445-13

2023-11-03

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('StudentSurvey.csv')
```

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 <- Survey %>%
  drop_na() %>% # remove NA values
  count(Year, Sex) %>% # all gender & year combinations
  arrange(match(Year, c('FirstYear','Sophomore', 'Junior', 'Senior')), .by_group=FALSE)
Survey              # arrange years in order
```

```
##        Year Sex  n
## 1 FirstYear   F 36
## 2 FirstYear   M 43
## 3 Sophomore   F 92
## 4 Sophomore   M 91
## 5    Junior   F 16
## 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				

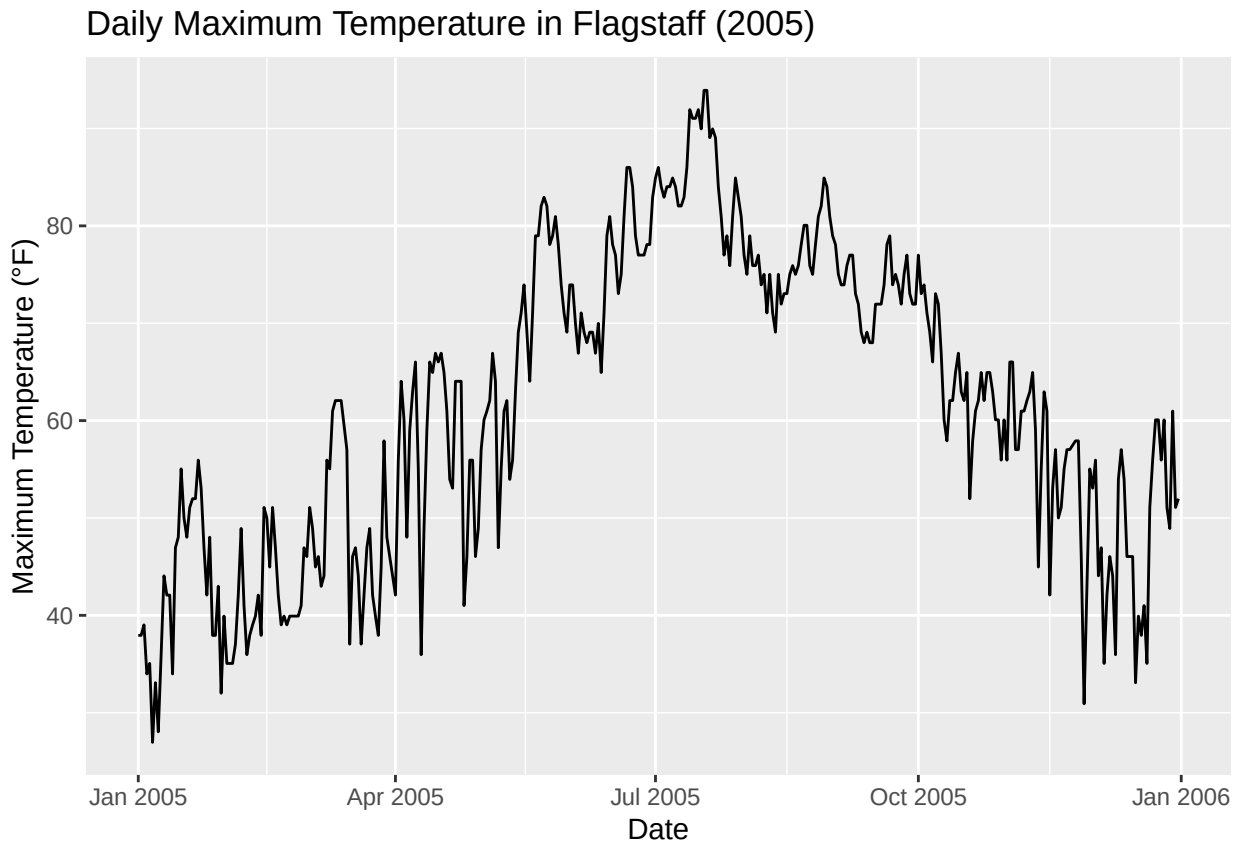
```
Survey2 <- Survey %>%
  pivot_wider( names_from=Year, values_from=n ) # make years columns
Survey2
```

