I. INTRODUCTION

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 datasets include three files:

- Enhanced Twitter Archive
- Image Predictions File
- Data extraction from Twitter API

II. GATHER DATA

1. Import necessary libraries

```
import numpy as np
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import requests
import tweepy
import json
from timeit import default_timer as timer
from tweepy import OAuthHandler
```

```
pip install tweepy
```

```
import tweepy
from tweepy import OAuthHandler
```

2. Read datasets

Archive tweets

```
#READ TWITTER ARCHIVE FILE
df = pd.read_csv('twitter-archive-enhanced.csv')
```

```
df.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
```

Image Predictions File

```
#Read image file
Image = requests.get('https://d17h27t6h515a5.cloudfront.net/topher/2017/August/599fd2ad_image-predictions/image-predictions.tsv')

with open('image-predictions.tsv', mode='wb') as file:
    file.write(Image.content)
```

```
#Read image file
image_df = pd.read_csv('image-predictions.tsv', sep='\t')
```

```
image_df.head()
```

	tweet_id	jpg_url	img_num	p1	p1_conf	p1_dog	p2	p2_conf	p2_d
0	666020888022790149	https://pbs.twimg.com/media/CT4udn0WwAA0aMy.jpg	1	Welsh_springer_spaniel	0.465074	True	collie	0.156665	Tr
1	666029285002620928	https://pbs.twimg.com/media/CT42GRgUYAA5iDo.jpg	1	redbone	0.506826	True	miniature_pinscher	0.074192	Tr
2	666033412701032449	https://pbs.twimg.com/media/CT4521TWwAEvMyu.jpg	1	German_shepherd	0.596461	True	malinois	0.138584	Tr
3	666044226329800704	https://pbs.twimg.com/media/CT5Dr8HUEAA-lEu.jpg	1	Rhodesian_ridgeback	0.408143	True	redbone	0.360687	Tr
4	666049248165822465	https://pbs.twimg.com/media/CT5IQmsXIAAKY4A.jpg	1	miniature_pinscher	0.560311	True	Rottweiler	0.243682	Tr

Definition of the attributes:

- p1 is the algorithm's #1 prediction for the image in the tweet
- p1_conf is how confident the algorithm is in its #1 prediction
- p1_dog is whether or not the #1 prediction is a breed of dog
- p2 is the algorithm's second most likely prediction
- p2_conf is how confident the algorithm is in its #2 prediction
- p2_dog is whether or not the #2 prediction is a breed of dog etc

Data extraction from Twitter API

```
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_secret)
api = tweepy.API(auth, wait_on_rate_limit=True)
```

```
# Tweet IDs for which to gather additional data via Twitter's API
tweet_ids = df.tweet_id.values
len(tweet_ids)
```

```
2356
```

```
count = 0
fails_dict = {}
start = timer()
# Save each tweet's returned JSON as a new line in a .txt file
with open('tweet_json.txt', 'w') as outfile:
    # This loop will likely take 20-30 minutes to run because of Twitter's rate limit
    for tweet_id in tweet_ids:
        count += 1
        print(str(count) + ": " + str(tweet_id))
        try:
            tweet = api.get_status(tweet_id, tweet_mode='extended')
            print("Success")
            json.dump(tweet._json, outfile)
            outfile.write('\n')
        except tweepy.TweepError as e:
            print("Fail")
            fails_dict[tweet_id] = e
            pass
end = timer()
print(end - start)
print(fails_dict)
```

```
-------
2354: 666033412701032449
Success
2355: 666029285002620928
Success
2356: 666020888022790149
Success
1982.4432694000025
{888202515573088257: TweepError([{'code': 144, 'message': 'No status found with that ID.'}]), 873697596434513921: TweepError
([{'code': 144, 'message': 'No status found with that ID.'}]), 872668790621863937: TweepError([{'code': 144, 'message': 'No
status found with that ID.'}]), 872261713294495745: TweepError([{'code': 144, 'message': 'No status found with that ID.'}]),
869988702071779329: TweepError([{'code': 144, 'message': 'No status found with that ID.'}]), 866816280283807744: TweepError
([{'code': 144, 'message': 'No status found with that ID.'}]), 861769973181624320: TweepError([{'code': 144, 'message': 'No
status found with that ID.'}]), 856602993587888130: TweepError([{'code': 144, 'message': 'No status found with that ID.'}]),
851953902622658560: TweepError([{'code': 144, 'message': 'No status found with that ID.'}]), 845459076796616705: TweepError
([{'code': 144, 'message': 'No status found with that ID.'}]), 844704788403113984: TweepError([{'code': 144, 'message': 'No
status found with that ID.'}]), 842892208864923648: TweepError([{'code': 144, 'message': 'No status found with that ID.'}]),
837366284874571778: TweepError([{'code': 144, 'message': 'No status found with that ID.'}]), 837012587749474308: TweepError
([{'code': 144, 'message': 'No status found with that ID.'}]), 829374341691346946: TweepError([{'code': 144, 'message': 'No
status found with that ID.'}]), 827228250799742977: TweepError([{'code': 144, 'message': 'No status found with that ID.'}]),
```

```
df_tweet_json = pd.DataFrame(columns=['tweet_id', 'retweet_count', 'favorite_count'])
with open('tweet-json.txt') as data_file:
    for line in data_file:
        tweet = json.loads(line)
        tweet_id = tweet['id_str']
        retweet_count = tweet['retweet_count']
        favorite_count = tweet['favorite_count']
        df_tweet_json = df_tweet_json.append(pd.DataFrame([[tweet_id, retweet_count, favorite_count]],
        columns=['tweet_id', 'retweet_count', 'favorite_count']))
        df_tweet_json = df_tweet_json.reset_index(drop=True)
```

```
tweet_count = df_tweet_json[['tweet_id', 'retweet_count', 'favorite_count']]
tweet_count
```

