Data Wrangling Report

This project will look at the Twitter data We Rate Dogs (@dog_rates). This account is devoted to humorously reviewing pictures of dogs doing adorable poses, often giving them scores above 10/10. It has acquired over 1.36 million followers since its debut. For more details on this infomation please view Origin (http://knowyourmeme.com/memes/theyre-good-dogs-brent)

This report is grouped into the four categories that form part of the data wrangling and analysis process

> 1) GATHER
> 2) ASSESS
> 3) CLEAN
> 4) INSIGHTS

The libraries used for the analysis:

In [969]:

```
import requests #this is used for making the web request
import os #this is used for file operations
import sqlalchemy as sql #this is used for integrating with SQL
import pandas as pd #this is used for storing the data structure, assessing the data an
d cleaning the data
import numpy as np
import sqlite3 #this is used for integrating with SQL
import tweepy #this is used to integrate with Twitter API
import json #this is used for storing the data structure
from IPython.display import Image #this is used to insert images
from bs4 import BeautifulSoup as bs #this is used to parse the html within the source v
ariable
```

1 GATHER

> During this step of the process I look at sourcing the following three data requirements
> 1.1 twitter-archive-enhanced.csv
> 1.2 image-predictions.tsv
> 1.3 tweet_json.txt
>
> In the next sections I will go into more detail on the code used to source this data.

1.1 twitter-archive-enhanced.csv

This file was provided and I could manually download it and place it within the working directory.

1.2 image-predictions.tsv

This file I automatically download using the Pyhton requests library. The code below shows how I achieved this.

In [970]:

```
#Initialise variables:
url = r'https://d17h27t6h515a5.cloudfront.net/topher/2017/August/599fd2ad_image-predict
ions/image-predictions.tsv' #Provided URL
filepath = os.path.join(os.getcwd(), 'image-predictions.tsv') #Location of the file
successcall = requests.codes.ok #Set success code

#Create function:
def writefiledata(filepath):
    '''This function takes as input the filepath and if it does not exist downloads and
creates it'''
    try:
        if os.path.exists(filepath) == False:
            request_object = requests.get(url)
            if request_object.status_code == successcall:
                with open(filepath,'wb') as file: #create file object in binary mode
                    file.write(request_object.content) #add content to file
                return('File downloaded successfully')
        else:
            return('File already exists')
    except:
        return('File download failed')

#Make the call:
writefiledata(filepath)
```

Out[970]:

```
'File already exists'
```

1.3 tweet_json.txt

In order to create this file I had to create an application Twitter account, obtain all the security tokens and read over the Twitter API documentation. The library I used to gather this information and gain access to Twitters API is the Tweepy library.

For more information on the resources I used please view:

> [Setting up a twitter application (https://www.slickremix.com/docs/how-to-get-api-keys-and-tokens-for-twitter/)](https://www.slickremix.com/docs/how-to-get-api-keys-and-tokens-for-twitter/)
> [Tweepy library (http://www.tweepy.org/)](http://www.tweepy.org/)
> [Rate limits (https://developer.twitter.com/en/docs/basics/rate-limiting)](https://developer.twitter.com/en/docs/basics/rate-limiting)

The code I built to interact with Twitter via the API is provided below

In [971]:

```
#Set file location:
filepath = os.path.join(os.getcwd(), 'tweet_json.txt')

#This data will provide the tweet id's I need to pass on to the API method 'get_status'
df = pd.read_csv('twitter-archive-enhanced.csv') #read in the archive file and store co
ntent in a dataframe object

#This list will keep track of the tweet's visited:
visited = [] #empty array at this stage

#These are constants provided during the API setup, I will note these only with 'xxx'
#Please note that in me doing this the code will not work as the authorization will rej
ect.
#This code was already used to gather the content and produce the document called tweet
_json.txt.
consumer_key = 'xxx'
consumer_secret = 'xxx'
access_token = 'xxx'
access_token_secret = 'xxx'

#Create authorization:
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)

#this creates the api object to ensure I keep to the rate limit
api = tweepy.API(auth, wait_on_rate_limit=True, wait_on_rate_limit_notify=True)

#Create function:
def calltwitter():
    '''This function will loop over the tweet ids:
    (1) call the API get_status method
    (2) Keep track of visited tweets
    (3) Write json data to file'''
    try:
        for tweet in df['tweet_id']:
            if tweet not in visited:
                status = api.get_status(tweet, tweet_mode='extended')._json #get json f
rom twitter
                visited.append(tweet)
                print(api.wait_on_rate_limit_notify) #show information as tweets are be
ing processed
                with open(filepath, 'a') as output: #Create file object in append mode
                    json.dump(status, output) #add json object to file
                    output.write('\n') #add new line
    except tweepy.error.TweepError:
        visited.append(tweet)
        calltwitter() #call function recursively


if os.path.exists(filepath) == False: #if the path does not exist
    calltwitter() #call function
    print('All API data downloaded successfully')
else:
    print('File already exists')
```

```
File already exists
```

