Lab 4: Importing Data

Introduction

In this lab we learn how to import new data from Excel or other sources into RStudio. Along the way we will also get more practice with RMarkdown reports and statistical analysis commands.

-

TODO

Overall Goals

TODO

Creating an RStudio Project

We start similar to the last lab. We will start a new project based on a prepared repository.

- In RStudio, go to File > New Project > Version Control > Git.
- Paste this URL in: https://github.com/HanoverStatsLabs/Lab-Data-Import.git
- Press Tab go to the next field — it should auto-complete the directory name.
- Make sure the parent directory is the correct one.
- Click Create Project.
- Your Files pane should now show the provided files, in particular a file titled Lab4Report.Rmd. Click the file name to edit the file.
- Recall that we will mostly be editing this report file, rather than working directly in the console. You may want to make the report window larger than the console window.

Importing Data

In the previous labs we have prepared the data for you in R's built-in system. But data you find in the real world is not typically in this form. It may be an Excel file, a comma-separated-format file (CSV) or many other formats. RStudio has built-in ways to read CSV and Excel files. Some other more obscure formats may need more work and specific instructions.

For now we will learn how to import a CSV file. R can read files directly from a URL, so there is no need to download the file.

- Go to File > Import Dataset > From CSV ...
- Paste in the following URL: https://hanoverstatslabs.github.io/resources/datasets/driving.csv
- Click on Update and you should see the data in the Preview Window. We will need to now make sure it is being read correctly before doing the actual import.

This data contains information recorded over the Fall 2007-2008 when one of the faculty was living in Louisville and commuting to Hanover. Each row correponds to a one-way trip and contains details of that trip such as date, direction, departure and arrival times and car mileage at the start and end of the trip.

- Checking for misalignments:
  - Sometimes programs misidentify the column separators and may misalign some columns.
  - Scroll through the file to make sure that what shows up in each column makes sense for that column.
  - You should find at least some unusual values in the time column. We will worry about them later.
- Setting column types:
  - Especially in this data, many columns have "non-standard" types. For example the first column (Day) is supposed to be a date. The system is smart enough to do the right thing in most cases, but if after importing you find that a column has been misidentified, then you can go back to the Import Dataset dialog and choose a type for each column by clicking at the pulldown in the column heading.
  - In that same pulldown you can also tell it to skip a column.
  - For now we will let RStudio guess the types.
- Copying the appropriate code:
  - The code for importing the dataset must be part of our final report. For that reason we need the code that does the importing added to our report file.
  - However RStudio will instead run that code in the console. So, before you click the Import button, copy the text in the "Code Preview" window, excluding the last ("View") command.
  - Click Import, and you should see the dataset. If it looks OK, then go to your report and add a code chunk containing the code your copied. You should set your code chunk options (gear button to the right of the chunk) to not show warnings, not show messages, and to output "Show nothing (run code)".
  - This will ensure that when anyone compiles the report, the dataset will be loaded.
  - Take a moment to look at these two lines of code. The first loads a new package ("library") called readr. This contains useful functions for loading new data. The second line uses the read_csv method from this package to load the data and stores it in the name driving. You can access this data in the rest of your report via this name.

Before we move on, there is one more step we must take with the data. The weekDay column is a categorical variable coded as 1 through 7 (Monday through Sunday). We must tell RStudio about that, as right now it treats it as quantitative. For that you would need to run the following commands (make sure to include them in the code chunk you created earlier).

```
days <- c("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun")
weekdayOrdered <- ordered(driving$weekDay, labels=days)
driving$weekDay <- weekdayOrdered
```

Sidebar: Workflow practices

Working in the reports as opposed to the console can get confusing. And complicated commands have many steps and many places where things can go wrong. A simple workflow to follow is the following:

1. Start a code chunk, using the Insert Pulldown.
2. Enter a single line of code and execute it via the "Run > Selected Line" from the pulldown. This will run just the current line, not the whole chunk.
3. Make sure that line's output is correct before proceeding to the next line. If that line was an assignment, there is no output by default, but you can use the Environment pane to look at the assigned value or a new line of code asking for that assigned value.

