Exercise 6: Base R vs. Tidyverse (Answer Key)

Marcy Shieh

10/15/2020

Prerequisites

1. Open you `ps811-exercises` folder.
2. Go to File > New File > R Script.
3. Save file as `exercise-5-code.R`.
4. Load R packages installed in Lecture 5.

Base R tasks

1. Download the food_coded.csv file from Kaggle.
2. Load the CSV file into your R environment.

   Open the `codebook_food.docx` file for guidance.

```
food <- read.csv(here("data", "food_coded.csv"))
```

3. Extract the first 95 rows.

```
food_95 <- food[1:95, ]
```

4. Look at the following variables using both name and column index/number.
   - GPA
   - calories_chicken
   - drink
   - fav_cuisine
   - father_profession
   - mother_profession

```
# name
food_95[ , c("GPA", "calories_chicken", "drink", "fav_cuisine", "father_profession", "mother_profession

# index number
food_95[ , c(1,4,16,26,25,45)]
```

5. Create a new variable for how healthy each person feels but convert the scale from 1 to 10 to 1 to 100.

```
food_95$healthy_feeling_100 <- food_95$healthy_feeling * 10
```

6. Filter to students who are female and have GPAs that are above 3.0.

```
subset(food_95, Gender == 1 & GPA > 3.0)
```

7. Arrange their favorite cuisine in alphabetical order.

```
food_95[order(food_95$fav_cuisine), ]
```

8. Find the mean and standard deviation for the following variables, and summarize them in a data frame.

   - chicken_calories
   - tortilla_calories
   - turkey_calories
   - waffle_calories

```
data.frame(chicken_calories.mean = mean(food_95$calories_chicken, na.rm = TRUE),
           chicken_calories.sd = sd(food_95$calories_chicken, na.rm = TRUE),
           tortilla_calories.mean = mean(food_95$tortilla_calories, na.rm = TRUE),
           tortilla_calories.sd = sd(food_95$tortilla_calories, na.rm = TRUE),
           turkey_calories.mean = mean(food_95$turkey_calories, na.rm = TRUE),
           turkey_calories.sd = sd(food_95$turkey_calories, na.rm = TRUE),
           waffle_calories.mean = mean(food_95$waffle_calories, na.rm = TRUE),
           waffle_calories.sd = sd(food_95$waffle_calories, na.rm = TRUE))
```

```
##   chicken_calories.mean chicken_calories.sd tortilla_calories.mean
## 1              586.1053             127.384               957.2105
##   tortilla_calories.sd turkey_calories.mean turkey_calories.sd
## 1             197.1644             552.5263           155.0392
##   waffle_calories.mean waffle_calories.sd
## 1             1074.895           246.5409
```

