- read_excel("../data-raw/US_Gov_Budget_1962_2020.xls", skip = 2)
head(US_Budget)
```

```
## # A tibble: 6 x 62
##   Function and Subfunc~1 ‘1962‘ ‘1963‘ ‘1964‘ ‘1965‘ ‘1966‘ ‘1967‘ ‘1968‘ ‘1969‘
##   <chr>                  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>
## 1 050 National Defense:  <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>
## 2 051 Department of Def~ <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>
## 3 Military Personnel     16331  16256  17422  17913  20009  22952  25118  26914
## 4 Operation and Mainten~ 11594  11874  11932  12349  14710  19000  20578  22227
## 5 Procurement            14532  16632  15351  11839  14339  19012  23283  23988
## 6 Research, Development~ 6319   6376   7021   6236   6259   7160   7747   7457
## # i abbreviated name: 1: ‘Function and Subfunction‘
## # i 53 more variables: ‘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>, ‘1997‘ <chr>, ...
```

b) Rename the `Function or subfunction` column to `Department`.

```
US_Budget %<>% rename("Department" = `Function and Subfunction`)
head(US_Budget, 10)
```

```
## # A tibble: 10 x 62
##    Department     ‘1962‘ ‘1963‘ ‘1964‘ ‘1965‘ ‘1966‘ ‘1967‘ ‘1968‘ ‘1969‘ ‘1970‘
##    <chr>          <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>
##  1 050 National ~ <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>
##  2 051 Departmen~ <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>
##  3 Military Pers~ 16331  16256  17422  17913  20009  22952  25118  26914  29032
##  4 Operation and~ 11594  11874  11932  12349  14710  19000  20578  22227  21609
##  5 Procurement    14532  16632  15351  11839  14339  19012  23283  23988  21584
##  6 Research, Dev~ 6319   6376   7021   6236   6259   7160   7747   7457   7166
##  7 Military Cons~ 1347   1144   1026   1007   1334   1536   1281   1389   1168
##  8 Family Housing 259    563    550    563    569    485    495    574    614
##  9 Other          -271   -1696  -717   -1127  -590   -76    1853   -1777  -1050
## 10 051 Subtotal,~ 50111  51147  52585  48780  56629  70069  80355  80771  80123
## # i 52 more variables: ‘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>, ‘1997‘ <chr>, ‘1998‘ <chr>,
## #   ‘1999‘ <chr>, ‘2000‘ <chr>, ‘2001‘ <chr>, ‘2002‘ <chr>, ‘2003‘ <chr>, ...
```

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

```
US_Budget %<>%
  filter(
    !str_detect(Department, regex("total|subtotal|on-budget|off-budget|not available",
                                  ignore_case = TRUE)))
head(US_Budget, 10)
```

```
## # A tibble: 10 x 62
##    Department     ‘1962‘ ‘1963‘ ‘1964‘ ‘1965‘ ‘1966‘ ‘1967‘ ‘1968‘ ‘1969‘ ‘1970‘
##    <chr>          <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>
##  1 050 National ~ <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>
##  2 051 Departmen~ <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>
##  3 Military Pers~ 16331  16256  17422  17913  20009  22952  25118  26914  29032
##  4 Operation and~ 11594  11874  11932  12349  14710  19000  20578  22227  21609
##  5 Procurement    14532  16632  15351  11839  14339  19012  23283  23988  21584
##  6 Research, Dev~ 6319   6376   7021   6236   6259   7160   7747   7457   7166
##  7 Military Cons~ 1347   1144   1026   1007   1334   1536   1281   1389   1168
##  8 Family Housing 259    563    550    563    569    485    495    574    614
##  9 Other          -271   -1696  -717   -1127  -590   -76    1853   -1777  -1050
## 10 053 Atomic en~ 2074   2041   1902   1620   1466   1277   1336   1389   1415
## # i 52 more variables: ‘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>, ‘1997‘ <chr>, ‘1998‘ <chr>,
## #   ‘1999‘ <chr>, ‘2000‘ <chr>, ‘2001‘ <chr>, ‘2002‘ <chr>, ‘2003‘ <chr>, ...
```

d) Create a new column for `ID_number` and parse the `Department` column for it.

```
US_Budget %<>% mutate(ID_number = str_match(Department, "^\\d{3}"), .before = Department)
head(US_Budget, 10)
```

