Navigating data

In the last module, we created our first report in R as a Quarto document, created objects using the `<-` assignment operator, and used number and character data to create a data frame. In this module, we are going to start navigating through data to get the information we need.

Objectives

1. Read data into R.
2. Use commands that tell us information about a data frame.
3. Create subsets of data based on positions and conditions.
4. Use subsets to calculate metrics from two data sources.
5. Report on the average age of residents, average age of agricultural producers, population size, percentage of residents in agricultural producer households, and percent of acres used for agriculture in your case study county.

Getting Started

(1) Open RStudio by double-clicking the project file you created in the last module, or, open RStudio and then open your project (see Module 1 step (7)). (2) Create a new Quarto file (refer to the "Create a Quarto Document" section in Module 1).

We'll be using a a simplified data set of population demographics for each county in Idaho from the American Community Survey. (3) Download `ID_counties_tot_pop_med_age.csv` from this GitHub data folder into the `data` folder you created in your project.

Accessing data in R

The data is downloaded to our computer, but R can't access it yet. We have to read the data into R and store it inside an object. To do this, we can use the `read.csv` function, which reads files with a .csv (comma separated values) extension. `read.csv` needs to know where the file is on your computer, but we can use a *relative path* since you are working inside a project. (4) Create a new code chunk and run this code:

```
population <- read.csv("data/ID_counties_tot_pop_med_age.csv")
```

This code tells R that within the project, there is a folder called `data`, and in that folder is a file called `ID_counties_tot_pop_med_age.csv`. We want `R` to read that file and store it as an object called `population`.

You may have noticed the `population` object appear in your Environment panel. We should check that the data looks right and get a feeling for what kind of data is in this file.

Inspecting data frames

The function I use the most when inspecting a new data frame is the `head` function. It will print the first few lines of a data frame.

(5) Use `head` to print the first 5 lines of the `population` data frame.

```
head(population, n = 5)
```

```
                  County Total_population Total_male Total_female Median_age
1       Ada County, Idaho           435117     217999       217118       36.4
2     Adams County, Idaho             3946       2021         1925       52.8
3   Bannock County, Idaho            84113      41867        42246       33.2
4 Bear Lake County, Idaho             5942       2928         3014       39.3
5   Benewah County, Idaho             9050       4568         4482       46.5
```

(6) Use the help file for `head` to find the function that will print the *last* part of the `population` data frame and print the last 5 rows of `population`.

Sometimes, our data will have a lot of columns and the `head` output can be really hard to read. In that case, it can be helpful to at least get a list of column names so we know what information might be available to us. (7) Use `colnames` to get the column names of `population`.

```
colnames(population)
```

```
[1] "County"           "Total_population" "Total_male"       "Total_female"
[5] "Median_age"
```

Finally, it can be helpful to check the type of data and some summary statistics for each column. You can do this using `summary`. (8) Use `summary` to get a summary of `population`.

```
summary(population)
```

```
   County          Total_population   Total_male      Total_female
 Length:44          Min.   :   886   Min.   :   443   Min.   :   443
 Class :character   1st Qu.:  7492   1st Qu.:  3849   1st Qu.:  3632
 Mode  :character   Median : 13004   Median :  6686   Median :  6340
                    Mean   : 37668   Mean   : 18878   Mean   : 18790
                    3rd Qu.: 30118   3rd Qu.: 15251   3rd Qu.: 14845
                    Max.   :435117   Max.   :217999   Max.   :217118
  Median_age
 Min.   :23.50
 1st Qu.:33.20
 Median :38.70
 Mean   :39.89
 3rd Qu.:46.62
 Max.   :53.40
```

Notice that `summary` can't compute summary statistics for character data like the county names. `summary` is a good check to make sure that all our number data is truly numbers and not characters. If you ever come across characters that should be numbers, you can try to get `R` to read them as numbers with `as.numeric` (more on this later!).

