Project #5: Wrangle Report -- "We Rate Dogs"

Before beginning any analysis, we need to gather the data that will be needed for the analysis, assess the data visually and programmatically, merge data where appropriate, clean the data, and store the data in a format that can be accessed for analysis.

In [16]:
```
#Import all necessary packages/libraries

import pandas as pd
import numpy as np
import requests
import os
import tweepy
from tweepy import OAuthHandler
import json
import matplotlib as plt
import datetime as dt
```

NOTE: Code is provided for explanatory purposes only here. Run code in wrangle_act to reproduce results.

Gather Data

The objective of the wrangling data project was to gather data from various sources, merge the data, clean the data, analyze the data, and finally present the data.

In the Udacity "We Rate Dogs" data wrangling project, we were first tasked with reading in a cvs data file provided by Udacity.

In [17]:
```
#Load the WeRateDogs Twitter archive data provided by Udacity.
twitter_archive = pd.read_csv('twitter-archive-enhanced.csv')
```

Second, we were tasked with reading in an image file programmatically using the Request library and reading in the images from a website.

In [18]:
```
image_url = ("https://d17h27t6h515a5.cloudfront.net/topher/2017/August/599fd2ad_

response = requests.get(image_url)
with open(os.path.join(os.getcwd(), image_url.split('/')[-1]), mode = 'wb') as f
        file.write(response.content)

#Read in image-prediction text file
image_prediction = pd.read_csv('image-predictions.tsv',sep='\t')
```

Lastly, we were tasked with querying a Twitter API for each tweet in the "We Rate Dogs" Twitter archive and saving it as a JSON text file. Then we were to save each retweet returned as a new line in a text file.

In [19]:
```
# Query Twitter API for each tweet in the Twitter archive and save JSON in a tex

consumer_key = 'HIDDEN'
consumer_secret = 'HIDDEN'
access_token = 'HIDDEN'
```

```
access_secret = 'HIDDEN'

auth = OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_secret)

api = tweepy.API(auth, wait_on_rate_limit=True)
```

In [20]:
```
#Read tweets into dataframe
tweets_list = []

#Use for loop to append each tweet into tweets_list
tweet_file = open('tweet_json.txt', "r")

for line in tweet_file:
    try:
        tweet = json.loads(line)

        tweets_list.append(tweet)
    except:
        continue

tweet_file.close()
```

In [21]:
```
#Create dataframe to store tweets
tweets_df = pd.DataFrame()

#Update column names
tweets_df['tweet_id'] = list(map(lambda tweet: tweet['id'], tweets_list))
tweets_df['retweet_count'] = list(map(lambda tweet: tweet['retweet_count'], twee
tweets_df['favorite_count'] = list(map(lambda tweet: tweet['favorite_count'], tw
```

Assess Data

Once we gathered all the necessary data for our project, we need to access and familiarize ourselves with the data both visually and programmatically.

Visual assessment can uncover data anomalies that will need to be cleaned to be prepped for analysis, such as missing or inconsistent data, data inaccuracy, and tidiness issues.

Programmatic allows us dive even deeper to uncover inaccurate data types and missing values (df.info()), exploring descriptive statistics (df.describe()), and identifying duplicate values (df.nunique()).

Exploring the data allows us to identify issues that need to be cleaned prior to analyzing the data.

Cleaning the Data

For this analysis, I began by merging all three datasets using the image file as the base file since we were only interested in looking ar records with images.

In [22]:
```
twitter_image = image_prediction.merge(twitter_archive, left_on='tweet_id', righ

twitter_image_tweets = twitter_image.merge(tweets_df, left_on='tweet_id', right_
```

Next, and as advised, I made a copy of the merged file prior to cleaning any data so that I could always refer back to the original if needed.

Next, I began to address data quality issues, beginning with the removal of extra characters that were added to the timestamp field.

In [23]:
```
tw_image_clean = twitter_image_tweets
```

In [24]:
```
tw_image_clean.timestamp  = tw_image_clean.timestamp.str.strip('+0000')
```

After removing the extra characters, I converted the timestamp field to a datetime data type.

