Data Wrangling Project - We Rate Dogs

By Ricardo Rosas

Hello World! In this document I will go through my data warangling efforts gathering, assessing, and cleaning and visualizing data of WeRateDog tweets. To see the code, go to my github account: rrosasl . Here I only provide a summary/subset of all the code I used

This is done as part of my [Data Analyst Nano Degree (https://www.udacity.com/course/data-analyst-nanodegree--nd002)](https://www.udacity.com/course/data-analyst-nanodegree--nd002)

Background

From the project instructions:

Real-world data rarely comes clean. Using Python and its libraries, you will gather data from a variety of sources and in a variety of formats, assess its quality and tidiness, then clean it. This is called data wrangling. You will document your wrangling efforts in a Jupyter Notebook, plus showcase them through analyses and visualizations using Python (and its libraries) and/or SQL.

The dataset that you will be wrangling (and analyzing and visualizing) is the tweet archive of Twitter user @dog_rates, also known as WeRateDogs. WeRateDogs is a Twitter account that rates people's dogs with a humorous comment about the dog. These ratings almost always have a denominator of 10. The numerators, though? Almost always greater than 10. 11/10, 12/10, 13/10, etc. Why? Because "they're good dogs Brent." WeRateDogs has over 4 million followers and has received international media coverage.

The data wrangling process

My final objective is to have a database with information about WeRateDogs tweets, including what dog type it is (e.g golden retriever), what stage they are (e.g. pupper), their WeRateDogs rating (e.g. 14/10!) and how many tweets and retweets they got.

To do this I had to go through the entire data wrangling process. What's that, you ask? Well, I had to

> Gather data from three different sources (a csv with all of WeRateDogs tweets, a database with predictions from a neural network, and finally twitter's own data which I scrapped containing information on the number of tweets and retweets).
>
> Assess the data. Real-world data is messy and after I gathered the data I realized it wasn't good to be analyzed right away. In this stage I assessed the data and made observations on the data quality and tidiness
>
> Clean the data. Here I prioritized some of the observations from the previous phase to clean them, i.e. adapt them in a way that helps me do further analysis.

For all the work, I used the following python libraries:

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import requests
import tweepy
import json

%matplotlib inline
```

Data Gathering

I had to gather data from three different sources. `

> 1) WeRateDogs twitter archive (twitter-archive-enhanced.csv'). Name = twitter_archive
>
> 2) Image predictions of what breed of dog it is (image_predictions.tsv) name = predictions Need to access these data using requests from udacity's url (https://d17h27t6h515a5.cloudfront.net/topher/2017/August/599fd2ad_image-predictions/image-predictions.tsv)
>
> 3) Each tweet's retweet count and favorite information from tweepy (http://www.tweepy.org/) (API for twitter) name = tweet_meta

WeRateDogs's twitter archive

For `twitter_archive` it was relatively straight forward. I already had a .csv file on my computer. I just needed to read it, which I did using:

In [ ]:

```
twitter_archive = pd.read_csv('twitter-archive-enhanced.csv')
```

Predictions on dog breed

For `predictions` the work was slightly more challenging. Although the data was also on a csv file, I needed to access it by requesting it from Udacity's server. I did it using the requests library. Here I faced some issues requesting the data which I solved by disabling verify (warning! not recommended unles you truly truly trust the data source)

In [ ]:

```
#Import using requests
url="https://d17h27t6h515a5.cloudfront.net/topher/2017/August/599fd2ad_image-prediction
s/image-predictions.tsv"
res = requests.get(url,verify=False) #I had an issue without the verify=False
with open('image_predictions.tsv',mode='wb') as file:
    file.write(res.content)
predictions = pd.read_csv('image_predictions.tsv',sep='\t') #I used sep = '\t' after re
ading in in a heloful post from Stackoverflow
#https://stackoverflow.com/questions/9652832/how-to-load-a-tsv-file-into-a-pandas-dataf
rame Thanks @huon
```

Tweet meta data (favorites and retweets)

The most challenging was to gather the meta data from `tweet_meta`. By meta data I refer to the retweets and favorites information. The process:

- I had to create a develope account with Twitter
- pip install tweepy and import it to my workspace
- set the api
- loop through all the tweets from `twitter_archive` to access the data and store it on a json file

In [ ]:

```
#Set up of Tweepy's API
consumer_key = 'here_your_own'
consumer_secret = 'here_your_own'
access_token = 'here_your_own'
access_secret = 'here_your_own'
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_secret)

api = tweepy.API(auth, wait_on_rate_limit = True, wait_on_rate_limit_notify = True)
with open('tweet_meta.txt', mode='w') as outfile:
    for tweet_id in twitter_archive['tweet_id']:
        try:
            tweet = api.get_status(tweet_id, tweet_mode ='extended') #Use API to gather
 tweets metadata
            json.dump(tweet._json,outfile) #If we dont use ._json there is an error
            outfile.write('\n') #This is to have each tweet in a new line
            correct.append(tweet_id)
        except Exception as e:
            incorrect.append(tweet_id)
```

Conclusion of Gathering phase

With this section I gathered data from twitter using an API, I requested data from another server using the requests library, and I opened a csv file in my workspace stored in my computer. This is an important first step... but we're not ready to analyze the data yet... for that we still need to go through the next step of the data wrangling process!

In [3]:

