CSI 2300: Introduction to Data Science

Lecture 10: Data Wrangling 101

Today's Topics

What is Data Wrangling?

- Some motivation
- Basic principles
- Data science workflow
- Saving reformatted data

How does wrangling fit into the data science work flow?

Details of a wrangle for Eagle Mountain (next video)

- combining files
- reshaping tables
- new variables
- writing out a new data frame

Data Wrangling

"Wrangling the data," or taking it from its initial format to a useable form, can take more than half of a data scientist's analysis time. Important principles of data wrangling are as follows:

1. Never overwrite the original data. Always preserve the original data.
2. Create an R script that loads the original data, does any manipulation needed, and then produces a clean dataset. This ensures **reproducibility.**
3. Data should be arranged as follows:
   - Each variable in a column
   - Each observation in a row
   - Each value as a cell

It is all about the data frames . . .

In this course most "wrangled" data will be in the form of a data frame. Although for some irregular data sets this may not be possible.

This step is just one in an overall data science project workflow that begins with the birth of a problem and ends with an analysis that is communicated to someone else. It is important to realize that that the wrangling step is not a negative and onerous aspect of this process but more about translating the raw information collected into a form that is focused for answering specific questions.

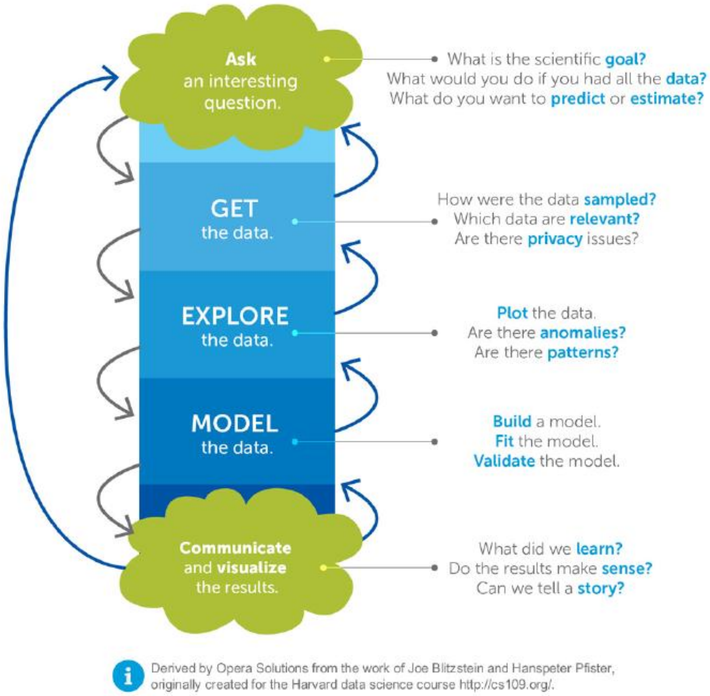

Figure 1: Typical data science workflow.

Note that this diagram glosses over the important wrangling step in the GET box.

- The process is iterative from beginning to end.
- **GET the data** can include collecting the raw data from original sources via
  - databases
  - crawling
  - streams
  - binaries
  - API
- **WRANGLE the data** which can include but is not limited to the following:
  - reorganizing the data
  - transforming variables or recoding categorical data
  - identifying and removing/imputing NAs
  - renaming variables
  - averaging values (e.g., to a lower resolution time stamp)
- Visualizing tools are used extensively, beginning in **EXPLORE the data** with examples such as
  - histograms
  - boxplots
  - scatterplots

    - time series plots

- **MODEL the data** based on the goal of the analysis

    - linear models
    - clustering
    - classification
    - forecasting

- **Communicate and visualize** the results, which can include

    - developing interactive apps
    - commenting and publicizing code
    - making recommendations for future data collecting efforts

- The beginning and ending of the process requires subject-matter expertise. The middle boxes are where statistical and computer science tools are used.

- Story-telling is taking an idea and turning it into a story that is compelling and that prompts people to take action.

Finally, in walking through this process it helpful to come back to one definition of a data scientist:

Someone who knows more statistics than a computer scientist, more computer science than a statistician, and can explain their results to audiences that are neither statisticians nor computer scientists.

Example, Eagle Mountain Lake: This is an example of some of the steps needed to transform the Eagle Mountain Lake raw data into the dataset that you have used and that follows these standards. When it is first downloaded from the website[1], each variable is saved in a different file. Within each variable's file, the date and time stamp are in the second column, but then the variable's value for each depth is given in the next twenty-one columns.

```
temp  <-read.csv(
  file = "dat/EagleMountain/temp_through_09_12_2019.csv", header=T)
DO    <-read.csv(
  file = "dat/EagleMountain/DO_through_09_12_2019.csv", header=T)
DOsat <-read.csv(
  file = "dat/EagleMountain/DOsat_through_09_12_2019.csv", header=T)
pH    <-read.csv(
  file = "dat/EagleMountain/pH_through_09_12_2019.csv", header=T)
cond  <-read.csv(
  file = "dat/EagleMountain/cond_through_09_12_2019.csv", header=T)

head(temp)
```

