WRANGLE ACT ON WERATEDOGS DATA

BY AMOS OMOFAIYE

TABLE OF CONTENTS

INTRODUCTION OF THE PROJECT

> *This project is part of the Bertelsman Scholarship for Data Analyst from Udacity. The main thrust is to analyze tweet data of **WeRateDogs** - a dog rating organization. This organization provides a humourous dog rating service. One notable thing is that their rating numerator is usually greater than the denominator. This is akin to awarding a student 12/10. They do this because the dog is too good to them. Other information about them can be found in the README.txt file. To begin with, the data must first be gathered from three sources. One of the files has been provided by Udacity (twitter-archive-enhanced.csv). The second dataset will be programmatically gathered from udacity through the address (https://d17h27t6h515a5.cloudfront.net/topher/2017/August/599fd2ad_image-predictions/image-predictions.tsv). This second dataset will be stored as **image-predictions.tsv** which is the file name. The third and the final dataset for this project will be programmatically gathered from twitter using their API. The data will be stored as **tweet-json.txt** and the needed data will be ectracted from it into a dataframe. After gathering the data will be cleaned, combined, stored, analyzed, visualized, and reported on. I will try to make the process as interactive as possible. Now, we look at the specific objectives of the project.*
>
> ### *Objectives of the project*
>
> *The objectives of the project are to:*
>
> - *gather data from three different sources.*
> - *assess the gathered data with the aim of identifying at least 8 quality issues and 2 tidiness issues.*
> - *clean the data with respect to the identified issues.*
> - *store the cleaned data in a file titled **twitter-archive-master.csv**.*
> - *analyze and visualize the stored data producing at least 3 insights and 1 visualization.*
> - *report the work by producing two documents namely internal (wrangle_report.pdf or html with 300-600 words) detailing the wrangling efforts and external (act_report.pdf or html, 250 words mininum) detailing the insights and visualizations.*

Gathering data

>

> *Now the main work begins. In this section I will gather the needed datasets. To do this I will need the following libraries requests, tweepy, json, pandas, and numpy. I will also import the libraries that I will later use for visualization. These will be matplotlib, seaborn, and pywaffle. Please note that because it took a long time before my twitter api application was approved, I used the tweet-json.txt file provided as an alternative. But I later got the api. I d not deem it necessary to redo the sesion again because when I extracted the data it was the same except that it took longer*

than 1hour 30 minutes before I could finish the download. Let us continue.

In [1]:

```
# Importing the needed libraries
import requests as rs
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
from pywaffle import Waffle
import tweepy
import json
```

In [ ]:

```
# Programmatically downloading the image-predictions.tsv using the requests library
url = 'https://d17h27t6h515a5.cloudfront.net/topher/2017/August/599fd2ad_image-predictions/image-p
redictions.tsv'
response = rs.get(url)
with open("image-predictions.tsv", mode="wb") as file:
    file.write(response.content)
```

In [2]:

```
# Loading the image-predictions dataset. Since it is a tsv, I had to use tab as the separator.
di = pd.read_csv('image-predictions.tsv', sep='\t')
```

In [3]:

```
# Checking to see whether it is correctly loaded. Note that di means dataframe of image
predictions
di.head()
```

Out[3]:

	tweet_id	jpg_url	img_num	p1	p1_conf	p1_dog	
0	666020888022790149	https://pbs.twimg.com/media/CT4udn0WwAA0aMy.jpg	1	Welsh_springer_spaniel	0.465074	True	
1	666029285002620928	https://pbs.twimg.com/media/CT42GRgUYAA5iDo.jpg	1	redbone	0.506826	True	miniature_
2	666033412701032449	https://pbs.twimg.com/media/CT4521TWwAEvMyu.jpg	1	German_shepherd	0.596461	True	
3	666044226329800704	https://pbs.twimg.com/media/CT5Dr8HUEAA-lEu.jpg	1	Rhodesian_ridgeback	0.408143	True	
4	666049248165822465	https://pbs.twimg.com/media/CT5IQmsXIAAKY4A.jpg	1	miniature_pinscher	0.560311	True	F

In [4]:

```
# Loading the given twitter archive enhanced file into a dataframe.
dt = pd.read_csv('twitter-archive-enhanced.csv')
```

In [5]:

```
# Checking to see whether it is correctly loaded. Note that dt means dataframe of twitter archive
enhanced
dt.head()
```

Out[5]:

	tweet_id	in_reply_to_status_id	in_reply_to_user_id	timestamp	source	text	retwee
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...	
1	892177421306343426	NaN	NaN	2017-08-01	<a href="http://twitter.com/download/iphone"	This is Tilly. She's just checking	

	tweet_id	in_reply_to_status_id	in_reply_to_user_id	timestamp	source	text	retwee
1	892177421306343426	NaN	NaN	00:17:27 +0000	href="http://twitter.com/download/iphone"	checking pupper you....	
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...	
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...	
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...	

The copied code due to lack of access to twitter API is in this cell. It will not be run

import tweepy from tweepy import OAuthHandler import json from timeit import default_timer as timer

Query Twitter API for each tweet in the Twitter archive and save JSON in a text file

These are hidden to comply with Twitter's API terms and conditions

consumer_key = 'HIDDEN' consumer_secret = 'HIDDEN' access_token = 'HIDDEN' access_secret = 'HIDDEN'

auth = OAuthHandler(consumer_key, consumer_secret) auth.set_access_token(access_token, access_secret)

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

NOTE TO STUDENT WITH MOBILE VERIFICATION ISSUES:

df_1 is a DataFrame with the twitter_archive_enhanced.csv file. You may have to

change line 17 to match the name of your DataFrame with twitter_archive_enhanced.csv

NOTE TO REVIEWER: this student had mobile verification issues so the following

Twitter API code was sent to this student from a Udacity instructor

Tweet IDs for which to gather additional data via Twitter's API

tweet_ids = da.tweet_id.values len(tweet_ids)

Query Twitter's API for JSON data for each tweet ID in the Twitter archive

count = 0 fails_dict = {} start = timer()

Save each tweet's returned JSON as a new line in a .txt file

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)

In [6]:

```
# Creating a list of data from the available tweet-json.txt file
tweets_data = []
tweet_file = open('tweet-json.txt', 'r')
for line in tweet_file:
    try:
        tweet = json.loads(line)
        tweets_data.append(tweet)
    except:
        continue
tweet_file.close()
```

In [7]:

```
# Checking the list to determine which columns that I will like to include apart from the compulso
ry
# tweet id, retweet count, and favorite count.
#tweets_data
```

In [8]:

```
# Creating a  dataframe for  the extracted tweet information. The dataframe will be named dj.
dj = pd.DataFrame()
# Add variables to df: tweet ID, retweet count, favorite count
dj['tweet_id'] = list(map(lambda tweet: tweet['id'], tweets_data))
dj['created_at'] = list(map(lambda tweet: tweet['created_at'], tweets_data))
dj['retweet_count'] = list(map(lambda tweet: tweet['retweet_count'], tweets_data))
dj['favorite_count'] = list(map(lambda tweet: tweet['favorite_count'], tweets_data))
dj['full_text'] = list(map(lambda tweet: tweet['full_text'], tweets_data))
dj['full_text'] = list(map(lambda tweet: tweet['full_text'], tweets_data))
dj['favorited'] = list(map(lambda tweet: tweet['favorited'], tweets_data))
dj['retweeted'] = list(map(lambda tweet: tweet['retweeted'], tweets_data))
```

In [9]:

```
# Checking to see how the dataframe looks. Note that dj means dataframe of extracted data from the
tweet-json.txt file.
dj.head()
```

Out[9]:

	tweet_id	created_at	retweet_count	favorite_count	full_text	favorited	retweeted
0	892420643555336193	Tue Aug 01 16:23:56 +0000 2017	8853	39467	This is Phineas. He's a mystical boy. Only eve...	False	False
1	892177421306343426	Tue Aug 01 00:17:27 +0000 2017	6514	33819	This is Tilly. She's just checking pup on you....	False	False
2	891815181378084864	Mon Jul 31 00:18:03 +0000 2017	4328	25461	This is Archie. He is a rare Norwegian Pouncin...	False	False

	tweet_id	created_at	retweet_count	favorite_count	full_text	favorited	retweeted
3	891689557279858688	Sun Jul 30 15:58:51 +0000 2017	8964	42908	This is Darla. She commenced a snooze mid meal...	False	False
4	891327558926688256	Sat Jul 29 16:00:24 +0000 2017	9774	41048	This is Franklin. He would like you to stop ca...	False	False

In [10]:

```
# Saving the dj dataframe to csv file
#dj.to_csv('tweet_extract.csv', index=False)
```

In [11]:

```
# Generating requirements.txt file for this project
#!pip freeze > requirements.txt
```

Assessing the gathered data

>

> *In this section I will assess the three datasets with the aim of identifying at least 8 data quality issues and 2 data tidiness issues.*
>
> *Data Quality issues will include such things as missing values, non-descriptive column headings, duplications, incorrect spellings etc. Generally, they may be categorized into issues of that undermine data completeness, uniqueness, timeliness, validity, accuracy, and consistency. Datasets with such issues are called **dirty datasets**.*
>
> *Data tidiness issues are seen in datasets that do not follow the tidy data principle. Such datasets are called **messy datasets**. Please note that a tidy dataset should have each variable forming a column, each observation forming a row, and each type of observational unit forming a table.*
>
> *With that preamble we delve into the assessment process. I shall assess the data in the following order: dj, dt, and di. Shall we?*

In [12]:

```
# Assesing the shape of the data extracted from tweet-json.txt. This is stored in the dj dataframe
.
dj.shape
```

Out[12]:

(2354, 7)

In [13]:

```
# Viewing some samples of the dataframe
dj.sample(5)
```

Out[13]:

	tweet_id	created_at	retweet_count	favorite_count	full_text	favorited	retweeted
820	770093767776997377	Mon Aug 29 03:00:36 +0000 2016	3520	0	RT @dog_rates: This is just downright precious...	False	False
211	851861385021730816	Tue Apr 11 18:15:55 +0000 2017	23	0	RT @eddie_coe98: Thanks @dog_rates completed m...	False	False
1619	684926975086034944	Thu Jan 07 02:38:10 +0000 2016	552	3849	Meet Bruiser & Charlie. They are the best ...	False	False
499	813112105746448384	Sun Dec 25 20:00:07 +0000 2016	3225	11515	Meet Toby. He's pupset because his hat isn't b...	False	False
167	859607811541651456	Wed May 03 03:17:27 +0000 2017	1704	19476	Sorry for the lack of posts today. I came home...	False	False

In [14]:

```
# Listing out its columns
# First, I will define a function to do this so that I call the function on any dataframe.
def column_lister(df):
    """
    This function will take a single argument. The argument must hold a dataframe with columns.
    The function will then return a borderless table showing the columns in the dataframe with the
ir indices.
    """
    for i,v in enumerate(df.columns):
        print(i,v)
column_lister(dj)
```

```
0 tweet_id
1 created_at
2 retweet_count
3 favorite_count
4 full_text
5 favorited
6 retweeted
```

twitter_json(dj) dataframe columns

> *First let me explain these columns in brief.*
>
> - *tweet_id is the number that identifies each tweet. Customarily, it shhould be unique that is without duplicates.*
> - *created_at shows the time the tweet was created. Since I will not expect an organization to create two tweets at a time, then the this column should contain unique values.*
> - *retweet_count is the number of times the tweet was retweeted by users.*
> - *favorite_count is the number of times the tweet was declared a favorite by users.*
> - *full_text is the message or the text of the tweet.*
> - *favorited answers the question whether the tweet was favorited or not by WeRateDogs. It should be either False or True.*
> - *retweeted answers the question whether the tweet was retweeted or not by WeRateDogs. It should be either False or True.*

In [15]:

```
# I want to check the tweet_id for uniqueness since it will be needed in joining the three data se
ts. To be useful for joinig it
# must be uniquely present in all the three dataframes.
dj.tweet_id.unique().size, dj.tweet_id.nunique()
```

