CSI 2300: Introduction to Data Science

Lecture 11: More on Data Wrangling

Today's Topics

A difficult wrangle: The Audi A4 data

- Looking for the "rows"
- Exploiting patterns
- Dealing with Murphy's Law

Missing Data

- Missing at random vs not at random
- Tools to find missing values
- Imputing missing data

Cleaning Data

- Global vs local outliers
- Handling outliers

File Organization

- Project workflow
- File naming conventions
- Looping over files – a glimpse at R ninjas

More complicated wrangling

Recall that the data in **RawCars.txt** was some text that was copied from the web page and then pasted into a text file. This is a search for used Audi A4 cars within a few hundred miles of Boulder, and the goal is to examine the relationship among asking price, mileage, and the model year. The strategy is that cars where the *predicted price* based on the other variables is over the asking price could be bargains, and those are the ones to look at more closely. Thinking ahead to the topic of outliers, note in this case that the "outlier" cars are not negative. They can be the great deals from this search.

The business at hand is to wrangle this nasty text file to a data frame with price, mileage, and model year. Once you get the technique, you should be able to pull off some other variables too that could be important, such as distance from Boulder or the kind of engine.

Finding the start and end

The key to dealing with irregular data like these is to find some tag that marks the beginning of each car's record. This will be different for every data set and needs some investigation.

Here is are the first few lines of the Audi raw data file:

```
See all 41 photos
USED
2018 Audi A4 2.0T Premium Plus
21,991 mi.
$28,999 FAIR PRICE
DELIVERY AVAILABLE
VIRTUAL APPOINTMENTS
 2018 Award Winner
Ext. Color: Black
Int. Color: Black
Transmission: Automatic
Drivetrain: AWD
 SaveCompare
Free CARFAX 1-Owner Report
Green Eyed Motors     4.6 (48 reviews)  3 mi. from 80305
Check Availability
See all 34 photos
USED
2017 Audi A4 2.0T Prestige
40,138 mi.
$29,389 FAIR PRICE
DELIVERY AVAILABLE
VIRTUAL APPOINTMENTS
Ext. Color: Black
Int. Color: Black
Transmission: Automatic
Drivetrain: AWD
 SaveCompare
CARFAX Report
Gebhardt BMW     4.7 (146 reviews)  3 mi. from 80305
Check Availability
```

What seems to repeat at regular intervals? I focused on the lines that always start with the model year, always 4 digits, then a space, then `Audi A4`. The scheme that I use assumes

that repeats of this pattern indicate where each car's information is. Note that in between these lines, the text can vary and most likely depends on how dealers tend to enter the information.

Reading the raw data finding the cars

The initial step is based on reading these lines as single character strings and then finding the lines that have the year/model line. We can use these line numbers to break up the vector of lines into separate information for each car. Take a look at the **scan** function to see what the arguments mean. This will help you understand that each element being read is still just a line of this file – a big long string. You can match them up with the file listing above.

```
AudiA4Raw <- scan("dat/rawCars.txt",what="a", sep="\n" )
# list first 5 lines read in
# note by default they are printed two items to a line -- confusing!
AudiA4Raw[1:5]
# [1] "See all 41 photos     "          "USED"
# [3] "2018 Audi A4 2.0T Premium Plus" "21,991 mi."
# [5] "$28,999 FAIR PRICE"

theTag <- substr(AudiA4Raw, 6, 12)

#Returns the row numbers that have rows matching "Audi A4"
CarsIndexStart <- which(theTag == "Audi A4")
head(CarsIndexStart)
# [1]  3 19 34 49 64 79

CarsIndexEnd <- CarsIndexStart[2:length(CarsIndexStart)] - 1

# add last line for the end of the last cars record
CarsIndexEnd <- c(CarsIndexEnd, length(AudiA4Raw))

#  156 cars found
length(CarsIndexStart)
# [1] 156
```

The tag we are using to find the first lines extracts the characters 6 through 12 on the line (need to count these from the raw data) and check if this string is (exactly) equal to `Audi A4`. Using this, we found 156 cars. Keep in mind Murphy's Law – anything that can go wrong will go wrong, so always be suspicious that your strategy is perfect. In this case, it works; however, if we had a line that had an extra space between the year and **Audi**, we would miss it.