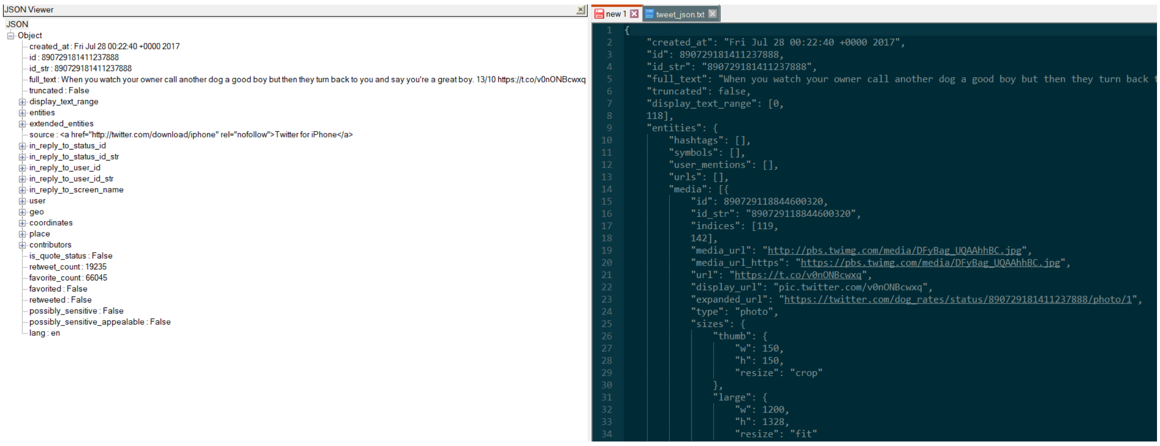
After the data was collected successfully, I had to read the file in and store the content into a dataframe. I noticed that 10 out of the 2356 archive tweet id's did not get a response back from the API and these 10 entries would have gone into the except block in the above code. The below image shows the structure of the JSON object I had to parse. The keys I'm interested in is:

- create_at
- id_str
- retweet_count
- favorite_count

In [972]:

```
#This is the code used to display the below image:
filename = os.path.join(os.getcwd(), 'JsonSample.png')
Image(filename)
```

Out[972]:



The code below shows how I'm constructing the dataframe - extracting only the keys i'm interested in

In [973]:

```
#Get file location:
filelocation = os.path.join(os.getcwd(), 'tweet_json.txt')
#Where to write the pandas object to:
csvfilelocation = os.path.join(os.getcwd(), 'twitterapidata.csv')

#Create empty array:
dataframe = []

#Create function:
def builddataframe():
    '''This function opens the file which holds the API data.
    (1) It loops over every line and pass the object to the json loads method
    (2) It strips out the keys of interest
    (3) Constructs a dictionary
    (4) Append the dictionary (observation) to an array
    (5) Pass the array (collection of observations) to the dataframe constructor
    (6) Returns a dataframe - twitterapidata'''
    try:
        with open(filelocation, 'r') as file: #create file object in read mode
            for line in file.readlines(): #go over every line
                content = json.loads(line) #consume the JSON line
                dict = {'tweet_id': str(content['id_str']),
                        'created_at': pd.to_datetime(content['created_at']),
                        'retweet_count': int(content['retweet_count']),
                        'favorite_count': int(content['favorite_count'])} #construct a
 dictionary object
                dataframe.append(dict) #add the observation to the array
        twitterapidata = pd.DataFrame(dataframe) #send the collection of observations t
o the dataframe constructor
        return(twitterapidata) #return the dataframe object
    except TypeError:
        return('Something is wrong')
    except ValueError:
        return('Something is wrong')

#Write conent to csv (create the file for the submission):
if os.path.exists(csvfilelocation) == False:
    builddataframe().to_csv(csvfilelocation, index=False) #write the dataframe object t
o csv file
else:
    print('File already exists')
```

```
File already exists
```

After I downloaded the above three files I wanted to gather this information into structure SQL environment to make the analysis easier. To achieve this I used the code below. As part of the analysis I also include the database file 'WrangleAndAnalyzeProjectDB.db' within the working directory