Out[15]:

```
(2354, 2354)
```

> *The dj dataframe contains 2354 rows and 7 columns. The tweet_id column as shown by the two methods above contains unique values and can therefore be used in joining the datasets. Let us continue.*

In [16]:

```
# Checking for null values
dj.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2354 entries, 0 to 2353
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   tweet_id        2354 non-null   int64 
 1   created_at      2354 non-null   object
 2   retweet_count   2354 non-null   int64 
 3   favorite_count  2354 non-null   int64 
```

```
 4   full_text       2354 non-null   object
 5   favorited       2354 non-null   bool
 6   retweeted       2354 non-null   bool
dtypes: bool(2), int64(3), object(2)
memory usage: 96.7+ KB
```

In [17]:

```
# Checking for duplicates
sum(dj.duplicated())
```

Out[17]:

```
0
```

In [18]:

```
# Checking for data types
dj.dtypes
```

Out[18]:

```
tweet_id            int64
created_at         object
retweet_count       int64
favorite_count      int64
full_text          object
favorited            bool
retweeted            bool
dtype: object
```

In [19]:

```
# Describing the quantitative variables in this dataframe
dj.describe()
```

Out[19]:

	tweet_id	retweet_count	favorite_count
count	2.354000e+03	2354.000000	2354.000000
mean	7.426978e+17	3164.797366	8080.968564
std	6.852812e+16	5284.770364	11814.771334
min	6.660209e+17	0.000000	0.000000
25%	6.783975e+17	624.500000	1415.000000
50%	7.194596e+17	1473.500000	3603.500000
75%	7.993058e+17	3652.000000	10122.250000
max	8.924206e+17	79515.000000	132810.000000

In [20]:

```
# Checking the observations that are above the 75th percentile
dj[dj['retweet_count'] > 3652].shape, dj[dj['favorite_count'] > 10122.25].shape
# Although there appears to be great difference, the difference warants further investigation to d
etermine outliers.
```

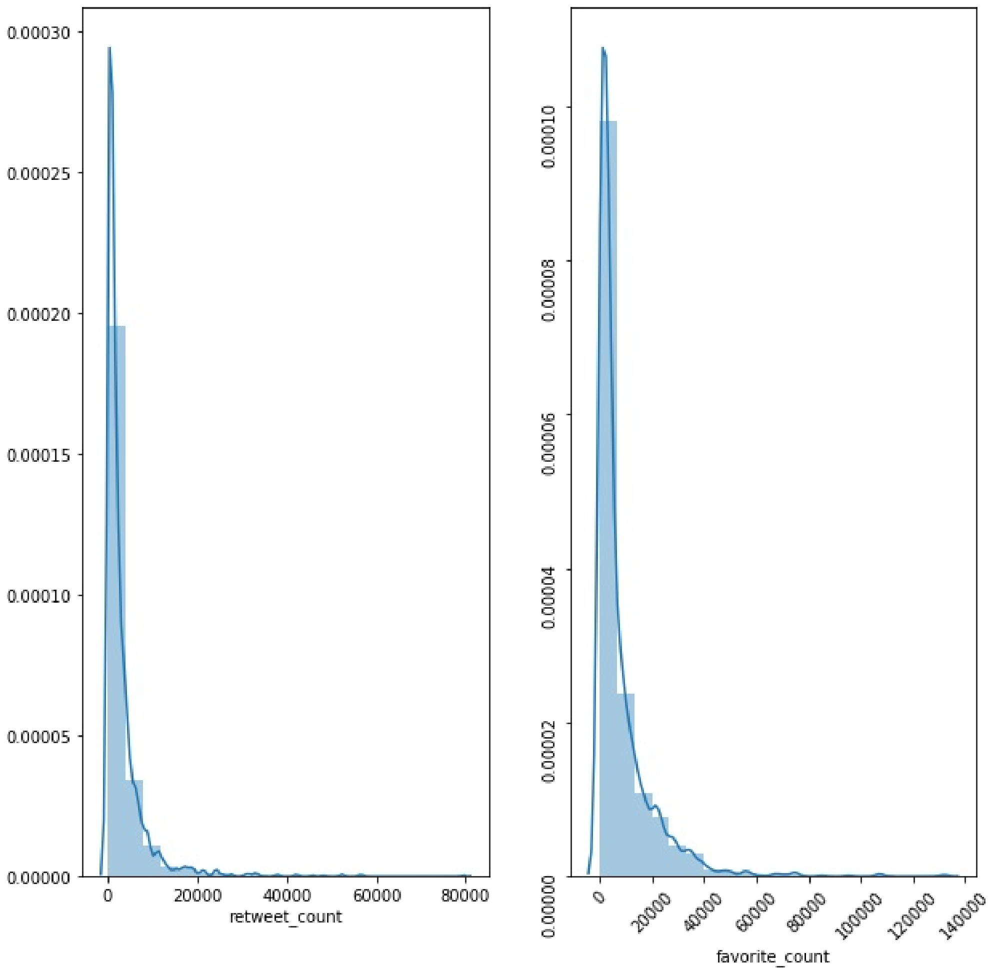
Out[20]:

```
((586, 7), (589, 7))
```

In [21]:

```
# Visualizing the distribution of retweet_count and favorite_count
plt.figure(figsize=(10,10))
plt.subplot(1,2,1)
sb.distplot(dj['retweet_count'], bins = 20)
plt.subplot(1,2,2)
```

```
sb.distplot(dj['favorite_count'], bins = 20)
plt.yticks(rotation=90)
plt.xticks(rotation=45);
```



> *The distribution of the retweet_count and the favorite_count are positively skewed with a few outliers. Let us continue.*

In [22]:

```
# Checking for variation in some columns
# dj.created_at.value_counts() # Varies and all values are unique
# dj.retweet_count.value_counts() # Varies with maximum being 5 and one tweet having zero retweet
# dj.favorite_count.value_counts() # Varies with 179 tweets having zero favorite
# dj.favorited.value_counts() # Varies but greatly tends towards False as only 8 tweets were favor
ited by WeRateDogs
# dj.retweeted.value_counts() # Does not vary. All values are False. No tweet was retweeted by WeR
ateDogs
```

Next, we assess the twitter-archive-enhanced dataset. This is stored in dt dataframe.

In [23]:

```
# Assesing the shape of the dt dataframe.
dt.shape
```

Out[23]:

```
(2356, 17)
```

In [24]:

```
# Viewing some samples of the dataframe
```

```
dt.sample(15,replace=False).T
```

Out[24]:

	1537	1917	
tweet_id	689877686181715968	674291837063053312	
in_reply_to_status_id	NaN	NaN	
in_reply_to_user_id	NaN	NaN	
timestamp	2016-01-20 18:30:32 +0000	2015-12-08 18:17:56 +0000	
source	<a href="http://twitter.com/download/iphone" r...	<a href="http://twitter.com/download/iphone" r...	<a href="http://twitt
text	This is Durg. He's trying to conquer his fear ...	This is Kenny. He just wants to be included in...	Unbelievable. We oı
retweeted_status_id	NaN	NaN	
retweeted_status_user_id	NaN	NaN	
retweeted_status_timestamp	NaN	NaN	
expanded_urls	https://twitter.com/dog_rates/status/689877686...	https://twitter.com/dog_rates/status/674291837...	https://twitter.com/dc
rating_numerator	9	11	
rating_denominator	10	10	
name	Durg	Kenny	
doggo	None	None	
floofer	None	None	
pupper	None	None	
puppo	None	None	

In [25]:

```
# Listing out the columns of the dt dataframe
column_lister(dt)
```

```
0 tweet_id
1 in_reply_to_status_id
2 in_reply_to_user_id
3 timestamp
4 source
5 text
6 retweeted_status_id
7 retweeted_status_user_id
8 retweeted_status_timestamp
9 expanded_urls
10 rating_numerator
11 rating_denominator
12 name
13 doggo
14 floofer
15 pupper
16 puppo
```

twitter-archive-enhanced(dt) dataframe columns

> *At this juncture, I will explain these columns in brief.*
>
> - *tweet_id is the assigned identity of each tweet. This is unique for each tweet.*
> - *in_reply to status_id is the identity number of the status reply.*
> - *in_reply_to_user_id captures the identity number of the user who replied to the status.*
> - *timestamp captures the time the tweet was created*
> - *source is the account from which the tweet originated*
> - *text is the message in each tweet that is the tweet content.*
> - *retweeted_status_id is the identity number of the retweet of the status.*
> - *retweeted_status_user_id is the identity number of the user who retweets the tweet.*
> - *retweeted_status_timestamp captures the time the retweet was made.*

- *expanded_urls is the uniform resource locator of each tweet.*
- *rating_numerator is the rating of the dog. This is the upper part of the rating fraction.*
- *rating_denominator is the standard on which the rating is based. The interesting thing is that this varies. It is usually 10 but it can be greatly more than 10.*
- *name captures the proper or given name of the dog.*
- *doggo is a dog stage for adults (full grown) dogs.*
- *floofer is a dog stage for fluffy dogs.*
- *pupper is a dog stage for young dogs.*
- *puppo is a dog stage for dogs developing from young to adult.*

In [26]:

```
# I want to check the tweet_id for uniqueness since it will be needed in joining the three data se
ts. To be useful for joinig it
# must be uniquely present in all the three dataframes.
dt.tweet_id.unique().size, dt.tweet_id.nunique()
```

Out[26]:

```
(2356, 2356)
```

In [27]:

```
# Checking for null values
dt.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2356 entries, 0 to 2355
Data columns (total 17 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   tweet_id                    2356 non-null   int64
 1   in_reply_to_status_id       78 non-null     float64
 2   in_reply_to_user_id         78 non-null     float64
 3   timestamp                   2356 non-null   object
 4   source                      2356 non-null   object
 5   text                        2356 non-null   object
 6   retweeted_status_id         181 non-null    float64
 7   retweeted_status_user_id    181 non-null    float64
 8   retweeted_status_timestamp  181 non-null    object
 9   expanded_urls               2297 non-null   object
 10  rating_numerator            2356 non-null   int64
 11  rating_denominator          2356 non-null   int64
 12  name                        2356 non-null   object
 13  doggo                       2356 non-null   object
 14  floofer                     2356 non-null   object
 15  pupper                      2356 non-null   object
 16  puppo                       2356 non-null   object
dtypes: float64(4), int64(3), object(10)
memory usage: 313.0+ KB
```

In [28]:

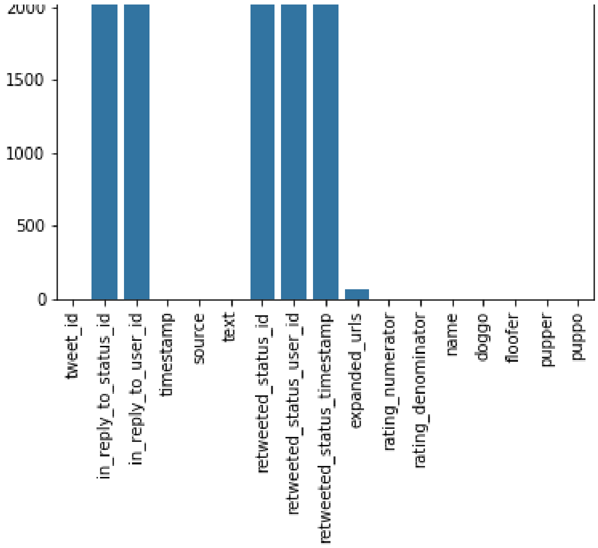
```
# Checking for duplicates
sum(dt.duplicated())
```

Out[28]:

```
0
```

In [29]:

```
# Visualizing the variables with null values
null_counts = dt.isnull().sum()
sb.barplot(null_counts.index.values, null_counts, color = sb.color_palette()[0])
plt.xticks(rotation=90);
```



In [30]:

```
# Checking each variable to determine how their values are.
# dt.rating_numerator.value_counts() # The highest rating numerator is 1776 and the lowest is 1
# dt.rating_denominator.value_counts() # The highest denominator is 170 and the lowest is 0.
# dt.name.value_counts() # 745 names are None while 55 other names are a. These doesn't look like
correct names
# dt.puppo.value_counts() # The sum of the four categories is far less than the number of tweet_id
. This means that most tweets
# are not associated with dog stage.
```

In [31]:

```
# Checking the information of the tweet with zero rating_denominator.
dt[dt['rating_denominator'] == 0]
```

Out[31]:

	tweet_id	in_reply_to_status_id	in_reply_to_user_id	timestamp	source	text	ret
313	835246439529840640	8.352460e+17	26259576.0	2017-02-24 21:54:03 +0000	<a href="http://twitter.com/download/iphone" r...	@jonnysun @Lin_Manuel ok jomny I know you're e...	