The R object being created is vector of line numbers **CarsIndexStart**. So for example, `CarsIndexStart[10]` will give us the line number in **AudiA4Raw** where the 10th car begins. `CarsIndexEnd[10]` is the line (we hope!) where the 10th car's information ends. So if this is all working, here is how we can extract the 10th car.

```
rangeCar10 <- CarsIndexStart[10]:CarsIndexEnd[10]
work <- AudiA4Raw[rangeCar10]
print(work)
#  [1] "2020 Audi A4 40 Premium Plus"
#  [2] "5,316 mi."
#  [3] "$43,675 GOOD DEAL"
#  [4] "DELIVERY AVAILABLE"
#  [5] "VIRTUAL APPOINTMENTS"
#  [6] "Ext. Color: Blue"
#  [7] "Int. Color: Gray"
#  [8] "Transmission: Automatic"
#  [9] "Drivetrain: AWD"
# [10] " SaveCompare"
# [11] "Free CARFAX Report"
# [12] "Audi Flatirons     5 (976 reviews)  7 mi. from 80305  Authorized Audi Dealer"
# [13] "Check Availability"
# [14] "See all 57 photos     "
# [15] "USED"
```

Looping through the cars

Half of our work is done! At this point, we can use other ways of extracting the values we want, and this becomes the 10th row of our data frame. To do this, we will use a **for** loop to loop over the 156 cars. There are three steps here:

1. create the space in several vectors to hold the information,
2. grab the lines that are for a particular car, and
3. grab the information in the single car's block of text.

We will reformat these to numbers and create the data frame after the `for` loop. The code below has been shortened to just grab year and price. See the R script **readAudiLecture11.R** for the complete example.

```
# number of cars
nCars <- length(CarsIndexStart)

# make space to hold each variable for each car
```

```
check <- year <- price <- mileage <- distance <- rep(NA, nCars)

for(k in 1:nCars){

     start <- CarsIndexStart[k]
     end   <- CarsIndexEnd[k]

     # idiot numbers to follow progress
     # this is commented out for the final run
     # cat("Car:", k, start, fill=TRUE )

     work <- AudiA4Raw[start:end]
   # here we use the scan function but just scan the first line of the
   # car record -- not from a file instead we give it the
   #  line we have already read in
     temp <- scan(text = work[1], what = "a", sep = " ", quiet=TRUE)
     year[k]  <- temp[1]
     check[k] <- temp[3] # better be "A4" !

   # price
     ind <- grep("$", work, fixed = TRUE)
     temp <- scan(text = work[ind], what = "a", sep = " ", quiet = TRUE )
     price[k] <- temp[1]

} # end for loop
```

Creating the final data frame

After this loop, we make sure that *all* the values in **check** are the string "A4". If not, we know something went wrong!

The final step is to create the final data frame. I like to do these numeric conversions after I have the whole vectors, but this could have been done within the **for** loop. Note that the complete data frame with more variables can be loaded from **AudiA4.rda**

