Agricultural Statistics: Part 1

In the last module, we began to navigate data. We used `read.csv` to read data into `R` and commands like `head` and `summary` to get information about that data. We used the `subset` function to filter data to a specific county of interest. We retrieved specific pieces of data with the command format `data_frame[row_number, column_number]` and `data_frame[,c("column_name1", "column_name2")]`, saved that information in objects, and used those objects in calculations. We used `c()` to make groups (technically known as vectors) of column names and row/column numbers. In this module, we are going to learn more data frame methods and focus on calculating statistics from the Census of Agriculture.

Objectives:

1. Apply percent calculations used in Module 2 to entire columns to calculate statistics on producer gender and race.
2. Use `unique` to calculate the variety of commodities produced by a county.
3. Use `ifelse` to assign categories and `aggregate` to calculate summary statistics for those categories (farm size, farm ownership).

Getting Started

(1) Open RStudio and your project. **(2)** Create a new Quarto file.

(3) Download the following `.csv` files into your `data` folder from this GitHub folder: `producers_race_2017_ID.csv`, `commodities_ID_2017.csv`, and `num_farms_areafiltered_tenure_1997_2017_ID.csv`.

Producer demographics

(4) Create a new code chunk and use `read.csv` to read in the data from `producers_race_2017_ID.csv`.
(5) Use `head` and at least one other function from Module 2 to inspect the data.

(6) Create a subset of this data for your county and print the columns `Year`, `County`, `Data.Item`, and `Value`. Your output should be similar to the output below.

```
   Year    County
9  2017 BEAR LAKE
10 2017 BEAR LAKE
11 2017 BEAR LAKE
12 2017 BEAR LAKE
13 2017 BEAR LAKE
14 2017 BEAR LAKE
                                                          Data.Item Value
9                              PRODUCERS, (ALL) - NUMBER OF PRODUCERS   667
10                       PRODUCERS, (ALL), FEMALE - NUMBER OF PRODUCERS   211
11                         PRODUCERS, (ALL), MALE - NUMBER OF PRODUCERS   456
12 PRODUCERS, AMERICAN INDIAN OR ALASKA NATIVE - NUMBER OF PRODUCERS     1
13                            PRODUCERS, HISPANIC - NUMBER OF PRODUCERS    10
14                               PRODUCERS, WHITE - NUMBER OF PRODUCERS   661
```

For efficiency, we will now work with the enitre `Value` column instead of one number at a time, as we did in Module 2.

(7) Check the data type of the `Value` column using the code below:

```
class(my_county_producers$Value)
```

```
[1] "character"
```

If the data type is `"integer"` or `"numeric"` you're good to go! We can do math with those types. **(8)** If the data type is `"character"`, use `as.numeric`, as well as `library(stringr)` and `str_remove_all` if necessary, to convert the `Value` column to numbers. You will need to overwrite the whole `Value` column to convert it. Notice how the syntax has changed slightly from Module 1 steps (21)-(26) – we are working with a whole column (indicated by the `$`) rather than a single object.

```
library(stringr)

my_county_producers$Value <- str_remove_all(string = my_county_producers$Value,
                                            pattern = ",")

my_county_producers$Value <- as.numeric(__________$_____)
```

(9) Print the columns `Year`, `County`, `Data.Item`, and `Value` in your subset again to make sure the conversion to numbers worked. Your subset should look the same as the one you printed earlier, but all commas in the `Value` column should be removed.

(10) Save the value (a single number) for total number of producers in your county into a new object called `total_producers`. (Hint: Find the `subset` of your county subset in which `Data.Item` is equal to the label for the total number of producers. Then, save the `Value` column into a new object.) Print your object containing the total number of producers in your county.

For the next step, we're going to use a different kind of filter that generates `TRUE/FALSE` values like `==` does. In the `stringr` library, we will use the function `str_detect`. **(11)** If you haven't read in the `stringr` library yet with `library(stringr)`, do so now. **(12)** Use the code below to save the number of male producers into a new object. Make sure to include the space before `MALE`. Since `FEMALE` also includes the letters `MALE`, we're using a space to show `R` that there are no letters before the word `MALE`.

```
male_producers <- subset(my_county_producers,
                         str_detect(string = Data.Item,
                                    pattern = " MALE"))

# save only the value for calculations
male_producers <- male_producers$Value

print(male_producers)
```

```
[1] 456
```

(13) Calculate the percentage of producers that are male in your county using the `total_producers` and `male_producers` objects.

(14) Use the same process in steps (12)-(13) to calculate the percentage of female producers and the percentage of producers of each race for your county.