	tweet_id	retweet_count	favorite_count
0	892420643555336193	8853	39467
1	892177421306343426	6514	33819
2	891815181378084864	4328	25461
3	891689557279858688	8964	42908
4	891327558926688256	9774	41048
...	...	...	...
2349	666049248165822465	41	111
2350	666044226329800704	147	311
2351	666033412701032449	47	128
2352	666029285002620928	48	132
2353	666020888022790149	532	2535

2354 rows × 3 columns

Data extraction from Twitter API provides data of retweet count and favorite count.

III. ASSESSING

```
#Assesing Archive Tweet file
df.head()
```

	tweet_id	in_reply_to_status_id	in_reply_to_user_id	timestamp	source	text	retweeted_status_id	retweeted
0	892420643555336193	NaN	NaN	2017-08-01 16:23:56 +0000	<a href="http://twitter.com/download/iphone" r...	This is Phineas. He's a mystical boy. Only eve...	NaN	
1	892177421306343426	NaN	NaN	2017-08-01 00:17:27 +0000	<a href="http://twitter.com/download/iphone" r...	This is Tilly. She's just checking pup on you....	NaN	
2	891815181378084864	NaN	NaN	2017-07-31 00:18:03 +0000	<a href="http://twitter.com/download/iphone" r...	This is Archie. He is a rare Norwegian Pouncin...	NaN	
3	891689557279858688	NaN	NaN	2017-07-30 15:58:51 +0000	<a href="http://twitter.com/download/iphone" r...	This is Darla. She commenced a snooze mid meal...	NaN	
4	891327558926688256	NaN	NaN	2017-07-29 16:00:24 +0000	<a href="http://twitter.com/download/iphone" r...	This is Franklin. He would like you to stop ca...	NaN	

```
df.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
```

We can see quality issues in Archive tweet data as picture above including:

- Data is missing in the some columns: in_reply_to_status_id, in_reply_to_user_id, retweeted_status_id, retweeted_status_user_id, retweeted_status_timestamp, expanded_urls
- Wrong format. Source columns have HTML tags
- Duplicated data.This dataset includes retweets, which means there is duplicated data
- Wrong data type. Timestamp and retweeted_status_timestamp is an object instead of datetime

```
df.describe()
```