```
twitter_archive = pd.read_csv('twitter-archive-enhanced.csv')
predictions = pd.read_csv('image_predictions.tsv',sep='\t')
tweet_meta = pd.read_json('tweet_meta.txt',lines=True)
```

Data Assessment

Sweet! Now we have the three databases: `twitter_archive`, `predictions` and `tweet_meta`. Let's see how clean and tidy they are. Throughout this sections we will be talking a lot about data quality and data tidiness... but what does that mean?

Data Quality

From the instructor notes from Udacity: The four main data quality dimensions are:

- *Completeness*: do we have all of the records that we should? Do we have missing records or not? Are there specific rows, columns, or cells missing?
- *Validity* : we have the records, but they're not valid, i.e., they don't conform to a defined schema. A schema is a defined set of rules for data. These rules can be real-world constraints (e.g. negative height is impossible) and table-specific constraints (e.g. unique key constraints in tables).
- *Accuracy*: inaccurate data is wrong data that is valid. It adheres to the defined schema, but it is still incorrect. Example: a patient's weight that is 5 lbs too heavy because the scale was faulty.
- *Consistency* : inconsistent data is both valid and accurate, but there are multiple correct ways of referring to the same thing. Consistency, i.e., a standard format, in columns that represent the same data across tables and/or within tables is desired.

Data Tidiness

Source (https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html) For a dataset to be considered tidy it needs to meet the following criteria:

- Each variable forms a column.
- Each observation forms a row.
- Each type of observational unit forms a table.

Programmatic and visual assessment

In order to establish the shortcomings of the data, I used programmatic and visual assessment of the data. The functions I used for this assessment are (code example below):

- DatFrame.info()
- DataFrame.head()
- DataFrame.tail()
- DataFrame.sample()
- Dataframe.duplicated().sum()
- DataFrame.nunique()
- DataFrame.column.value_counts()

Examples of code using `twitter_archive`

In [13]:

```
twitter_archive.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2356 entries, 0 to 2355
Data columns (total 17 columns):
tweet_id                      2356 non-null int64
in_reply_to_status_id         78 non-null float64
in_reply_to_user_id           78 non-null float64
timestamp                     2356 non-null object
source                        2356 non-null object
text                          2356 non-null object
retweeted_status_id           181 non-null float64
retweeted_status_user_id      181 non-null float64
retweeted_status_timestamp    181 non-null object
expanded_urls                 2297 non-null object
rating_numerator              2356 non-null int64
rating_denominator            2356 non-null int64
name                          2356 non-null object
doggo                         2356 non-null object
floofer                       2356 non-null object
pupper                        2356 non-null object
puppo                         2356 non-null object
dtypes: float64(4), int64(3), object(10)
memory usage: 313.0+ KB
```

In [6]:

```
twitter_archive.head(1)
```

Out[6]:

	tweet_id	in_reply_to_status_id	in_reply_to_user_id	timestamp	
0	892420643555336193	NaN	NaN	2017-08-01 16:23:56 +0000	<a hr r...

In [7]:

```
twitter_archive.tail(1)
```

Out[7]:

	tweet_id	in_reply_to_status_id	in_reply_to_user_id	timestamp	
2355	666020888022790149	NaN	NaN	2015-11-15 22:32:08 +0000	<a hr r..

In [5]:

```
twitter_archive.sample()
```

Out[5]:

	tweet_id	in_reply_to_status_id	in_reply_to_user_id	timestamp
1929	674042553264685056	NaN	NaN	2015-12-08 01:47:22 +0000

In [8]:

```
twitter_archive.duplicated().sum()
```

Out[8]:

```
0
```

In [9]:

```
twitter_archive.nunique()
```

Out[9]:

```
tweet_id                      2356
in_reply_to_status_id           77
in_reply_to_user_id             31
timestamp                     2356
source                           4
text                          2356
retweeted_status_id            181
retweeted_status_user_id        25
retweeted_status_timestamp     181
expanded_urls                 2218
rating_numerator                40
rating_denominator              18
name                           957
doggo                            2
floofer                          2
pupper                           2
puppo                            2
dtype: int64
```

In [11]:

```
twitter_archive.name.value_counts()[:4]
```

Out[11]:

