assignment7

Noah Plant

2024-11-04

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(forcats)
library(ggplot2)
library(lubridate)
library(stringr)
```

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.2<-Survey%>%select("Year","Gender")


head(Survey.2)
```

```
##        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.3 <- mutate(Survey.2,Year = factor(Year))  # Converts the year column to a column of factors not

Survey.3 <- mutate(Survey.3,Year = fct_relevel(Year, 'FirstYear', 'Sophomore','Junior','Senior' ))

head(Survey.3)
```

```
##        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.4 <- mutate(Survey.3,Gender=factor(Gender))
Survey.4<- mutate(Survey.4, Gender=fct_recode(Gender, 'Male' = 'M'),
                  Gender=fct_recode(Gender, 'Female' = 'F'))

head(Survey.4)
```

```
##        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.4<-drop_na(Survey.4)
numResponses<-summarise(Survey.4,count(Survey.4,Year,Gender))
```

```
## Warning: Returning more (or less) than 1 row per `summarise()` group was deprecated in
## dplyr 1.1.0.
## i Please use `reframe()` instead.
## i When switching from `summarise()` to `reframe()`, remember that `reframe()`
##   always returns an ungrouped data frame and adjust accordingly.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

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


numResponses
```

```
##        Year Gender  n
## 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.5<-numResponses %>% pivot_wider(
                                  names_from=Year ,
                                  values_from = n)


Survey.5
```