	tweet_id	in_reply_to_status_id	in_reply_to_user_id	retweeted_status_id	retweeted_status_user_id	rating_numerator	rating_denominator
count	2.356000e+03	7.800000e+01	7.800000e+01	1.810000e+02	1.810000e+02	2356.000000	2356.000000
mean	7.427716e+17	7.455079e+17	2.014171e+16	7.720400e+17	1.241698e+16	13.126486	10.455433
std	6.856705e+16	7.582492e+16	1.252797e+17	6.236928e+16	9.599254e+16	45.876648	6.745237
min	6.660209e+17	6.658147e+17	1.185634e+07	6.661041e+17	7.832140e+05	0.000000	0.000000
25%	6.783989e+17	6.757419e+17	3.086374e+08	7.186315e+17	4.196984e+09	10.000000	10.000000
50%	7.196279e+17	7.038708e+17	4.196984e+09	7.804657e+17	4.196984e+09	11.000000	10.000000
75%	7.993373e+17	8.257804e+17	4.196984e+09	8.203146e+17	4.196984e+09	12.000000	10.000000
max	8.924206e+17	8.862664e+17	8.405479e+17	8.874740e+17	7.874618e+17	1776.000000	170.000000

- Invalid Range: rating_numerator contains max value 1776 and rating_denominator contains max value of 17 rating_denominator is out of standard value of 10 at some places which can be Inaccurate data.

```
df[df.name.str.islower()].name.value_counts()
```

```
a               55
the              8
an               7
very             5
quite            4
one              4
just             4
getting          2
actually         2
mad              2
not              2
light            1
such             1
his              1
all              1
by               1
unacceptable     1
incredibly       1
infuriating      1
space            1
life             1
my               1
officially       1
this             1
old              1
Name: name, dtype: int64
```

- Some of dogs have unnormal name such as 'None', or 'a', or 'an.' or 'O' or 'by' and some more lower case words as names

```
#Assessing tweet Json file
df_tweet_json.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2354 entries, 0 to 2353
Data columns (total 3 columns):
tweet_id          2354 non-null object
retweet_count     2354 non-null object
favorite_count    2354 non-null object
dtypes: object(3)
memory usage: 55.3+ KB
```

- Tweet ID is object instead of int64

```
#Assessing Image file
image_df.head()
```

jpg_url	img_num	p1	p1_conf	p1_dog	p2	p2_conf	p2_dog	p3	p3_conf	p3_dog
ɔm/media/CT4udn0WwAA0aMy.jpg	1	Welsh_springer_spaniel	0.465074	True	collie	0.156665	True	Shetland_sheepdog	0.061428	True
:om/media/CT42GRgUYAA5iDo.jpg	1	redbone	0.506826	True	miniature_pinscher	0.074192	True	Rhodesian_ridgeback	0.072010	True
ɔm/media/CT4521TWwAEvMyu.jpg	1	German_shepherd	0.596461	True	malinois	0.138584	True	bloodhound	0.116197	True
com/media/CT5Dr8HUEAA-lEu.jpg	1	Rhodesian_ridgeback	0.408143	True	redbone	0.360687	True	miniature_pinscher	0.222752	True
:om/media/CT5IQmsXIAAKY4A.jpg	1	miniature_pinscher	0.560311	True	Rottweiler	0.243682	True	Doberman	0.154629	True

- Inconsistent and wrong format of typing. Some are uppercase, some are lower case. Use "_" instead of " "

Summary - Quality Issues

Archive tweet file

1. Data is missing in the some columns: in_reply_to_status_id, in_reply_to_user_id, retweeted_status_id, etweeted_status_user_id, retweeted_status_timestamp, expanded_urls.
2. Duplicated data.This dataset includes retweets, which means there is duplicated data
3. Wrong format. Source columns have HTML tags
4. Wrong data type. Timestamp and retweeted_status_timestamp is an object instead of datetime
5. Invalid Range: rating_numerator contains max value 1776 and rating_denominator contains max value of 17 rating_denominator is out of standard value of 10
6. Some of dogs have unnormal name such as 'None', or 'a', or 'an.' or 'O' or 'by' and some more lower case words as names

JSON file:

7. Tweet ID is object instead of int64

Image file

8. Inconsistent and wrong format of typing. Some are uppercase, some are lower case. Use "_" instead of " ".

Summary - Tidiness Issues

df

1. The variable for the dog's stage (dogoo, floofer, pupper, puppo) is spread in different columns

image_df

2. This data set is part of the same observational unit as the data in the archive_df

df_tweet_json

3. This data set is also part of the same observational unit as the data in the archive_df

IV. CLEAN DATA

Before cleaning data, the student would like to make a copy of dataset first.

```
#Make a copy before cleaning

clean_df= df.copy()
clean_image_df = image_df.copy()
clean_tweet_json = df_tweet_json.copy()
```