9. Summarize GPA and weight within the gender and cuisine variables.

```
# you need to turn the variables into numeric variables
food_95$GPA <- as.numeric(food_95$GPA)
```

```
## Warning: NAs introduced by coercion
```

```
food_95$weight <- as.numeric(food_95$weight)
```

```
## Warning: NAs introduced by coercion
```

```
food_95$Gender <- as.factor(food_95$Gender)
food_95$cuisine <- as.factor(food_95$cuisine)

# you want to summarize the GPA and weight variables (so you put it on the outside of the formula)
# grouped by gender and cuisine
aggregate(formula = cbind(Gender, cuisine) ~ GPA + weight,
          data = food_95,
          FUN = function(x){
            c(mean = mean(x), sd = sd(x))
          })
```

```
##      GPA weight Gender.mean Gender.sd cuisine.mean cuisine.sd
## 1  3.500    100   1.0000000        NA    1.0000000         NA
## 2  3.900    105   1.0000000        NA    1.0000000         NA
## 3  3.904    110   1.0000000        NA    1.0000000         NA
## 4  3.200    112   1.0000000        NA    2.0000000         NA
## 5  3.605    113   1.0000000        NA    1.0000000         NA
## 6  4.000    115   1.0000000        NA    1.0000000         NA
## 7  3.500    116   1.0000000        NA    1.0000000         NA
```

```
## 8  2.900    120   1.0000000        NA    1.0000000         NA
## 9  3.000    120   1.0000000        NA    1.0000000         NA
## 10 3.600    123   1.0000000        NA    1.0000000         NA
## 11 2.800    125   1.0000000        NA    1.0000000         NA
## 12 3.000    125   1.0000000        NA    1.0000000         NA
## 13 3.300    125   1.0000000        NA    1.0000000         NA
## 14 3.700    127   1.0000000        NA    1.0000000         NA
## 15 2.800    128   1.0000000        NA    2.0000000         NA
## 16 3.200    129   1.0000000        NA    6.0000000         NA
## 17 3.400    130   1.0000000        NA    1.0000000         NA
## 18 3.670    130   1.0000000        NA    6.0000000         NA
## 19 3.000    135   1.0000000        NA    1.0000000         NA
## 20 3.400    135   1.0000000        NA    1.0000000         NA
## 21 3.600    135   1.0000000 0.0000000    1.0000000  0.0000000
## 22 3.800    135   1.0000000        NA    1.0000000         NA
## 23 3.300    137   1.0000000        NA    1.0000000         NA
## 24 2.600    140   1.0000000        NA    1.0000000         NA
## 25 2.800    140   1.0000000        NA    1.0000000         NA
## 26 3.100    140   2.0000000        NA    1.0000000         NA
## 27 3.400    140   2.0000000        NA    1.0000000         NA
## 28 3.500    140   1.0000000        NA    1.0000000         NA
## 29 3.730    140   1.0000000        NA    1.0000000         NA
## 30 4.000    140   1.0000000        NA    1.0000000         NA
## 31 2.800    145   2.0000000        NA    1.0000000         NA
## 32 3.200    145   1.0000000        NA    1.0000000         NA
## 33 3.900    145   2.0000000        NA    1.0000000         NA
## 34 4.000    145   1.0000000        NA    6.0000000         NA
## 35 3.000    150   1.0000000        NA    1.0000000         NA
## 36 3.650    150   1.0000000        NA    1.0000000         NA
## 37 3.700    150   1.0000000 0.0000000    3.5000000  3.5355339
## 38 3.830    150   2.0000000        NA    2.0000000         NA
## 39 3.890    150   1.0000000        NA    1.0000000         NA
## 40 3.200    155   2.0000000        NA    1.0000000         NA
## 41 3.500    155   1.5000000 0.7071068    1.5000000  0.7071068
## 42 3.654    155   1.0000000        NA    1.0000000         NA
## 43 3.700    155   1.0000000        NA    1.0000000         NA
## 44 3.700    160   2.0000000        NA    1.0000000         NA
## 45 2.200    165   2.0000000        NA    1.0000000         NA
## 46 3.200    165   2.0000000        NA    1.0000000         NA
## 47 3.500    165   2.0000000        NA    2.0000000         NA
## 48 3.500    167   2.0000000        NA    6.0000000         NA
## 49 3.830    167   1.0000000        NA    1.0000000         NA
## 50 3.800    168   2.0000000        NA    1.0000000         NA
## 51 3.600    169   2.0000000        NA    1.0000000         NA
## 52 3.200    170   1.0000000        NA    1.0000000         NA
## 53 3.400    170   1.0000000        NA    5.0000000         NA
## 54 3.600    170   2.0000000        NA    1.0000000         NA
## 55 3.700    170   1.0000000        NA    1.0000000         NA
## 56 3.000    175   2.0000000 0.0000000    1.0000000  0.0000000
## 57 3.300    175   2.0000000        NA    1.0000000         NA
## 58 3.500    175   2.0000000        NA    1.0000000         NA
## 59 3.750    175   2.0000000        NA    6.0000000         NA
## 60 3.800    175   2.0000000        NA    1.0000000         NA
## 61 2.600    180   1.0000000        NA    2.0000000         NA
```

```
## 62 3.200    180   2.0000000        NA    2.0000000         NA
## 63 3.300    180   1.0000000        NA    1.0000000         NA
## 64 3.800    180   2.0000000        NA    1.0000000         NA
## 65 3.100    185   2.0000000        NA    1.0000000         NA
## 66 3.300    185   2.0000000        NA    1.0000000         NA
## 67 3.700    185   2.0000000 0.0000000    1.5000000  0.7071068
## 68 2.400    187   2.0000000        NA    6.0000000         NA
## 69 2.250    190   1.0000000        NA    6.0000000         NA
## 70 3.000    190   1.0000000        NA    1.0000000         NA
## 71 3.300    190   2.0000000        NA    1.0000000         NA
## 72 3.500    190   1.0000000 0.0000000    1.5000000  0.7071068
## 73 3.350    192   1.0000000        NA    6.0000000         NA
## 74 3.870    195   2.0000000        NA    3.0000000         NA
## 75 3.100    200   2.0000000        NA    1.0000000         NA
## 76 3.300    200   2.0000000        NA    1.0000000         NA
## 77 4.000    205   2.0000000        NA    1.0000000         NA
## 78 3.900    210   2.0000000        NA    1.0000000         NA
## 79 3.680    260   2.0000000        NA    6.0000000         NA
## 80 3.400    264   2.0000000        NA    1.0000000         NA
## 81 3.292    265   2.0000000        NA    1.0000000         NA
```

```
# in the table, the NA variables in the standard deviation columns mean that
# the summary has only aggregated one value, ergo...no standard deviation
```

Tidyverse tasks

1. Download the facebook-fact-check.csv file from Kaggle.
2. Load the CSV file into your R environment.

```
facebook <- read.csv(here("data", "facebook-fact-check.csv"))
```

3. Extract the last 500 rows.

   Hint: Check out the top_n() page to figure out how to extract the lasst 500 rows instead of the firsst 500 rows.

```
facebook_500tidy <- facebook %>%
  top_n(-500)
```

```
## Selecting by comment_count
```

