Data Wrangling Report

1. Gathering Data

About the Dataset(s)

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

WeRateDogs downloaded their Twitter archive and sent it to Udacity via email exclusively for you to use in this project. This archive contains basic tweet data (tweet ID, timestamp, text, etc.) for all 5000+ of their tweets as they stood on August 1, 2017. More on this soon.

Data had been gathered as below sourced and loaded to pandas data frames:

1- Twitter archive data as the form of csv format (twitter-archive-enhanced.csv)

2- loaded Image Predictions File (image_predictions.tsv) also tried to automate loading but didn't complete

3- Additional Data via the Twitter API since there was issue in credential and account approval so i used tweet-json.txt as source to avoid delays however i added relevant portion of code for later execution

4-Created a dataFrame with tweet ID, retweet count, favorite count mainly to get followers

2- Assess Data

- Started Displaying data captured from data frames through samples
- Started checking metadata.
- Build some insights to capture as much as possible quality and tidiness issues

Identified DQ Issues:

1- DQ Issue 1 Capturing dog tagging for doggo and floofer at the same time

2- DQ Issue 2 Capturing dog tagging for doggo and pupper at the same time

3- DQ Issue 3 Capturing dog tagging for doggo and puppo at the same time

4- DQ Issue 4 Discrading 4 fields that will not lead to solid analysis as a results of lot's of missing values ['retweeted_status_id', 'retweeted_status_user_id', 'retweeted_status_timestamp','expanded_urls']

5- DQ Issue 5 lot's of null values in archive_df.in_reply_to_status_id

6- DQ Issue 6 having lot's of null values in archive_df.in_reply_to_user_id

7- DQ Issue 7 in_reply_to_status_id having float data type need to be integer

8- DQ Issue 8 in in_reply_to_user_id having float data type need to be integer

9- DQ Issue 9 in timestamp should be datetime datatype

10- DQ issue 10 archive_df name should be string datatype

11- DQ Issue 11 unify tweet_id data type as string in all sources as there is no in use for calculation

12- DQ Issue 12 Validiate rating_numerator & rating_denominator to ensure no zero ratings

13 –DQ Issue 13 for 4 coulmns doggo,floofer,pupper and puppo in archive tweet need to replace 'None' with the NaN to show that it is missing values for 4 coulmns

14-DQ issue 14 need to change 'rating_numerator' and 'rating_denominator' from int to float to have proper measurement

15-DQ Issue 15: As recommended in review comments will keep only those rows in archive-clean that are original tweets and delete rest (i.e. retweeted_status_id column is null) hence tweet_id in tweets archive data frame need to be consistent with image predictions file

16-DQ Issue 16 columns in image prediction data files need to have more descriptive name

17- DQ Issue 17 in tweets json file need to rename columns to be more descriptive from id to tweet_id

Tidiness Issues:

#1- Tidness Issue 1: As part of analysis dogs classifications doggo,floofer,pupper and puppo should be merged into one column # dog_classification

#2 Tidness Issue 2: Copies of the original pieces of data are made prior to cleaning a tidy master dataset (or datasets, if appropriate) with all pieces of gathered data is created.

3. Cleanup

- Create a copy of archive_df data to cleanup data (archive_clean)
- Create copy from image_df to cleanup data (image_clean)
- Create copy from tweets_df to cleanup data (tweets_clean)
- Fixed majority of identified quality and tidness as below

DQ Issue 15

```
#baseline
len(archive_clean[archive_clean.retweeted_status_id.isnull() == False])
#Clean
#capturing as recommended in review comments will keep only those rows in archive-clean that are original tweets and delete rest (i.e. retweeted_status_id column is null)
archive_clean = archive_clean[archive_clean.retweeted_status_id.isnull()]
#Clean
#fixing will make sure that all tweets ids in archive clean consistent with image_df
archive_clean = archive_clean[archive_clean.tweet_id.isin(image_clean.tweet_id)]
#Test
#Fixing DQ Issue 15
len(archive_clean[archive_clean.tweet_id.isin(image_clean.tweet_id)])
#Test
#Fixing DQ Issue 15
len(archive_clean[archive_clean.retweeted_status_id.isnull() == False])
```

DQ issue 14

```
#Define
#DQ issue 14 need to change 'rating_numerator' and 'rating_denominator' from int to float to have proper measurement
#Clean
#Reqtested DQ Issue 14 to 1st change 'rating_numerator'and 'rating_denominator' from int to float
archive_clean[['rating_numerator', 'rating_denominator']] =archive_clean[['rating_numerator', 'rating_denominator']].astype('float')
#Test modification
archive_clean.info()
```

