Wrangle Report

November 22, 2017

In this project, we were tasked with wrangling and analyzing data related to the WeRateDogs Twitter account. This process invloved gathering data from multiple sources, assessing the quality and tidiness of the data, cleaning the data, then performing analysis on the newly created masterset of data.

Gathering Data In this project, we need to use data from three different sources:

- A file, titled `twitter_archive_enhanced.csv`, which is given to us
- A file, titled `image_predictions.tsv`, which we needed to download programmatically from an online resource
- The Twitter API, which we required us to interact with Twitter's REST API to programmatically download tweet data

The first file, `twitter_archive_enhanced.csv`, was downloaded and then uploaded to the Jupyter notebook server. The second file, `image_predictions.tsv`, was downloaded in the `wrangle_act.ipynb` notebook using Python's request library. The Twitter API was accessed in the `twitter_data_fetch.ipynb` notebook using the `tweepy` library.

In order to access the Twitter API, I had to create an 'application' on Twitter and use the keys they provide to authenticate. Then, I used the tweet IDs in the `twitter_archive_enhanced.csv` to query the API for relevant information (specifically, retweet and favorite counts). I compiled the data in a pandas DataFrame, then saved it as a txt file, 'tweet_json.txt', in JSON format.

After collecting all the data, I imported each file into it's own pandas DataFrame.

Assessing Data After importing the data, I printed each DataFrame to visually inspect them. I noted each column and described what I inferred the column to represent. Right away, I noticed issues where some columns that should have had data that was just a string of digits (such as `retweeted_status_id`) was being represented as a float in scientific notation.

I then used each DataFrame's `info()` method and noted the column types. I saw that there were some columns that were being represented as integers where they should have been objects, such as `tweet_id`.

Next, I examined the "dog types" (e.g., "pupper", "doggo", etc.) that were present in each observation. I realized that the data was untidy in that one variable was being represented in four columns. I also noticed that 14 observations had mutliple dog types (which can't happen).

I also noticed that there were some observations in which the name of the dog was entered as "quite". This was a mistake made during parsing of the tweet body. There were a number of similar naming mistakes through the set.

I used the DataFrame's `value_counts()` method to display the counts for each value in the `rating_numerator` column and saw that a majority of the values were between 4 and 14. I examined a tweet with a `rating_numerator` of 1 to find that the tweet body mentions a '3 1/2 legged' dog, indicating another parsing error.

Similar to analyzing the `rating_numerator`, I examined the `rating_denominator` to find a vast majority of denominators entered were 10.

Many more observations were made during this process, some of which had direct impact on how cleaned the data while others were used to gain context in the set.

Cleaning Data In this section, I noted multiple quality and tidiness issues to be cleaned. First, I made copies of each of the DataFrames using their `copy()` method. Next, I went through each issue I noted and defined the problem, coded a solution, and tested it.

Some of the issues were as straight forward as replacing values (such as replacing 'doggo' in the `doggo` column of tweet `855851453814013952` with `None`) while others required converting column data types (such as from object to datetime).

For the issue where values were interpreted as floats instead of objects, I had to convert the value to be represented as a string of digits instead of in scientifc notation, then cast the column as an object type.

In the case where we had to convert the doggo, floofer, pupper, and puppo columns to one column, I had to iterate through each observation and place the result of which column had text in it into a new `dog_type` column. I also had to handle the case where multiple dog types were present.

Storing Data After cleaning the data, I merged each DataFrame based on their `twitter_id` columns (or in the case of the Twitter API data, `id`) using an inner join. The result was one DataFrame with all the data present and cleaned. I then saved the DataFrame as a csv file, `twitter_archive_master.csv`, using the DataFrame's `to_csv()` method.