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

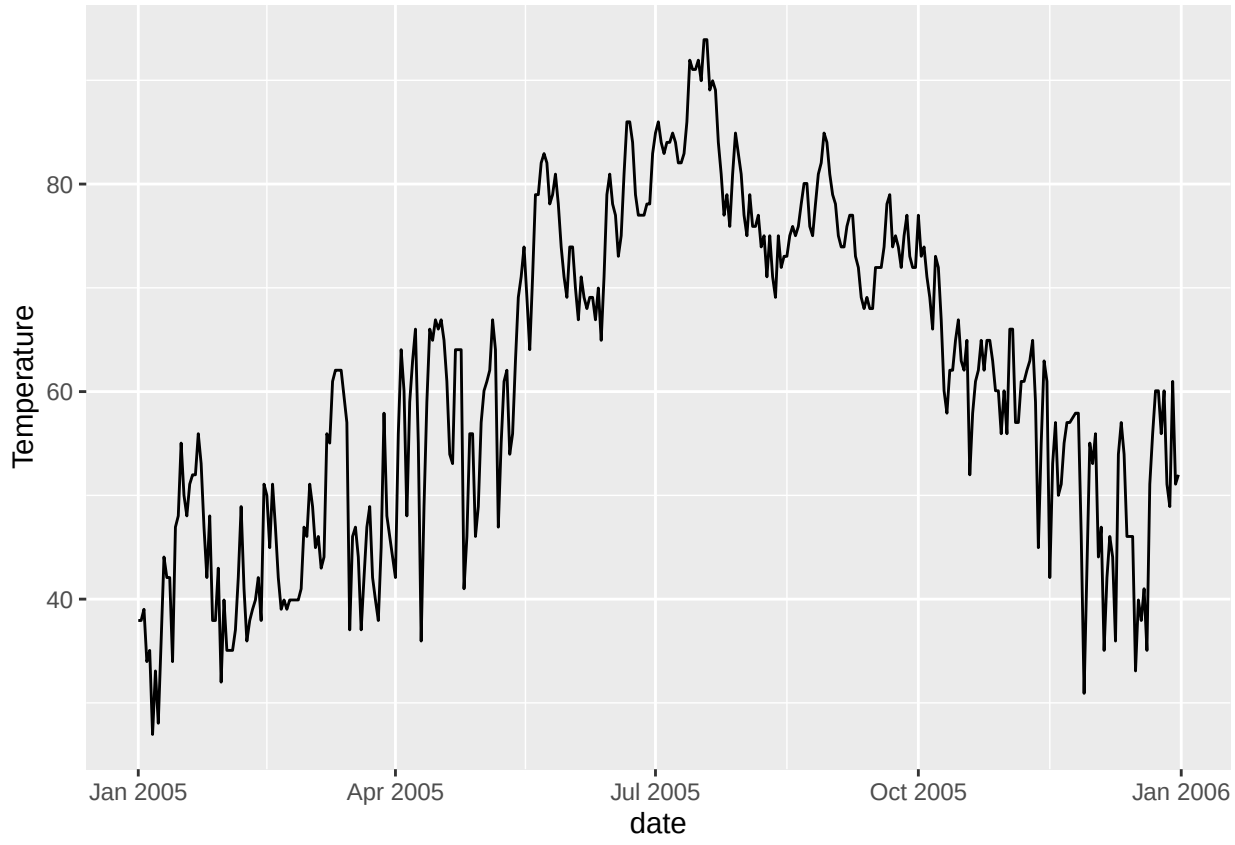
Exercise 2

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.*

```
temp <- read.csv('https://raw.githubusercontent.com/BuscagliaR/STA_444_v2/master/data-raw/FlagMaxTemp.cs


temp<-select(temp,-X)
temp.2<-temp%>%filter(Year==2005)

temp.3<-temp.2%>% pivot_longer(X1:X31,names_to="day",values_to="Temperature")
```

```r
temp.3<-temp.3%>% mutate(day=str_replace(day,pattern='X',replacement='')) # Format strings so we can ma

temp.4<-temp.3%>% mutate(date=make_date(year=Year,month=Month,day=day)) # Create date objects instead o


temp.4<-temp.4%>%select(4,5)
temp.4<-temp.4%>%drop_na()


plot<-ggplot(data=temp.4,aes(x=date,y=Temperature))+geom_line()
plot
```




```r
#head(temp.2)
```

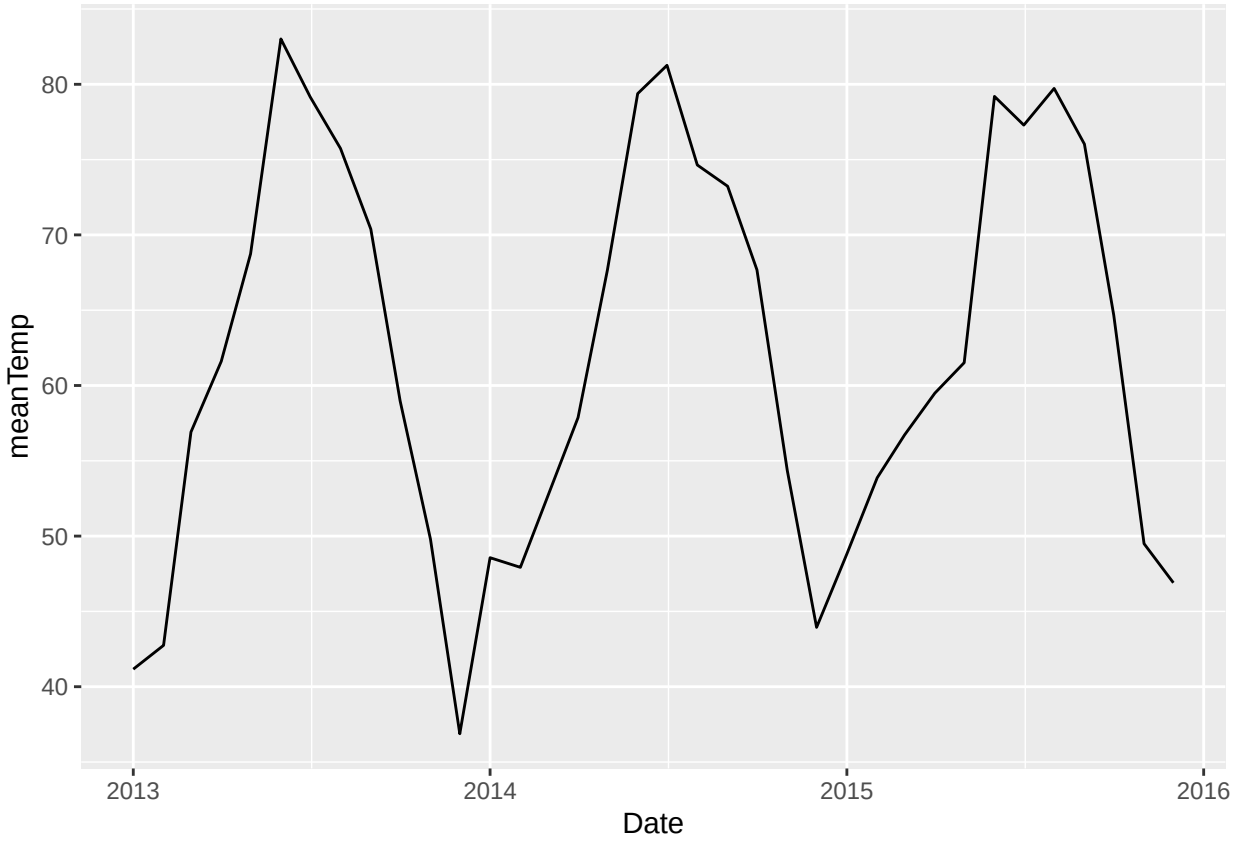
b) Create a line graph that gives the monthly average maximum temperature for 2013 - 2015. *Again the x-axis should be the date and span 3 years.*

```r
tempB.1<-temp%>%filter(Year>=2013 & Year <=2015)
tempB.2<-tempB.1%>% pivot_longer(X1:X31,names_to="Day",values_to="Temperature")

tempB.2<-tempB.2%>%drop_na()


tempB.3<-tempB.2%>%
```

```
  group_by(Year,Month)%>%
  summarise(meanTemp=mean(Temperature))
```