Tidiness Issue 1 :

```
#Define
#Validate Tidiness Issue
archive_clean.loc[(archive_clean[['doggo', 'floofer', 'pupper', 'puppo']] != 'None').sum(axis=1) > 1]

#check puppo counts
archive_clean.puppo.value_counts()

#check doggo counts
archive_clean.doggo.value_counts()
```

```
#check floofer counts
archive_clean.floofer.value_counts()
```

```
#check pupper counts
archive_clean.pupper.value_counts()
```

```
#Check counts
archive_clean.groupby(["doggo", "floofer", "pupper","puppo"]).size().reset_index().rename(columns={0:
"count"})
```

```
#Clean doggo
```

```
#Remember to convert 'None' or np.NaN to empty string "" for all columns prior to running the
following code e.g. df.doggo.replace('None', '', inplace=True) and df.doggo.replace(np.NaN, '',
inplace=True)
archive_clean.doggo.replace('None', '', inplace=True)
```

```
#Clean doggo
```

```
#Remember to convert 'None' or np.NaN to empty string "" for all columns prior to running the
following code e.g. df.doggo.replace('None', '', inplace=True) and df.doggo.replace(np.NaN, '',
inplace=True)
archive_clean.doggo.replace(np.NaN, '', inplace=True)
```

```
#Test doggo
archive_clean.sample(50)
```

```
#Clean floofer
```

```
#Remember to convert 'None' or np.NaN to empty string "" for all columns prior to running the
following code e.g. df.floofer.replace('None', '', inplace=True) and df.floofer.replace(np.NaN, '',
inplace=True)
archive_clean.floofer.replace('None', '', inplace=True)
```

```
#Clean floofer
```

```
#Remember to convert 'None' or np.NaN to empty string "" for all columns prior to running the
following code e.g. df.floofer.replace('None', '', inplace=True) and df.floofer.replace(np.NaN, '',
inplace=True)
archive_clean.floofer.replace(np.NaN, '', inplace=True)
```

```
#Test floofer
archive_clean.sample(50)

#Clean pupper

#Remember to convert 'None' or np.NaN to empty string "" for all columns prior to running the
following code e.g. df.pupper.replace('None', '', inplace=True) and df.pupper.replace(np.NaN, '',
inplace=True)
archive_clean.pupper.replace('None', '', inplace=True)

#Clean pupper

#Remember to convert 'None' or np.NaN to empty string "" for all columns prior to running the
following code e.g. df.pupper.replace('None', '', inplace=True) and df.pupper.replace(np.NaN, '',
inplace=True)
archive_clean.pupper.replace(np.NaN, '', inplace=True)

#Test pupper
archive_clean.sample(150)

#Clean puppo

#Remember to convert 'None' or np.NaN to empty string "" for all columns prior to running the
following code e.g. df.puppo.replace('None', '', inplace=True) and df.puppo.replace(np.NaN, '',
inplace=True)
archive_clean.puppo.replace('None', '', inplace=True)



#Clean puppo

#Remember to convert 'None' or np.NaN to empty string "" for all columns prior to running the
following code e.g. df.puppo.replace('None', '', inplace=True) and df.puppo.replace(np.NaN, '',
inplace=True)
archive_clean.puppo.replace(np.NaN, '', inplace=True)

#Test puppo
archive_clean.sample(150)

#Clean

# use loc to add a new column dog_class = doggo, floofer, pupper or poppo. NaN will be used if not any
of the previous
archive_clean['dog_classification'] = archive_clean.doggo + archive_clean.floofer + archive_clean.pupper
+ archive_clean.puppo
```

archive_clean.loc[archive_clean.dog_classification == 'doggopupper', 'dog_classification'] = 'doggo, pupper'

archive_clean.loc[archive_clean.dog_classification == 'doggopuppo', 'dog_classification'] = 'doggo, puppo'
archive_clean.loc[archive_clean.dog_classification == 'doggofloofer', 'dog_classification'] = 'doggo, floofer'

#Test

archive_clean.dog_classification.value_counts()
archive_clean.sample(100)

#As part of Tidness Issue 1 fixing will convert the dog_classification datatype to categorical
archive_clean.dog_classification = archive_clean.dog_classification.astype('category')

#As part of Tidness Issue 1 fixing will drop the all dogs classifications colmns : doggo, floofer, pupper and puppo
archive_clean.drop(['doggo', 'floofer', 'pupper', 'puppo'], axis=1, inplace=True)

#Test consolidation
archive_clean.dog_classification.value_counts()

#Test Metadata
archive_clean.info()

archive_clean.sample(100)