In [32]:

```
# Describing the quantitative variables in this dataframe
dt.describe()
# The rating numerator and rating denominator variables appear to have outliers.
```

Out[32]:

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

In [33]:

```
# Checking the observations that are above the 75th percentile
dt[dt['rating_numerator'] > 12].shape, dt[dt['rating_denominator'] > 10].shape
# Although there appears to be great difference especially in rating_numerator,
# the difference warants further investigation to determine outliers.
```

Out[33]:

```
((433, 17), (20, 17))
```

In [34]:

```
# Checking the observations that are above the twice the 75th percentile
dt[dt['rating_numerator'] > 24].shape, dt[dt['rating_denominator'] > 20].shape
# These appears so little. They are therfore outliers. We maydecide to visualize them using histog
ram.
```

Out[34]:

```
((23, 17), (13, 17))
```

Now I will assess the image predictions dataset stored in di dataframe

In [35]:

```
# Assesing the shape of the di dataframe.
di.shape
```

Out[35]:

```
(2075, 12)
```

In [36]:

```
# Viewing some samples of the dataframe
di.sample(5,replace=False)
```

Out[36]:

	tweet_id	jpg_url	img_num	p1	p1_conf	p1_dog	
1567	794205286408003585	https://pbs.twimg.com/media/CwWVe_3WEAAHAvx.jpg	3	pedestal	0.662660	False	fo
1895	850019790995546112	https://pbs.twimg.com/media/C8vgfTsXgAA561h.jpg	3	Shetland_sheepdog	0.759907	True	
318	671763349865160704	https://pbs.twimg.com/media/CVKVM3NW4AAdi1e.jpg	1	prayer_rug	0.445334	False	dc
218	670073503555706880	https://pbs.twimg.com/media/CUyUSuWXIAAZKYF.jpg	1	malamute	0.601886	True	Siberian_
1126	727524757080539137	https://pbs.twimg.com/media/Chiv6BAW4AAiQvH.jpg	2	Pomeranian	0.958834	True	Chih

In [37]:

```
# Listing out the columns of the di dataframe
column_lister(di)
```

```
0 tweet_id
1 jpg_url
2 img_num
3 p1
4 p1_conf
5 p1_dog
6 p2
7 p2_conf
8 p2_dog
9 p3
10 p3_conf
11 p3_dog
```

image predictions (di) dataframe columns

Image predictions (di) dataframe columns

> *At this juncture, I will explain these columns in brief. Please note that some of the columns in this dataset are generated from an image prediction software.*
>
> - *tweet_id is the id of each tweet. This should be unique.*
> - *jpg_url is the uniform resource locator of the image of the dog being tweeted about*
> - *img_num captures the specific image number from a list of possible images depicted by numbers.*
> - *p1 is the algorithm's first prediction for the image in the tweet.*
> - *p1_conf captures how confident the algorithm is in the first prediction.*
> - *p1_dog captures whether the first prediction is a breed of dog or not.*
> - *p2 is the algorithm's most likely second prediction for the image in the tweet.*
> - *p2_config captures how confident the algorithm is in the second prediction.*
> - *p2_dog captures whether the second prediction is a breed of dog or not.*
> - *p3 is the algorithm's most likely third prediction for the image in the tweet.*
> - *p3_config captures how confident the algorithm is in the third prediction.*
> - *p3_dog captures whether the third prediction is a breed of dog or not.*

In [38]:

```
# Customarily, I want to check the tweet_id for uniqueness since it will be needed in joining the three data sets.
# To be useful for joinig it
# must be uniquely present in all the three dataframes.
di.tweet_id.unique().size, di.tweet_id.nunique()
```

Out[38]:

```
(2075, 2075)
```

In [39]:

```
# Checking for null values
di.info()
# There are no null values in this dataset. Let's continue.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2075 entries, 0 to 2074
Data columns (total 12 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   tweet_id  2075 non-null   int64  
 1   jpg_url   2075 non-null   object 
 2   img_num   2075 non-null   int64  
 3   p1        2075 non-null   object 
 4   p1_conf   2075 non-null   float64
 5   p1_dog    2075 non-null   bool   
 6   p2        2075 non-null   object 
 7   p2_conf   2075 non-null   float64
 8   p2_dog    2075 non-null   bool   
 9   p3        2075 non-null   object 
 10  p3_conf   2075 non-null   float64
 11  p3_dog    2075 non-null   bool   
dtypes: bool(3), float64(3), int64(2), object(4)
memory usage: 152.1+ KB
```

In [40]:

```
# Describing the numerical variables
di.describe()
# It appears 75% of the images are 1.
```

Out[40]:

	tweet_id	img_num	p1_conf	p2_conf	p3_conf
count	2.075000e+03	2075.000000	2075.000000	2.075000e+03	2.075000e+03
mean	7.384514e+17	1.203855	0.594548	1.345886e-01	6.032417e-02
std	6.785203e+16	0.561875	0.271174	1.006657e-01	5.090593e-02

	tweet_id	img_num	p1_conf	p2_conf	p3_conf
min	6.660209e+17	1.000000	0.044333	1.011300e-08	1.740170e-10
25%	6.764835e+17	1.000000	0.364412	5.388625e-02	1.622240e-02
50%	7.119988e+17	1.000000	0.588230	1.181810e-01	4.944380e-02
75%	7.932034e+17	1.000000	0.843855	1.955655e-01	9.180755e-02
max	8.924206e+17	4.000000	1.000000	4.880140e-01	2.734190e-01

In [41]:

```
# Checking for duplicates
sum(di.duplicated())
```

Out[41]:

0

Issues

Data Quality Issues

> *The identified data quality issues are listed below.*
>
> 1. *The retweeted column of the dj(twitter-json) dataframe contains only False values. A variable is expected to vary. But this variable does not vary and as such should not be a part of the dataframe.*
> 2. *The created_at column should be datetime and not string. Specifically, the data in it is supposed to be splitted into three columns with the first two being string and the last being datetime*
> 3. *In the dt dataframe, the number of nulls in retweeted_status_id, retweeted_status_timestamp, retweeted_status_user_id, in_reply_to_status_id, in_reply_to_user_id are too many. I don't think such columns should be included in the dataset.*
> 4. *In the dt dataframe, the timestamp data type should be datetime not object. It should also be splitted to other columns such as year, month, day, and time. It looks similar to created_at column of dj dataframe.*
> 5. *The text column looks similar to the full_text column of dj dataframe, so one of them may be dropped after combining the two dataframes.*
> 6. *In the dt dataframe, checking the rating_denominator column shows that one of the denominators is 0. This shouldn't be.*
> 7. *Also, the names in this dt dataframe has 775 None and 55 'a'. There are other names like an, the, this, quite, interesting, just, his, not, o, unacceptable, one, getting, and infuriating. These are not valid names. Moreover, they are all in non-title case whereas names should be in title case.*
> 8. *In the dt dataframe, the columns containing None should have been represented by a blank that is NaN (Not a Number) since None is not a name.*
> 9. *Also, the column for columns for dog stages have some having dog stage values that are more than one. At least three of such are present. For instance, tweet_ids 85401017255294900, 85585145381401300, and 81777768676452300 all have two dog stages. I don't think a dog can be in two growth stages at a time.*
> 10. *Also in the name column, some names are not written in English Alphabet. There are at least 9 of such names in the dt dataframe.*
> 11. *There are outliers in the rating_numerator and rating_denominator columns of the dt dataframe. They may distort analysis if not attended to.*
> 12. *In the di (image predictions) dataframe some column names are not descriptive. Examples are p1, p2, p3, p1-config, p2_dog, etc*
> 13. *Also in this di dataframe, most of the names captured by p1,p2, and p3 are not in title case. Since they are proper names of dogs, I think they should be in title case.*
>
> ### *Tidiness Issues*
>
> *The identified tidiness issues are listed below.*
>
> 1. *The created_at column of the dj(twitter-json) dataframe and the timestamp column of the dt dataframe contain three different variables namely Day of the week, Month of the year, and time. These components should be in different columns since they are different variables.*
> 2. *In the dt dataframe, the four columns namely doggo, floofer, pupper, and puppo all captures a stage of dog. Since a dog is expected to belong to only one of these stages, only one variable is needed to capture them not four.*
> 3. *In the dt dataframe, there should be a column to capture actual dog rating (that is a standardized dog rating)*

4. Since all the three datasets contain related information and have the same unique identifier (tweet_id), they should be in the same sheet (dataframe). So, it will be necessary to combine them although some values will be lost because they don't have equal size.

Now let us proceed to addressing these quality and tidiness issues.

Cleaning the assessed data

>

> *This cleaning effort will address only the issues identified above. Cleaning shall proceed in the order dj, to dt, and then di. This was the order of assessing. Also, I will first tackle completeness issues (missing data issues), then the tidiness issues, and finally address the remaining data quality issues for each data frame. Shall we? Of course, we shall!*

In [42]:

```
# To begin I will make a copy of all my dataframes.
dj_clean = dj.copy()
dt_clean = dt.copy()
di_clean = di.copy()
# I will follow the Define, Code, and Test format.
```

Cleaning1 : Fixing missing data and extraneous variables in the dj_clean dataframe

Define: Drop the retweeted column because it contains only false values and therefore does not vary.

>

Code

In [43]:

```
#dj_clean.columns
# Dropping the retweeted column
dj_clean.drop('retweeted', axis=1, inplace=True)
```

Test

In [44]:

```
# Testing to see the effect of the code
dj_clean.columns
```

Out[44]:

```
Index(['tweet_id', 'created_at', 'retweet_count', 'favorite_count',
       'full_text', 'favorited'],
      dtype='object')
```

Cleaning 2: Removing Missing Values (NaNs) and unneeded columns in dt_clean dataframe

> *It is a requirement of this project that only original tweets should be used.Therefore, the values in retweeted_status_id, retweeted_status_timestamp, retweeted_status_user_id, in_reply_to_status_id, in_reply_to_user_id should not be used in analysis.*
>
> ***Define:***
>
> 1. *Remove the non-null values in the following columns: retweeted_status_id, retweeted_status_timestamp, retweeted_status_user_id, in_reply_to_status_id, in_reply_to_user_id.*

2. *Thereafter, drop the columns.*

Please note that the columns could have just been dropped without the first part but the project requires it.

