WeRateDogs Twitter Data from 2015 to 2017

Data Wrangling Project - Report

Table of Contents

1. Introduction

For the final project of the Data Wrangling unit, we were tasked with examining twitter data for popular handle WeRateDogs [@dog_rates](https://twitter.com/dog_rates) and performing in-depth, but not exhaustive, data wrangling procedures on several sets of data. The end goal was to exhibit our data gathering, accessing, and cleaning abilities. The result we would be few simple insights and a basic visualization, with more focus on the data wrangling process rather than the product. I'll go into brief detail on each step below.

2. Gathering

Three data sources, which were turned into dataframes, were used for this project:

1. "twitter-archive-enhanced.csv" → `df_archive`
2. "image-predictions.tsv" → `df_image`
3. "tweet_json.txt" → `df_tweets`

(1) "twitter-archive-enhanced.csv" was made available for download in advance by Udacity instuctors. It contains about 3000 tweets and their date from 2015 to 2017. The data includes a tweet ID, tweet text, date tweeted, tweet URL, extracted dog ratings (typically out of 10, but with a numerator like 12 or 13 for good humor), the dog's name, so-called dog "stages" (such as the young "puppo" to the older "doggo"), and other data points. It was turned into a dataframe using pandas `read_csv()` function.

(2) "image-predictions.tsv" is a file prepared by Udacity instructors that need to be downloaded via URL. Using the `requests` package and the `get()` function to access the file, I used the `os` package to open and write the file, then turned it into a dataframe using Pandas as above. The file contains results from running the WeRateDogs tweet archive images through a neural network to try and classify the breeds of dogs. The resulting file contains a table of image preditions (top 3), and each corresponding tweet ID, image URL, and the image number that corresponded to the most confident prediction.

(3) "tweet_json.txt" is a text file in JSON format provided by Udacity instructors. I had initially tried to gather this data independently from the Twitter API and Python's Tweepy libary but I ran into time-consuming errors (as another user explained here) when trying to query the Twitter API for each tweet's JSON data. So I instead opted to use the provided data which I turned into a dataframe once again using Pandas. The file contains tweet IDs, retweet count, and favorite ("like") count.

3. Assessing

We were instructed to "detect and document at least eight quality issues and two tidiness issue" using both visual and programmatic assessment. To assess the quality and tidyness of the data, I performed multiple data exploration functions on each dataframe. These functions included, but were not limited to:

- `sample()`
  - Used this function to get a sampling of rows from each df to see how the columns and data were written, if there were any NaN entries, any unnecessary data or columns, any commonalities or differences between each dataset, and simply to see if any issues could be detected by browsing through entries.
- `info()`
  - Here I could see differences between column names and their respective datatypes. If something that should be a number but was an "object" rather than "int64", I could identify it here. I could also see differences in number of rows.
- `duplicated()`
  - Here I could string together several `list()` methods to detect if columns were duplicated between dataframes. Identifying such columns would be useful for merging and joining dfs if necessary.
- `describe()`
  - This function uses basic summary statistics to gather insights across the numerical data. Here I can see if say the max or min value of the rating numerator or denominator are suspiciously high or low.
- `value_counts()`
  - Here I can look at specific columns to count the values of all variables and have them shown in order. I can notice if certain values are suspiciously high or low.

I identified the following issues:

Quality issues

#	Dataframe	Issue
1	df_archive	Contains duplicate tweets (ie. retweets, as evidence of 181 count for `retweeted_status_id`)
2	df_archive	Unnecessary columns (`in_reply_to_status_id`, `in_reply_to_user_id`, `retweeted_status_id`, `source`, `retweeted_status_id`, `retweeted_status_user_id`, `retweeted_status_timestamp`)
3	df_archive	Different `tweet_id` count from `df_image` (suggests some tweets in `df_archive` do not have images)