Retrieving data from certain locations in a data frame

Data frames have rows and columns, and we can ask for data by referring to its location in the data frame. We do this using the format `data_frame[row_index, column_index]`. For example, the code below gives the value in the first row and first column of our data:

```
population[1, 1]
```

```
[1] "Ada County, Idaho"
```

If you want a whole row, specify the row number and leave the column index blank. This code gives the first row:

```
population[1, ]
```

```
             County Total_population Total_male Total_female Median_age
1 Ada County, Idaho           435117     217999       217118       36.4
```

You can do the same process to get a whole column. You can also use a `$` and the column name to get a column. The two lines of code below produce the same output. **(9)** Try both of these in the Console. Choose one for the report, print the `County` column in a code chunk, and make sure your county is included.

```
population[, 1]

population$County
```

We can also choose multiple rows and/or columns using `c()` to create a vector of row or column numbers to select. A colon (`:`) will select a range of numbers, so `c(1:5, 7)` will be a vector containing 1, 2, 3, 4, 5, 7. Finally, we can use column names instead of numbers for more readable code (be sure to use quotation marks). **(10)** Change the code below to print a custom range of rows and columns:

```
population[c(1:4, 6), c("County", "Median_age")]
```

```
                  County Median_age
1       Ada County, Idaho       36.4
2     Adams County, Idaho       52.8
3   Bannock County, Idaho       33.2
4 Bear Lake County, Idaho       39.3
6   Bingham County, Idaho       33.3
```

Retrieving data based on a value

Since you are creating a case study for one county, you don't need information for all the others. We can create a subset of data based on a *condition*. To do this, we need to know a little bit about conditions in `R`.

In `R`, `TRUE` will select a value and `FALSE` will not. For example, in the code below, the `TRUE` and `FALSE` codes select columns 1 and 5.

```
population[c(1:4, 6), c(TRUE, FALSE, FALSE, FALSE, TRUE)]
```

```
                   County Median_age
1       Ada County, Idaho       36.4
2     Adams County, Idaho       52.8
3   Bannock County, Idaho       33.2
4 Bear Lake County, Idaho       39.3
6   Bingham County, Idaho       33.3
```

Normally, `TRUE` and `FALSE` codes are the result of a function. We use greater than ($>$, `>`), less than ($<$, `<`), greater than or equal to ($\geq$, `>=`), less than or equal to ($\leq$, `<=`), and equal to ($=$, `==`) to find the rows and columns we're interested in.

To get data for one county, we will use the `subset` function. Inside `subset`, you specify a condition that `R` should match, and it will only return rows of data where that condition is `TRUE`. **(11)** Use `subset` and `==` to get the data for your case study county from `population`.

```
my_county_population <- subset(population, County == "Bear Lake County, Idaho")

print(my_county_population)
```

```
                   County Total_population Total_male Total_female Median_age
4 Bear Lake County, Idaho             5942       2928         3014       39.3
```

(12) Add the total population and median age of residents for your county to our shared Google sheet.

Try it with Census of Agriculture data

(13) Download the file `producer_age_2017_ID.csv` from the GitHub data folder into your `data` folder.

(14) Read the data into an object in `R` and use at least two of the functions in the "Inspecting data frames" section to get some information about the data. (Note: `#` is a comment in code and is not run. I used a comment here to tell you some information about the code without breaking up my code chunk.) Remember that objects names (to the left of the `<-`) can be whatever you want, and you will use this object name in your code later.

```
# fill in the blanks
_______ <- read.csv("data/_________________")

# use at least two of these:
head(_______)
colnames(________)
summary(_______)
```

You may notice that many of these columns have the same information in every row. This is a "raw" dataset, and we will spend lots of time this semester learning how make data like this "clean."

(15) Identify the column that has the county names and check the format of the county names by printing the column.

```
print(______$_______) # data_frame$column_name_of_county_column
```

(16) Using `subset` (similar to step 11) and `data_frame$Column_name`, retrieve and print the median age of agricultural producers in your county (hint: median age is in the `Value` column of the data frame you created in step 14). Give your subset a different name than the data frame from step 14 to avoid overwriting it.

```
_______ <- subset(_______, ______ == "name of your county in correct format")

print(______$Value)
```

(17) Add the median age of producers to the shared Google sheet.