Code

In [45]:

```
# Checking before coding
dt_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2356 entries, 0 to 2355
Data columns (total 17 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   tweet_id                    2356 non-null   int64  
 1   in_reply_to_status_id       78 non-null     float64
 2   in_reply_to_user_id         78 non-null     float64
 3   timestamp                   2356 non-null   object 
 4   source                      2356 non-null   object 
 5   text                        2356 non-null   object 
 6   retweeted_status_id         181 non-null    float64
 7   retweeted_status_user_id    181 non-null    float64
 8   retweeted_status_timestamp  181 non-null    object 
 9   expanded_urls               2297 non-null   object 
 10  rating_numerator            2356 non-null   int64  
 11  rating_denominator          2356 non-null   int64  
 12  name                        2356 non-null   object 
 13  doggo                       2356 non-null   object 
 14  floofer                     2356 non-null   object 
 15  pupper                      2356 non-null   object 
 16  puppo                       2356 non-null   object 
dtypes: float64(4), int64(3), object(10)
memory usage: 313.0+ KB
```

In [46]:

```
# Removing the non null values in in_reply_to_status_id, in_reply_to_user_id
# Here, I will use the tilda ~ shortcut. This is like the not operator.
# First, I will gather such values
reply = (~dt_clean.in_reply_to_status_id.isnull())
reply.sum() # This as expected is 78
# Second, I will remove the gathered values from the dataframe.
dt_clean = dt_clean[~reply]
dt_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2278 entries, 0 to 2355
Data columns (total 17 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   tweet_id                    2278 non-null   int64  
 1   in_reply_to_status_id       0 non-null      float64
 2   in_reply_to_user_id         0 non-null      float64
 3   timestamp                   2278 non-null   object 
 4   source                      2278 non-null   object 
 5   text                        2278 non-null   object 
 6   retweeted_status_id         181 non-null    float64
 7   retweeted_status_user_id    181 non-null    float64
 8   retweeted_status_timestamp  181 non-null    object 
 9   expanded_urls               2274 non-null   object 
 10  rating_numerator            2278 non-null   int64  
 11  rating_denominator          2278 non-null   int64  
 12  name                        2278 non-null   object 
 13  doggo                       2278 non-null   object 
 14  floofer                     2278 non-null   object 
 15  pupper                      2278 non-null   object 
 16  puppo                       2278 non-null   object 
dtypes: float64(4), int64(3), object(10)
memory usage: 320.3+ KB
```

In [47]:

```
# Removing the non null values in retweeted_status_id, retweeted_status_timestamp,
retweeted_status_user_id
# Here, I will use the tilda ~ shortcut. This is like the not operator.
# First, I will gather such values
retweet = (~dt_clean.retweeted_status_id.isnull())
retweet.sum() # This as expected is 181
# Second, I will remove the gathered values from the dataframe.
dt_clean = dt_clean[~retweet]
# To see the effect of the code
dt_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2097 entries, 0 to 2355
Data columns (total 17 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   tweet_id                    2097 non-null   int64
 1   in_reply_to_status_id       0 non-null      float64
 2   in_reply_to_user_id         0 non-null      float64
 3   timestamp                   2097 non-null   object
 4   source                      2097 non-null   object
 5   text                        2097 non-null   object
 6   retweeted_status_id         0 non-null      float64
 7   retweeted_status_user_id    0 non-null      float64
 8   retweeted_status_timestamp  0 non-null      object
 9   expanded_urls               2094 non-null   object
 10  rating_numerator            2097 non-null   int64
 11  rating_denominator          2097 non-null   int64
 12  name                        2097 non-null   object
 13  doggo                       2097 non-null   object
 14  floofer                     2097 non-null   object
 15  pupper                      2097 non-null   object
 16  puppo                       2097 non-null   object
dtypes: float64(4), int64(3), object(10)
memory usage: 294.9+ KB
```

In [48]:

```
# Next, I will remove the columns
dt_clean.drop(['in_reply_to_status_id',
                    'in_reply_to_user_id',
                    'retweeted_status_id',
                    'retweeted_status_user_id',
                    'retweeted_status_timestamp'], axis=1, inplace=True)
```

Test

In [49]:

```
# Checking to see that retweeted_status_id, retweeted_status_timestamp,
# retweeted_status_user_id, in_reply_to_status_id, in_reply_to_user_id no longer exist in the dt_c
lean dataframe.
dt_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2097 entries, 0 to 2355
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   tweet_id            2097 non-null   int64
 1   timestamp           2097 non-null   object
 2   source              2097 non-null   object
 3   text                2097 non-null   object
 4   expanded_urls       2094 non-null   object
 5   rating_numerator    2097 non-null   int64
 6   rating_denominator  2097 non-null   int64
 7   name                2097 non-null   object
 8   doggo               2097 non-null   object
 9   floofer             2097 non-null   object
 10  pupper              2097 non-null   object
```

```
 11  puppo               2097 non-null   object
dtypes: int64(3), object(9)
memory usage: 213.0+ KB
```

Cleaning 3: Changing names in dt_clean and di_clean to title

Define:

1. *In the dt_clean dataframe, change name values to title case. This includes name, doggo, floofer, pupper, and puppo*
2. *In the di_clean dataframe, change the values in p1,p2,p3 to title case*

Code

In [50]:

```
# Before the change
#dt_clean.name.value_counts()
#di_clean.p1.value_counts()
#di_clean.p2.value_counts()
#di_clean.p3.value_counts()
```

In [51]:

```
# Now, shall we?
dt_clean['name'] = [name.capitalize() for name in dt_clean['name']]
dt_clean['doggo'] = [doggo.capitalize() for doggo in dt_clean['doggo']]
dt_clean['floofer'] = [floofer.capitalize() for floofer in dt_clean['floofer']]
dt_clean['pupper'] = [pupper.capitalize() for pupper in dt_clean['pupper']]
dt_clean['puppo'] = [puppo.capitalize() for puppo in dt_clean['puppo']]
#dt_clean.name.value_counts()
di_clean['p1'] = [p1.capitalize() for p1 in di_clean['p1']]
di_clean['p2'] = [p2.capitalize() for p2 in di_clean['p2']]
di_clean['p3'] = [p3.capitalize() for p3 in di_clean['p3']]
di_clean['p3'].value_counts()
```

Out[51]:

```
Labrador_retriever    79
Chihuahua             58
Golden_retriever      48
Eskimo_dog            38
Kelpie                35
                      ..
Pier                   1
Pickup                 1
Valley                 1
Kerry_blue_terrier     1
Pot                    1
Name: p3, Length: 408, dtype: int64
```

Test

In [52]:

```
# Checking to see that all names in dt_clean and di_clean are now capitalized. This concerns name,
p1, p2, and p3.
dt_clean.name.value_counts()
dt_clean.doggo.value_counts()
dt_clean.floofer.value_counts()
dt_clean.pupper.value_counts()
dt_clean.puppo.value_counts()
di_clean.p1.value_counts()
di_clean.p2.value_counts()
di_clean.p3.value_counts()
```

Out[52]:

```
Labrador_retriever    79
Chihuahua             58
Golden_retriever      48
```

```
Eskimo_dog            38
Kelpie                35
                      ..
Pier                   1
Pickup                 1
Valley                 1
Kerry_blue_terrier     1
Pot                    1
Name: p3, Length: 408, dtype: int64
```

Cleaning 4: Dealing with None and Combining dog stages to one column

Define:

1. *In the dt_clean dataframe, replace None in the name and dog stages columns with a blank.*
2. *Thereafter, combine the dog stages to one column and drop the original dog stages columns and columns with more than one dog_stage.*
3. *Finally, fill all nulls with 'Not_available'. Do this also for the nulls in name.*
4. *Leave the remaining names intact since I don't know what to replace them with.*

Code

In [53]:

```
# Before the change
dt_clean.name.value_counts()
```

Out[53]:

```
None        603
A            55
Lucy         11
Charlie      11
Cooper       10
           ...
Leonidas      1
Fido          1
Chevy         1
Champ         1
Tango         1
Name: name, Length: 955, dtype: int64
```

In [54]:

```
# Replacing None with blank
dt_clean.name.replace('None', '', inplace=True)
dt_clean.doggo.replace('None', '', inplace=True)
dt_clean.floofer.replace('None', '', inplace=True)
dt_clean.pupper.replace('None', '', inplace=True)
dt_clean.puppo.replace('None', '', inplace=True)
```

In [55]:

```
# Test for the effect
#dt_clean.name.value_counts()
#dt_clean.sample(20, replace=False) # Change effected.
```

In [56]:

```
# combining the dog stages column and separating those with
dt_clean['dog_stage'] = dt_clean[['doggo', 'floofer', 'pupper', 'puppo']].agg(''.join, axis=1)
dt_clean.dog_stage.replace('', 'Not_available', inplace=True)
dt_clean.dog_stage.value_counts()
```

Out[56]:

```
Not_available    1761
Pupper            221
Doggo              72
Puppo              23
```

```
Puppo              23
Floofer             9
DoggoPupper         9
DoggoPuppo          1
DoggoFloofer        1
Name: dog_stage, dtype: int64
```

In [57]:

```
# Dropping dog_stage values with  more than one dog_stages
dt_clean.drop(dt_clean[dt_clean.dog_stage=='DoggoPupper'].index,inplace=True)
dt_clean.drop(dt_clean[dt_clean.dog_stage=='DoggoPuppo'].index,inplace=True)
dt_clean.drop(dt_clean[dt_clean.dog_stage=='DoggoFloofer'].index,inplace=True)
```

In [58]:

```
# Checking to see the effect
dt_clean.dog_stage.value_counts()
```

Out[58]:

```
Not_available    1761
Pupper            221
Doggo              72
Puppo              23
Floofer             9
Name: dog_stage, dtype: int64
```

In [59]:

```
# Dropping the original dog stages columns
dt_clean.head(0)
dt_clean.drop(['doggo','floofer','pupper','puppo'], axis=1, inplace=True)
dt_clean.head(0)
```

Out[59]:

tweet_id	timestamp	source	text	expanded_urls	rating_numerator	rating_denominator	name	dog_stage

In [60]:

```
# Replacing blanks in name with 'Not_Available'
dt_clean.name.replace('', 'Not_Available', inplace=True)
dt_clean.name.value_counts()
```

Out[60]:

```
Not_Available    597
A                 55
Charlie           11
Lucy              11
Oliver            10
                ...
Chevy              1
Champ              1
Chaz               1
Ashleigh           1
Tango              1
Name: name, Length: 952, dtype: int64
```

Test

In [61]:

```
# Using samples to check for the effects
dt_clean.sample(5, replace=False)
```

Out[61]:

	tweet_id	timestamp	source	text	expanded_urls	rat
2123	670385711116361728	2015-11-27 23:36:23 +0000	<a href="http://twitter.com/download/iphone" r...	Meet Larry. He's a Panoramic Benzoate. Can sho...	https://twitter.com/dog_rates/status/670385711...	
112	870804317367881728	2017-06-03 00:48:22 +0000	<a href="http://twitter.com/download/iphone" r...	Real funny guys. Sending in a pic without a do...	https://twitter.com/dog_rates/status/870804317...	
1878	675047298674663426	2015-12-10 20:19:52 +0000	<a href="http://twitter.com/download/iphone" r...	This is a fluffy albino Bacardi Columbia mix. ...	https://twitter.com/dog_rates/status/675047298...	
809	771500966810099713	2016-09-02 00:12:18 +0000	<a href="http://twitter.com/download/iphone" r...	This is Dakota. He's just saying hi. That's al...	https://twitter.com/dog_rates/status/771500966...	
1999	672523490734551040	2015-12-03 21:11:09 +0000	<a href="http://twitter.com/download/iphone" r...	When she says she'll be ready in a minute but ...	https://twitter.com/dog_rates/status/672523490...	