(15) Add the producer demographics you calculated to the shared Google sheet. Some racial groups will not be listed on the Google sheet and not all groups may be present in your county. If there are no producers of a certain race for your county, add a 0 to that cell. If you have racial groups that don't have a column in the Google sheet, add those percentages together and add the total to the `Percent_producers_other` column.

Unique commodities

One measure of high quality farmland is how many different types of food it can produce. Therefore, it is useful to calculate how many types of food each county is currently producing.

(16) Create a new code chunk and use `read.csv` to read in the data from `commodities_ID_2017.csv`. This dataset contains information about the number of farms and acres that are growing each type of crop. **(17)** Inspect the data with at least two functions from Module 2.

This file has a lot more information than we need from it currently. The most important column for us right now is the `Commodity` column.

(18) Subset the commodities data to your county and print the `Commodity` column.

You will probably notice that there are some repeats. We can't just count the number of rows in the subset, we need the number of *unique* commodities for your county. **(19)** Use the `unique` function to print the commodities your county produces with no repeats and save this list in an object with a name that makes sense to you. **(20)** Use that object as the argument for the `length` function to get the number of commodities your county produces. Add this number to the Google sheet.

Farm Tenure (Ownership status)

(21) Create a new code chunk and use `read.csv` to read in the data from `num_farms_areafiltered_tenure_1997_2017_ID.csv`. This dataset has information about the number of farms in different acreage categories and ownership categories, as well as the total number of farms for each county in the years 1997 and 2017. **(22)** Inspect the data with at least two functions from Module 2.

(23) Subset the farm operations data to your county *and* to the year 2017. Print the `County`, `Domain`, `Domain.Category`, and `Value` columns.

These data contain different kinds of information. The `Value` column shows the number of farms in each `Domain.Category`. We are going to filter out both farm ownership and farm size information from these data.

(24) Check the data type of the `Value` column using `class()`. If the data type is `"character"`, use `as.numeric`, as well as `str_remove_all` if necessary, to convert the `Value` column to numbers. (Hint: see steps 7-8.)

(25) Save the total number of farms in your county in an object called `total_farms` by retrieving the number in the `Value` column where `Domain` is equal to `TOTAL`. (Hint: use `subset`.)

(26) Add the total number of farms for your county to the shared Google sheet.

(27) Create a new object and save a subset of the farm operations data where `Domain` is equal to `TENURE`. Print the `Domain` and `Domain.Category` columns of this subset to check that it worked.

We are now going to use a very versatile function called `aggregate`. `aggregate` calculates a summary statistic for every group in your data. Since we only have one row per ownership group in this data, we don't need to use `aggregate` here. However, we're going to practice it here so we can use it in a more complicated use case later.

The format of the `aggregate` function that we will use is is `aggregate(___ ~ ___, data = ___, FUN = ___)`. Think of the `~` as the word "by." So, we will aggregate one column *by* another column. You will use the name of your data frame in the `data` argument and the summary statistic you want in the `FUN` (function) argument. Some common examples are `mean`, `max`, `min`, and `sum`. We will use `sum` in this report since we want to add up all the farms in each category.

(28) Use the code below to find the number of farms in each ownership category. Make sure to change the data argument to the name of the data frame object you created in step (27). Print your result.

```
my_county_tenure_agg <- aggregate(Value ~ Domain.Category,
                                  data = my_county_tenure,
                                  FUN = sum)

print(my_county_tenure_agg)
```

```
       Domain.Category Value
1 TENURE: (FULL OWNER)   256
2 TENURE: (PART OWNER)   121
3     TENURE: (TENANT)    18
```

Now, we're going to create a new column in this data frame to save the percentage of farms in each category. Creating a new column in a data frame follows the same pattern as accessing an existing column. We can print existing columns with the `$` in `data_frame$column_name`, and we can create new columns with `data_frame$new_column_name <-`. Everything after the `<-` assignment will go into the new column. **(29)** Use the code below to create a new column in the `my_county_tenure_agg` data frame that calculates and saves the percentage of farms in each category in the correct row.

```
my_county_tenure_agg$Percent_farms <- my_county_tenure_agg$Value / total_farms * 100
```

Notice how we can apply the same math equation to a whole column (`Value`) to create a new column with the answer to that equation in the correct rows.

(30) Create a new column called `Percent_farms_rounded` with the percentage of farms in each category rounded to 1 decimal place. **(31)** Print the `my_county_tenure_agg` data frame and add the rounded values for each category to the Google sheet.

Farm size

(32) Create a new object and use the data from step (24) to save a subset of the farm operations data where `Domain` is equal to `AREA OPERATED`. Print the `Domain` and `Domain.Category` columns of this subset to check that it worked.

