Wrangle Report

Data wrangling is split into 3 steps

1. Gather
2. Assess
3. Clean

Gather

We have 3 sources of data for this project:

1. download twitter-archive-enhanced.csv which is done manually from the project overview page
2. download image-predictions.tsv programmatically using requests library
3. any data from twitter API that we see fit

- Downloading manually is no problem so we will skip that part

- Downloading programmatically consists of using requests library to access the link and then accessing the content of that response which is the TSV file

```
In [4]: #use requests library to access the link in order to download it
        response=requests.get(url)
        response

Out[4]: <Response [200]>
```

```
In [ ]: #writing the content of the response to a file
        if not os.path.isfile(file_name):
            with open (file_name,'wb') as file:
                file.write(response.content)
```

Twitter API was the long part as I had to apply for a developer account answer a number of questions but I gained access eventually

```
In [9]: #accessing API
        consumer_key=[redacted]
        consumer_secret=[redacted]
        access_token=[redacted]
        access_secret=[redacted]
```

```
In [10]: auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
         auth.set_access_token(access_token, access_secret)
         api=tweepy.API(auth,wait_on_rate_limit=True,wait_on_rate_limit_notify=True)
```

Getting tweets using their id and storing in a file

```
In [12]: #getting the tweets data using their id,the errors list is to catch any tweet that can't be found
errors = []
if not os.path.isfile('tweet_json.txt'):
    # create the file and write on it
    with open ('tweet_json.txt', 'w') as file:
        for tweet_id in archive_df['tweet_id']:
            try:
                status = api.get_status(tweet_id, wait_on_rate_limit=True,  wait_on_rate_limit_notify=True, tweet_mode = 'extende
                json.dump(status._json, file)
                file.write('\n')
            except Exception as e:
                print("Error on tweet id {}".format(tweet_id) + ";" + str(e))
                errors.append(tweet_id)
```

Getting the tweets id , favorite_count and retweet to create the desired DF

```
In [16]: tweets_list = []

with open('tweet_json.txt', 'r') as file:
    for line in file:
        tweet = json.loads(line)
        tweet_id = tweet['id']
        retweet_count = tweet['retweet_count']
        fav_count = tweet['favorite_count']
        tweets_list.append({'tweet_id':tweet_id,
                            'retweet_count': retweet_count,
                            'favorite_count': fav_count})

tweet_df=pd.DataFrame(tweets_list)
tweet_df
```

Assessing

Now we have all the data we need and we can start assessing

Assessing is done on 2 levels:

1. Visually by looking at the DF either on pandas or Excel
2. Programmatic assessment using pandas functions(info,head,tail,shape,etc..)

Long story short I have come up with some issues as below

In Archive DF

issues in archive_df

- timestamp not in time date format
- timestamp column contains 2 different data (date and time)
- some tweets doesn't have an image
- some tweets are replys
- some tweets are retweets not the original one
- tweet id should be string not int
- doggo floofer pupper puppo columns has none instead of being empty
- dog stage is divided into 4 columns
- after removing retweets and replys the columns related to them are no longer needed
- convert rating from 2 columns to a single column

In Images DF

issues in images

- tweet id should be string not int
- column headers should be changed to a more describtive headers
- need to check if all tweets in file are orignial tweets or some are retweets and replys

---

In Tweet DF

issues in tweet df

- tweet_id should be string not int
- retweets count and favourite count can be added as 2 columns in the archive DF

Cleaning

Now we can start cleaning these issues

Cleaning the data ranges from using simple methods to convert column data type to matching data from 2 different data frames in order to eliminate irrelevant data

For ex

To remove any tweet that doesn't have an image we use the image prediction file in order to use the tweets id in it as reference as these are the tweets with images in them

Code

In [40]:
```
# creating a list of tweet_ids with images "tweets_with_image" and confirming its length
tweets_with_image = list(images_clean.tweet_id.unique())
len(tweets_with_image)
```

Out[40]: 2075

In [41]:
```
# confirming that all the tweets with images exist in the archive dataset
len(tweets_with_image) == archive_df_clean.tweet_id.isin(tweets_with_image).sum()
```

Out[41]: True

In [42]:
```
# Cleaning--leaving only tweets with ids exisiting in the images file
archive_df_clean = archive_df_clean[archive_df_clean.tweet_id.isin(tweets_with_image)]
```

Removing tweets that are reply

3-Remove tweets that are replys using in_reply_to_status_id column as guide

Code

In [44]:
```
replys_tweets=archive_df_clean.in_reply_to_status_id.notnull()
archive_df_clean=archive_df_clean[~replys_tweets]
```

Converting tweet id from in to string

Archive DF

5- Convert Tweet ID to string using astype method

In [48]:
```
archive_df_clean['tweet_id']=archive_df_clean['tweet_id'].astype(str)
```

Replacing the word ‘None’ with empty value

6- Replace None with empty string in doggo floofer pupper puppo columns

In [51]:
```
archive_df_clean['doggo']=archive_df_clean['doggo'].str.replace('None','')
archive_df_clean['floofer']=archive_df_clean['floofer'].str.replace('None','')
archive_df_clean['pupper']=archive_df_clean['pupper'].str.replace('None','')
archive_df_clean['puppo']=archive_df_clean['puppo'].str.replace('None','')
```

Cleaning goes on and on till we reach a semi clean dataset

Then we store that data in a master df

Storing data

In [199]:
```
archive_df_clean.to_csv('twitter_archive_master.csv',index=False)
```

Now we can use this data to generate some insights about it