Calculations with two data frames

It would be useful to know the percentage of people in each county whose families are agricultural producers. The Census of Agriculture reports the number of people in producer households in each county. However, a raw value isn't as useful for comparison between counties as a percentage; it's better to normalize by total population so we can compare this value to other counties. To do that, we need to do math with data from two different data frames.

(18) Download the file `producers_persons_in_household_2017_ID.csv` from the Google Drive data folder into your `data` folder.

(19) Read the data into an object in `R` with `read.csv` and use at least two of the functions in the "Inspecting data frames" section (one of them may be the code below) to get some information about the data. The format will be similar to `producer_age_2017_ID.csv`.

```
head(households)[, c("County", "Data.Item", "Value")]
```

```
      County                                                Data.Item Value
1    BANNOCK PRODUCERS - PERSONS IN HOUSEHOLD, MEASURED IN PERSONS 2,617
2  BEAR LAKE PRODUCERS - PERSONS IN HOUSEHOLD, MEASURED IN PERSONS 1,510
3    BINGHAM PRODUCERS - PERSONS IN HOUSEHOLD, MEASURED IN PERSONS 4,976
4 BONNEVILLE PRODUCERS - PERSONS IN HOUSEHOLD, MEASURED IN PERSONS 3,970
5      BUTTE PRODUCERS - PERSONS IN HOUSEHOLD, MEASURED IN PERSONS   726
6    CARIBOU PRODUCERS - PERSONS IN HOUSEHOLD, MEASURED IN PERSONS 1,677
```

Notice how you can use `[]` with `head`, which is possible because `head` produces a new data frame. Since I knew the column names were similar to the column names from the other Census of Agriculture data, I chose to look at only a few interesting columns in `head` instead of all of them.

To do math with values, we can't get away with printing a subset and writing down the correct value. We need to use subsets and column names to retrieve the exact value for our calculation. We will store each value in a separate object, and then do math with the objects to find the percentage we're interested in. **Modify** the code below to calculate the percentage of people in producer households for your county.

(20) First, get the total population from the subset you created in step 11. Print the value to make sure it's correct.

```
my_county_total_population <- my_county_population[, "Total_population"]

print(my_county_total_population)
```

```
[1] 5942
```

(21) Next, find the number of people in producer households for your county (Hint: refer to step 16). This time, save the value to a new object before printing it.

```
# First, make a subset of households called my_county_households here

my_county_households_total <- my_county_households$Value

print(my_county_households_total)
```

```
[1] "1,510"
```

Notice how the the number of people in producer households is printed in quotation marks. R thinks this is a word, not a number, because there is a comma. If we try to do math now, R will give us an error because it doesn't know how to do math with a number and a word. (22) Try this in the Console (using your object names, if they differ from mine) and read the error message.

```
(my_county_households_total / my_county_total_population) * 100
```

```
Error in my_county_households_total/my_county_total_population: non-numeric argument to binar
```

In this message, R tells us that an error occurred (`Error`), where in the code it occurred (`in my_county_households_total/my_county_total_popuation`), and its best description of what went wrong (`non-numeric argument to binary operator`). The key information for us here is that there is a "non-numeric" object, even though we are trying to do math, which needs numbers.

To do our calculation, we need to remove the comma, and then tell R that this value is a number.

To remove the comma, we need to use a library called `stringr`, which makes working with strings, which are combinations of characters, much easier. (23) Install this library in the Console with:

```
install.packages('stringr')
```

(24) Then, in a code chunk in your Quarto document, tell R to access this library.

```
library(stringr)
```

(25) We will use the `str_remove_all` (string remove all) command. We need to tell `str_remove_all` what to find (all the commas) and where to find it (in the `my_county_households_total` object). Take a look at the help file if you'd like more information.

