SOLUTIONS
Skill Task 2: Data manipulation (dplyr)

PS 811: Statistical Computing

February 25, 2020

Abstract

This document contains solutions for Skills Task 2. Although you were asked to submit a plain `.R` file, I am using `.Rmd` so I can provide more comments and interpretation of the assignment.

1 Setup

As always, I load my packages first.

```
library("here")
library("tidyverse")
```

Import the data using `read_csv()` as before. I tend to import my raw data with an object name ending in `_raw`. This way, my revised data get the prettier name.

Note also that I can use the logic of the pipe to build my file path with `here()` *and then* pipe that path into `read_csv()`. This works exactly the same, but sometimes I do this to make the code look prettier. I can then pipe into `print()` to show me the results, which is something I *always* do.

```
cafe_raw <-
  here("data", "CAFE.csv") %>%
  read_csv() %>%
  print()
## # A tibble: 100 x 5
##    Senator           State Contribution Party_Code Vote
##    <chr>             <chr>        <dbl>      <dbl> <chr>
##  1 Murkowski, Frank  AK           19700        200 Yea
##  2 Stevens, Ted      AK           13000        200 Yea
##  3 Sessions, Jeff    AL            9500        200 Yea
```

```
##  4 Shelby, Richard   AL           25000        200 Yea
##  5 Hutchinson, Tim   AR            4900        200 Yea
##  6 Lincoln, Blanche  AR            5500        100 Yea
##  7 McCain, John      AZ           29350        200 Nay
##  8 Kyl, Jon          AZ           14500        200 Yea
##  9 Boxer, Barbara    CA            1500        100 Nay
## 10 Feinstein, Dianne CA            9750        100 Nay
## # ... with 90 more rows
```

2 Investigate data

```
names(cafe_raw)
## [1] "Senator"      "State"        "Contribution" "Party_Code"   "Vote"
nrow(cafe_raw)
## [1] 100
ncol(cafe_raw)
## [1] 5
summary(cafe_raw)
##    Senator              State            Contribution      Party_Code
##  Length:100         Length:100         Min.   :     0   Min.   :100.0
##  Class :character   Class :character   1st Qu.:  3788   1st Qu.:100.0
##  Mode  :character   Mode  :character   Median :  9750   Median :150.0
##                                        Mean   : 13806   Mean   :151.3
##                                        3rd Qu.: 19775   3rd Qu.:200.0
##                                        Max.   :133250   Max.   :328.0
##      Vote
##  Length:100
##  Class :character
##  Mode  :character
##
##
##
```

3 Did the bill pass: count()

Supply a data frame and a variable name directly to `count()`. This is also one way (of many) to tabulate any variable to see what values it has.

```
count(cafe_raw, Vote)
## # A tibble: 2 x 2
```

```
##   Vote      n
##   <chr> <int>
## 1 Nay      38
## 2 Yea      62
```

Because the Yeas outnumber the Nays, it looks like the bill passed the Senate.

4 New variables: `mutate()`

Here is where I modify the dataset to its more "final form." This is where I give the result a new name that is prettier and easier to work with for the rest of my analysis.

Before we recode this variable, we can see what the existing values are, using `count()`. Let's look at the `Party_Code` variable, since we've already seen `Vote`.

```
count(cafe_raw, Party_Code)
## # A tibble: 3 x 2
##   Party_Code     n
##        <dbl> <int>
## 1        100    50
## 2        200    49
## 3        328     1
```

There are `100`s (Democrats), `200`s (Republicans), and `328`s (Independents). We need to recode all of these.

```
cafe <- cafe_raw %>%
  mutate(
    Party = case_when(Party_Code == 100 ~ "Democrat",
                      Party_Code == 200 ~ "Republican",
                      Party_Code == 328 ~ "Independent"),
    Yea_Vote = case_when(Vote == "Yea" ~ 1,
                         Vote == "Nay" ~ 0)
  )
```

The `case_when()` function is like an "if…else…" statement, if you're familiar with that. You read it by saying "If `Party_Code` is 100, the result is `"Democrat"`," and so on.[1] You can always check to see that R did what you want by using `count()` to compare the new and old variables.

[1] The `case_when()` function is similar to `ifelse()`, which you may have seen in your previous R work. We prefer `case_when()` because it does the same thing but with more flexibility to match multiple conditions and multiple outputs, whereas `ifelse()` only does one thing at a time.

```
count(cafe, Vote, Yea_Vote)
## # A tibble: 2 x 3
##   Vote  Yea_Vote     n
##   <chr>    <dbl> <int>
## 1 Nay          0    38
## 2 Yea          1    62
count(cafe, Party_Code, Party)
## # A tibble: 3 x 3
##   Party_Code Party           n
##        <dbl> <chr>       <int>
## 1        100 Democrat       50
## 2        200 Republican     49
## 3        328 Independent     1
```

There are a few important workflow tips to point out here.

1. You use an equals sign within `mutate()` to create a new variable. This is different from assigning entirely new objects in the general R workspace, where we use the assignment operator `<-`.
2. You can add however many variables you want to `mutate()`. Just separate each new variable declaration with a comma.
3. If we didn't match a category using `case_when()`, for example if we left out the part about `328`, the result would default to `NA` for those cases. It often isn't wise to let R handle this choice for you by default, which is why it is important to investigate which values are in your data before you recode (e.g. using `count()`).
4. In situations like this where you have a function *within* `mutate()`, you need to make sure that you have closed all of your parentheses.

5 Vote within Party

With this new data, find out which party was more supportive of the bill. This is where we use `group_by()` and then pass the result directly to `summarize()`.