Cleaning 5: Splitting created_at column of the dj_clean and Merging the three datasets

Define:

1. *Split the created_at column of dj_clean to month, day, and time.*
2. *Give dexcriptive names to the non-descriptive columns in dj_clean (that is p1,p2,p3 and the others)*
3. *Merge the three dataframes into one dataframe named df_all*

Code

In [62]:

```
# Before splitting the created_at column of dj_clean
dj_clean.created_at.value_counts()
```

Out[62]:

```
Thu Feb 25 19:04:13 +0000 2016    1
Thu Nov 26 22:16:09 +0000 2015    1
Tue Aug 30 23:58:40 +0000 2016    1
Sun Jul 17 01:05:25 +0000 2016    1
Mon Jan 25 02:17:57 +0000 2016    1
                                 ..
Wed Mar 30 15:34:51 +0000 2016    1
Wed Mar 09 03:45:22 +0000 2016    1
Wed Mar 09 22:24:31 +0000 2016    1
Wed Nov 25 17:49:14 +0000 2015    1
Wed Jun 01 23:52:28 +0000 2016    1
Name: created_at, Length: 2354, dtype: int64
```

In [63]:

```
dt_clean.timestamp.value_counts()
```

Out[63]:

```
2016-02-17 02:17:19 +0000    1
2017-02-09 01:27:41 +0000    1
2016-12-04 19:02:24 +0000    1
2016-09-26 17:29:48 +0000    1
2016-03-14 00:49:23 +0000    1
```

```
                                  ..
2016-01-18 01:38:15 +0000    1
2017-07-11 00:00:02 +0000    1
2016-02-09 03:35:31 +0000    1
2016-10-03 15:42:44 +0000    1
2015-12-06 22:54:44 +0000    1
Name: timestamp, Length: 2086, dtype: int64
```

> *Taking another look at the created_at column, I realized it is similar to the timestamp column of dt_clean. Moreover, the timestamp column is much cleaner and will be easier to work with with minimal code than the created_at column. Since I will later merge the dataframes, I will at this stage drop created_at column and focus on timestamp so as to guide against unnecessary duplication. Thanks for the inconvinience, if any.*

In [64]:

```
# Dropping the created_at column of dj_clean
dj_clean.drop('created_at', axis=1, inplace=True)
# Checking for the effect
dj_clean.head(0)
```

Out[64]:

tweet_id	retweet_count	favorite_count	full_text	favorited

In [65]:

```
# Convenverting timestamp of dt_clean to datetime datatype
# Checking before changing the data type
dt_clean.dtypes
dt_clean['timestamp'] = pd.to_datetime(dt_clean['timestamp'])
# Checking to see the effect
dt_clean.dtypes
```

Out[65]:

```
tweet_id                              int64
timestamp               datetime64[ns, UTC]
source                               object
text                                 object
expanded_urls                        object
rating_numerator                      int64
rating_denominator                    int64
name                                 object
dog_stage                            object
dtype: object
```

In [66]:

```
# Splitting the timestamp column
# Creating the year column
dt_clean['year'] = dt_clean['timestamp'].dt.year
# Creating the month column
dt_clean['month'] = dt_clean['timestamp'].dt.month
# Creating the day column
dt_clean['day'] = dt_clean['timestamp'].dt.day
# Creating the time column
dt_clean['time'] = dt_clean['timestamp'].dt.time
# Creating the week_day column to capture days in string
dt_clean['week_day'] = dt_clean['timestamp'].dt.dayofweek
days = {0:'Sun',1:'Mon',2:'Tues',3:'Weds',4:'Thurs',5:'Fri',6:'Sat'}
dt_clean['week_day'] = dt_clean['week_day'].apply(lambda x: days[x])
# Checking
dt_clean.head(2)
```

Out[66]:

	tweet_id	timestamp	source	text	expanded_urls	rati

	tweet_id	timestamp	source	text	expanded_urls	rati
0	892420643555336193	2017-08-01 16:23:56+00:00	<a href="http://twitter.com/download/iphone" r...	This is Phineas. He's a mystical boy. Only eve...	https://twitter.com/dog_rates/status/892420643...	
1	892177421306343426	2017-08-01 00:17:27+00:00	<a href="http://twitter.com/download/iphone" r...	This is Tilly. She's just checking pup on you....	https://twitter.com/dog_rates/status/892177421...	

In [67]:

```
# Creating year_month
dt_clean['year_month'] = dt_clean['timestamp'].dt.month
months = {1:'Jan',2:'Feb',3:'Mar',4:'Apr',5:'May',6:'Jun',7:'Jul',8:'Aug',9:'Sep',10:'Oct',11:'Nov',
12:'Dec'}
dt_clean['year_month'] = dt_clean['year_month'].apply(lambda x: months[x])
dt_clean.head(1)
```

Out[67]:

	tweet_id	timestamp	source	text	expanded_urls	rati
0	892420643555336193	2017-08-01 16:23:56+00:00	<a href="http://twitter.com/download/iphone" r...	This is Phineas. He's a mystical boy. Only eve...	https://twitter.com/dog_rates/status/892420643...	

In [68]:

```
# Dropping dj_clean full_text column and renaming dt_clean text column to tweet
dj_clean.drop('full_text', axis=1,inplace=True)
dt_clean.rename(columns={'text':'tweet'}, inplace=True)
# Checking
dj_clean.head(1)
dt_clean.head(1)
```

Out[68]:

	tweet_id	timestamp	source	tweet	expanded_urls	rati
0	892420643555336193	2017-08-01 16:23:56+00:00	<a href="http://twitter.com/download/iphone" r...	This is Phineas. He's a mystical boy. Only eve...	https://twitter.com/dog_rates/status/892420643...	

In [69]:

```
# Now we give descriptive names to the variables in di_clean dataframe
di_clean.rename(columns={'p1':'predicted_image1',
                        'p2': 'predicted_image2',
                        'p3':'predicted_image3',
                        'p1_conf':'confidence_on_image1',
                        'p2_conf':'confidence_on_image2',
                        'p3_conf':'confidence_on_image3',
                        'p1_dog':'image1_dog?',
                        'p2_dog':'image2_dog?',
                        'p3_dog':'image3_dog?'}, inplace=True)
```

In [70]:

```
# Checking to see change in the names
di_clean.head(1)
```

Out[70]:

	tweet_id	jpg_url	img_num	predicted_image1	confidence_on_image1	imag
0	666020888022790149	https://pbs.twimg.com/media/CT4udn0WwAA0aMy.jpg	1	Welsh_springer_spaniel	0.465074	

In [71]:

```
# Before combining on tweet_id, I want to check for equality in id and remove those id that are in
one dataframe and not in
# the other
missing1 = (~dj_clean.tweet_id.isin(list(dt_clean.tweet_id)))
missing1.sum() # This is 268, we need to remove these.
dj_clean = dj_clean[~missing1]
```

In [72]:

```
# Checking to see that they are equal in rows.
dt_clean.shape,dj_clean.shape
# Now merging the dt_clean and dj_clean dataframes to form df_all1
df_all1 = pd.merge(dt_clean, dj_clean, on='tweet_id', how = 'inner')
# Checking to see the effect
df_all1.head(2)
```

Out[72]:

	tweet_id	timestamp	source	tweet	expanded_urls	rati
0	892420643555336193	2017-08-01 16:23:56+00:00	<a href="http://twitter.com/download/iphone" r...	This is Phineas. He's a mystical boy. Only eve...	https://twitter.com/dog_rates/status/892420643...	
1	892177421306343426	2017-08-01 00:17:27+00:00	<a href="http://twitter.com/download/iphone" r...	This is Tilly. She's just checking pup on you....	https://twitter.com/dog_rates/status/892177421...	

In [73]:

```
# Next, I will merge df_all1 with di_clean to form df_all
# I want to check for id that are di_clean but not in df_all1
missing2 = (~di_clean.tweet_id.isin(list(df_all1.tweet_id)))
missing2.sum()
# This is a whopping 114
# Now I will remove them
di_clean = di_clean[~missing2]
# Checking
di_clean.shape,df_all1.shape # With what I have here, It is evident that we will lose some more da
ta points
# because they are in only one of the dataframes that I want to merge. But I shall go ahead
# Now merging the di_clean and df_all1 dataframes to form df_all
df_all = pd.merge(df_all1, di_clean, on='tweet_id', how = 'inner')
# Checking to see the effect
df_all.head(2)
```

Out[73]:

	tweet_id	timestamp	source	tweet	expanded_urls	rati
0	892420643555336193	2017-08-01 16:23:56+00:00	<a href="http://twitter.com/download/iphone" r...	This is Phineas. He's a mystical boy. Only eve...	https://twitter.com/dog_rates/status/892420643...	

	tweet_id	timestamp	source	tweet	expanded_urls	rati
1	892177421306343426	2017-08-01 00:17:27+00:00	<a href="http://twitter.com/download/iphone" r...	This is Tilly. She's just checking pup on you....	https://twitter.com/dog_rates/status/892177421...	

2 rows × 29 columns

Test

In [74]:

```
# Listing out the columns after merging the three dataframes to test that we have achieved the aim
#df_all.dtypes,
#df_all.shape
#column_lister(df_all)
df_all.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1961 entries, 0 to 1960
Data columns (total 29 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   tweet_id              1961 non-null   int64
 1   timestamp             1961 non-null   datetime64[ns, UTC]
 2   source                1961 non-null   object
 3   tweet                 1961 non-null   object
 4   expanded_urls         1961 non-null   object
 5   rating_numerator      1961 non-null   int64
 6   rating_denominator    1961 non-null   int64
 7   name                  1961 non-null   object
 8   dog_stage             1961 non-null   object
 9   year                  1961 non-null   int64
 10  month                 1961 non-null   int64
 11  day                   1961 non-null   int64
 12  time                  1961 non-null   object
 13  week_day              1961 non-null   object
 14  year_month            1961 non-null   object
 15  retweet_count         1961 non-null   int64
 16  favorite_count        1961 non-null   int64
 17  favorited             1961 non-null   bool
 18  jpg_url               1961 non-null   object
 19  img_num               1961 non-null   int64
 20  predicted_image1      1961 non-null   object
 21  confidence_on_image1  1961 non-null   float64
 22  image1_dog?           1961 non-null   bool
 23  predicted_image2      1961 non-null   object
 24  confidence_on_image2  1961 non-null   float64
 25  image2_dog?           1961 non-null   bool
 26  predicted_image3      1961 non-null   object
 27  confidence_on_image3  1961 non-null   float64
 28  image3_dog?           1961 non-null   bool
dtypes: bool(4), datetime64[ns, UTC](1), float64(3), int64(9), object(12)
memory usage: 406.0+ KB
```

Cleaning 6: Dropping 0 denominator and creating a standardized rating column

Define:

1. *Drop the tweet with rating_denominator equal zero (0).*
2. *Create rating column that is more comparative*
3. *Save the df_all dataframe to twitter_archive_master.csv*

Code

In [75]:

```
# Checking to see if the zero rating denominator still exists
df_all.rating_denominator.value_counts()
# It doesn't. It must have been dropped during merging or thereabout. Now let us continue
```

Out[75]:

```
10      1944
50         3
80         2
11         2
170        1
150        1
120        1
110        1
90         1
70         1
40         1
20         1
7          1
2          1
Name: rating_denominator, dtype: int64
```

In [76]:

```python
# Creating a rating column that is a division of standadized rating_numerator mean by rating_denom
inator_mean
# First we get the different means
mean1 = df_all.rating_numerator.mean()
mean2 = df_all.rating_denominator.mean()
mean1,mean2
```

Out[76]:

```
(12.228454869964304, 10.479857215706271)
```

In [77]:

```python
# Second we create the desired rating column
df_all['rating'] = df_all['rating_numerator'] * mean1 / df_all['rating_denominator'] * mean2
```

Test

In [78]:

```python
#column_lister(df_all)
#df_all.info()
df_all.sample(5)
```

Out[78]:

	tweet_id	timestamp	source	tweet	expanded_urls
1474	675740360753160193	2015-12-12 18:13:51+00:00	<a href="http://twitter.com/download/iphone" r...	Here's a pupper licking in slow motion. 12/10 ...	https://twitter.com/dog_rates/status/675740360...
973	709042156699303936	2016-03-13 15:43:18+00:00	<a href="http://twitter.com/download/iphone" r...	This is Klevin. He's addicted to sandwiches (y...	https://twitter.com/dog_rates/status/709042156...
1849	667915453470232577	2015-11-21 04:00:28+00:00	<a href="http://twitter.com/download/iphone" r...	Meet Otis. He is a Peruvian Quartzite. Pic spo...	https://twitter.com/dog_rates/status/667915453...
1924	666447344410484738	2015-11-17 02:46:43+00:00	<a href="http://twitter.com/download/iphone" r...	This is Scout. She is a black Downton Abbey. I...	https://twitter.com/dog_rates/status/666447344...
1861	667793409583771648	2015-11-20	<a href="http://twitter.com/download/iphone"	Dogs only please. Small cows	https://twitter.com/dog_rates/status/667793409

1961	667793409583771648	19:55:30+00:00	rel="http://twitter.com/download/iphone	Small cows and other non can...	https://twitter.com/dog_rates/status/667793409...
	tweet_id	timestamp	source	tweet	expanded_urls

5 rows × 30 columns

Storing the cleaned data

> *Now I will store df_all into twitter_archive_master.csv*

In [79]:

```
# I have decided to keep timestamp, rating_numerator, and rating_denominator in the df_all datafra
me. They may be needed later
df_all.to_csv('twitter_archive_master.csv', index=False)
```

Analyzing the stored data

> *Now I will begin to analyze the data. I will first pose some questions in order to guide the analysis. Hope you are not yet tired?*
>
> *Research questions:*
>
> 1. *Do retweet_count, and favorite_count vary overtime?*
> 2. *What are the most popular names of dog?*
> 3. *What are the most popular dog_stages?*
> 4. *What variables affect rating?*
>
> *Visualizations shall be carried out where deemed useful. Moreover, some other columns may be created to make things easier. Now let's start. I'm very excited.*

In [80]:

```
# Setting seaborn grid
sb.set_style('darkgrid')
```

In [81]:

```
# Loading the dataset
df = pd.read_csv('twitter_archive_master.csv')
```

In [82]:

```
# Brief assessment of the dataframe
#df.head()
#df.shape
df.describe()
```

Out[82]:

	tweet_id	rating_numerator	rating_denominator	year	month	day	retweet_count	favorite_count
count	1.961000e+03	1961.000000	1961.000000	1961.000000	1961.000000	1961.000000	1961.000000	1961.000000
mean	7.357626e+17	12.228455	10.479857	2015.844977	7.168281	16.033146	2769.170321	8907.657828
std	6.751967e+16	41.739741	6.870651	0.699443	4.121154	8.936649	4682.802592	12238.973877
min	6.660209e+17	0.000000	2.000000	2015.000000	1.000000	1.000000	16.000000	81.000000
25%	6.758228e+17	10.000000	10.000000	2015.000000	3.000000	8.000000	624.000000	1971.000000
50%	7.084699e+17	11.000000	10.000000	2016.000000	7.000000	16.000000	1360.000000	4110.000000
75%	7.877176e+17	12.000000	10.000000	2016.000000	11.000000	24.000000	3227.000000	11363.000000
max	8.924206e+17	1776.000000	170.000000	2017.000000	12.000000	31.000000	79515.000000	132810.000000

tweet_id	rating_numerator	rating_denominator	year	month	day	retweet_count	favorite_count	i

In [83]:

```
column_lister(df)
```

```
0 tweet_id
1 timestamp
2 source
3 tweet
4 expanded_urls
5 rating_numerator
6 rating_denominator
7 name
8 dog_stage
9 year
10 month
11 day
12 time
13 week_day
14 year_month
15 retweet_count
16 favorite_count
17 favorited
18 jpg_url
19 img_num
20 predicted_image1
21 confidence_on_image1
22 image1_dog?
23 predicted_image2
24 confidence_on_image2
25 image2_dog?
26 predicted_image3
27 confidence_on_image3
28 image3_dog?
29 rating
```

Do retweet_count, and favorite_count vary overtime?

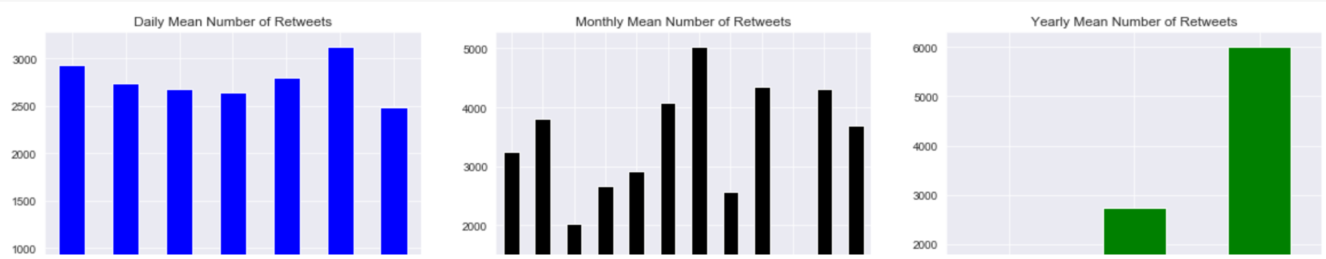
In [84]:

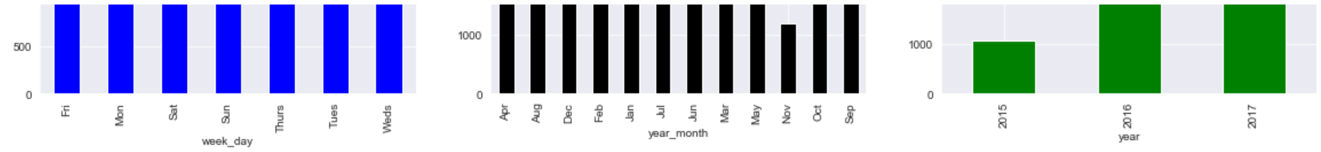
```
# Examining retweet count and favourite count based on months
daily_retweet_mean = df.groupby('week_day')['retweet_count'].mean()
monthly_retweet_mean = df.groupby('year_month')['retweet_count'].mean()
yearly_retweet_mean = df.groupby('year')['retweet_count'].mean()

#monthly_retweet_mean,
#monthly_favorite_mean
```

In [85]:

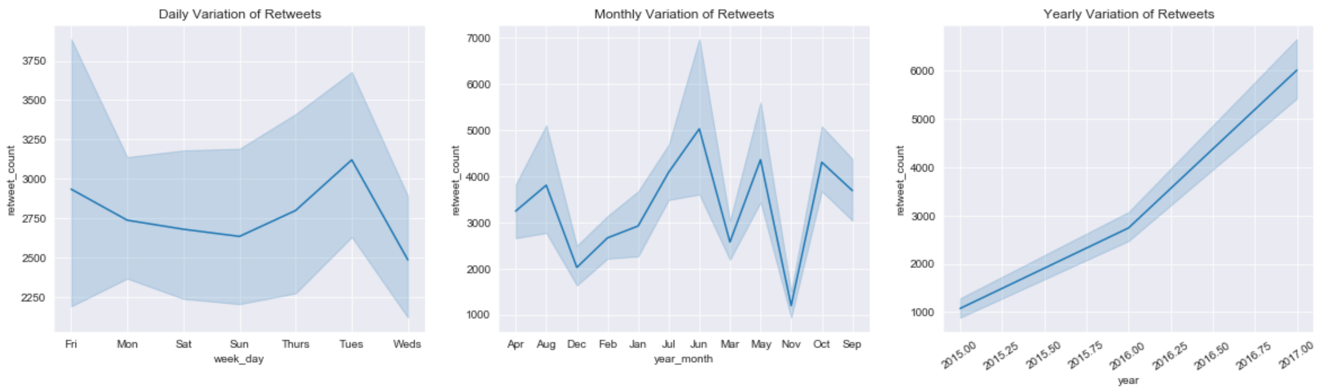
```
# Visualization of Retweets 1
plt.figure(figsize=(20,5))
plt.subplot(1,3,1)
daily_retweet_mean.plot(kind='bar',color='b')
plt.title('Daily Mean Number of Retweets')
plt.subplot(1,3,2)
monthly_retweet_mean.plot(kind='bar',color='black')
plt.title('Monthly Mean Number of Retweets')
plt.subplot(1,3,3)
yearly_retweet_mean.plot(kind='bar',color='g')
plt.title('Yearly Mean Number of Retweets');
```





In [86]:

```
# Visualization of retweets 2
plt.figure(figsize=(20,5))
plt.subplot(1,3,1)
sb.lineplot(y='retweet_count', x='week_day', data=df)
plt.title('Daily Variation of Retweets')
plt.subplot(1,3,2)
sb.lineplot(y='retweet_count', x='year_month', data=df)
plt.title('Monthly Variation of Retweets')
plt.subplot(1,3,3)
sb.lineplot(y='retweet_count', x='year', data=df)
plt.title('Yearly Variation of Retweets')
plt.xticks(rotation=35);
```



> *It can be seen that retweets vary greatly. Retweets were at their lowest in 2015, had a kink in 2016 and has continued to rise till 2017. Retweets are lowest in November and highest in June. Generally November, December, January, February, and March are associated with low levels of retweets while the months of May, June, July, August, and October are associated with high levels of retweets. Perhaps socialization is more in this period of time because it is the summer. Also, low retweets may be associated with the winter or cooler months. On a daily basis, retweets are highest on Tuesdays and lowest on Wednesdays.*

In [87]:

```
# Visualizing favourites count
daily_favorite_mean = df.groupby('week_day')['favorite_count'].mean()
monthly_favorite_mean = df.groupby('year_month')['favorite_count'].mean()
yearly_favorite_mean = df.groupby('year')['favorite_count'].mean()
yearly_favorite_mean
```

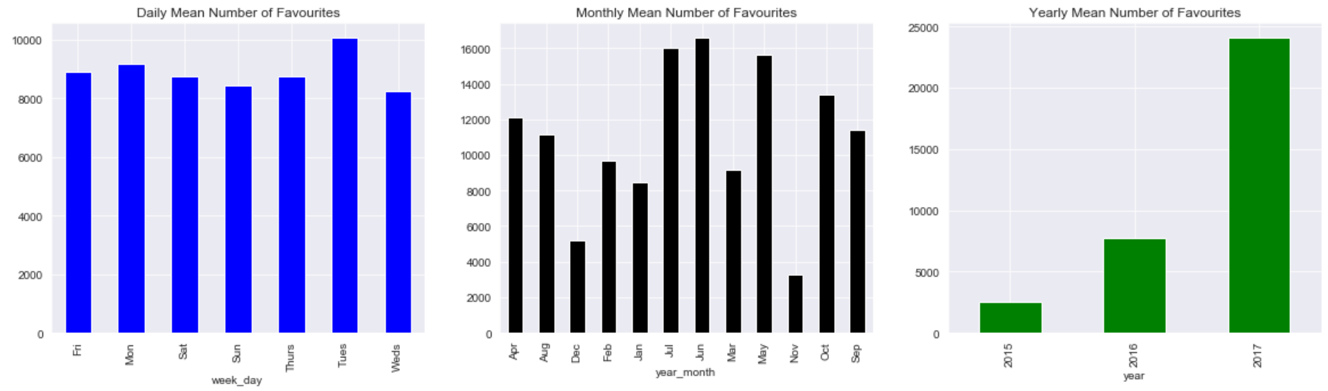
Out[87]:

```
year
2015     2492.277863
2016     7734.215707
2017    24072.076923
Name: favorite_count, dtype: float64
```

In [88]:

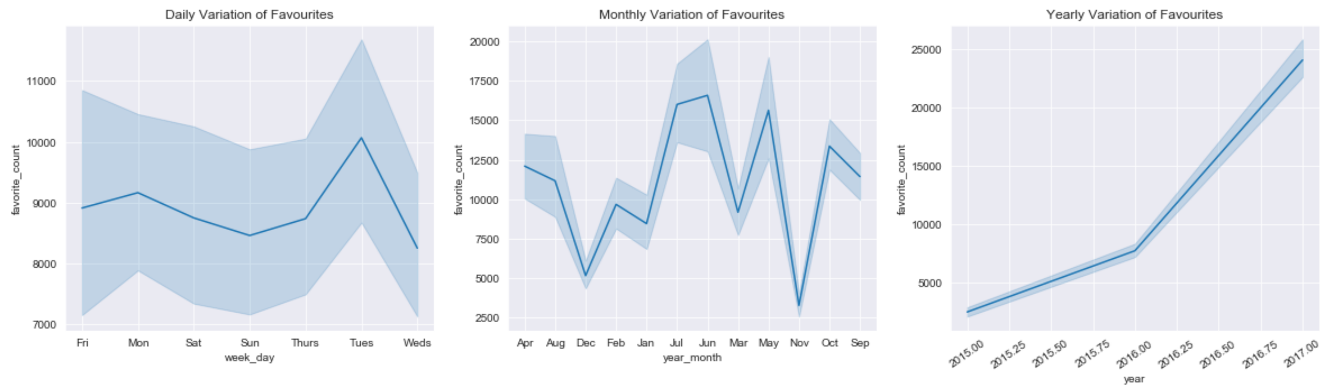
```
# Visualization of Favourites 1
plt.figure(figsize=(20,5))
plt.subplot(1,3,1)
daily_favorite_mean.plot(kind='bar',color='b')
plt.title('Daily Mean Number of Favourites')
plt.subplot(1,3,2)
monthly_favorite_mean.plot(kind='bar',color='black')
```

```
plt.title('Monthly Mean Number of Favourites')
plt.subplot(1,3,3)
yearly_favorite_mean.plot(kind='bar',color='g')
plt.title('Yearly Mean Number of Favourites');
```