```
## # A tibble: 2 x 5
##   Sex   FirstYear Sophomore Junior Senior
##   <chr>     <int>     <int>  <int>  <int>
## 1 F            36        92     16     10
## 2 M            43        91     17     26
```

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.*

```
# load temp data
TempData <- read.csv('https://raw.githubusercontent.com/dereksonderegger/444/master/data-raw/FlagMaxTemp

PrepData <- TempData %>%
  filter(Year == 2005) %>% # data in 2005
  select(-X) %>%  # remove 'X' col out of preference
  pivot_longer(3:33, names_to = 'Day',values_to = 'MaxTemp') %>% # X1:X31 cols
  mutate(Date=make_date(year=Year, month=Month, day=str_replace_all(Day, pattern='X', replacement='')))
# make date col

PrepData <- drop_na(PrepData) # remove NA values

ggplot(PrepData, aes(x=Date,y=MaxTemp)) + geom_line() + # date vs temp
    labs(title = "Daily Maximum Temperature in Flagstaff (2005)",
       x = "Date",
       y = "Maximum Temperature (°F)")
```



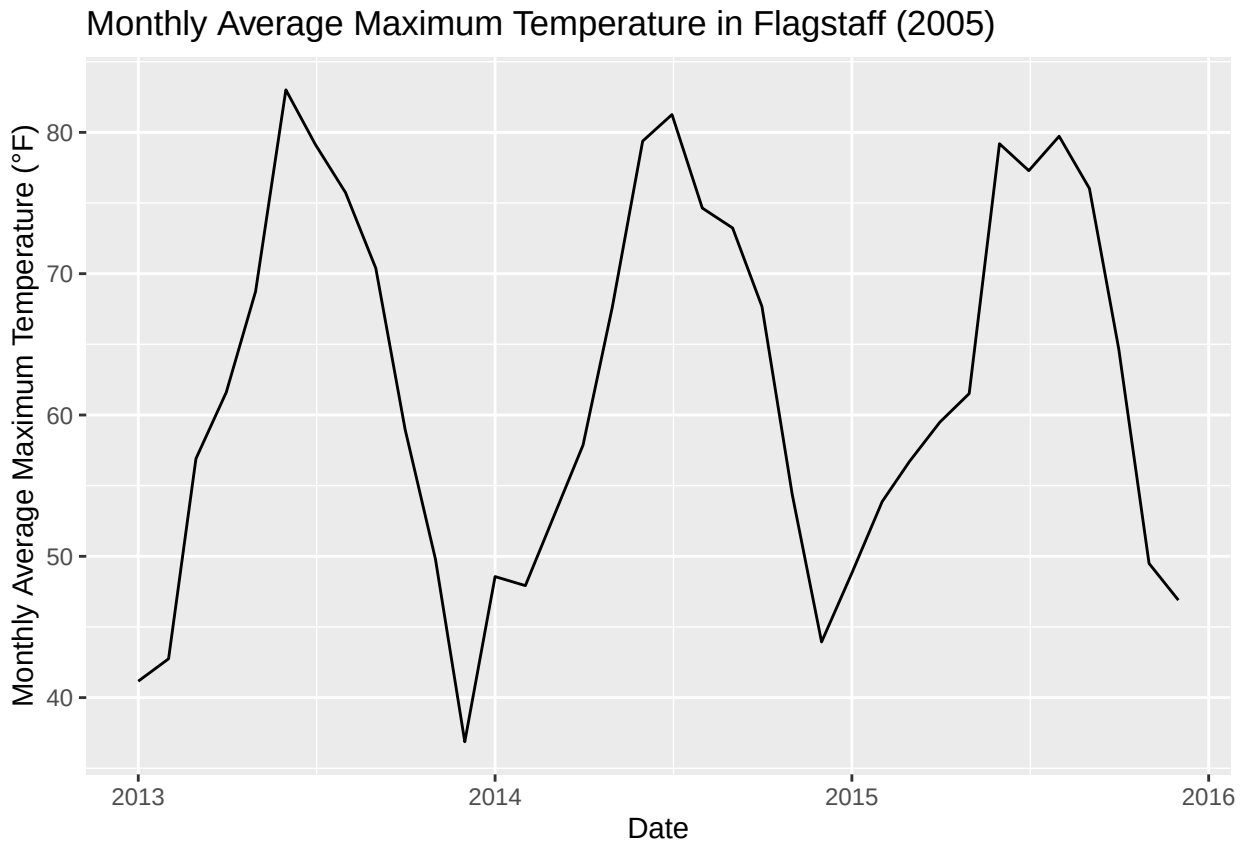
b) Create a line graph that gives the monthly average maximum temperature for 2013 - 2015. *Again the x-axis should be the date and the axis* *spans 3 years.*

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

MA.PrepData <- TempData %>%
  filter(Year >= 2013 & Year <= 2015) %>% # years 2013-2015
  select(-X) %>%
  pivot_longer(3:33, names_to = 'Day',values_to = 'MaxTemp') %>%
  mutate(Date=make_date(year=Year, month=Month, day=str_replace_all(Day, pattern='X', replacement='')))

MA.PrepData <- drop_na(MA.PrepData)

MA <- MA.PrepData %>%
  group_by(Year, Month) %>% # each month, year combination
  summarise(MA.Temp = mean(MaxTemp)) # mean of temp
```