<u>Tidiness Issue 2 Joining the tables</u>

#Stored cleaned data for archive_df in archive_clean.csv
archive_clean.to_csv('archive_clean.csv', encoding='utf-8', index=False)

#Stored cleaned data for image_df in image_clean.csv
image_clean.to_csv('image_clean.csv', encoding='utf-8', index=False)

#Stored cleaned data for tweets_df in tweets_clean1-Work on 3 Data Quality issues related to datatypes
tweets_clean.to_csv('tweets_clean.csv', encoding='utf-8', index=False)

#Also stored API data frame to file
api_df.to_csv('api_df.csv', encoding='utf-8', index=False)

#Merging archive_clean and image_clean on tweets_stat_fin data frame based tweet_id
tweets_stat_fin = pd.merge(archive_clean, image_clean,how='outer', on=['tweet_id'])

#Merging tweets_clean on tweets_stat_fin data frame based tweet_id
tweets_stat_fin = pd.merge(tweets_stat_fin, tweets_clean, how = 'outer', on=['tweet_id'])

#As requested merged all data frames to one and exported file as below
tweets_stat_fin.to_csv('twitter_archive_master.csv', encoding='utf-8', index=False)

DQ Issue 16

#Define DQ Issue 16 columns in image prediction data files need to have more descriptive name

#Clean

#DQ Issue 16 fixing for better tidness will rename 6 columns to have better meaningful visibility

image_clean= image_clean.rename(columns={'p1':'prediction1', 'p2':'prediction1', 'p3':'prediction3', 'p1_conf':'prediction1_confidence', 'p2_conf':'prediction2_confidence', 'p3_conf':'prediction3_confidence'})

#Test
image_clean.info()

DQ Issue 17

#DQ Issue 17 fixing for better tidness will rename id column to have better meaningful visibility
tweets_clean = tweets_clean.rename(columns={'id':'tweet_id'})

#Test
tweets_clean.info()

DQ Issue 5

#Define DQ Issue 5 missing data which not allowing chaning metadata for in_reply_to_status_id

#Clean

#Fixing DQ Issue 5 through fill missing data to allow chaning metadata for in_reply_to_status_id & in_reply_to_user_id
archive_clean.in_reply_to_status_id = archive_clean.in_reply_to_status_id.fillna(0)

#Test
archive_clean.in_reply_to_status_id.sample(100)

DQ Issue 6

#Define DQ Issue 6 missing data which not allowing chaning metadata for in_reply_to_user_id

#Clean

#Fixing DQ Issue 5 through fill missing data to allow chaning metadata for in_reply_to_status_id & in_reply_to_user_id
archive_clean.in_reply_to_user_id = archive_clean.in_reply_to_user_id.fillna(0)

#Test
archive_clean.in_reply_to_user_id.sample(100)

DQ Issue 7
#Define DQ Issue 7 in_reply_to_status_id having float data type need to be integer

#Clean
#Fixing DQ Issue 7 through change in_reply_to_status_id to integer type
archive_clean.in_reply_to_status_id = archive_clean.in_reply_to_status_id.astype(np.int64)

#Test
archive_clean.info()

DQ Issue 8
#DQ Issue 8 in in_reply_to_user_id having float data type need to be integer
#Clean
#Fixing DQ Issue 8 through change in_reply_to_user_id to integer type
archive_clean.in_reply_to_user_id = archive_clean.in_reply_to_user_id.astype(np.int64)

#Test
archive_clean.info()

DQ Issue 9
#Define
#Fixing DQ Issue 9 through change timestamp to datetime data type

#Clean
#Fixing DQ Issue 9 through change timestamp to datetime data type
archive_clean.timestamp = pd.to_datetime(archive_clean.timestamp)

#Test
archive_clean.info()

DQ Issue 10
#Define
#Fixing DQ Issue 10 through change Name data type string to be able to analyze

#Clean
#Fixing DQ Issue 10 through change Name data type string to be able to analyze
archive_clean['name'] = archive_clean['name'].astype('str')

```
#Test
archive_clean.info()
```

DQ Issue 11

#Define
#Fixing DQ Issue 11 through unity tweet_id data type as string

 #Clean

```
#Fixing DQ Issue 11 through unity tweet_id  data type as string
archive_clean['tweet_id'] = archive_clean['tweet_id'].astype('str')
image_clean['tweet_id'] = image_clean['tweet_id'].astype('str')
tweets_clean['tweet_id'] = tweets_clean['tweet_id'].astype('str')
api_df['id'] = api_df['id'].astype('str')
```

```
#Test
archive_clean.info()
image_clean.info()
tweets_clean.info()
api_df.info()
```