```
None       745
a           55
Charlie     12
Cooper      11
Name: name, dtype: int64
```

Conclusion of Data Assessment

After looking at the visual and programmatic assessment, I listed observations for the three databases (`twitter_archive`, `predictions`, and `tweet_meta`) for both quality and data tidiness.

Example: `tweet_meta` observations

Data Quality

- id is an int and not a string
- 14 missing values comparing twitter_archive and tweet_meta

Data tidiness

- All columns except id, favorite_count, and retweet_count are irrelevant for this analysis

Data Cleaning

Although I had made many observations in the assessment phase. It was not necessary to clean all of them for my analsyis. In the section of Data Cleaning, I decided how to 'repair' the relevant observations using code.

The observations I prioritized to clean are:

- drop all unecessary columns for `twitter_archive`, `predictions`, `tweet_meta`
- change type of tweet_id from integer to strig for `twitter_archive`, `predictions`, `tweet_meta`
- `tweet_meta` : Change column name from id to tweet_id
- `predictions`: change column names
- `predictions`: drop rows where p1 is not a dog
- `predictions`: lower case p1, p2, p3
- `twitter_archive`: change breed of dog to a single column
- `twitter_archive`: dog stage change from none to NaN
- `twitter_archive`: drop / adapt rows where denominator is not /10
- merge the relevant columns fro the three datasets into one dataframe

Steps:

- Copy dataframes (in case I do something wrong we can always revert to the original :))
- Remove unecessary columns
- Code to clean the observations listed above
- Test code
- Create combined dataframe with the relevant observation

Although this process sounds linear it is actually quite iterative.

Below a summary:

In [ ]:

```
#drop all unecessary columns for `twitter_archive`, `predictions`, `tweet_meta`

twitter_archive_clean = twitter_archive_clean[['tweet_id','text','timestamp','rating_nu
merator','rating_denominator','name','doggo','floofer','pupper','puppo','expanded_urls'
]]
predictions_clean.drop('img_num',axis=1,inplace=True)
meta_data_clean = meta_data_clean[['id','favorite_count','retweet_count']]

#change type of tweet_id from integer to strig for `twitter_archive`, `predictions`, `t
weet_meta`
twitter_archive_clean['tweet_id'] = twitter_archive_clean['tweet_id'].astype(str)
predictions_clean['tweet_id'] = predictions_clean['tweet_id'].astype(str)
meta_data_clean['tweet_id'] = meta_data_clean['tweet_id'].astype(str)

#`predictions`: change column names
predictions_clean.rename(columns={'p1':'prediction_1','p2':'prediction_2','p3':'predict
ion_3'},inplace=True)

#`predictions`: drop rows where p1 is not a dog
predictions_clean = predictions_clean[predictions_clean['p1_dog'] == True]

#`predictions`: lower case p1, p2, p3
predictions = ['prediction_1','prediction_2','prediction_3']
for prediction in predictions:
    predictions_clean[prediction] = predictions_clean[prediction].str.lower()

#- `twitter_archive`: change breed of dog to a single column
twitter_archive_clean = pd.melt(twitter_archive_clean, id_vars = ['tweet_id','timestam
p','text','rating_numerator','rating_denominator','name','expanded_urls'],var_name='dog
_stage',value_vars=['doggo','floofer','pupper','puppo'])

twitter_archive_clean.drop_duplicates(subset='tweet_id',inplace=True)
twitter_archive_clean.shape

#Clean new column dog_stage
twitter_archive_clean.drop('dog_stage',axis=1,inplace=True)
twitter_archive_clean.rename(columns={'value':'dog_stage'},inplace=True)

#`twitter_archive`: dog stage change from none to NaN
twitter_archive_clean['dog_stage'].replace('None',np.NaN,inplace=True)

#Drop columns where denominator is not 10 in twitter_archive
twitter_archive_clean = twitter_archive_clean[twitter_archive_clean.rating_denominator
== 10]

#Change timestamp format to date time
twitter_archive_clean['timestamp'] = pd.to_datetime(twitter_archive_clean.timestamp)

#merge the relevant columns fro the three datasets into one dataframe
df_combined = pd.merge(twitter_archive_clean, predictions_clean,on='tweet_id',how='lef
t')
df_combined = pd.merge(df_combined, meta_data_clean,on='tweet_id',how='left')
```

Testing clean up

In order to see if the clean up was sucessful I used the same functions as in the assessment phase (e.g. .info(), .sample(), etc)

Storing, Analyzing and visualizing the wrangled data

With the new clean versions and the combined data set, I now wanted to save it locally for further analysis and visualize some of my precious data

In [ ]:

```
#Export to csv file
df_combined.to_csv('weratedogs_combined.csv')
meta_data_clean.to_csv('meta_data_clean.csv')
twitter_archive_clean.to_csv('twitter_archive_clean.csv')
predictions_clean.to_csv('predictions_clean.csv')
```

To visualize the data I used pyplot for histograms about retweet and favorite counts and seaborn's fantastic pairplot to visually explore relationships between variables (retweet_count, rating, favorite_count, and dog stage)

Further information

Hope you enjoyed this! If you have any questions on this find me on GitHub (username rrosasl)