[1] no longer active

```
#   Observation      DateTime     X0   X0.5     X1   X1.5     X2   X2.5     X3
# 1           0  4/25/19 0:00 19.156 19.137 19.193 19.229 19.171 19.239 19.240
# 2           1  4/25/19 2:00 19.053 19.093 19.008 19.032 19.019 19.056 18.923
# 3           2  4/25/19 4:00 18.987 18.919 18.945 18.953 18.983 18.894 18.901
# 4           3  4/25/19 6:00 18.961 18.919 18.927 18.954 18.912 18.901 18.905
# 5           4  4/25/19 8:00 18.979 18.954 18.980 18.887 18.970 18.962 18.962
# 6           5 4/25/19 10:00 18.767 18.852 18.790 18.833 18.832 18.818 18.764
#     X3.5     X4   X4.5     X5   X5.5     X6   X6.5     X7   X7.5     X8   X8.5
# 1 19.191 18.980 18.752 18.735 18.713 18.584 18.607 18.544 18.407 18.444 18.299
# 2 18.954 18.986 18.924 18.939 18.979 18.917 18.936 18.939 18.866 18.773 18.447
# 3 18.905 18.957 18.959 18.933 18.891 18.995 18.912 18.406 18.337 18.303 18.274
# 4 18.926 18.914 18.915 18.906 18.926 18.962 18.877 18.915 18.923 18.946 18.157
# 5 18.962 18.966 18.977 18.992 18.882 18.996 18.905 18.272 18.195 18.212 18.249
# 6 18.873 18.771 18.815 18.850 18.770 18.804 18.868 18.763 18.667 18.266 18.143
#       X9   X9.5    X10
# 1 18.292 18.374 18.149
# 2 18.319 18.211 18.057
# 3 18.144 17.928 17.875
# 4 17.944 17.763 17.716
# 5 18.042 17.911 17.980
# 6 18.207 18.173 17.969
```

How should the data be arranged to meet the row-column-cell criteria? There are measurements at 21 depths for each date-time stamp, so we want to repeat each date-time stamp 21 times, each one associated with a different depth. Then, we have the date-time stamp in the first column, the depth in the second column, and the five variables in each of the next columns. It actually doesn't take very many lines of code to make this happen. The steps are as follows:

- Read in the five separate .csv files.
- Repeat each entry of the time-date stamp 21 times (one for each depth).
- Create a sequence of the 21 depths. Repeat this sequence for the number of unique time stamps present.
- Take the 21 columns of each variable, transpose it (flipping columns and rows), and create a single vector.
- Combine the time stamps, depths, and the five variables into a single data frame.

```
##---------------------------------------
## Rearranging data to have variables
## in columns and observations in rows
## with depth as a new column
##---------------------------------------

#There are measurements at 21 depths for each date-time stamp.
#We want to repeat each date-time stamp 21 times, each one
#associated with a different depth.
```

```r
new.Date.Time.col<-rep(temp[,2],each=21)

#Create a column of depths of the values {0, 0.5, 1.0, ..., 10.0}
#repeated for each observation.

depth<-seq(0,10,by=0.5)
new.depth.col<-rep(depth, dim(temp)[1])

#Reformat the variables so that they are in a single column.
#What does each of the commands in one of these lines do?

temp.one.col <- c(t(as.matrix( temp[ ,3:23])))
DO.one.col   <- c(t(as.matrix(   DO[ ,3:23])))
DOsat.one.col<- c(t(as.matrix(DOsat[ ,3:23])))
pH.one.col   <- c(t(as.matrix(   pH[ ,3:23])))
cond.one.col <- c(t(as.matrix( cond[ ,3:23])))

#Combine all columns into a data frame and rename them
all.data<-data.frame(DateTime = new.Date.Time.col,
                     Depth = new.depth.col,
                     Temp = temp.one.col,
                     DO = DO.one.col,
                     DOsat = DOsat.one.col,
                     pH = pH.one.col,
                     Cond = cond.one.col)

head(all.data)
#       DateTime Depth   Temp     DO   DOsat    pH    Cond
# 1 4/25/19 0:00   0.0 19.156 10.455 116.370 8.586 421.801
# 2 4/25/19 0:00   0.5 19.137 10.468 115.732 8.578 421.859
# 3 4/25/19 0:00   1.0 19.193 10.411 115.345 8.617 419.710
# 4 4/25/19 0:00   1.5 19.229 10.414 115.121 8.618 419.609
# 5 4/25/19 0:00   2.0 19.171 10.419 114.568 8.571 421.432
# 6 4/25/19 0:00   2.5 19.239 10.351 114.830 8.570 421.246
dim(all.data)
# [1] 35532     7

##--------------------------------------
## Save the restructured data
##--------------------------------------

write.csv(all.data, file="dat/EML_through_09_12_2019.csv",
          row.names = FALSE, col.names = TRUE)
# Warning in write.csv(all.data, file = "dat/EML_through_09_12_2019.csv", :
```

```r
# attempt to set 'col.names' ignored

save( all.data, file="dat/EML_through_09_12_2019.rda")
# list out contents of the directory
dir()
# [1] "dat"                  "fig"                  "prog_10_inclass.pdf"
# [4] "prog_10_inclass.Rmd" "prog_10_lecture.pdf" "prog_10_lecture.Rmd"
```

Once you have wrangled the raw data into a new, more user-friendly form, you will want to save it. There are multiple options, as follows:

- The `write.csv()` command will create a `.csv` file. You then use the `read.csv()` command to load the file in working memory.

The `write.csv()` command takes as arguments a matrix or a data frame along with the file name and path to where the file will be saved.

- The `save()` command will create a `.Rdata` file. The `load()` command is used to read the file back into working memory, but it cannot be assigned to anything, and you'll have to hunt a bit to see what the object is called in R's memory. The `ls()` command can be used to list all of the objects in working memory.

It also accepts as arguments a matrix or data frame with the file and file path.

- The `saveRDS()` command will create a `.Rds` file with similar arguments `write.csv()` and `save()`. The `readRDS()` is used to read in a new `.Rds` file, and it can be assigned to an object name.