First, I will demonstrate `summarize()` without grouping. Summarizing is different from adding new variables because we are *collapsing the data down* into a data frame of summary statistics. Each result must be length 1 (a summary!).

```
# I use the na.rm = TRUE argument
#   in order to skip NAs, should there be any.
summarize(
  cafe,
  Prop_Yea = mean(Yea_Vote, na.rm = TRUE)
)
```

```
## # A tibble: 1 x 1
##   Prop_Yea
##      <dbl>
## 1     0.62
```

The result is a data frame with a variable called `Prop_Yea`, containing the mean of the original `Yea_Vote` variable (which is the proportion of `1`s).

To calculate this concept within party groups, use `group_by()` before `summarize()`. Instead of returning a one-row data frame, it will return a data frame that contains one row for each group.

```
cafe %>%
  group_by(Party) %>%
  summarize(
    Prop_Yea = mean(Yea_Vote, na.rm = TRUE)
  )
## # A tibble: 3 x 2
##   Party       Prop_Yea
##   <chr>          <dbl>
## 1 Democrat       0.38
## 2 Independent    0
## 3 Republican     0.878
```

Looks like Republicans were more supportive of the bill.

It's possible that you didn't catch the trick that you can find the proportion of `1`s by calculating the mean. Instead you might have used `count()` or something. Here's how you might do that.

```
# note that the data are grouped by party,
# so when you calculate prop,
# sum(n) is the sum of n WITHIN PARTY
cafe %>%
  group_by(Party) %>%
  count(Vote) %>%
  mutate(prop = n / sum(n))
## # A tibble: 5 x 4
## # Groups:   Party [3]
##   Party       Vote      n  prop
##   <chr>       <chr> <int> <dbl>
## 1 Democrat    Nay      31 0.62
## 2 Democrat    Yea      19 0.38
## 3 Independent Nay       1 1
```

```
## 4 Republican  Nay       6 0.122
## 5 Republican  Yea      43 0.878
```

6 Bonus: co-voting

How do we know if two Senators co-voted? We `Vote` variable would sum to 2 (two Yeas) or 0 (two Nays) within a state. States where the senators voted differently would only sum to 1. So we can do this by grouping on state and then summing `Yea_Vote`.

```
cafe %>%
  group_by(State) %>%
  summarize(
    sum_vote = sum(Yea_Vote)
  )
## # A tibble: 50 x 2
##    State sum_vote
##    <chr>    <dbl>
##  1 AK           2
##  2 AL           2
##  3 AR           2
##  4 AZ           1
##  5 CA           0
##  6 CO           2
##  7 CT           0
##  8 DE           1
##  9 FL           0
## 10 GA           2
## # ... with 40 more rows
```

If you did this approach, great. But you should know that this method can run into problems. Imagine a scenario where a senator died in office, and then the state is represented only by one senator. If the remaining senator voted Nay on the bill, the sum of `Yea_Vote` would be 0. This can also happen if a state has two senators but one of the senators doesn't vote on the bill. If you calculate the sum without also checking that there were two votes, you might find the wrong answer.

This is an important thing to keep in mind about programming. There are often many ways to get the same answer *under specific assumptions*, but your answer can differ if the assumptions don't hold. It is often advisable to program your routine using the more rigorous approach as a way of guarding against violated assumptions in the future, should your data ever change, or should the method be applied in a different setting.

So here is how you can take a stricter approach. If the sum of `Yea_Vote` is 2, we know

that this means two Yea votes. However, we only know that a sum of 0 means two Nay votes if we also check that two senators voted. The following approach checks all of those conditions.

```
cafe %>%
  group_by(State) %>%
  summarize(
    sum_yea = sum(Yea_Vote),
    sum_valid_votes = sum(Yea_Vote == 1 | Yea_Vote == 0)
  )
## # A tibble: 50 x 3
##    State sum_yea sum_valid_votes
##    <chr>   <dbl>           <int>
##  1 AK          2               2
##  2 AL          2               2
##  3 AR          2               2
##  4 AZ          1               2
##  5 CA          0               2
##  6 CO          2               2
##  7 CT          0               2
##  8 DE          1               2
##  9 FL          0               2
## 10 GA          2               2
## # ... with 40 more rows
```

The `sum_valid_votes` line says, sum the number of cases where `Yea_Vote` is `1` OR `Yea_Vote` is `0`. If this variable doesn't equal `2` for a state, then that state did not have two senators casting relevant votes.

You could combine these methods into one variable that indicates co-voting.

```
# the sum of yea_vote is either 2 or 0 (using the | symbol for "or"),
# AND (&) the number of Yeas or Nays (|) for a state must be 2
state_covotes <- cafe %>%
  group_by(State) %>%
  summarize(
    co_vote =
      (sum(Yea_Vote) == 2 | sum(Yea_Vote) == 0) &
      (sum(Yea_Vote == 1 | Yea_Vote == 0) == 2)
  ) %>%
  print()
## # A tibble: 50 x 2
##    State co_vote
##    <chr> <lgl>
```

```
##  1 AK    TRUE
##  2 AL    TRUE
##  3 AR    TRUE
##  4 AZ    FALSE
##  5 CA    TRUE
##  6 CO    TRUE
##  7 CT    TRUE
##  8 DE    FALSE
##  9 FL    TRUE
## 10 GA    TRUE
## # ... with 40 more rows
```

The result is a logical variable, so it is either `TRUE` or `FALSE`. You can always convert a logical into a number, and logicals actually behave exactly like `1`s and `0`s.

```
as.numeric(TRUE)
## [1] 1
sum(c(TRUE, TRUE, FALSE))
## [1] 2
mean(c(TRUE, FALSE))
## [1] 0.5
```