wrangle_report

September 19, 2023

0.1 Reporting: wrangle_report

This report is a high level overview of the wrangling efforts that went into the gathering, assessing, and cleaning of the WeRateDogs twitter data.

0.2 Data Gathering

First, I gathered data from three different sources. 1. Directly downloaded the provided WeRateDogs Twitter archive data (twitter-archive-enhanced.csv) 2. Used the requests library to programmatically download the image_predictions.tsv file from a url. 3. Queried the Twitter API using the Tweepy library (tweet_json.txt). Note: I had verification issues with the twitter API, so I used the code and file provided by a Udacity instructor.

I then read all three files into separate dataframes: df_archive, df_predictions, df_tweets.

0.3 Assessment and Cleaning

Note: After assessing each dataframe visually and programmatically and before cleaning the data, I made a copy of each dataframe. This is to preserve the original data in case we need it again.

First, I assessed df_archive visually and programmatically and discovered a variety of tidiness and quality issues that needed to be cleaned.

- Converted tweet_id to a string because we would not be doing calculations with this identifier.
- Converted the timestamp column to datetime.
- The dog stages (doggo, floofer, pupper, and puppo) were in four separate columns. I combined all into one dog_stage column and dropped the original four columns. In the case where a record had two or more dog stages listed, I chose to use the first stage encountered in the following order: 1) doggo, 2) floofer, 3) pupper, 4) puppo.
- Per project instructions, we only wanted original tweets with images in our dataset. I removed rows where expanded_urls was null, in_reply_to_status_id was not null, and retweeted_status_id was not null. This ensured we removed retweets and replies and only kept original tweets with images. I then dropped expanded_urls, in_reply_to_status_id, and retweeted_status_id columns to keep the data tidy.
- I discovered many lowercase "names" that were not actually dog names. I changed all lowercase records in the names column to None and then changed None to NaN. This left only real names.
- Removed rows where rating_numerator or rating_denominator = 0 so it would not affect any calculations we may have wanted to do with those columns.

- Identified 17 rows where the rating_numerator was not equal to 10. Since there were so few records, I simply droppped them. Had there been more than 17, I would have tried to normalize them to equal 10.

Then, I visually and programmatically assesed df_predictions and cleaned the issues.

- Converted tweet_id to a string because we would not be doing calculations with this identifier.
- Redundant prediction columns in df_predictions were handled by choosing the best prediction based on confidence and assigning the breed prediction to a new column called dog_breed and the accompanying confidence to a new column called confidence. The True predictions were actual dog breeds, while the False predictions were classifying the images as things like laptop, sarong, orange, etc. All True records became a dog_breed, while False records were changed to "Not a Dog". I then dropped all unneccessary columns.
- In the process of cleaning up the redundant predictions, I also cleaned up instances where the prediction was not an actual dog. We do not want predictions where the dog breed is laptop or sarong! Using the True/False Boolean made it easy to clean this while in the process of handling the redundant predictions.
- The dog_breeds were still pretty ugly with underscores between two-word breed names and non-standard capitalization. I replaced the underscores with a space and capitalized the first letter in each word in the dog_breeds column. This also capitalized the A in "Not A Dog" records, but that is okay. It's a grammar technicality.

I then visually and programmatically assessed df_tweets, which had only one issue.

- Converted tweet_id to a string because we would not be doing calculations with this identifier.

After I finished cleaning the issues, I combined all three dataframes into one master dataframe called df_tweet_archive_clean.

While checking the df_tweet_archive_clean table for errors, I discovered a few more records that did not have images (jpg_url had null records). I dropped these nulls from the dataframe. I also noticed that img_num was a float, when it should have been an integer since we cannot have 1.5 images. I changed this, and then saved the cleaned master dataset to the file twitter_archive_master.csv.