```
## # A tibble: 10 x 63
##    ID_number[,1] Department     ‘1962‘ ‘1963‘ ‘1964‘ ‘1965‘ ‘1966‘ ‘1967‘ ‘1968‘
##    <chr>         <chr>          <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>
##  1 050           050 National ~ <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>
##  2 051           051 Departmen~ <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>
##  3 <NA>          Military Pers~ 16331  16256  17422  17913  20009  22952  25118
##  4 <NA>          Operation and~ 11594  11874  11932  12349  14710  19000  20578
##  5 <NA>          Procurement    14532  16632  15351  11839  14339  19012  23283
##  6 <NA>          Research, Dev~ 6319   6376   7021   6236   6259   7160   7747
##  7 <NA>          Military Cons~ 1347   1144   1026   1007   1334   1536   1281
##  8 <NA>          Family Housing 259    563    550    563    569    485    495
##  9 <NA>          Other          -271   -1696  -717   -1127  -590   -76    1853
## 10 053           053 Atomic en~ 2074   2041   1902   1620   1466   1277   1336
## # 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>,
## #   ‘1997‘ <chr>, ‘1998‘ <chr>, ‘1999‘ <chr>, ‘2000‘ <chr>, ‘2001‘ <chr>, ...
```

e) If all (or just 2015?) the year values are missing, then the `Department` corresponds to `Function` name. Otherwise `Department` corresponds to the `Subfunction`. Create columns for `Function` and `Subfunction`. *Hint: Directly copy `Department` to `Subfunction`. Then using an `if_else()` statement to copy either `NA` or `Department` to `Function` depending on if the 2015 column is an `NA` (use the function `is.na()`). Once you have `Function` with either the `Function` name or an `NA`, you can use the `tidyr::fill` command to replace the NA values with whatever is on the row above. Check out the help files to see how to use it.*

```r
US_Budget %<>% mutate(
  Function = if_else(is.na(`2015`), Department, NA),
  Subfunction = Department
) %>%
  fill(Function)
head(US_Budget)
```

```
## # A tibble: 6 x 65
##   ID_number[,1] Department      ‘1962‘ ‘1963‘ ‘1964‘ ‘1965‘ ‘1966‘ ‘1967‘ ‘1968‘
##   <chr>         <chr>           <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>
## 1 050           050 National D~ <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>
## 2 051           051 Department~ <NA>   <NA>   <NA>   <NA>   <NA>   <NA>   <NA>
## 3 <NA>          Military Perso~ 16331  16256  17422  17913  20009  22952  25118
## 4 <NA>          Operation and ~ 11594  11874  11932  12349  14710  19000  20578
## 5 <NA>          Procurement     14532  16632  15351  11839  14339  19012  23283
## 6 <NA>          Research, Deve~ 6319   6376   7021   6236   6259   7160   7747
## # i 56 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>,
## #   ‘1997‘ <chr>, ‘1998‘ <chr>, ‘1999‘ <chr>, ‘2000‘ <chr>, ‘2001‘ <chr>, ...
```

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

```r
US_Budget %<>% filter(!is.na(`2015`))
head(US_Budget)
```

```
## # A tibble: 6 x 65
##   ID_number[,1] Department      ‘1962‘ ‘1963‘ ‘1964‘ ‘1965‘ ‘1966‘ ‘1967‘ ‘1968‘
##   <chr>         <chr>           <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>
## 1 <NA>          Military Perso~ 16331  16256  17422  17913  20009  22952  25118
## 2 <NA>          Operation and ~ 11594  11874  11932  12349  14710  19000  20578
## 3 <NA>          Procurement     14532  16632  15351  11839  14339  19012  23283
## 4 <NA>          Research, Deve~ 6319   6376   7021   6236   6259   7160   7747
## 5 <NA>          Military Const~ 1347   1144   1026   1007   1334   1536   1281
## 6 <NA>          Family Housing  259    563    550    563    569    485    495
## # i 56 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>,
## #   ‘1997‘ <chr>, ‘1998‘ <chr>, ‘1999‘ <chr>, ‘2000‘ <chr>, ‘2001‘ <chr>, ...
```

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

```r
US_Budget %<>% pivot_longer(`1962`:`2021 estimate`,
                            names_to = "Year",
                            values_to = "Amount") %>%
  select(Function, Subfunction, Year, Amount)
head(US_Budget)
```

```
## # A tibble: 6 x 4
##   Function                            Subfunction        Year  Amount
##   <chr>                               <chr>              <chr> <chr>
## 1 051 Department of Defense-Military: Military Personnel 1962  16331
## 2 051 Department of Defense-Military: Military Personnel 1963  16256
## 3 051 Department of Defense-Military: Military Personnel 1964  17422
## 4 051 Department of Defense-Military: Military Personnel 1965  17913
## 5 051 Department of Defense-Military: Military Personnel 1966  20009
## 6 051 Department of Defense-Military: Military Personnel 1967  22952
```

h) Remove rows that have Amount value of ....

```r
US_Budget %<>% filter(Amount != "..........")
tail(US_Budget)
```

```
## # A tibble: 6 x 4
##   Function                               Subfunction                Year  Amount
##   <chr>                                  <chr>                      <chr> <chr>
## 1 950 Undistributed Offsetting Receipts: 959 Other undistributed o~ 2016~ -12925
## 2 950 Undistributed Offsetting Receipts: 959 Other undistributed o~ 2017~ -15700
## 3 950 Undistributed Offsetting Receipts: 959 Other undistributed o~ 2018~ -8050
## 4 950 Undistributed Offsetting Receipts: 959 Other undistributed o~ 2019~ -575
## 5 950 Undistributed Offsetting Receipts: 959 Other undistributed o~ 2020~ -1115
## 6 950 Undistributed Offsetting Receipts: 959 Other undistributed o~ 2021~ -965
```

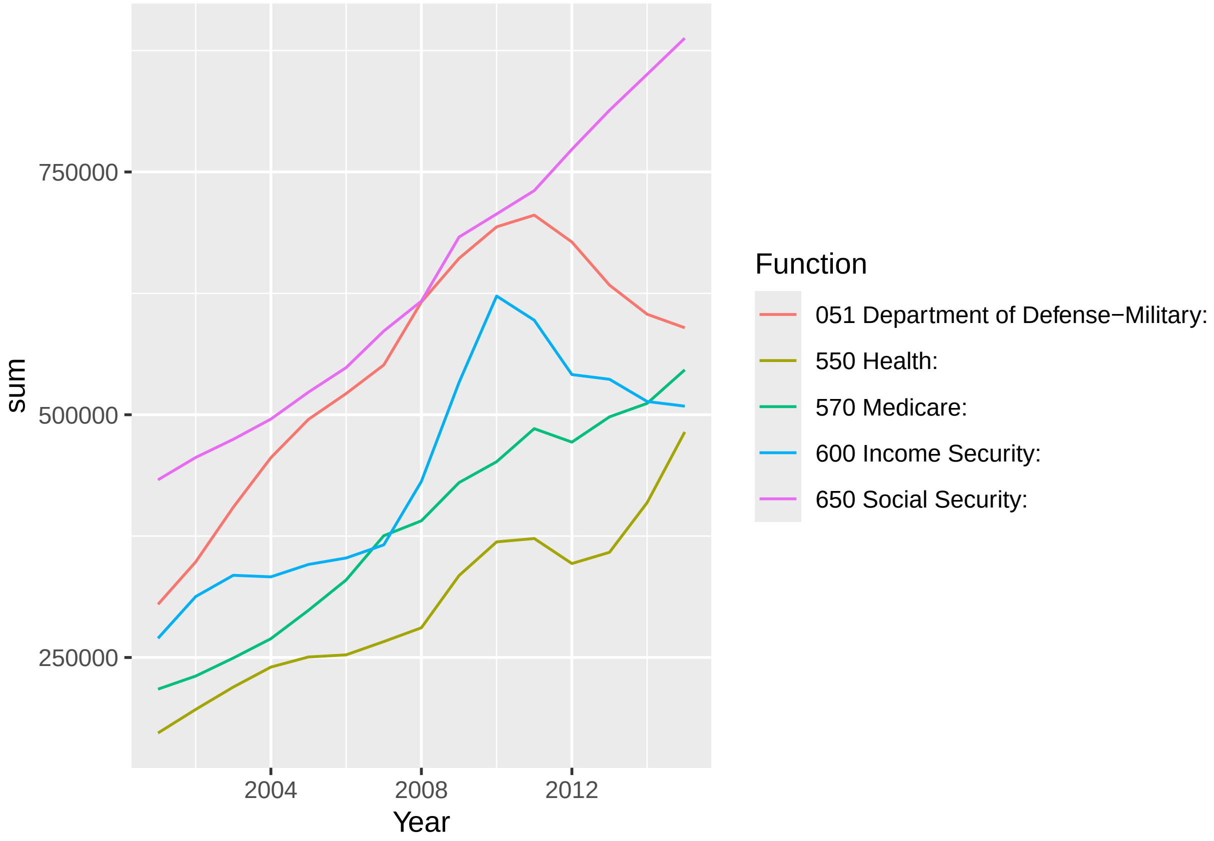
i) Make sure that Year and Amount are numeric. *Hint: it is OK to get rid of the estimate rows for 2016+*

```r
US_Budget %<>% mutate(Year = as.numeric(Year), Amount = as.numeric(Amount)) %>%
  filter(!is.na(Year), !is.na(Amount))
tail(US_Budget)
```

```
## # A tibble: 6 x 4
##   Function                               Subfunction                 Year Amount
##   <chr>                                  <chr>                      <dbl>  <dbl>
## 1 950 Undistributed Offsetting Receipts: 959 Other undistributed o~  2007 -13700
## 2 950 Undistributed Offsetting Receipts: 959 Other undistributed o~  2008  -1779
## 3 950 Undistributed Offsetting Receipts: 959 Other undistributed o~  2009 -16690
## 4 950 Undistributed Offsetting Receipts: 959 Other undistributed o~  2010   -197
## 5 950 Undistributed Offsetting Receipts: 959 Other undistributed o~  2014  -1221
## 6 950 Undistributed Offsetting Receipts: 959 Other undistributed o~  2015 -30128
```

j) 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.*

```
US_Budget %<>% summarize(sum = sum(Amount), .by = c(Function, Year))

US_Budget.plotData <- US_Budget %>%
  filter(str_detect(Function,
                    "Defense|Health|Medicare|Income Security|Social Security"),
                    between(Year, 2001, 2015))

ggplot(US_Budget.plotData, aes(x = Year, y = sum, color = Function)) +
  geom_line()
```


sum
Year
Function
051 Department of Defense−Military:
550 Health:
570 Medicare:
600 Income Security:
650 Social Security: