Data Wrangling Report

Table of Contents

Gather

Gathering data is a crucial step in data wrangling or data preprocessing. In order to analyze data effectively, I need to ensure that I have the right kind of data and that the data is clean and accurate. This involves several steps, including defining the problem or objective, identifying sources, collecting the data, cleaning and validating the data, and transforming and organizing the data. Overall, data gathering is important in data wrangling because it helps ensure that the data I have is clean, accurate, and relevant to the problem I am trying to solve. This can save time and effort in later stages of data analysis, and help produce more accurate and reliable insights.

I gathered this data from 3 sources in 3 different formats.

1. CSV file (archive sent by WeRateDogs twitter account).
2. TSV file downloaded through the `requests` library.
3. Tweepy: A Twitter API.

The first source is pretty easy and straight forward to work with. I just needed to read the CSV file. The second source, however, was a little trickier, but still manageable. I needed to download the file from the internet (using `requests.get()`) and save it locally, only then do I read it from the TSV file that I just downloaded. The third source was the most challenging for these three reasons:

1. Access and authentication: To use the Twitter API, you need to obtain access and authentication credentials, which can be a complicated process. Twitter requires developers to go through an approval process to ensure that they are using the API for legitimate purposes and to prevent abuse.

2. Rate limits and quotas: Twitter limits the number of requests you can make to the API within a certain time frame. This can limit the amount of data you can collect at a time, and may require you to manage your requests carefully.

3. Data structure: The data returned by the Twitter API can be complex, with multiple levels of nested objects and different data types. Understanding the structure of the data and how to extract the relevant information can be challenging. For example: `"sizes": {"large": {"w": 540, "h": 528, "resize": "fit"}, "thumb": {"w": 150, "h": 150, "resize": "crop"}, "small": {"w": 540, "h": 528, "resize": "fit"}, "medium": {"w": 540, "h": 528, "resize": "fit"}}}]}, "extended_entities": {"media": [{"id": 892420639486877696, "id_str": "892420639486877696", "indices": [86, 109], "media_url": "htt...`

Assess

After gathering, assessing the gathered data is a must. Data may contain errors, missing values, or inconsistencies, and assessing the data can help identify and correct these issues. Assessing data also helps identify any biases or limitations in the data. For example, if you are analyzing data from a survey, assessing the data can help identify any biases in the sample or any questions that were poorly worded, leading to inaccurate responses. Assessing data can also help to identify patterns or trends in the data that may not be immediately apparent. This can help you to gain insights into the data and identify areas for further analysis.

I assessed the data **both** visually and programatically, and both assessments were useful for identifying certain issues. The issues I identified were:

Quality

1. Null values in `retweeted_`... columns and in `in_reply_to_` columns.
2. Timestamp includes +0000, which is uneeded because it's the same timezone for all the records.
3. Missing Values replaced by `None` when it should be `NaN`
4. Wrong Data Types:-
   - Data Type of `pupper`, `doggo`, `floofer`, and `puppo` is *object*, but it representes a *bool*.
   - Data Type of `tweet_id` in df_tweet_meta is *object* when it should be *int64*
   - Data Type of `retweeted_status_timestamp` and `timestamp` is *object* when it should be *datetime*
5. Text formatted poorly in the image predictions table.
6. Some predictions aren't dogs.
7. 3 different predictions for type of dog.
8. Irrelevant Data: ID 1004 in the archive table.
9. Twitter ID: 666287406224695296 in archive table.. wrong rating extracted. Should be 9/10 not 1/2.
10. 23 records have a denominator that is other than 10 in the archive table.
11. Image Predictions have missing records for some tweets. (Archive record count: 2356.. Image Prediction record count: 2075)
12. Some records in the archive aren't tweets, instead they are retweets.
13. Meaningless data & Missing Values: `in_reply_to_status_id`, `retweeted_status_id`, `retweeted_status_user_id`, and `in_reply_to_user_id`.
14. Sometimes the `name` column is "a" when the text contains: "This is `a` --".. this can be fixed but for the sake of simplicity, I won't fix it in this notebook.
15. Outliers in the `rating_numerator`

Tidiness

1. Type of dog should be only one column, not four.
2. Html tags form the source column in the archive are not required.

- Programatically, I checked for duplicates (none were found), null values, outliers, missing data, wrong datatypes, etc..
- Visually, I checked for consistency and unnecessary/irrelevant data

Clean

This is a very important step as it means that the data I'll analyze or give to an AI model is complete and would give plausible results.

Here, I started tackling each issue individualy through the define-code-test framework. I first made a copy of my data (I am grateful I did so because I messed up the data a million times). Then, I started cleaning.

1. I removed irrelevant data by using `.drop()` to remove tweets and messed up records.
2. I treated missing values through `drop()`ing the respective columns that were uneeded too.
3. "None" instead of "NaN". In the type of dog columns, I changed None to False and the other value to True to make the 4 columns booleans. In the name column, I changed "None" to `np.nan`.
4. I fixed the incorrect Data Types by simply using `.astype()` to correct the Data Type. I also removed the localization (+0000) from the timestamp.
5. I fixed the broken ratings for a certain record (hard coded) and `.drop()`ed all columns with a rating_denominator other than 10 to have a unified rating system.
6. I fixed that the type of dog was expressed in 4 columns by using `.apply()`. This is used to extract the index of the first occurrence of a True value in one of the 4 columns. Here's how it works:

- The first part of the code `.apply(lambda x: x.index[x == True], axis=1)` applies a lambda function to each row of the DataFrame. The lambda function returns the index of each True value in the row.
- The second part of the code `.apply(lambda x: x[0] if len(x) > 0 else np.nan)` applies another lambda function to the result of the previous lambda function. This lambda function returns the first index value if there is at least one True value in the row, or NaN if there are no True values in the row.

1. I fixed the poorly formatted predictions by replacing `_` with a space using the `.str.replace()` function.
2. I fixed the multiple predictions by selected the highest confidence one and the one that IS a dog. I simply check if the highest confidence is a dog, if yes use it, if not then do the same thing for the second highest confidence..
3. I removed the HTML tags from the "source" column by `.str.extract()`ing the link from the href attribute.
4. Finally, I replaced outliers in the rating by the mean of the "non-outliered" data.

After cleaning the whole data, I still needed to merge them into one, master dataset to be able to analyze it.

I also added month and day columns that will further help me analyze the trends in the data.

After finishing everything, I stored the master dataset to a CSV file.