Define- Code - Test

1. Remove unessentual attributes: in_reply_to_status_id, in_reply_to_user_id, retweeted_status_id, retweeted_status_user_id, and retweeted_status_timestamp

2. Removing HTML tags from source column

3. Delete retweets

4. Change the timestamp to correct datetime format

5. Dog ratings get standardized for denom of 10

6. Replace 'a', 'an', 'the', 'None' and other lower case words with NaN in name column

7. Convert the tweet_id in tweet_counts_clean into int64 type for merging

8. Create one column for the various dog types: doggo, floofer, pupper, puppo, 'doggo, puppo', 'doggo, pupper', 'doggo, floofer'

9 & 10. Merge the copied df_clean, image_df_clean, and tweet_json_clean dataframes

Result

```
clean_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2237 entries, 0 to 2355
Data columns (total 12 columns):
tweet_id              2237 non-null int64
timestamp             2237 non-null datetime64[ns, UTC]
source                2237 non-null object
text                  2237 non-null object
expanded_urls         2182 non-null object
rating_numerator      2237 non-null float64
rating_denominator    2237 non-null int64
name                  1427 non-null object
doggo                 2237 non-null object
floofer               2237 non-null object
pupper                2237 non-null object
puppo                 2237 non-null object
dtypes: datetime64[ns, UTC](1), float64(1), int64(2), object(8)
memory usage: 227.2+ KB
```

Unnecessary columns are removed. Variables (timestamp) are formatted in correct data type.

```
# Check the distribution of source values
clean_df.source.value_counts()
```

```
<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>     2221
<a href="http://vine.co" rel="nofollow">Vine - Make a Scene</a>                          91
<a href="http://twitter.com" rel="nofollow">Twitter Web Client</a>                       33
<a href="https://about.twitter.com/products/tweetdeck" rel="nofollow">TweetDeck</a>      11
Name: source, dtype: int64
```

```
ce = clean_df.source.replace('<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>', 'Twitter for i
ce = clean_df.source.replace('<a href="http://vine.co" rel="nofollow">Vine - Make a Scene</a>', 'Vine - Make a Scene')
ce = clean_df.source.replace('<a href="http://twitter.com" rel="nofollow">Twitter Web Client</a>', 'Twitter Web Client')
ce = clean_df.source.replace('<a href="https://about.twitter.com/products/tweetdeck" rel="nofollow">TweetDeck</a>', 'TweetDeck')
```

```
#TEST
clean_df.source.value_counts()
```

```
Twitter for iPhone     2221
Vine - Make a Scene      91
Twitter Web Client       33
TweetDeck                11
Name: source, dtype: int64
```

HTML tags are removed from source column,

```
: # 5. Check for invalid names
  clean_df[clean_df.name.str.islower() == True].name.value_counts()
```

```
a               55
the              8
an               6
very             5
one              4
quite            3
just             3
not              2
getting          2
actually         2
by               1
this             1
officially       1
life             1
old              1
his              1
such             1
light            1
all              1
unacceptable     1
incredibly       1
infuriating      1
space            1
my               1
mad              1
Name: name, dtype: int64
```

```
: # Replace the invalid names with a string 'None'
  for item in clean_df.name:
      if item.islower() == True:
          clean_df.name = clean_df.name.replace(item, 'None')
```

```
: # Replace 'None' string in the name column with NaN value
  clean_df.name = clean_df.name.replace('None', np.nan)
```

```
: clean_df[clean_df.name.str.islower() == True].name.value_counts()
```

```
Series([], Name: name, dtype: int64)
```

```
: sum(clean_df.name.str.islower() == True)
```

```
0
```

```
: sum(clean_df.name.isnull())
```

```
815
```

Invalid names are replaced by NaN value.

```
#test
clean_df['rating_numerator'].value_counts()
```

```
10.00    447
11.00    446
13.00    327
9.00     157
8.00     100
7.00      54
14.00     50
5.00      36
6.00      33
3.00      19
4.00      15
2.00      11
1.00       7
0.00       2
15.00      2
11.27      1
9.50       1
17.00      1
13.50      1
11.26      1
```

```
clean_df['rating_denominator'].value_counts()
```

```
10    2237
Name: rating_denominator, dtype: int64
```