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

```
ggplot(MA, aes(x=as.Date(paste(Year,Month,"01",sep="-")),y=MA.Temp)) +
  geom_line() + # date vs mean monthly temps
  labs(title = "Monthly Average Maximum Temperature in Flagstaff (2005)",
```

```
       x = "Date",
       y = "Monthly Average Maximum Temperature (°F)")
```



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.

```
B <- B %>% arrange(First.Name) # reorder 'First.Name'
B
```

```
## # A tibble: 3 x 2
```

```
##   First.Name Pet
##   <chr>      <chr>
## 1 Alice      Rabbit
## 2 Bob        Cat
## 3 Charlie    Dog
```

```
ABC <- cbind(A,B$Pet) # A + Pet col in B
ABC
```

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

```
ABC2 <- inner_join(A, B, by=c('Name'='First.Name')) # join A & B
ABC2
```

```
## # 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‘.

```
C <- tibble( First.Name='Alice', Pet='Guinea Pig') # new row
B2 <- add_row(B, C) # add row
B2
```

```
## # A tibble: 4 x 2
##   First.Name Pet
##   <chr>      <chr>
## 1 Alice      Rabbit
## 2 Bob        Cat
## 3 Charlie    Dog
## 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.
The inner join is easier to program and by using cbind the program will cause an errror because of the differing number of rows in each data set.
Error in data.frame(..., check.names = FALSE) :
arguments imply differing number of rows: 3, 4

```
#fourthree <- cbind(A, B2$Pet)
#fourthree
```

```
# combine data frames
four3 <- inner_join(A, B2, by = c("Name" = "First.Name"))
four3
```

```
## # 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
```

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')
Customers
```

```
## # A tibble: 4 x 5
##   PersonID Name               Street             City      State
##      <dbl> <chr>              <chr>              <chr>     <chr>
## 1        1 Derek Sonderegger  231 River Run      Flagstaff AZ
## 2        2 Aubrey Sonderegger 231 River Run      Flagstaff AZ
## 3        3 Robert Buscaglia   754 Forest Heights Flagstaff AZ
## 4        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' )
Retailers
```

```
## # A tibble: 3 x 5
##   RetailID Name           Street             City      State
##      <dbl> <chr>          <chr>              <chr>     <chr>
## 1        1 Kickstand Kafe 719 N Humphreys St Flagstaff AZ
## 2        2 MartAnnes      112 E Route 66     Flagstaff AZ
## 3        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) )
Cards
```

```
## # A tibble: 4 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 2022-09-20 00:00:00
## 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
```

```
Transactions <- Transactions %>%
  mutate( DateTime = lubridate::ymd_hms(DateTime))
Transactions
```

```
## # A tibble: 7 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
```

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.