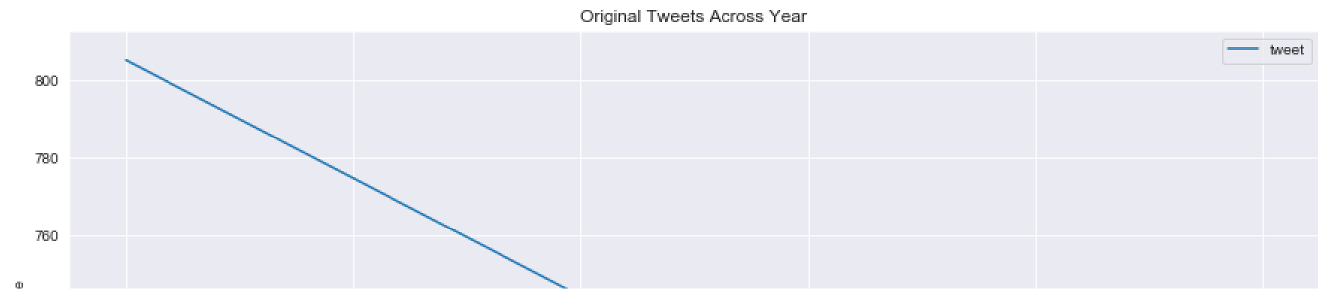
In [89]:

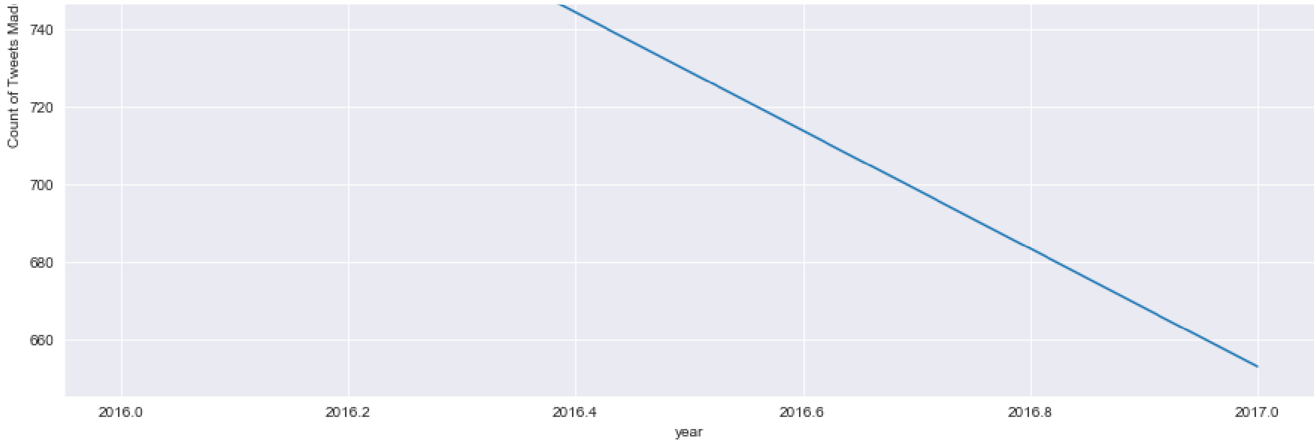
```
# Visualization of retweets 2
plt.figure(figsize=(20,5))
plt.subplot(1,3,1)
sb.lineplot(y='favorite_count', x='week_day', data=df)
plt.title('Daily Variation of Favourites')
plt.subplot(1,3,2)
sb.lineplot(y='favorite_count', x='year_month', data=df)
plt.title('Monthly Variation of Favourites')
plt.subplot(1,3,3)
sb.lineplot(y='favorite_count', x='year', data=df)
plt.title('Yearly Variation of Favourites')
plt.xticks(rotation=35);
```



In [118]:

```
# Checking yearly tweets overtime
df1 = df[['year', 'tweet']].groupby(['year']).count()
df1.tweet.nlargest(5)
# Use moving averages to smooth the line
df1['tweet'] = df1['tweet'].rolling(window=2).mean()
# Plot
df1.plot(figsize=(15, 8), title='Original Tweets Across Year')
plt.ylabel('Count of Tweets Made')
plt.savefig('yearly_tweets.png');
```





In [127]:

```
# Tweets per year
df2 = df[['year', 'tweet']].groupby(['year']).count()
df2
```

Out[127]:

	tweet
year	
2015	655
2016	955
2017	351

In [131]:

```
# Retweets per year
df2 = df[['year', 'retweet_count']].groupby(['year']).sum()
df2
```

Out[131]:

	retweet_count
year	
2015	705898
2016	2617771
2017	2106674

In [132]:

```
# Favorites per year
df3 = df[['year', 'favorite_count']].groupby(['year']).sum()
df3
```

Out[132]:

	favorite_count
year	
2015	1632442
2016	7386176
2017	8449299

> *It can be seen that the pattern of favorite_count is similar to that of retweet_count across years, months, and days.*

> *With regards to original tweets, most tweets were made in 2016 and reduced drastically in 2017. But retweets were still substantial in 2017. Favourites were highest in 2017. This portrays that they keep digging the achhives and retweeting. Now on to the next question.*
>
> ***What are the most popular breeds of dog?***

In [138]:

```
# The breeds of dog are captured in name
df.name.value_counts().nlargest(12)
# Now I need to omit the Not Availables
```

Out[138]:

```
Not_Available    519
A                 55
Charlie           11
Lucy              10
Oliver            10
Cooper            10
Penny              9
Tucker             9
Winston            8
Sadie              8
Daisy              7
Lola               7
Name: name, dtype: int64
```

In [139]:

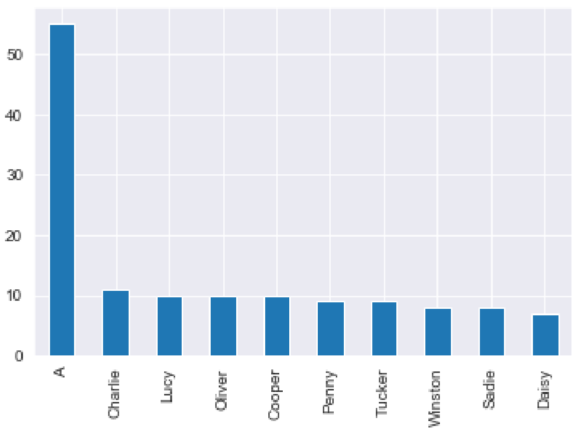
```
popular_dogs = df.name.value_counts().nlargest(12)[1:-1]
popular_dogs
# Since we have decided to leave such names as 'A', 'The', etc, then the ten most popular names
# are displayed below.
```

Out[139]:

```
A          55
Charlie    11
Lucy       10
Oliver     10
Cooper     10
Penny       9
Tucker      9
Winston     8
Sadie       8
Daisy       7
Name: name, dtype: int64
```

In [140]:

```
popular_dogs.plot(kind='bar')
plt.savefig('popular_dogs_bar.png');
```

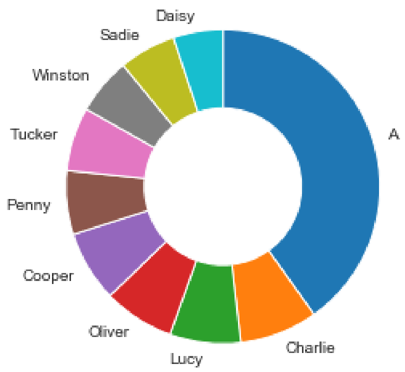


In [141]:

```
# Making a waffle for it
fig = plt.figure(FigureClass = Waffle,
                rows= 7,
                columns = 11,
                values = list(popular_dogs.values),
                labels = list(popular_dogs.index),
                figsize=(7,6),
                icons = 'dog',
                icon_legend = True,
                legend = {'loc':'upper left',
                         'bbox_to_anchor':(1.1,1)})
plt.savefig('popular_dogs_waffle.png');
```

In [142]:

```
# Making a donut to show popular_dogs
plt.pie(popular_dogs,labels=popular_dogs.index,startangle=90,counterclock=False,wedgeprops={'width'
:0.5})
plt.axis('square')
plt.savefig('popular_dogs_donut.png');
```



> *As shown by the grouping, the bar chart, the Waffle plot,and the donut, the five most popular dog names are A, Charlie, Oliver, Cooper, and Lucy. Now, let us move on to the next question.*
>
> ***What are the most popular dog_stages?***

In [143]:

```
# Dog stages are captured in the dog_stage variable
df.dog_stage.value_counts()
```

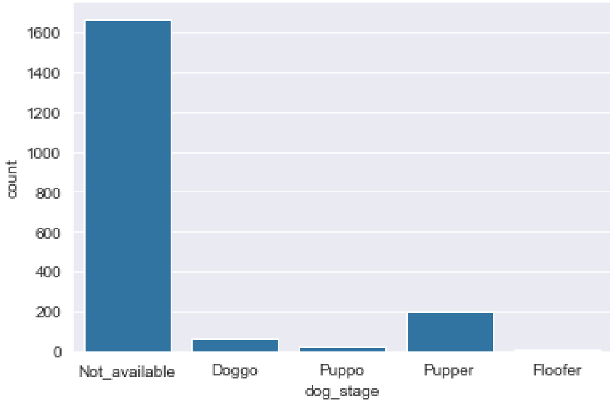
Out[143]:

```
Not_available    1668
Pupper            201
Doggo              63
```

```
Puppo              22
Floofer             7
Name: dog_stage, dtype: int64
```

In [144]:

```
# First view with the Not Availables
sb.countplot(df['dog_stage'], color= sb.color_palette()[0])
plt.savefig('dog_stage_bar.png');
```



In [146]:

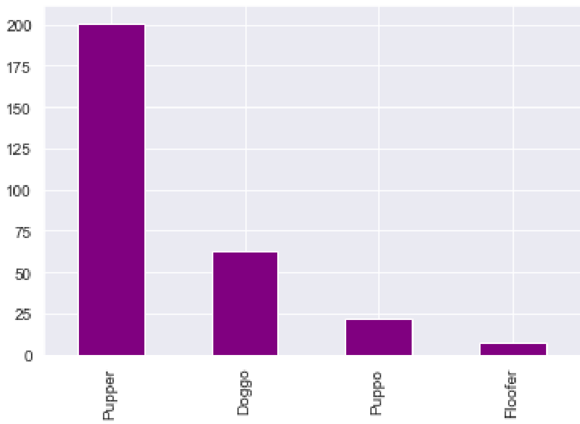
```
# Now I will remove the Not Availables
popular_dog_stage = df.dog_stage.value_counts()[1:]
popular_dog_stage
```

Out[146]:

```
Pupper      201
Doggo        63
Puppo        22
Floofer       7
Name: dog_stage, dtype: int64
```

In [147]:

```
# Now we visualize it in bars and waffle
popular_dog_stage.plot(kind='bar', color = 'purple')
plt.savefig('dog_stage_bar_ordered.png');
```



