wrangle_act

September 25, 2019

```
In [1]: import datetime as dt
        import json
        import numpy as np
        import pandas as pd
        import re
        import requests
        import tweepy
```

Gather

```
In [2]: archive = pd.read_csv("twitter-archive-enhanced.csv")
        archive.set_index("tweet_id", inplace = True)
        archive.head(2)
```

```
Out[2]:                     in_reply_to_status_id  in_reply_to_user_id  \
        tweet_id
        892420643555336193                    NaN                  NaN
        892177421306343426                    NaN                  NaN

                                            timestamp  \
        tweet_id
        892420643555336193  2017-08-01 16:23:56 +0000
        892177421306343426  2017-08-01 00:17:27 +0000

                                                                       source  \
        tweet_id
        892420643555336193  <a href="http://twitter.com/download/iphone" r...
        892177421306343426  <a href="http://twitter.com/download/iphone" r...

                                                                         text  \
        tweet_id
        892420643555336193  This is Phineas. He's a mystical boy. Only eve...
        892177421306343426  This is Tilly. She's just checking pup on you...

                            retweeted_status_id  retweeted_status_user_id  \
        tweet_id
        892420643555336193                  NaN                       NaN
        892177421306343426                  NaN                       NaN
```

```
                   retweeted_status_timestamp  \
tweet_id
892420643555336193                        NaN
892177421306343426                        NaN

                                                        expanded_urls  \
tweet_id
892420643555336193  https://twitter.com/dog_rates/status/892420643...
892177421306343426  https://twitter.com/dog_rates/status/892177421...

                    rating_numerator  rating_denominator     name doggo  \
tweet_id
892420643555336193                13                  10  Phineas  None
892177421306343426                13                  10    Tilly  None

                   floofer pupper puppo
tweet_id
892420643555336193    None   None  None
892177421306343426    None   None  None
```

```
In [3]: tsv_url = "https://d17h27t6h515a5.cloudfront.net/topher/2017/August/599fd2ad_image-predi
        r = requests.get(tsv_url)

        with open(tsv_url.split('/')[-1], mode = 'wb') as file:
            file.write(r.content)

        images = pd.read_csv('image-predictions.tsv', sep = '\t')
        images.head(2)
```

```
Out[3]:             tweet_id                                          jpg_url  \
        0  666020888022790149  https://pbs.twimg.com/media/CT4udn0WwAA0aMy.jpg
        1  666029285002620928  https://pbs.twimg.com/media/CT42GRgUYAA5iDo.jpg

           img_num                      p1   p1_conf  p1_dog                  p2  \
        0        1  Welsh_springer_spaniel  0.465074    True              collie
        1        1                 redbone  0.506826    True  miniature_pinscher

            p2_conf  p2_dog                   p3   p3_conf  p3_dog
        0  0.156665    True    Shetland_sheepdog  0.061428    True
        1  0.074192    True  Rhodesian_ridgeback  0.072010    True
```

```
In [4]: consumer_key = 'IudBZvlwcR2jN5zeo0wA90TvB'
        consumer_secret = 'uGBvxVxZHMXbagMIQuvK1Xy2J2ZJWQIl04dDeVwEThsgYyCkh3'
        access_token = '100709392-z5blkywHLee7cRZD7A00fqc40aCPG9jd6FDcByTW'
        access_token_secret = 'pMPhV9bWAu5PYso3Y3IVD1X12iauxEXLKgiTXYLhrYVNA'

        auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
        auth.set_access_token(access_token, access_token_secret)

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

```
In [ ]: df = []
        exceptions = []
        tweet_id = images['tweet_id']

        for id in tweet_id:
            try:
                page = api.get_status(id)
                favorites = page['favorite_count']
                retweets = page['retweet_count']
                time = pd.to_datetime(page['created_at'])
                df.append({'tweet_id': int(id),
                                'favorites': int(favorites),
                                'retweets': int(retweets)})

            except Exception as e:
                exceptions.append(id)
```

```
In [ ]: exceptions
```

```
In [ ]: exceptions2 = []
        for e in exceptions:
            try:
                page = api.get_status(e)
                favorites = page['favorite_count']
                retweets = page['retweet_count']
                time = pd.to_datetime(page['created_at'])
                df.append({'tweet_id': int(e),
                                'favorites': int(favorites),
                                'retweets': int(retweets)})

            except Exception:
                exceptions2.append(id)
```

```
In [ ]: df = pd.DataFrame(df, columns = ['tweet_id', 'favorites', 'retweets'])
        df.to_csv('tweet_json.txt', encoding = 'utf-8')
```

```
In [ ]: df = pd.read_csv('tweet_json.txt', encoding = 'utf-8',index ='False')
        df.set_index('tweet_id', inplace = True)
        df.tail()
```

```
In [ ]: images.set_index('tweet_id', inplace = True)
        df2 = pd.merge(left=archive, right=images, left_index=True, right_index=True, how='left'
        df2 = pd.merge(left=df2, right=df, left_index=True, right_index=True, how='left')
        df2.to_csv('df2copy.csv', encoding = 'utf-8')
```

```
In [ ]: #Assessing Data
        archive.info()
```

```
In [ ]: archive.name.value_counts()
```

```
In [ ]: archive.rating_denominator.value_counts()
```

```
In [ ]: archive.rating_numerator.value_counts()
```

```
In [ ]: archive.head()
```

```
In [ ]: images.info()
```

```
In [ ]: tweet = pd.read_csv("tweet_json.txt", encoding = 'utf-8',index ='False')
        tweet.info()
```

Quality Several columns have empty values, like in_reply_to_status, in_reply_to_user_id, retweeted_status_id, retweeted_status_user_id, retweeted_status_timestamp. The name column has many entries which do not look like names. The most frequent entry in name column is "a", which is not a name. The numerator and denominator columns have unusual values. The timestamp column is an object. It has to be a datetime object. There are 2075 rows in the images dataframe and 2356 rows in the archive dataframe. In several columns, null values are not treated as null values. Tidiness The dog stages have values as columns, instead of one column filled with their values. We don't need the Unnamed: 0 column from the tweet dataframe. The columns for dog breed predictions can be condensed.

```
In [ ]: df = pd.read_csv("df2copy.csv")
```

```
In [ ]: Clean
```

```
In [ ]: #delete extra column
        del(df['Unnamed: 0'])
```

```
In [ ]: #test
        df.columns
```

```
In [ ]: #convert timestamp to datetime
        df['timestamp'] = pd.to_datetime(df['timestamp'])
```

```
In [ ]: df.info()
```

```
In [ ]: # removing retweets
        df = df[pd.isnull(df['retweeted_status_id'])]
        df.shape[0]
```

```
In [ ]: df.drop(['retweeted_status_id', 'retweeted_status_user_id', 'retweeted_status_timestamp'
```

```
In [ ]: #test
        df.columns
```

In [ ]:

```
# Condensing dog type columns
dog_type = []

x = ['pupper', 'puppo', 'doggo', 'floof']
y = ['pupper', 'puppo', 'doggo', 'floof']

for row in df['text']:
    row = row.lower()
    for word in x:
        if word in str(row):
            dog_type.append(y[x.index(word)])
            break
    else:
        dog_type.append('None')

df['dog_type'] = dog_type
```

In [ ]:

```
df['dog_type'].value_counts()
```

In [ ]:

```
# removing extra columns
df.drop(['doggo', 'floofer', 'pupper', 'puppo'], axis=1, inplace=True)
```

In [ ]:

```
#test
df.columns
```

In [ ]:

```
#Condensing dog breed predictions
breed = []
conf= []

def breed_conf(row):
    if row['p1_dog']:
        breed.append(row['p1'])
        conf.append(row['p1_conf'])
    elif row['p2_dog']:
        breed.append(row['p2'])
        conf.append(row['p2_conf'])
    elif row['p3_dog']:
        breed.append(row['p3'])
        conf.append(row['p3_conf'])
    else:
        breed.append('Unidentifiable')
        conf.append(0)

df.apply(breed_conf, axis = 1)

df['breed'] = breed
df['confidence'] = conf
```

In [ ]:

```
#removing the processed columns
df.drop(['p1', 'p1_conf', 'p1_dog', 'p2', 'p2_conf', 'p2_dog', 'p3', 'p3_conf', 'p3_dog'
In [ ]: #test
        df.head(2)
```

```
In [ ]: #removing useless columns
        df['in_reply_to_status_id'].value_counts()
```

```
In [ ]: df['in_reply_to_user_id'].value_counts()
```

```
In [ ]: df.drop(['in_reply_to_status_id', 'in_reply_to_user_id'], axis=1, inplace=True)
```

```
In [ ]: #test
        df.columns
```

```
In [ ]: # Extract Dog Rates and Dog Count
        rates = []

        #raw_rates = lambda x: rates.append(re.findall(r'(\d+(\.\d+)|(\d+))\/(\d+0)', x, flags=

        df['text'].apply(lambda x: rates.append(re.findall(r'(\d+(\.\d+)|(\d+))\/(\d+0)', x, fla

        rating = []
        dog_count = []

        for item in rates:

            # for tweets with no rating, but a picture, so a dog_count of 1
            if len(item) == 0:
                rating.append('NaN')
                dog_count.append(1)

            # for tweets with single rating and dog_count of 1
            elif len(item) == 1 and item[0][-1] == '10':
                rating.append(float(item[0][0]))
                dog_count.append(1)

            # for multiple ratings
            elif len(item) == 1:
                a = float(item[0][0]) / (float(item[0][-1]) / 10)
                rating.append(a)
                dog_count.append(float(item[0][-1]) / 10)
                # for tweets with more than one rating
            elif len(item) > 1:
                total = 0
                r = []
                for i in range(len(item)):
                    if item[i][-1] == '10': #one tweet has the phrase '50/50' so I'm coding to e
                        r.append(item[i])
                for rate in r:
                    total = total + float(rate[0])
                a = total / len(item)
                rating.append(a)
                dog_count.append(len(item))

            # if any error has occurred
            else:
                rating.append('Not parsed')
                dog_count.append('Not parsed')

        df['rating'] = rating # not need to also add denominator since they are all 10!
        df['dog_count'] = dog_count
        df['rating'].value_counts()
```

```
In [ ]: df.drop(['rating_numerator', 'rating_denominator'], axis=1, inplace=True)
```

```
In [ ]: #test
        df.info()
```

```
In [ ]: df['dog_count'].value_counts()
```

```
In [ ]: # extract names
        df['text_split'] = df['text'].str.split()
```

```
In [ ]: names = []

        # use string starts with method to clean this up

        def extract_names(row):

            # 'named Phineas'
            if 'named' in row['text'] and re.match(r'[A-Z].*', row['text_split'][(row['text_spli
                    names.append(row['text_split'][(row['text_split'].index('named') + 1)]) # 'H
            elif row['text'].startswith('Here we have ') and re.match(r'[A-Z].*', row['text_spli
                    names.append(row['text_split'][3].strip('.').strip(','))

            # 'This is Phineas'
            elif row['text'].startswith('This is ') and re.match(r'[A-Z].*', row['text_split'][2
                    names.append(row['text_split'][2].strip('.').strip(','))

            # 'Say hello to Phineas'
            elif row['text'].startswith('Say hello to ') and re.match(r'[A-Z].*', row['text_spli
                    names.append(row['text_split'][3].strip('.').strip(','))

            # 'Meet Phineas'
            elif row['text'].startswith('Meet ') and re.match(r'[A-Z].*', row['text_split'][1]):
                    names.append(row['text_split'][1].strip('.').strip(','))

            else:
                names.append('Nameless')
        df.apply(extract_names, axis=1)

        df['names'] = names

In [ ]: df['names'].value_counts()

In [ ]: df.drop(['text_split'], axis=1, inplace=True)

In [ ]: df.loc[df['names'] == 'Nameless', 'names'] = None
        df.loc[df['breed'] == 'Unidentifiable', 'breed'] = None
        df.loc[df['dog_type'] == 'None', 'dog_type'] = None
        df.loc[df['rating'] == 0.0, 'rating'] = np.nan
        df.loc[df['confidence'] == 0.0, 'confidence'] = np.nan

In [ ]: #test
        df.info()

In [ ]: #saving the cleaned file
        df.to_csv('twitter_archive_master.csv', encoding = 'utf-8')

In [ ]: Analysis

In [ ]: %matplotlib inline
        #import matplotlib
        import matplotlib.pyplot as plt

In [ ]: df = pd.read_csv('twitter_archive_master.csv')
        df['timestamp'] = pd.to_datetime(df['timestamp'])
        df.set_index('timestamp', inplace=True)

In [ ]: # Retweets, Favorites and Ratings Correlation
        df[['favorites', 'retweets']].plot(style = '.', alpha = 0.4)
        plt.title('Favorites and Retweets with Time')
        plt.xlabel('Date')
        plt.ylabel('Count');

In [ ]: df.plot(y ='rating', ylim=[0,14], style = '.', alpha = 0.4)
        plt.title('Rating with Time')
        plt.xlabel('Date')
        plt.ylabel('Rating');

In [ ]: df[['favorites', 'rating', 'retweets']].corr(method='pearson')
```

So Brant was right, there are more ratings above 10. Still don't know the reason why there are so much high ratings.

So let's see if dogs with higher ratings were getting more favorites and retweets. According to me, if the dogs are getting better they should be getting more favorites and retweets along with the higher rating. There is a strong correlation between favorites and retweets. This means that if the tweet is good in general then there will be more retweets and favorites.

Yet there is no correlation between rating and retweets or rating and favorites. It can be because the dogs are not actually getting better. It can be that 'lower quality' dogs are given funnier captions. In this case, it is the caption that is getting more retweets and favorites, rather than the dog itself.

```
In [ ]: # Dog Stages Stats
        df.boxplot(column='rating', by='dog_type');
```

```
In [ ]: df.groupby('dog_type')['rating'].describe()
```

```
In [ ]: df.reset_index(inplace=True)
        df.groupby('dog_type')['timestamp'].describe()
```

So puppers are getting much lower rates than the other dog types. They have several low outliers which decrease the mean to 10.6.

Floofers are consistently rated above 10. I don't know whether they are really good or the rating just gets higher with time. Maybe we can see if 'floof' is a newer term.

Here we see that 'floof' is not a new term, first seen on January 2016. So we can say that floofer are consistently good dogs.

```
In [ ]: # Most Rated Breeds
        top=df.groupby('breed').filter(lambda x: len(x) >= 20)
        top['breed'].value_counts().plot(kind = 'bar')
        plt.title('The Most Rated Breeds');
```

It's difficult to know why these breeds are the top breeds. It could be because they are commonly owned. Or they could be the easiest to identify by the AI that identified them.

```
In [ ]: top.groupby('breed')['rating'].describe()
```

```
In [ ]: df['rating'].describe()
```

```
In [ ]: df[df['rating'] <= 14]['rating'].describe()
```

Here we have a statistical comparison of the top breeds with all the ratings. Only one of the top breeds has a mean higher than the total population mean. This is because of these two ratings: 420 and 1776.

Excluding outliers bring down the mean to 10.55.

```
In [ ]:
```