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

```
tempB.4<-tempB.3%>%mutate(Date=make_date(year=Year,month=Month))

p2<- ggplot(data=tempB.4,aes(x=Date,y=meanTemp))+geom_line()

p2
```



Exercise 3

For this problem we will consider two simple data sets.

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

```
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<-rename(A,First.Name=Name)  # In order to join the column names must match.

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

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

```
C
```

```
## # A tibble: 3 x 3
##   First.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.

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.*

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

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

```
D
```

```
## # A tibble: 4 x 3
##   First.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
```

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`.

```
flights<-nycflights13::flights
airports<-nycflights13::airports
airlines<-nycflights13::airlines


flights.2<-flights%>%filter(month==2 & day==14)
#flights.2

flight.airline<-inner_join(flights.2,airlines)
```

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

```
flight.airline.2<-flight.airline%>%filter(name=='Virgin America')

allInfo<-inner_join(flight.airline.2,airports,by=c("dest"="faa"))

allInfo.2<-allInfo%>%select(carrier,name.y,dep_time,air_time)

allInfo.2
```

```
## # A tibble: 10 x 4
##    carrier name.y              dep_time air_time
##    <chr>   <chr>                  <int>    <dbl>
##  1 VX      Los Angeles Intl         706      347
##  2 VX      San Francisco Intl       732      344
##  3 VX      Los Angeles Intl         909      341
##  4 VX      Mc Carran Intl           934      307
##  5 VX      San Francisco Intl      1029      351
##  6 VX      Los Angeles Intl        1317      349
##  7 VX      Los Angeles Intl        1706      335
##  8 VX      San Francisco Intl      1746      358
##  9 VX      San Francisco Intl      1852      355
## 10 VX      Los Angeles Intl        2017      337
```

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))

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
```

```
## # 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
```

```
## # 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
```

```
## # 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.*

```
derek<-Customers%>%
  filter(Name=='Derek Sonderegger')%>%
  inner_join(Cards)%>%
  inner_join(Transactions)
```

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

```
derek.2<-inner_join(derek,Retailers,by=c("RetailID"="RetailID"))


derek.2
```

```
## # 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 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.*

```
exp.date=ymd_hms('2019-10-15 16:28:21')

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

personID<-aubrey$PersonID

aubreyCard<-Cards%>%filter(PersonID==personID)

cardID<-aubreyCard$CardID

close.date<-aubreyCard$Exp_DateTime

newCard='10'

Cards<-Cards%>%mutate(Exp_DateTime=if_else(PersonID==personID,exp.date,Exp_DateTime))

Cards<-Cards%>%add_row(CardID=newCard,PersonID=personID,Issue_DateTime=exp.date,Exp_DateTime=close.date

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

Valid_Cards <- Cards %>%
  filter(CardID == card, Issue_DateTime <= datetime, datetime <= Exp_DateTime)


Valid_Cards
```

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

```
# If the transaction is valid, insert the transaction into the table
if( nrow(Valid_Cards) == 1){
  # Some code to insert the transaction
  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 10                      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 <- ymd_hms('2019-10-16 14:30:21')
amount <- 4.98

Valid_Cards <- Cards %>%
```

```
  filter(CardID == card, Issue_DateTime <= datetime, datetime <= Exp_DateTime)

Valid_Cards
```

```
## # A tibble: 0 x 4
## # i 4 variables: CardID <chr>, PersonID <dbl>, Issue_DateTime <dttm>,
## #   Exp_DateTime <dttm>
```

```
# If the transaction is valid, insert the transaction into the table
if( nrow(Valid_Cards) == 1){
  # Some code to insert the transaction
  Transactions<-Transactions%>%add_row(CardID=card,RetailID=retailid,DateTime=datetime,Amount=amount)
}else{
  print('Card Denied')
}
```

```
## [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 10                      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.

```
aubreyE<-Customers%>%
  filter(Name=='Aubrey Sonderegger')%>%
  inner_join(Cards)%>%
  inner_join(Transactions)
```

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

```
aubreyE.2<-inner_join(aubreyE,Retailers,by=c("RetailID"="RetailID"))


aubreyE.2
```

```
## # 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      10     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>
```