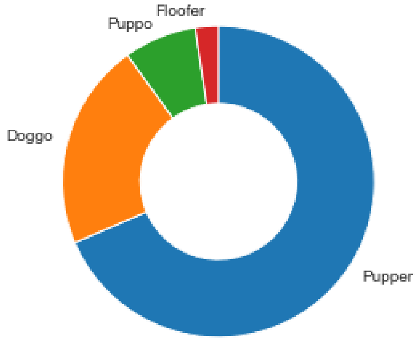
In [148]:

```
# Making a waffle for it
fig = plt.figure(FigureClass = Waffle,
                 rows= 7,
                 columns = 11,
                 values = list(popular_dog_stage.values),
                 labels = list(popular_dog_stage.index),
                 figsize=(7,6),
```

```
                 icons = 'chair',
                 icon_legend = True,
                 legend = {'loc':'upper left',
                           'bbox_to_anchor':(1.1,1)});
plt.savefig('dog_stage_waffle.png');
```

In [149]:

```
# Making a donut to show popular_dog_stage
plt.pie(popular_dog_stage,labels=popular_dog_stage.index,startangle=90,counterclock=False,wedgeprop
s={'width':0.5})
plt.axis('square')
plt.savefig('dog_stage_donut.png');
```



> *As shown by the grouping, the bar chart, the Waffle plot,and the donut, the most popular dog stage is Pupper followed by Doggo. Now, let us move on to the last question.*
>
> ***What variables affect rating?***

In [167]:

```
# First I will describe rating
df.rating.describe()
# Next I will categorize rating for visualizations by creating a performance column
```

Out[167]:

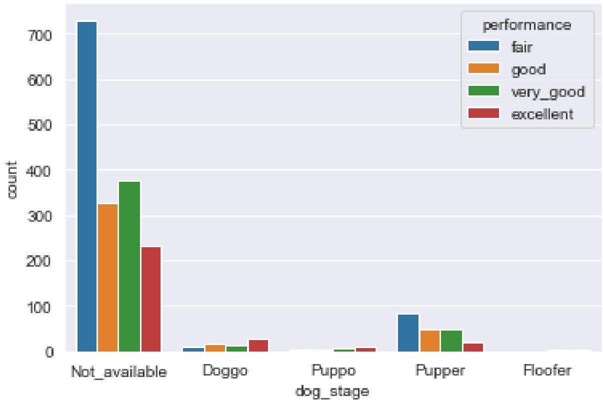
```
count      1961.000000
mean        149.906561
std         525.556286
min           0.000000
25%         128.152461
50%         140.967707
75%         153.782953
max       22759.877075
Name: rating, dtype: float64
```

In [169]:

```
bin_edges = [0.00,128.16,140.97,153.79,22759.88]
bin_names = ['fair','good','very_good','excellent']
df['performance'] = pd.cut(df['rating'], bin_edges, labels=bin_names)
```

In [171]:

```
# Visualizing dog_stage with performance
sb.countplot(data=df,x='dog_stage', hue='performance')
plt.savefig('dog_stage_performance.png');
```



In [172]:

```
# Looking at the counts
df.groupby(['dog_stage','performance'])['performance'].count()
```

Out[172]:

```
dog_stage      performance
Doggo          fair             9
               good            16
               very_good       12
               excellent       26
Floofer        fair             1
               good             1
               very_good        2
               excellent        3
Not_available  fair           730
               good           328
               very_good      378
               excellent      231
Pupper         fair            85
               good            49
               very_good       47
               excellent       20
Puppo          fair             4
               good             2
               very_good        6
               excellent       10
Name: performance, dtype: int64
```

> *In terms of rating, dogs at the doggo and Pupper stages appear to have more excellent rating than other named dogs. Both most dogs with excellent rating are not named.*

In [174]:

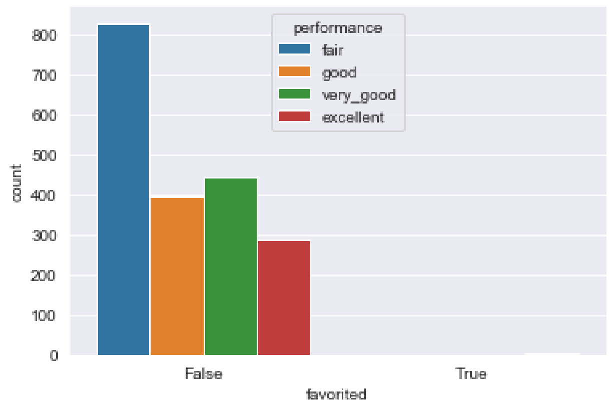
```
df.favorited.value_counts()
```

Out[174]:

```
False    1956
True        5
Name: favorited, dtype: int64
```

In [173]:

```
# Visualizing dog_stage with performance
sb.countplot(data=df,x='favorited', hue='performance')
plt.savefig('favorited_performance.png');
```



In [175]:

```
# Looking at the counts
df.groupby(['favorited','performance'])['performance'].count()
```

Out[175]:

```
favorited  performance
False      fair           828
           good           395
           very_good      445
           excellent      287
True       fair             1
           good             1
           very_good        0
           excellent        3
Name: performance, dtype: int64
```

> *Being favorited does not appear to have anything to do with high performance*

In [188]:

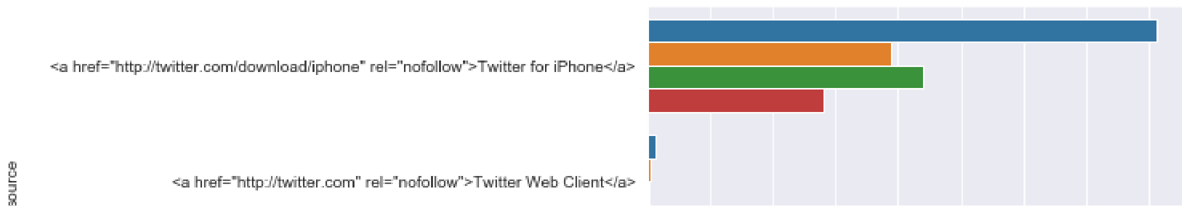
```
df.source.value_counts()
```

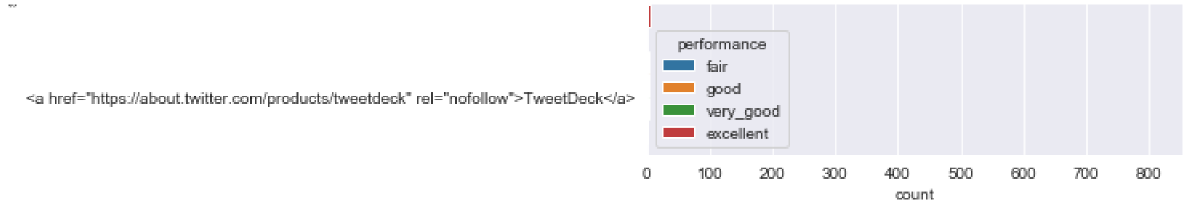
Out[188]:

```
<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>     1922
<a href="http://twitter.com" rel="nofollow">Twitter Web Client</a>                       28
<a href="https://about.twitter.com/products/tweetdeck" rel="nofollow">TweetDeck</a>      11
Name: source, dtype: int64
```

In [191]:

```
# Visualizing dog_stage with performance
sb.countplot(data=df,y='source', hue='performance')
plt.savefig('source_performance.png');
```





In [190]:

```
df.groupby(['source','performance'])['performance'].count()
```

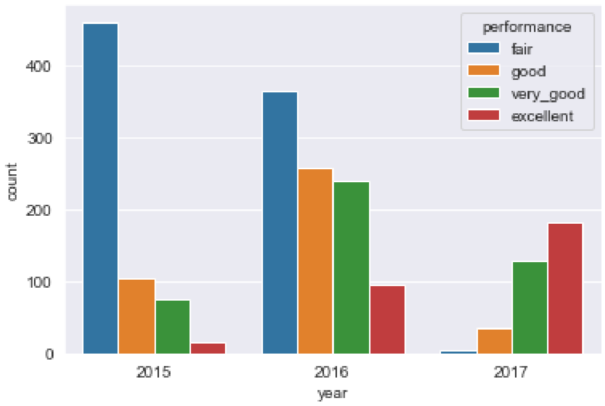
Out[190]:

```
source                                                                         performance
<a href="http://twitter.com" rel="nofollow">Twitter Web Client</a>             fair
13
                                                                               good
                                                                               very_good
3
                                                                               excellent
7
<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>    fair
813
                                                                               good
                                                                               very_good
439
                                                                               excellent
281
<a href="https://about.twitter.com/products/tweetdeck" rel="nofollow">TweetDeck</a>   fair
3
                                                                               good
                                                                               very_good
3
                                                                               excellent
2
Name: performance, dtype: int64
```

> *Although most tweets originate from Twitter for iPhone 25 percent of the tweets originating from Twitter Web Client has excellent rating. But it is difficult to conclude that high rating is associated with a platform.*

In [197]:

```
# Visualizing dog_stage with performance
sb.countplot(data=df,x='year', hue='performance')
plt.savefig('year_performance.png');
```



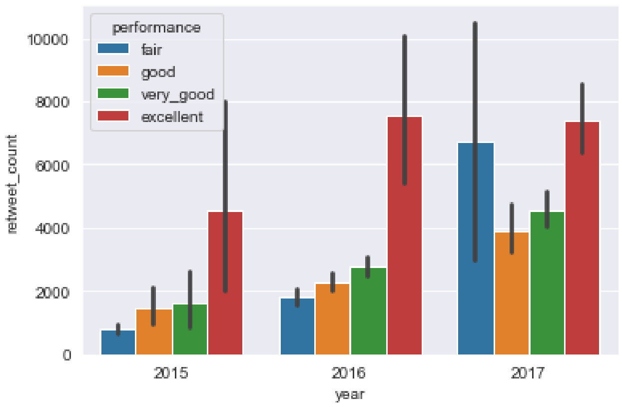
> *In 2015, a great percentage of the ratings are fair but this reduced in 2016 and it is barely existent in 2017. Could we say dogs became better, or WeRateDogs became smarter in rating?*

In [202]:

```
sb.barplot(data=df,x='year', y='favorite_count', hue='performance')
plt.savefig('year_favorite_count_performance.png');
```

In [199]:

```
sb.barplot(data=df,x='year', y='retweet_count', hue='performance')
plt.savefig('year_retweet_count_performance.png');
```



> *Finally, the share of different performance levels in favorite and retweet counts kept increasing across the years. It therefore appears that rating has little to do about whether a tweet will be retweeted or favorited. Thanks.*

Limitations of the Analysis

> *The major limitation of this analysis is the inconsistencies across the different datasets. These necessitated different set of cleanings that resulted in losing some data points. Moreover, I did not do much analysis on the image aspect. Were the data points to be included, the results may be slightly impacted. Another limitation is my current skill level. It is still highly rudimentary.*
>
> ## *Conclusions from the analysis and visualizations*
>
> *Conclusively, I have gathered data from three datasets, assessed them, cleaned them, stored the cleaned dataset, and also analyzed and visualized the cleaned dataset. I have been able to discover that the most popular dog stage is Pupper while the most popular dog name is 'A', followed by 'Charlie'. I have also found that tweets decreases overtime while favorites keep increasing. Moreover, I found that the source of the text, the year the text was made, and the developmental stage of the dog affects dogs' rating. The supporting documents, attached, should be consulted to see more of the insights.*
>
> ## *Bibliograpphy*
>
> - https://www.datacamp.com/
> - https://seaborn.pydata.org/controlling-figure-asthetics

- https://stackoverflow.com/questions/25707558/json-valueerror-expecting-property-name-line-1-column-2-char-1
- https://docs.python.org/3/library/json.html#json.loads
- http://www.jeannicholashould.com/tidy-data-in-python.html
- https://developer.twitter.com/en/docs/tweets/post-and-engage/api-reference/get-statuses-show-id.html
- https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/tweet-object.html
- https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/user-object
- https://media.readthedocs.org/pdf/tweepy/latest/tweepy.pdf
- https://stackabuse.com/reading-and-writing-json-to-a-file-in-python/

In [203]:

```
# Saving the little adjustments in the initial master csv
df.to_csv('final_werate.csv', index=False)
```

In [ ]:

```

```