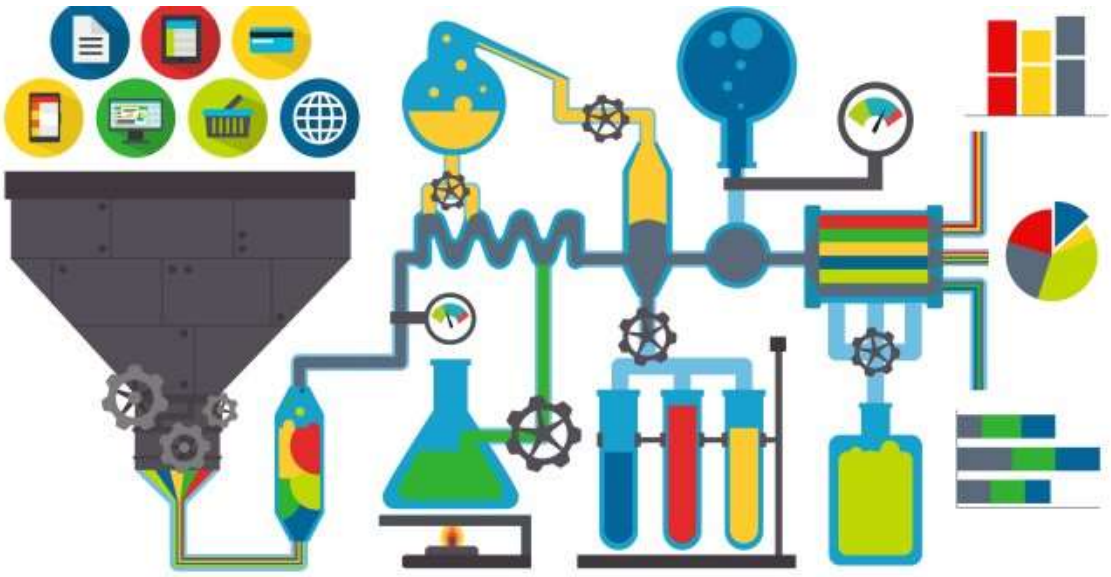

Data Wrangling Writeup

UDACITY DAND PROJECT 7

James R. Small | September 7, 2018

Data Gathering

My analysis started with retrieving and assessing the data. First, I downloaded "twitter-archive-enhanced.csv" from Udacity and opened it up in Microsoft Excel, v2016. Excel is a great tool for quickly assessing smaller datasets. I adjusted the column widths, "froze" the title row so it doesn't scroll off when paging through the data, and added header row "filters." Excel data filters allow you to see a summary of the column's contents including if data is missing ("blanks"). Excel also works great with large monitors allowing you to get a nice wide view and quickly scroll down the columns. Next, I retrieved "image-predictions.tsv." Importing this into Excel (it doesn't include ".tsv" files as a "native" filetype), I followed the same procedure with column widths, freezing, and filtering to set it up as the second spreadsheet (tab) within my workbook.

Finally, I worked on retrieving complete information about each Tweet from the set included in "twitter-archive-enhanced.csv." To facilitate this, I iteratively developed a Python module to parse out all Twitter status IDs from the CSV file, and then fetch all status information using tweepy to interface with Twitter's API. As this was somewhat involved, I created a third spreadsheet (tab) in my workbook which includes the tweepy model fields for the Twitter API status (tweet) and user objects. This sheet describes what each field is for and includes hyperlinks to the relevant tweepy/Twitter API documentation. This proved a useful reference as APIs and libraries constantly evolve leaving stale information and some ambiguity. Leveraging these references, Stack Overflow, and the tweepy source code I was able to complete the Python script and retrieve the data.

Data Wrangling

With the data in hand, I loaded the three sources described above into pandas dataframes. Beginning with a look at the column types and data I noticed several issues. First off, tweets (statuses), users, and other objects all have unique Identifiers (IDs). These identifiers look like integers but can require up to 53 bits to store. The problem with storing these ids as ints is that they're identifiers and not numeric data. In addition, it requires care that they're maintained as 64-bit ints and not 32-bit ints or else they'll become invalid. The Twitter documentation recommends storing these as strings to avoid these problems. So, I converted these all to strings. The objects retrieved from the Twitter API also contained both a numeric and string format for each identifier. The numeric columns were pruned keeping only the string typed ones.

Next, date/time columns were converted to the datetime type. True/False columns were converted to the Boolean type. Columns which were empty, contained only a handful of values, or were always false/null were removed. In addition, columns related to the authenticating ID used to retrieve objects from Twitter were eliminated. Columns that contained embedded objects were simplified to contain only a reference to the object ID and not the entire object. Some of the statuses could not be obtained from Twitter

because of various errors. This was captured in error status code and message columns. This also brought an interesting pandas/numpy type limitation to my attention. Integer types can't store nulls/NaNs. Trying to store this value type in an int column will coerce it into another data type. This required sometimes using other data types or special values to represent null as described in the cleaning section of the Notebook.

The status errors required some reconciling of the original dataset. The statuses with retrieval errors were removed. In addition, per the project specifications – quoted tweets, retweets, and reply tweets were also purged. Thus, only original tweets were retained. For the dog ratings, multiple issues were found. Given the size of the dataset, trying to determine which ratings were correctly extracted and which weren't seemed too hard. Instead, I just extracted the ratings myself. By iterating, I was able to reasonably capture the data. There are some corner cases where a single tweet contains multiple ratings. My solution to this was to average the ratings. Occasionally the rating doesn't refer to a dog but given this was a very small percentage of the time I just accepted it.

For dog names, I was not able to find a good way to extract the names. Therefore, I cleaned this data in place. The only problem I found is that sometimes non-names were selected. These were easy to filter out with regular expressions though – they always started with a lowercase letter, whereas names always started with uppercase. For the dog "stages" there were also some issues. Since these were easy to extract I just did it this way. The only tricky part here was there were case differences. This was overcome by extracting the stage using a case-insensitive regular expression. The other issue was there were a handful of columns with interesting nested data. For these, if there was only one embedded dataset then the relevant data was added to the main dataframe as new columns. Where there were multiple datasets within the embedded data, new dataframes were created to maintain "tidiness."