We are interested in the number of small, mid-size, and large farms in each county. You might notice that we do not have those labels in the data right now. We need to make our own categories based on the data labels we have. These are the size ranges we will use to define different farm sizes:

- Small: 1-99.9 acres
- Mid-size: 100-999 acres
- Large: Over 1,000 acres

To assign these labels to the correct rows, we are going to use the `ifelse` function. The inputs for the `ifelse` function are `test`, `yes`, and `no`. The test is similar to the statement we use in the `subset` function. However, instead of making a new data frame, we tell `ifelse` what to do if the answer to the test is yes and what to do if the answer is no.

Our test will use `str_detect` to find the different numbers we want to label. We will detect different values of `Domain.Category` and create a new `label` column with `ifelse`.

(33) Create a new column called `label` and fill it with `NA`. `NA` in `R` means "no data".

```
my_county_sizes$label <- NA
```

(34) Use `ifelse` to test if `Domain.Category` contains ",000". Every category with this pattern is over 1,000 acres. If yes, then let `label` be "Large farms." If no, let `label` keep its current value.

```
my_county_sizes$label <- ifelse(test =
                                  str_detect(string = my_county_sizes$Domain.Category,
                                             pattern = ",000"),
                                yes = "Large farms",
                                no = my_county_sizes$label)
```

(35) Print the `Domain.Category` and `label` columns to check the result of this code.

```
print(my_county_sizes[, c("Domain.Category", "label")])
```

```
                        Domain.Category       label
17 AREA OPERATED: (1,000 TO 1,999 ACRES) Large farms
18     AREA OPERATED: (1.0 TO 9.9 ACRES)        <NA>
19   AREA OPERATED: (10.0 TO 49.9 ACRES)        <NA>
20     AREA OPERATED: (100 TO 139 ACRES)        <NA>
21     AREA OPERATED: (140 TO 179 ACRES)        <NA>
22     AREA OPERATED: (180 TO 219 ACRES)        <NA>
23  AREA OPERATED: (2,000 OR MORE ACRES) Large farms
24     AREA OPERATED: (220 TO 259 ACRES)        <NA>
25     AREA OPERATED: (260 TO 499 ACRES)        <NA>
26   AREA OPERATED: (50.0 TO 69.9 ACRES)        <NA>
27     AREA OPERATED: (500 TO 999 ACRES)        <NA>
28   AREA OPERATED: (70.0 TO 99.9 ACRES)        <NA>
```

Another pattern we can see is that the small farm categories contain a decimal point. We can use `str_detect` to find all strings with a period, but we have to be careful. Since a period is a special character in `R`, we have to put `\\` before it to let `R` know that we want to find a period character instead of its special meaning.

(36) Use `ifelse` to test if `Domain.Category` contains a period. If yes, then let `label` be "Small farms." If no, let `label` keep its current value.

```
my_county_sizes$label <- ifelse(test =
                                  str_detect(string = my_county_sizes$Domain.Category,
                                             pattern = "\\."),
                                yes = "Small farms",
                                no = my_county_sizes$label)
```

(37) Print the `Domain.Category` and `label` columns to check the result of this code.

(38) Finally, use `ifelse` to test if `label` still equals `NA`. If yes, then let `label` be "Mid-sized farms." If no, let `label` keep its current value.

```
_______$label <- ifelse(test = is.na(_______$_______),
                                yes = "__________",
                                __ = _______$_____)
```

(39) Print the `Domain.Category` and `label` columns to check the result of this code.

(40) `aggregate` the `Value` column (number of farms) *by* the `label` column. Use `sum` as your function.

```
my_county_sizes_agg <- aggregate(Value ~ label,
                                 data = my_county_sizes,
                                 FUN = sum)
```

(41) Create a new column with the percentage of farms in each size category (Hint: step (29)).

(42) Create another column with the percentage rounded to one decimal place. Print the data frame and add the rounded percentages to the shared Google sheet.

Finishing up

(43) At the end of your report, copy this list and paste it outside of a code chunk to create bullet points. Fill in the data you calculated for your county. Make sure that all of this information is also in the team Google Sheet for comparison with other counties. If you calculated anything extra that not on this list, be sure to add it so you have a record for later.

```
- Percent male producers:
- Percent female producers:
- Percent American Indian or Alaska native producers:
- Percent Hispanic producers:
- Percent white producers:
- Percent Black or African American producers:
- Any other producer race demographics:
- Number of commodities produced:
- Percent full-owned farms:
- Percent part-owned farms:
```

```
- Percent tenant farms:
- Percent large farms:
- Percent mid-sized farms:
- Percent small farms:
```

(44) Go back through your report and add short explanations for what each code chunk does in your own words if you haven't done so already. (45) Render your report to a PDF and email it to [INSERT EMAIL HERE].

Statement of original and referenced work:

The entirety of this module is original work authored by Carolyn Koehn.

License