```
CustName <- 'Derek Sonderegger' # given customer name

Customer <- Customers %>% # filter 'Customers' for 'Derek Sonderegger'
  filter( Name == CustName )

Statement <- Customer %>% # join tables by shared cols
  left_join(Cards,by ='PersonID') %>% # Customer$PersonID<->Cards$PersonID
  left_join(Transactions,by='CardID') %>% # Cards$CardID<->Transactions$CardID
  left_join(Retailers,by='RetailID')#Transactions$RetailID<->Retailers$RetailID
Statement
```

```
## # 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>
```

```
Statement2 <- Statement %>%
  select(Name.x, Name.y, DateTime, Amount) # show name, store, time, amount
Statement2
```

```
## # A tibble: 3 x 4
##   Name.x            Name.y         DateTime            Amount
##   <chr>             <chr>          <dttm>               <dbl>
## 1 Derek Sonderegger Kickstand Kafe 2019-10-01 08:31:23   5.68
## 2 Derek Sonderegger Kickstand Kafe 2019-10-02 08:26:31   5.68
## 3 Derek Sonderegger Kickstand Kafe 2019-10-02 08:30:09   9.23
```

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.*

```
CustName2 <- 'Aubrey Sonderegger' # given name

Customer2 <- Customers %>% # filter 'Customers' for 'Aubrey Sonderegger'
  filter( Name == CustName2 )

Info <- Customer2 %>% # use PersonID to get CardID
  left_join(Cards,by ='PersonID') # Customer$PersonID<->Cards$PersonID

cardID <- Info$CardID # save vars
personID <- Info$PersonID
CloseTime <- ymd_hms("2019-10-15 4:28:21 PM") # close card time

Cards <- Cards %>%
```

```
  filter(CardID == cardID) %>% # if card ID is Audrey's, close it
  mutate(Exp_DateTime = CloseTime) # new expiration is close time

NewCardRow <- tibble( # create new card info
  CardID = '3875649120094678',
  PersonID = personID,
  Issue_DateTime = CloseTime, # card issued as old one is closed
  Exp_DateTime = ymd_hms('2022-9-20 0:00:00')
  )

Cards <- bind_rows(Cards, NewCardRow) # issue new card
Cards
```

```
## # A tibble: 2 x 4
##   CardID           PersonID Issue_DateTime      Exp_DateTime
##   <chr>               <dbl> <dttm>              <dttm>
## 1 5628927579821287        2 2019-09-20 00:00:00 2019-10-15 16:28:21
## 2 3875649120094678        2 2019-10-15 16:28:21 2022-09-20 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.

```
card <- '3875649120094678' # Audrey's card number
retailid <- 1 # kickstand
datetime <- ymd_hms('2019-10-16 14:30:21')
amount <- 4.98

NewRow <- tibble(
  CardID = card,
  RetailID = retailid,
  DateTime = datetime,
  Amount = amount
)

Transactions <- bind_rows(Transactions, NewRow) # new transaction entry
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 3875649120094678        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' # Audrey's old card number
retailid <- 3 # REI
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){
    NewRow <- tibble(
    CardID = card,
    RetailID = retailid,
    DateTime = datetime,
    Amount = amount
  )

  Transactions <- bind_rows(Transactions, NewRow)
}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.

```
statement <- Customer2 %>% # join tables by shared cols
  left_join(Cards,by ='PersonID') %>% # Customer$PersonID<->Cards$PersonID
  left_join(Transactions,by='CardID') %>% # Cards$CardID<->Transactions$CardID
  left_join(Retailers,by='RetailID')#Transactions$RetailID<->Retailers$RetailID
statement
```

```
## # 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      38756~ 2019-10-15 16:28:21
## # 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>
```

```
statement2 <- statement %>%
  select(Name.x, CardID, Name.y, DateTime, Amount)
statement2 # show name, card number, retailer, amount
```

```
## # A tibble: 2 x 5
##   Name.x             CardID           Name.y         DateTime            Amount
##   <chr>              <chr>            <chr>          <dttm>               <dbl>
## 1 Aubrey Sonderegger 5628927579821287 REI            2019-10-05 18:58:57  68.5
## 2 Aubrey Sonderegger 3875649120094678 Kickstand Kafe 2019-10-16 14:30:21   4.98
```