In [25]:
```
tw_image_clean['timestamp'] = pd.to_datetime(tw_image_clean['timestamp'])
```

Next, I converted all 'rating_denominators' to 10 to be consistent since there were some denominators identified in the data as 0 and 170.

In [26]:
```
tw_image_clean['rating_denominator'] = 10
```

Next, I filtered out all retweets since we were instructed not included these in our final data file.

In [27]:
```
tw_image_clean = tw_image_clean.query('retweeted_status_id == "NaN"')
```

After filtering out retweets, several columns became redundant so I filtered those out as well.

In [28]:
```
tw_image_clean.drop('retweeted_status_id', axis = 1, inplace = True)
tw_image_clean.drop('retweeted_status_user_id', axis = 1, inplace = True)
tw_image_clean.drop('retweeted_status_timestamp', axis = 1, inplace = True)
tw_image_clean.drop('in_reply_to_status_id', axis = 1, inplace = True)
tw_image_clean.drop('in_reply_to_user_id', axis = 1, inplace = True)
```

```
/opt/conda/lib/python3.6/site-packages/pandas/core/frame.py:3697: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stabl
e/indexing.html#indexing-view-versus-copy
  errors=errors)
```

Next, I addressed the next tidiness issue which was the notation of dog state in four separate columns. To address this issue, I merged these four columns into one dog_stage column.

In [29]:
```
tw_image_clean.replace('None', np.nan, inplace=True)
tw_image_clean['dog_stage'] = tw_image_clean[['doggo', 'floofer', 'pupper', 'pup
```

```
/opt/conda/lib/python3.6/site-packages/pandas/core/frame.py:3798: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stabl
e/indexing.html#indexing-view-versus-copy
  method=method)
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:2: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stabl
```

```
e/indexing.html#indexing-view-versus-copy
```

In [30]:
```
tw_image_clean = tw_image_clean.drop(['doggo', 'floofer', 'pupper', 'puppo'], ax
```

This unveiled a few other data quality issues. The first being that the majority of the data for the dog_stage field was missing. The next being a handful of records fell into buckets outside of the four main dog stages. To clean these records, I used visual assessment to review each record individually and assign it to the correct bucket.

In [31]:
```
tw_image_clean[tw_image_clean['dog_stage'] == "doggofloofer"]

tw_image_clean.drop(tw_image_clean.loc[tw_image_clean['tweet_id'] == 85401017255
```

Next, I looked at dog names and again found many missing values. In some cases, the dog name appeared in the text, but this was not consistent. In addition, there was no way to know if several images of the same dog appeared or if different dogs appeared with the same name. At this point, I decided I did not have enough information to make any dependable insights so I updated all names with None values to Nan and moved on. I could have extracted some names from text, but this would not have solved for the issues identified above.

In [32]:
```
tw_image_clean.dog_stage = tw_image_clean['dog_stage'].replace('None', np.NaN)
```

Next, I noticed that text entries where non-dog images had been submitted included the phrase “We only rate dogs” or “I only rate dogs” so I dropped all of these records since these images aligned with non-dog entries.

In [33]:
```
searchfor = ['We only rate dogs.', 'I only rate dogs.']
tw_image_clean = tw_image_clean[~tw_image_clean.text.str.contains('|'.join(searc
```

Next, I removed the html code from the source column as potential data element to explore further.

In [34]:
```
source_dict = {'<a href="http://twitter.com/download/iphone" rel="nofollow">Twit
'<a href="http://vine.co" rel="nofollow">Vine - Make a Scene</a>' : 'Vine - Make
'<a href="http://twitter.com" rel="nofollow">Twitter Web Client</a>' :'Twitter W
'<a href="https://about.twitter.com/products/tweetdeck" rel="nofollow">TweetDeck

def clean_source(tw_image_clean):
    if tw_image_clean['source'] in source_dict.keys():
        sourceab = source_dict[tw_image_clean['source']]
        return sourceab
    else:
        return tw_image_clean['source']

tw_image_clean['source'] = tw_image_clean.apply(clean_source, axis=1)
```

Store Data

Finally, I stored and exported the data for further analysis.

In [ ]:
```

```