In the next command, I am *overwriting* the value of `my_county_households_total`. I don't want to keep the one with a comma in it, so I'll store the result of `str_remove_all` in the same object name. Only overwrite an object if you're sure you won't need the old value!

```
my_county_households_total <- str_remove_all(string = my_county_households_total,
                                             pattern = ",")
print(my_county_households_total)
```

```
[1] "1510"
```

The comma is gone, but the number is still in quotation marks. We need to explicitly tell `R` that this is a number. **(26)** Overwrite the variable again (or create a new variable if you prefer) and use `as.numeric` to make this value a number.

```
my_county_households_total <- as.numeric(my_county_households_total)

print(my_county_households_total)
```

```
[1] 1510
```

(27) Finally, we can calculate the percentage we need:

```
(my_county_households_total / my_county_total_population) * 100
```

```
[1] 25.41232
```

(28) Use the code above and `<-` to store the percentage in an object.

```
______ <- (_________ / __________) * 100
```

(29) Then, use the `round` function on the object from step (28) to round the percentage to one decimal point. Use `?round` in the Console if you need help.

```
round(______, ______ = 1)
```

(30) Add your rounded percentage of people in producer households to the team Google Sheet.

Try it with another Census of Agriculture dataset

(31) Download the file `cropland_pastureland_total_ares_ID_1997-2017.csv` from the GitHub data folder into your `data` folder.

(32) Read the data into an object in `R` and use at least two of the functions in the inspecting data frames section to get some information about the data. The format will be similar to other data sets in this module. This dataset has information about (1) total cropland, (2) total pastureland, and (3) total land overall in each county for 1997, 2002, 2007, 2012, and 2017.

(33) Make a subset of this data for your county and save it in an object (Hint: refer to step 16).

You will need to separate agricultural land and total land into separate objects to calculate the percentage. The key to separating these items is in the `Commodity` column. **(34)** Use the `Commodity` column from your subset created in step 33 as the input to the `unique` function to see the unique values in that column.

```
unique(your_subset_data_frame$________)
```

Besides separating out agricultural land from land area, you also want to do this calculation for only one year. Let's calculate percent agricultural land from the most recent census (2017). You can use an `&` in `subset` to match more than one condition.

(35) Create a new subset for your county with rows where `Commodity` is `AG LAND` AND `Year` is 2017. **(36)** Print the subset to check your work.

```
my_county_ag_land <- subset(your_subset_data_frame,
                            Commodity == "AG LAND" & Year == 2017)

print(my_county_ag_land[, c("County", "Commodity", "Data.Item", "Value")])
```

(37) Create a new subset for your county with rows where `Commoditiy` is total land area (use the other data value from `unique`) AND `Year` is 2017.

```
my_county_total_land <- subset(____________,
                               Commodity == "_______" & _____________)
```

(38) Print the `Value` column for each subset. If the values are characters instead of numbers, modify the code below to save the `Value` columns in new objects where the values are stored as numbers.

```
ag_land_comma_removed <- str_remove_all(string = my_county_ag_land[ ,"Value"],
                                        pattern = ",")
__________ <- as.numeric(ag_land_comma_removed)

__________ <- str_remove_all(_____ = my_county_total_land[ ,"_____"],
                             _______ = "____")
__________ <- as.numeric(__________)
```

Your total land object should contain one number, and your agricultural land object should be a vector of two values (total cropland and total pastureland). **(39)** Check that this is correct by printing the two objects you created in step (38).

(40) Calculate the percentage of agricultural land in your county by completing the code chunk below. Since you have two numbers that add up to total agricultural land, you will have to use `sum` in your equation.

```
______ <- (sum(_________) / __________) * 100
```

(41) Use `round` to round your answer to one decimal point. Use the object you created in step (40) as an argument in the `round` function.

(42) Add the percentage of agricultural land in your county 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 (this format will create bullet points in your rendered report). 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 is not on this list, be sure to add it so you have a record for later.

```
- Total population:
- Median age of residents:
- Median age of agricultural producers:
- Percentage of people in producer households:
- Percentage of land used for agriculture in 2017:
```

(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:

This document is original work authored by Carolyn Koehn, created while consulting the Data Carpentry R for Ecology lesson, which is licensed under a Creative Commons Attribution 4.0 International License, as a guide. The order of topics and some code from sections "Inspecting data frames," "Retrieving data from certain locations in a data frame," and "Retrieving data based on a value" are based on the Data Carpentry lesson. The rest of the module is original work.

License