For example, when trying the above commands, we may do the following:

1. Type the first command, (dayNames <- ...) and run it.
2. Add a second command saying just dayNames and run it. This will print out what dayNames is. You can then remove that second line.
3. Alternatively you can look in the Environment pane, to make sure that the dayNames value looks correct.
4. Repeat for each line.

Explanation of the three lines

The three lines we used above introduced some new features.

- The first is the use of the c command, to "concatenate" items into a list. The result is technically called a "vector" in R.
- The second line used the ordered command to turn the weekDay variable from the data into a categorical (ordinal) variable. We had to provide it the variable's name (driving$weekDay) and a list of the labels to use (labels=dayNames).
- This also introduced the "dollar sign" syntax. This has the general form <dataset>$<columnName> and it is used to access an individual column in a dataset. It is not needed when we use the data=<dataset> syntax, but not all commands can use that syntax.
- In general you only need this syntax when changing a variable in a dataset, or want to create a new variable in a dataset.

Statistical Investigations

Week Days

The first set of questions relates to the various days of the week present in the dataset. You should do both a frequency table (tally) and a barchart of the weekDay variable.

1. The instructor was teaching only four days a week. Determine the workday on which the instructor was not teaching.
2. Looking at the weekday frequencies, explain why most of them are even. How do you explain those that are not?
3. What would you expect the typical value for the workday frequencies to be, based on Hanover College calendar? How does that match the data? How do you explain the workday values that deviate from the expected typical?

Distance Traveled

In this set of question we will investigate the distribution of the miles variable, which records the miles traveled per trip as a difference of the mileages recorded in the car's odometer. You will likely need several outputs:

- a summary (favstats) of the miles variable,
- a histogram of the miles variable,
- a histogram of the miles variable, but where the dataset has been filtered to only contain the rows with miles <= 48.
- a boxplot of miles against direction (possibly filtered as above), and
- a median or summary of the miles against direction.

4. What is the typical driving distance for the instructor? Does that roughly match the distance between Louisville and Hanover?
5. The distribution should appear skewed to the right with a very sharp drop-off to the left, and some high outliers to the right. Explain why we might expect these three behaviors in this dataset.
6. When focusing on the values below 48 miles, you should notice that what appeared to be one mode is actually two modes, of roughly the same frequency, about half a mile apart from each other. Provide a plausible explanation for presence of these modes that would account for their similar frequencies.

Driving Times

The (elapsed) time variable is calculated as the difference in minutes between the instructor's departure time and arrival time. We should do the following summaries:

- A favstats summary and a histogram of the time variable.

- Two xyplots (scatterplots) of the time variable against the miles variable. In the first plot we exclude three particularly unusual/incorrect time values, and in the second plot we restrict further to the "miles < 50" region containing the typical distances. Use the following commands for this:

```
scatter1 <- xyplot(time~miles, data=driving %>% filter(time <= 90 & time > 40))
print(scatter1)
scatter2 <- xyplot(time~miles, data=driving %>% filter(time <= 90 & time > 40 & miles <
print(scatter2)
```

  Note that we actually "stored" the graphs in names, and referred to those names to display the graphs in the print commands. In general you can store graphs this way and process them further in later commands.

Here are some questions to consider:

7. Discuss the distribution of the time variable, including any unusual observations and plausible explanations for the various aspects of the distribution.

8. Describe the relationship between the time variable and the miles variable. What is the overall pattern, how does it make sense? Are there deviations? How strong is the relationship?

9. For this question, we are adding to the scatterplots from earlier straight lines at the 55 mph speed limit (60 minutes per 55 miles, which gives the value for the slope b):

```
ladd(panel.abline(a=0, b=60/55), plot=scatter1)
ladd(panel.abline(a=0, b=60/55), plot=scatter2)
```

   How are the various points in the graph positioned relative to these lines? What does this tell you about the instructor's driving habits?

10. We will now color the points in the scatterplot based on the direction (we will learn more about the special graph settings used in a future lab):

```
xyplot(time~miles, data=driving %>% filter(time <= 90 & time > 40 & miles < 50), groups=
```

   What pattern do you see in the colors, how do you explain it?