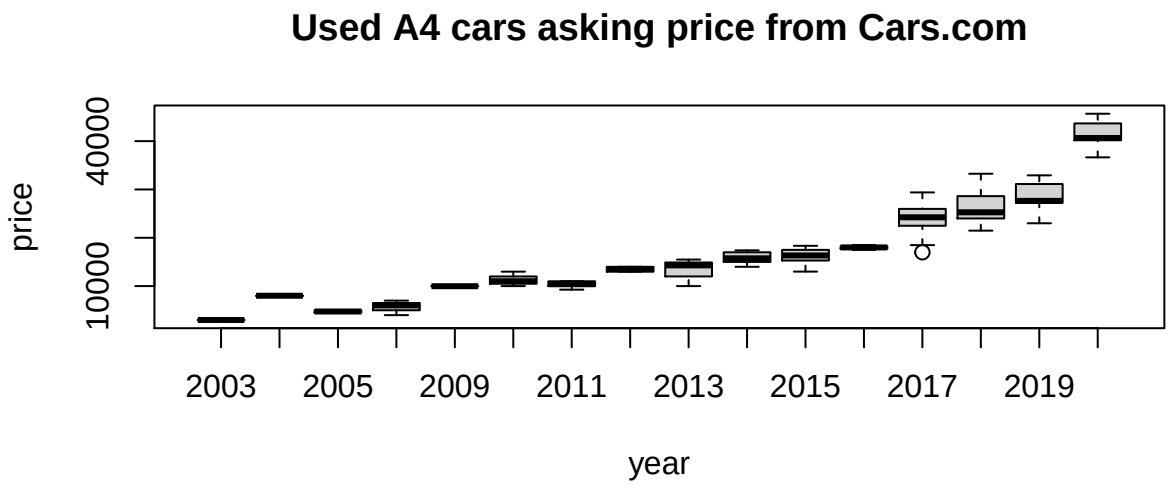
```
year <- as.numeric(year)

# remove the $ and the comma in the prices
#  "" means nothing so replace with nothing ...
price <- as.numeric(gsub('[$,]', '', price))

AudiExample <- data.frame(year = year, price = price)
```

And it is hard to resist not making a plot to check the results and start our shopping!

```
boxplot(price ~ year, data = AudiExample)
title("Used A4 cars asking price from Cars.com")
```



Data Cleaning

Typically, part of wrangling involves making the data consistent and removing extraneous parts. This is often called *cleaning*, but it is also part of the process of creating a useful data frame in R to analyze the data and answer questions. In some cases, it may be useful to convert the original data frame from the wrangling and make another copy that has some additional changes. For example, one might want to remove all the rows of the Audi A4 data set that have at least one missing value. This reduced data set is easier to use to fit lines. However, one also has to be aware that maybe those omitted rows (cars) are the bargains.

Below are some more general ideas to keep in mind.

- **Faulty data can lead to faulty conclusions.** Make sure your data are consistent and accurate.
- For more techincal data, the wrangling/cleaning should be done working with a subject-matter expert.
- Avoid making any changes to data in spread sheets. It is better to have that documented as a step in your R script. If this is not possible, make a copy of the spread sheets after you make the changes so that they can be compared to the originals.
- Put in as many checks as possible in your work.
  - These can include numbers of observations when counted using different parts of the data.

- Check that variables fall within expected ranges. (But don't inadvertently exclude that special discovery where you can win the Nobel prize!)
- Plotting raw data often helps to catch problems.
- Pay attention to which variables are numeric, which are character strings, which are dates, and which are *factor* objects.

Outliers

How do you identify an "outlier"? How is a categorical data "outlier" different from a quantitative data outlier? Perhaps just a difference in characters, possibly a misspelled or mis-typed word. For observational data, a classic outlier is adding an extra digit, or swapping digits. For example, a latte at Starbucks is reported as $8.25 instead of $2.85.

"Outliers" in right-skewed distributions can be larger in the right tail than in the left before being classified as an outlier. And vice-versa for left-skewed distributions.

Multiple kinds of outliers:

- **Global Outliers:** values that are large relative to the entire range of data.
- **Local Outliers:** values that are large in context relative to those around them. For example, a 20 degree Farenheit temperature in the summer would be an outlier in Waco but not in the winter.

Options for handling outliers include:

- Remove it.
- Flag the value. Actually, it is always a good idea to create a new column that keeps track of which observations have been identified as outliers. And create a new data set if they are replaced or edited.
- Give that value less weight in the overall analysis.
- Impute it with a reasonable value.