#	Dataframe	Issue
4	df_archive	`name` column contains name 'None'
5	df_archive	`name` column contains entries 'a' and 'quite' (i.e. non-names that start with lower-case)
6	df_archive	`text` column contains hyperlink info (starting with 'https')
7	df_archive	`rating_numerator` column has values as low as 0 and as high as 1776 (typically between 10 and 15) and `rating_denominator` column has values as low as 0 and as high as 170 (typically 10)
8	df_archive	`timestamp` column is 'object' Dtype and 'tweet_id' is 'int64' Dtype
9	df_tweets	All columns, despite being numbers, are 'object' Dtype and the shared observation is labeled `tweet ID`
10	df_image	Has multiple image predictions when only one is necessary

Tidiness issues

#	Dataframe	Issue
1	df_archive	Variables as column headers (`doggo` , `flooder` , `pepper` , `puppy`)
2	df_tweets + df_image	Share same observational unit as `df_archive` so they don't need to be separate dataframes

4. Cleaning

Corresponding to the above quality and tidiness issues, I defined and fixed the issues as follows.

Quality:

1. Remove unnecessary retweets using boolean masking to select only entries that have null values (ie. that are "True") for `retweeted_status_id`
2. Drop unnecessary columns
3. Drop rows that are not common between `df_archive` and `df_image` using the `isin()` function to align the `tweet_id` count
4. Examine `name` column entries that contain "None" to confirm that they are entered correctly, and then fix entries if necessary.
5. Fix misentered names in the `name` column
6. Remove hyperlink data from *text* column in the _df*archive* dataframe using regex and string splitting.
7. For entries with irregular denominators (i.e. not 10), normalize both the numerator and denominator to a standard denominator of 10. For entries with irregular numerators (i.e. outliers outside of the 95th percentile but have denominators of 10), either normalize the entries using the overall median or fix an error
8. Change dtype of `timestamp` column to `datatime` using `to_datetime`
9. Change dtype of `tweet ID` , `retweet count` , and `favorite count` to `int` using the `astype` function. Rename `tweet ID` to `tweet_id` so that it matches the naming convention of the other tables
10. Drop all columns except for `tweet_id` , `jpg_url` , and `p1` . Rename 'p1' to 'breed'.

Tidiness:

1. Extract dog stage names in `text` and, if found, add them to a new column `dog_stages`.
2. Merge `df_tweets_clean` to `df_archive_clean` to create `df_master`. Merge `df_image_clean` to `df_master`

5. Conclusions

I had a difficult time with regex in this assignment. I had to pause and do an entirely separate course on regex to feel confortable even attempting what I knew I wanted to do. I ended up reusing some of my code (just twice) so I turned one block into a function.

```
# Use a custom function to extract the name of the dog and
place the dog name in the `name` column
def dog_name_finder(df):
    """ Use a regex to extract the name of the dog and place
the dog name in the 'name' column.

    Keyword arguments:
    df -- String in 'text' column must contain the one of the
following patterns to be matched:
    - "named *dog name*"
    - "name is *dog name*"

    """
    x = df.text.str.extract(r'(?:(?:name\sis\s)|(?:named\s))
([a-zA-Z]+)')
    df['dog_name'] = x[0]
    df = df[x[0].isnull() == False]
    return df

none_names = dog_name_finder(none_names)
none_names
```

There was a lot of trial an error in making this code robust enough to find the dog names in text but not too far reaching so as to get noise. After examining the tweet text visually in a spreadsheet, I could see patterns like "named *dog name*", "name is *dog name*", and "This is *dog name*". So I intially included expanded code to extract all three of those combinations, but realized "This is *dog name*" found some names but more often found "This is *a dog breed* so I got many "a" and "an" which was a problem I was fixing as a seperate issue. I revised my code and had better results.

I also changed the order and added quality and tidiness issues as I went along. I did my intial assessment and listed my issues, but solving one issue often presented another. For example, I'd go back and move the 5th issue up to the 1st place because solving one issue would be best done earlier. The wrangling process was far from straightforward and I had to be flexible and adaptive.

I came to the conclusion that I need to improve the speed and general python abilities. This project took longer than expected. That's down to my lack of experience but also the

fact that data wrangling is a particularly time-consuming process. Once I had a clean, tidy master dataframe, I could easily pull out any insights from the data. The more time I put into wrangling, the lower chance I'll run in to problems at later stages.

In [ ]: