

wrangle_act

January 20, 2018

1 Gather

```
In [ ]: import requests
import numpy as np
import pandas as pd
import csv
import tweepy
from pathlib import Path
import json
```

Let's load the files into dataframes for use in the following assessment and cleaning.

```
In [ ]: # fetch the image predictions file if we don't already have it...
my_file = Path("/home/workspace/image-predictions_2.tsv")
if my_file.is_file():
    # request the image predictions file from provided url
    ip_r = requests.get('https://d17h27t6h515a5.cloudfront.net/topher/2017/August/599fd2...')
    # write it to file
    with open('/home/workspace/image-predictions.tsv', 'wb') as f:
        f.write(ip_r.content)

# load the image predictions file into data
image_predictions = pd.read_csv("/home/workspace/image-predictions.tsv", delimiter = "\t")

# load the twitter archive into data
twitter_archive = pd.read_csv("/home/workspace/twitter-archive-enhanced.csv")
```

The code below was used to gather tweet json from the twitter API. It is commented out to prevent it from running again accidentally.

```
In [ ]: # set credentials to the twitter api
consumer_key = '7NlGqlD9Xn4B0veghhn5W4MED'
consumer_secret = 'o8Yz1J1kKPouFWEf5bYp93omy1ULP27ha8MNzpPreC5TytJRv2'
access_token = '917796802363084801-J0uEjWUuz1v1Vbi7b2veIAeC34m6Zvn'
access_secret = 'dXznm8gkBzrgngDK08qQsQuG4ayDm3VsiXZmA0fhZz3oe'
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_secret)
```

```

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

# container for the json received from api
tweet_json_list = []

def fetch_tweets(status_id_list):
    try:
        tweet_list = api.statuses_lookup(status_id_list)
    except tweepy.TweepError as e:
        return 0
    pass
    for i in range(len(tweet_list)):
        tweet_json_list.append(json.dumps(tweet_list[i]._json))

# fetch the tweet data from Twitter API if we don't already have it...
my_file = Path("/home/workspace/tweet_json.txt")
if my_file.is_file() == False:
    # create a list of groups of tweet ids 100 ids in length each
    counter = 0
    tweet_id_groups = []
    tweet_id_group = []
    for index, row in twitter_archive.iterrows():
        if counter < 99:
            tweet_id_group.append(row['tweet_id'])
            counter+=1
        else:
            g = list(tweet_id_group)
            tweet_id_groups.append(g)
            tweet_id_group.clear()
            counter = 0
    # add any lingering ids
    if len(tweet_id_group) > 0:
        g = list(tweet_id_group)
        tweet_id_groups.append(tweet_id_group)
    # write the json to the file
    for index in range(len(tweet_id_groups)):
        fetch_tweets(tweet_id_groups[index])
    # write out the beginning of the file
    with open("/home/workspace/tweet_json.txt", "w") as output:
        output.write("[")
        output.close()
    # write out the rows to a file
    with open("/home/workspace/tweet_json.txt", "a") as output:
        for index, line in enumerate(tweet_json_list):
            output.write(str(line))
            if index < len(tweet_json_list)-1:
                output.write(",")
            output.write("]")

```

```
output.close()
```

Let's load the tweet json into a dataframe

```
In [ ]: # fetch the tweet data from Twitter API if we don't already have it...
my_file = Path("/home/workspace/tweet_json.txt")
if my_file.is_file() == True:
    with open('/home/workspace/tweet_json.txt') as json_data:
        tweet_json_raw = json.load(json_data)
        tweet_json_raw = json.dumps(tweet_json_raw)
        tweet_json = pd.read_json(tweet_json_raw)
```

2 Assess

```
In [ ]: twitter_archive
```

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

```
In [ ]: twitter_archive.describe()
```

```
In [ ]: image_predictions
```

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

```
In [ ]: image_predictions.describe()
```

```
In [ ]: tweet_json
```

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

```
In [ ]: # let's examine the 'variable columns' to see if they can be melted into a single 'dog_t
for index, row in twitter_archive_clean.iterrows():
    count=0
    if row['doggo'] != 'None':
        count+=1
    if row['floofer'] != 'None':
        count+=1
    if row['pupper'] != 'None':
        count+=1
    if row['puppo'] != 'None':
        count+=1
    if count >=2:
        print("tweet id " + str(row['tweet_id']) + " has multiple dog types")
```

Note: There were four tweet ids that had multiple dog types, and upon visual examination it appears that the situation of a single dog having multiple dog types is legitimate. However, examination did reveal one tweet (Tweet ID: 759793422261743616) that contained two dogs. This is essentially two records in one, and will need to be manually removed.

2.1 Quality Issues

2.1.1 tweet archive **table**:

- some records are retweets (not original ratings)
- some records contain multiple dogs (multiple records in one)
- some dog names are incorrect or absent ('None')
- columns doggo, floofer, pupper, puppo would be better represented as boolean
- there are empty values (NaN) in the breed column *this issue was added later in the process, iteratively.
- date/time data occurs as a raw timestamp

2.1.2 image prediction **table**:

- there are tweets with no detected dog
- column p1_conf values have varying number of digits.
- column p2_conf values have varying number of digits.
- column p3_conf values have varying number of digits.

2.2 Tidiness Issues

2.2.1 tweet archive **table**:

- table could be improved by adding a breed column
 - there are unused columns
 - data in tweet archive and image predictions
 - column retweet_count in table tweet_json is isolated from the other datasets
-

3 Clean

```
In [ ]: twitter_archive_clean = twitter_archive.copy()
        image_predictions_clean = image_predictions.copy()
        tweet_json_clean = tweet_json.copy()
```

Here we summarize the cleaning tasks that will be carried out below:

3.1 Quality Issues

3.1.1 tweet archive **table**:

- remove records that are retweets
- remove records that contain incorrect or absent dog names
- manually remove records containing multiple dogs (tweet_id = 759793422261743616)
- re-format column values for doggo, floofer, pupper, puppo to boolean (true/false)
- remove empty breeds (NaN)
- convert raw timestamp into formatted date/time

3.1.2 image prediction **table**:

- remove tweets with no detected dog
- round column p1_conf values to nearest hundredth
- round column p2_conf values to nearest hundredth
- round column p3_conf values to nearest hundredth

3.2 Tidiness Issues

3.2.1 tweet archive **table**:

- create breed column from image predictions table breed values p1 p2 p3.
- remove unused columns
- add column retweet_count from table tweet_json to tweet_archive table
- inner join the tables tweet_archive and image_predictions

tweet_archive remove records that are retweets

Define Identify tweets that have a value in their retweeted_status_id column, and drop it from the dataset.

Code

```
In [ ]: for index, row in twitter_archive_clean.iterrows():
        if not np.isnan(row['retweeted_status_id']):
            twitter_archive_clean.drop(index, inplace=True)

        for index, row in twitter_archive_clean.iterrows():
            if not np.isnan(row['retweeted_status_user_id']):
                twitter_archive_clean.drop(index, inplace=True)

        for index, row in twitter_archive_clean.iterrows():
            if not np.isnan(row['retweeted_status_timestamp']):
                twitter_archive_clean.drop(index, inplace=True)
```

Test

```
In [ ]: for index, row in twitter_archive_clean.iterrows():
        if not np.isnan(row['retweeted_status_id']):
            print("dataset still contains retweets")

        for index, row in twitter_archive_clean.iterrows():
            if not np.isnan(row['retweeted_status_user_id']):
                print("dataset still contains retweets")

        for index, row in twitter_archive_clean.iterrows():
            if not np.isnan(row['retweeted_status_timestamp']):
                print("dataset still contains retweets")
```

tweet archive remove records that contain incorrect or absent dog names

Define Identify and remove tweets that have a name value of 'None' or a value that is lowercase (lowercase values are often mistakenly processed non-name words like 'a', 'an', 'the', etc).

Code

```
In [ ]: for index, row in twitter_archive_clean.iterrows():
        if row['name'] == 'None' or row['name'].islower():
            twitter_archive_clean.drop(index, inplace=True)
```

Test

```
In [ ]: for index, row in twitter_archive_clean.iterrows():
        if row['name'] == 'None' or row['name'].islower():
            print("dataset still contains incorrect names")
```

tweet archive manually remove records containing multiple dogs (tweet_id = 759793422261743616)

Define Remove the data point with tweet_id of 759793422261743616, since it contains two dogs in a single data point.

Code

```
In [ ]: twitter_archive_clean = twitter_archive_clean[twitter_archive_clean.tweet_id != 759793422261743616]
```

Test

```
In [ ]: for index, row in twitter_archive_clean.iterrows():
        if row['tweet_id'] == 759793422261743616:
            print("dataset still contains data point where tweet_id = 759793422261743616")
```

tweet archive re-format column values for doggo, floofer, pupper, puppo to boolean (true/false)

Define Iterate over each of the columns doggo, floofer, pupper, puppo and if the value is 'None' set to a boolean value of False, otherwise set to True.