4. Look at the even-numbered column indices only. Identify them by name.

```
select(facebook_500tidy, 2, 4, 6, 8, 10, 12)
# post_id, Page, Date.Published, Rating, share_count, comment_count

# with names:
select(facebook_500tidy, post_id, Page, Date.Published, Rating, share_count, comment_count)
```

5. Using `mutate`, create a new variable called `post_type_coded` that renames each post type to the following:
   - link = 1
   - photo = 2
   - text = 3
   - video = 4

Hint: You want to make sure that these text categories are *equal* to these numeric values.

```
# this introduces the case_when command, which I did not cover (sorry!)...
# but could have been uncovered via googling...

# there is probably an easier way but I think this is easier to visualize
# essentially, you are saying setting it up as:
# if the value in Post.Type is "link"
# you want it as post_type_coded == 1; if not, you want it to be post_type_coded == 0
# and, if the value in Post.Type if "photo"
# you want it you want it as post_type_coded == 2; if not, you want it to be whatever value that is alr
# and you keep building :)

facebook_500tidy <-
  mutate(facebook_500tidy,
         post_type_coded = ifelse(
           Post.Type == "link", 1, 0),
         post_type_coded = ifelse(
           Post.Type == "photo", 2, post_type_coded),
         post_type_coded = ifelse(
           Post.Type == "text", 3, post_type_coded),
         post_type_coded = ifelse(
           Post.Type == "video", 4, post_type_coded))

# a more elegant way:
facebook_500tidy <-
  mutate(facebook_500tidy,
         post_type_coded =
           ifelse(Post.Type == "link", 1,
           ifelse(Post.Type == "photo", 2,
           ifelse(Post.Type == "text", 3,
                  4))))
```

6. Arrange page names in reverse order.

```
arrange(facebook_500tidy, desc(Page))
```

7. Find the mean and standard deviation for the following variables, and summarize them.
    - share_count
    - reaction_count
    - comment_count

```
summarise(facebook_500tidy,
          share_count.mean = mean(share_count, na.rm = TRUE),
          share_count.sd = sd(share_count, na.rm = TRUE),
          reaction_count.mean = mean(reaction_count, na.rm = TRUE),
          reaction_count.sd = sd(reaction_count, na.rm = TRUE),
          comment_count.mean = mean(comment_count, na.rm = TRUE),
          comment_count.sd = sd(comment_count, na.rm = TRUE))
```

```
##   share_count.mean share_count.sd reaction_count.mean reaction_count.sd
## 1         65.47234       143.6311            173.5813          275.0488
##   comment_count.mean comment_count.sd
## 1           14.70238         8.881969
```

8. Summarize the mean and standard deviations in Question 7 with the "mainstream" values in the

category variable.

```
# this shows all the values

facebook_500tidy %>%
  group_by(Category) %>%
  summarise(share_count.mean = mean(share_count, na.rm = TRUE),
            share_count.sd = sd(share_count, na.rm = TRUE),
            reaction_count.mean = mean(reaction_count, na.rm = TRUE),
            reaction_count.sd = sd(reaction_count, na.rm = TRUE),
            comment_count.mean = mean(comment_count, na.rm = TRUE),
            comment_count.sd = sd(comment_count, na.rm = TRUE)) %>%
  ungroup()
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 3 x 7
##   Category share_count.mean share_count.sd reaction_count.~ reaction_count.~
##   <chr>               <dbl>          <dbl>            <dbl>            <dbl>
## 1 left                 94             65.4            815.             774.
## 2 mainstr~             12.3           21.8             66.8             72.9
## 3 right               110.           183.             255.             321.
## # ... with 2 more variables: comment_count.mean <dbl>, comment_count.sd <dbl>
```

```
# but you may want to look at ONLY mainstream values in the category variable...
# if you had more than 3 levels in the category variable, this might be helpful in identifying the one

facebook_500tidy %>%
  group_by(Category == "mainstream") %>%
  summarise(share_count.mean = mean(share_count, na.rm = TRUE),
            share_count.sd = sd(share_count, na.rm = TRUE),
            reaction_count.mean = mean(reaction_count, na.rm = TRUE),
            reaction_count.sd = sd(reaction_count, na.rm = TRUE),
            comment_count.mean = mean(comment_count, na.rm = TRUE),
            comment_count.sd = sd(comment_count, na.rm = TRUE)) %>%
  ungroup()
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 7
##   `Category == "m~ share_count.mean share_count.sd reaction_count.~
##   <lgl>                      <dbl>          <dbl>            <dbl>
## 1 FALSE                      110.           182.             268.
## 2 TRUE                        12.3           21.8             66.8
## # ... with 3 more variables: reaction_count.sd <dbl>, comment_count.mean <dbl>,
## #   comment_count.sd <dbl>
```

Submit

Email me (mshieh2@wisc.edu) the link to your `ps811-exercises` repository when you are done.