Rating numerator is converted into standard of 10 and 20.

```
#7. Format name. Replace the instance of _ with ` and change the values to upper case inp1, p2 and p3
clean_image_df.p1 = clean_image_df.p1.str.replace('_', ' ').str.capitalize()
clean_image_df.p2 = clean_image_df.p2.str.replace('_', ' ').str.capitalize()
clean_image_df.p3 = clean_image_df.p3.str.replace('_', ' ').str.capitalize()
```

```
#Test
clean_image_df[['p1', 'p2', 'p3']].sample(20)
```

	p1	p2	p3
837	Boxer	Bull mastiff	Saint bernard
1044	Great pyrenees	Kuvasz	Irish wolfhound
245	Minivan	Beach wagon	Car wheel
1664	Bath towel	Pillow	Great dane
2058	Golden retriever	Tibetan mastiff	Labrador retriever
1238	Dingo	Timber wolf	Ibizan hound
2074	Orange	Bagel	Banana
1539	Dishwasher	Golden retriever	Chow
1242	Coil	Dugong	Rain barrel
1167	Pembroke	Seat belt	Cardigan
166	Jellyfish	Coral reef	Goldfish
121	Labrador retriever	Golden retriever	Chesapeake bay retriever
1804	Afghan hound	Borzoi	Doormat
260	Lacewing	Sulphur butterfly	Leafhopper
235	Seat belt	Toy terrier	Beagle
901	Chihuahua	Schipperke	Pug
53	Hyena	African hunting dog	Coyote

Unnecessary instances are removed from the predicted names.

```
# Combine the columns into one column
clean_df['dog_stage'] = clean_df['doggo'] + clean_df['floofer'] + clean_df['pupper'] + clean_df['puppo']
```

```
# Drop unnecessary columns
clean_df = clean_df.drop(['doggo', 'floofer', 'pupper', 'puppo'], axis=1)
```

```
#Test
clean_df.dog_stage.value_counts()
```

```
                1882
pupper           231
doggo             79
puppo             24
doggopupper       10
floofer            9
doggopuppo         1
doggofloofer       1
Name: dog_stage, dtype: int64
```

```
clean_df.loc[clean_df.dog_stage == 'doggopupper', 'dog_stage'] = 'doggo, pupper'
clean_df.loc[clean_df.dog_stage == 'doggopuppo', 'dog_stage'] = 'doggo, puppo'
clean_df.loc[clean_df.dog_stage == 'doggofloofer', 'dog_stage'] = 'doggo, floofer'
```

```
#test
clean_df.dog_stage.value_counts()
```

```
                1882
pupper           231
doggo             79
puppo             24
doggo, pupper     10
floofer            9
doggo, floofer     1
doggo, puppo       1
Name: dog_stage, dtype: int64
```

Dog stages are converted into one column.

```
#10 & 11 Merge the copied df_clean, image_df_clean, and tweet_json_clean dataframes
from functools import reduce
data = [clean_df, clean_image_df, clean_tweet_json]
data_df = reduce(lambda left, right:  pd.merge(left, right,on = 'tweet_id'), data)
```

Data is merged into one final dataset for better analysis, inconsistency and transparency.

```
#test
data_df.describe()
```

	tweet_id	rating_numerator	rating_denominator	img_num	p1_conf	p2_conf	p3_conf
count	1.969000e+03	1969.000000	1969.0	1969.000000	1969.000000	1969.000000	1.969000e+03
mean	7.371736e+17	10.554611	10.0	1.202133	0.593177	0.134028	6.050621e-02
std	6.798196e+16	2.191284	0.0	0.559267	0.272044	0.099926	5.092227e-02
min	6.660209e+17	0.000000	10.0	1.000000	0.044333	0.000010	2.160900e-07
25%	6.758981e+17	10.000000	10.0	1.000000	0.362835	0.054322	1.648340e-02
50%	7.095570e+17	11.000000	10.0	1.000000	0.588230	0.117566	4.981050e-02
75%	7.931355e+17	12.000000	10.0	1.000000	0.841987	0.194207	9.150480e-02
max	8.924206e+17	15.000000	10.0	4.000000	0.999984	0.488014	2.734190e-01

V. STORE DATA

```
# storing final as csv
data_df.to_csv('data_df.csv', encoding='utf-8', index=False)
```

```
#TEST

data_df = pd.read_csv('data_df.csv')
data_df.info()
```