Code

```
In [ ]: for index, row in twitter_archive_clean.iterrows():
        if row['doggo'] == 'None':
            twitter_archive_clean.set_value(index, 'doggo', False)
        else:
            twitter_archive_clean.set_value(index, 'doggo', True)
        if row['floofer'] == 'None':
            twitter_archive_clean.set_value(index, 'floofer', False)
        else:
            twitter_archive_clean.set_value(index, 'floofer', True)
        if row['pupper'] == 'None':
            twitter_archive_clean.set_value(index, 'pupper', False)
        else:
            twitter_archive_clean.set_value(index, 'pupper', True)
        if row['puppo'] == 'None':
            twitter_archive_clean.set_value(index, 'puppo', False)
        else:
            twitter_archive_clean.set_value(index, 'puppo', True)
```

Test

```
In [ ]: for index, row in twitter_archive_clean.iterrows():
        if row['doggo'] != True and row['doggo'] != False:
            print("tweet " + str(row['tweet_id']) + " has an incorrect doggo value")
        if row['floofer'] != True and row['floofer'] != False:
            print("tweet " + str(row['tweet_id']) + " has an incorrect floofer value")
        if row['pupper'] != True and row['pupper'] != False:
            print("tweet " + str(row['tweet_id']) + " has an incorrect pupper value")
        if row['puppo'] != True and row['puppo'] != False:
            print("tweet " + str(row['tweet_id']) + " has an incorrect puppo value")
```

image predictions - remove tweets with no detected dog

Define Remove all data points in which all three detection columns (p1_dog, p2_dog, p3_dog) are False.

Code

```
In [ ]: for index, row in image_predictions_clean.iterrows():
        if row['p1_dog'] == False and row['p2_dog'] == False and row['p3_dog'] == False:
            image_predictions_clean.drop(index, inplace=True)
```

Test

```
In [ ]: for index, row in image_predictions_clean.iterrows():
        if row['p1_dog'] == False and row['p2_dog'] == False and row['p3_dog'] == False:
            print("tweet " + str(row['tweet_id']) + " failed to detect a dog")
```

image_predictions - round columns p1_conf, p2_conf, p3_conf values to nearest hundredth

Define Round values in column p1_conf, p2_conf, p3_conf to the nearest hundredth.

Code

```
In [ ]: decimal_places = 4
image_predictions_clean['p1_conf'] = image_predictions_clean['p1_conf'].apply(lambda x:
image_predictions_clean['p2_conf'] = image_predictions_clean['p2_conf'].apply(lambda x:
image_predictions_clean['p3_conf'] = image_predictions_clean['p3_conf'].apply(lambda x:
```

Test

```
In [ ]: import decimal
for index, row in image_predictions_clean.iterrows():
    e1 = abs(decimal.Decimal(str(row['p1_conf']))).as_tuple().exponent
    e2 = abs(decimal.Decimal(str(row['p2_conf']))).as_tuple().exponent
    e3 = abs(decimal.Decimal(str(row['p3_conf']))).as_tuple().exponent
    if(e1 > 2):
        print("tweet " + str(row['tweet_id']) + " has a p1_conf value that is not rounded")
    if(e2 > 2):
        print("tweet " + str(row['tweet_id']) + " has a p2_conf value that is not rounded")
    if(e3 > 3):
        print("tweet " + str(row['tweet_id']) + " has a p3_conf value that is not rounded")
```

tweet_archive - create breed column from image predictions table breed values p1 p2 p3.

Define Create breed column from image predictions table p1 p2 p3 taking the positive dog detection of the highest confidence value. First evaluate p1 and if it is false, then evaluate p2 and so on.

Code

```
In [ ]: for index, row in twitter_archive_clean.iterrows():
    id_df = image_predictions_clean.loc[image_predictions_clean['tweet_id'] == row['tweet_id']]
    if id_df.empty == False:
        p1 = id_df['p1'].values[0]
        p2 = id_df['p2'].values[0]
        p3 = id_df['p3'].values[0]
        p1_is_dog = id_df['p1_dog'].values[0]
        p2_is_dog = id_df['p2_dog'].values[0]
        p3_is_dog = id_df['p3_dog'].values[0]
        if p1_is_dog == True:
            twitter_archive_clean.set_value(index, 'breed', p1)
```



```

elif p2_is_dog == True:
    twitter_archive_clean.set_value(index, 'breed', p2)
elif p3_is_dog == True:
    twitter_archive_clean.set_value(index, 'breed', p3)

```

Test Let's see if the field was created.

```
In [ ]: twitter_archive_clean
```

We can see that the breed column was created, but there were some data points that still don't have a breed (NaN) so these should be removed. Let's add another quality task to remove empty (NaN) breeds.

tweet archive - remove empty breeds (NaN)

Define Identify and remove tweets an empty value (NaN) in the breed column.

Code

```
In [ ]: for index, row in twitter_archive_clean.iterrows():
        if row['breed'] == 'NaN':
            twitter_archive_clean.drop(index, inplace=True)

```

Test

```
In [ ]: for index, row in twitter_archive_clean.iterrows():
        if row['breed'] == 'NaN':
            print('tweet ' + str(row['tweet_id']) + " has an empty breed")

```

let's also do a visual inspection to be sure...

```
In [ ]: twitter_archive_clean
```

tweet archive - remove unused columns

Define Remove all columns that are not useful to our data analysis purposes. Specifically, let's get rid of in_reply_to_status_id, in_reply_to_user_id, source, retweeted_status_id, retweeted_status_user_id, retweeted_status_timestamp, expanded_urls.

Code

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

```

Test

```
In [ ]: twitter_archive_clean
```

tweet archive - add column retweet_count from table tweet_json

Define Add the column retweet_count from table tweet_json to the table tweet_archive.

Code

```
In [ ]: # add the column
for index, row in twitter_archive_clean.iterrows():
    tj_df = tweet_json_clean.loc[tweet_json_clean['id_str'] == row['tweet_id']]
    if tj_df.empty == False:
        rtc = tj_df['retweet_count'].values[0]
        if np.isnan(rtc):
            rtc = 0
        twitter_archive_clean.set_value(index, 'retweet_count', rtc)

In [ ]: # clean up bad NaN values
for index, row in twitter_archive_clean.iterrows():
    value = row['retweet_count']
    if np.isnan(value):
        twitter_archive_clean.set_value(index, 'retweet_count', 0)

In [ ]: # change
for index, row in twitter_archive_clean.iterrows():
    value = row['retweet_count']
    val_int = int(value)
    if type(value) != int:
        twitter_archive_clean.set_value(index, 'retweet_count', val_int)

In [ ]: # convert to integer data type
twitter_archive_clean["retweet_count"] = twitter_archive_clean['retweet_count'].astype(int)
```

Test Let's see if the new column was created

```
In [ ]: twitter_archive_clean
```

tweet archive - inner join the tables tweet_archive and image_predictions

Define Merge the data in the tables tweet_archive and image_predictions using an inner join.

Code

```
In [ ]: twitter_archive_master = pd.merge(twitter_archive_clean, image_predictions_clean, on=['tw
```

Test

```
In [ ]: twitter_archive_master
```

4 Analysis

Now let's do some visual analysis to see what we can find out about this dataset. Particularly, we're interested in which breeds are most common in this dataset. Let's start by breaking down the counts of the categorical variable breed.

```
In [ ]: # get the total count of all datapoints
        total_datapoint_count = len(twitter_archive_clean)

        # get counts of the categorial variable 'breed'
        breed_df = pd.DataFrame(twitter_archive_clean['breed'].value_counts())
        breed_count = len(breed_df)
        subtract = breed_count - 5

        print(str(breed_count) + " breeds")
```

```
In [ ]: # let's take a look at some of these breeds
        print(breed_df)
```

```
In [ ]: breed_df.drop(breed_df.tail(subtract).index,inplace=True)

        name_list = []
        values_list = []
        named_datapoint_count = 0
        for index, row in breed_df.iterrows():
            name_list.append(index)
            values_list.append(row['breed'])
            named_datapoint_count+=row['breed']

        other_count = total_datapoint_count - named_datapoint_count

        # add counts for other breeds
        name_list.append('Other')
        values_list.append(other_count)
```

4.0.1 Visualization

```
In [ ]: import matplotlib.pyplot as plt

        # Pie chart, where the slices will be ordered and plotted counter-clockwise:
        labels = name_list
        sizes = values_list

        fig1, ax1 = plt.subplots()
```

```
ax1.pie(sizes, labels=labels, autopct='%1.1f%%', shadow=True, startangle=90)
ax1.axis('equal')

plt.show()
```

```
In [ ]: print(values_list)
```

4.0.2 Insights

- There are 110 different breeds in the dataset.
- The 5 most frequent breeds are Golden Retriever, Labrador Retriever, Pembroke, Chihuahua and Pug.
- The most frequent breed is Golden Retriever (7.5%).

Let's store the cleaned dataframe as a new file.

```
In [ ]: twitter_archive_master.to_csv("/home/workspace/twitter_archive_master.csv", encoding='utf-8')
```

5 Resources

- <https://stackoverflow.com/questions/16476924/how-to-iterate-over-rows-in-a-dataframe-in-pandas> (Dataframe row iteration code example)
- <https://stackoverflow.com/questions/18039057/python-pandas-error-tokenizing-data> (Dataframe csv reader code example)
- <https://stackoverflow.com/questions/6189956/easy-way-of-finding-decimal-places> (Testing decimal points)