Missing Values

Missing values can appear as NAs, sometimes as -9999, or sometimes as just empty slots in a spreadsheet. An entry of `0` is not the same thing as an NA! Values may be missing at random or not at random. *Missing at random* means that there is no pattern to the missing values. *Missing not at random* means that there is some feature of the process that is systematically under-represented. For example, in launching weather balloons in the early 1900's, they would not launch the instruments in bad weather, so the missing observations would not capture the most extreme winds.

How to find missing values coded as NA in a dataset? Here are some useful R commands:

- `is.na()`
- `na.omit()`
- `complete.cases()`

Ways to impute missing data are:

- Replace with the overall mean or median.
- Copy values from other observations whose other variable values are similar.
- Use the average of values nearby in space or time.

In DWN's opinion, it is usually not good practice to fill in a missing value. Most statistical methods can work around having missing data and do not need them filled in.

*Example, Radiosonde Outliers:** For historical radiosonde launches, researchers at the National Climatic Data Center (NCDC) have developed a set of quality control steps that eliminates the need for both an expected launch profile and prior probabilities of detecting random errors for the historical radiosonde temperature data (Durre et al., 2008). Included in this set of steps are checks on the plausibility of a particular value, repeated runs of values vertically or through time, and inconsistent values compared with those nearby in the vertical or horizontal planes. A climatological check identifies errors that are inconsistent with a station's climatological features.

Anderson, A. N., Browning, J. M., Comeaux, J., Hering, A. S., and Nychka, D. (2016) "A simulation study to compare statistical quality control methods for error detection in historical radiosonde temperatures," *International Journal of Climatology*, 36: 28–42.

Durre, I., Vose, R. S., and Wuertz, D. B. (2008) "Robust automated quality assurance of radiosonde temperatures," *Journal of Applied Meteorology and Climatology*, 47: 2081–2095.

A Radiosonde data set

The R dataset in **CorpusCristi.rda** are the Radiosonde observations for the year 2017 from Corpus Cristi, TX. Balloons are released twice a day (at 0 and 12 UTC), and several variables are reported as the balloon ascends:

- dateTime (days and hour as a date object)
- pressure (mbar),
- height (meters),
- temp (degrees C),
- RH (relative humidity %),
- DPDP (dewpoint depression degrees C),

Figure 1: Launching a radiosonde balloon, picture from National Weather Service.

- windDir (in degrees),
- windSpeed (in meters/sec).

Note that a wind speed of 10 meters/sec is about 22.36 mph.

The wrangling of these data took quite a few steps, and some parts are similar to working with the Audi data.

```
load("dat/CorpusCristi.rda")
head(CorpusCristi)
#     dateTime pressure height  temp   RH DPDP windDir windSpeed
# 1 2017-01-01     1000     78  19.9 78.7  3.8     198       5.4
# 2 2017-01-01      925    746  18.9 71.2  5.3     241       6.6
# 3 2017-01-01      850   1475  17.8 55.1  9.1     286       3.8
# 4 2017-01-01      700   3106   6.3 69.3  5.2     260      10.3
# 5 2017-01-01      500   5782 -11.1 23.9 16.7     255      22.7
# 6 2017-01-01      400   7465 -20.3 79.0  2.7     238      29.3
```

File Organization and Naming

There are some basic guidelines for organizing files for a data science project. Generally, within a folder, for a large project you'll have subfolders for

- Data
  - Raw, original data (sometime in a separate folder if multiple files)
  - Cleaned data
  - Binary R files
- Code
- Figures
- Reports
- Documentation about the data

File names can be long, but they should be descriptive. Headers at the top of each file that describe the purpose of the file and when it was last edited can also be helpful. Version control is also important when working with a team.

For the Radiosonde data we choose to keep everything in one subfolder because there were not multiple files.

Summary of Data Wrangling Workflow

1. Always preserve the original, raw data.
2. Create a script that takes the original, raw data and transforms it into clean, well-formatted data. You may need to merge multiple files into a single file. Keep a copy of what you have done.
3. One observation in each row, and one variable in each column.
4. Save the new dataset, and then perform analysis on this clean version.
5. Handle outliers and missing values thoughtfully, ideally considering the context in which they occur and your analysis questions.