DQ Issue 12

#Define zero values from the numertor and denuminator ratings

#Clean
# Fixing DQ Issue 12 Exclude zero values from the numertor and denuminator ratings

```
archive_clean = archive_clean[archive_clean['rating_numerator'] != 0 ]
archive_clean = archive_clean[archive_clean['rating_denominator'] != 0 ]
```

```
#Test
archive_clean['rating_numerator'].sample(100)
archive_clean['rating_denominator'].sample(100)
```

DQ Issue 13

#Define DQ Issue 13 through replacing the value 'None' with the NaN to show that it is missing values for 4 coulmns doggo,floofer,pupper and puppo

archive_clean = archive_clean.replace('None', np.nan)
#Clean doggo DQ Issue 13

#Remember to convert 'None' or np.NaN to empty string "" for all columns prior to running the following code e.g. df.doggo.replace('None', '', inplace=True) and df.doggo.replace(np.NaN, '', inplace=True) (please do this for all other three categories too i.e. 'pupper', 'puppo', and 'floofer').
archive_clean.doggo.replace('None', '', inplace=True)

#Clean doggo DQ Issue 13
#Remember to convert 'None' or np.NaN to empty string "" for all columns prior to running the following code e.g. df.doggo.replace('None', '', inplace=True) and df.doggo.replace(np.NaN, '', inplace=True) (please do this for all other three categories too i.e. 'pupper', 'puppo', and 'floofer').
archive_clean.doggo.replace(np.NaN, '', inplace=True)

#Test doggo
archive_clean.sample(50)

#Clean floofer DQ Issue 13
#Remember to convert 'None' or np.NaN to empty string "" for all columns prior to running the following code e.g. df.floofer.replace('None', '', inplace=True) and df.floofer.replace(np.NaN, '', inplace=True)
archive_clean.floofer.replace('None', '', inplace=True)

#Clean floofer DQ Issue 13
#Remember to convert 'None' or np.NaN to empty string "" for all columns prior to running the following code e.g. df.floofer.replace('None', '', inplace=True) and df.floofer.replace(np.NaN, '', inplace=True)
archive_clean.floofer.replace(np.NaN, '', inplace=True)

#Test floofer
archive_clean.sample(50)

#Clean pupper
#Remember to convert 'None' or np.NaN to empty string "" for all columns prior to running the following code e.g. df.pupper.replace('None', '', inplace=True) and df.pupper.replace(np.NaN, '', inplace=True)
archive_clean.pupper.replace('None', '', inplace=True)

#Clean pupper DQ Issue 13
#Remember to convert 'None' or np.NaN to empty string "" for all columns prior to running the

following code e.g. df.pupper.replace('None', '', inplace=True) and df.pupper.replace(np.NaN, '', inplace=True)
archive_clean.pupper.replace(np.NaN, '', inplace=True)

#Test pupper
archive_clean.sample(150)

#Clean puppo DQ Issue 13
#Remember to convert 'None' or np.NaN to empty string "" for all columns prior to running the following code e.g. df.puppo.replace('None', '', inplace=True) and df.puppo.replace(np.NaN, '', inplace=True)
archive_clean.puppo.replace('None', '', inplace=True)

#Clean puppo DQ Issue 13
#Remember to convert 'None' or np.NaN to empty string "" for all columns prior to running the following code e.g. df.puppo.replace('None', '', inplace=True) and df.puppo.replace(np.NaN, '', inplace=True)
archive_clean.puppo.replace(np.NaN, '', inplace=True)

#Test puppo
archive_clean.sample(150)

5- Store:

#Stored cleaned data for archive_df in archive_clean.csv
archive_clean.to_csv('archive_clean.csv', encoding='utf-8', index=False)

#Stored cleaned data for image_df in image_clean.csv
image_clean.to_csv('image_clean.csv', encoding='utf-8', index=False)

#Stored cleaned data for tweets_df in tweets_clean1-Work on 3 Data Quality issues related to datatypes
tweets_clean.to_csv('tweets_clean.csv', encoding='utf-8', index=False)

#Also stored API data frame to file
api_df.to_csv('api_df.csv', encoding='utf-8', index=False)

#Merging archive_clean and image_clean on tweets_stat_fin data frame based tweet_id
tweets_stat_fin = pd.merge(archive_clean, image_clean,how='outer', on=['tweet_id'])

#Merging tweets_clean on tweets_stat_fin data frame based tweet_id
tweets_stat_fin = pd.merge(tweets_stat_fin, tweets_clean, how = 'outer', on=['tweet_id'])

#As requested merged all data frames to one and exported file as below
tweets_stat_fin.to_csv('twitter_archive_master.csv', encoding='utf-8', index=False)