In [974]:

```
#Initiate the variables:
dblocation = os.path.join(os.getcwd(), r'WrangleAndAnalyzeProjectDB.db') #creates the d
atabase file location
dbconnection = 'sqlite:///' + dblocation #creates the connection object in case the dat
abase does not exist


#Create database called WrangleAndAnalyzeProjectDB:
if os.path.exists(dblocation) == False: #if the database does not exist
    db = sql.create_engine(dbconnection) #Create the SQL engine
else:
    db = sqlite3.connect(dblocation) #connect to the existing engine

#create dataframes (read in the files):
twitterarchive = pd.read_csv('twitter-archive-enhanced.csv')
imageprediction = pd.read_csv('image-predictions.tsv', delimiter='\t')
twitterapidata = pd.read_csv('twitterapidata.csv')

#Create original SQL objects to store each files content:
twitterarchive.to_sql('twitterarchive', db, if_exists='replace', index=False)
imageprediction.to_sql('imageprediction', db, if_exists='replace', index=False)
twitterapidata.to_sql('twitterapidata', db, if_exists = 'replace', index=False)
```

```
C:\Users\Byron\Applications\DataScienceToolkit\Anaconda\envs\DataAnalystna
nodegreeterm2\lib\site-packages\pandas\core\generic.py:1534: UserWarning:
The spaces in these column names will not be changed. In pandas versions <
0.14, spaces were converted to underscores.
  chunksize=chunksize, dtype=dtype)
```

Now that the data is stored within SQL I can collect the data from there to view and assess. The two cells below collects the data and creates copies of the originals.

In [975]:

```
#Initiate the variables:
dblocation = os.path.join(os.getcwd(), r'WrangleAndAnalyzeProjectDB.db') #creates the d
atabase file location
dbconnection = 'sqlite:///' + dblocation #creates the connection object

#This code creates the data frames from the database:
twitterarchive = pd.read_sql(sql='select * from twitterarchive',con=dbconnection)
imageprediction = pd.read_sql(sql='select * from imageprediction',con=dbconnection)
twitterapidata = pd.read_sql(sql='select * from twitterapidata',con=dbconnection)
```

In [976]:

```
#This code creates copies of the originals:
twitterarchive_clean = twitterarchive.copy(deep=True)
imageprediction_clean = imageprediction.copy(deep=True)
twitterapidata_clean = twitterapidata.copy(deep=True)
```

2 ASSESS

Now that the data has been gathered and stored, the next step in the wrangling process is to assess the data. During this process I will look at two catergories:

> 2.1 Messy Data (Structure issues)
> 2.2 Dirty Data (Quality issues)

2.1 Messy Data

> During this investigation I want to accomplish a tidy dataset. This is a dataset with the following characteristics:
>
> > - Each variable is a column
> > - Each row is an observation
> > - Each type of observational unit forms a table

2.1.1 Convert wide data into long data

- TABLE twitterarchive

> The variables: doggo, floofer, pupper and puppo can be converted into one new catergorical variable called 'dogstage' with 4 categories.

- TABLE imageprediction

> The prediction variables can be converted as follow:
>
> p1, p2 and p3 can be converted into one variable - prediction (string variable)
> p1_conf, p2_conf and p3_conf can be converted into one variable - confidencerating (float variable)
> p1_dog, p2_dog and p3_dog can be converted into one variable - clasification (boolean variable indicating if the breed is of type dog or not)

2.2 Dirty Data

During this investigation I will look at the following criteria:

- Datatypes
- Duplication
- Completeness
- Accuracy
- Validity
- Consistency

2.2.1 Datatypes

The code below gives general information about each dataset

In [977]:

```
twitterarchive_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2356 entries, 0 to 2355
Data columns (total 17 columns):
tweet_id                      2356 non-null int64
in_reply_to_status_id         78 non-null float64
in_reply_to_user_id           78 non-null float64
timestamp                     2356 non-null object
source                        2356 non-null object
text                          2356 non-null object
retweeted_status_id           181 non-null float64
retweeted_status_user_id      181 non-null float64
retweeted_status_timestamp    181 non-null object
expanded_urls                 2297 non-null object
rating_numerator              2356 non-null int64
rating_denominator            2356 non-null int64
name                          2356 non-null object
doggo                         2356 non-null object
floofer                       2356 non-null object
pupper                        2356 non-null object
puppo                         2356 non-null object
dtypes: float64(4), int64(3), object(10)
memory usage: 313.0+ KB
```

In [978]:

```
imageprediction_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2075 entries, 0 to 2074
Data columns (total 12 columns):
tweet_id    2075 non-null int64
jpg_url     2075 non-null object
img_num     2075 non-null int64
p1          2075 non-null object
p1_conf     2075 non-null float64
p1_dog      2075 non-null int64
p2          2075 non-null object
p2_conf     2075 non-null float64
p2_dog      2075 non-null int64
p3          2075 non-null object
p3_conf     2075 non-null float64
p3_dog      2075 non-null int64
dtypes: float64(3), int64(5), object(4)
memory usage: 194.6+ KB
```

In [979]:

```
twitterapidata_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2346 entries, 0 to 2345
Data columns (total 5 columns):
Unnamed: 0        2346 non-null int64
created_at        2346 non-null object
favorite_count    2346 non-null int64
retweet_count     2346 non-null int64
tweet_id          2346 non-null int64
dtypes: int64(4), object(1)
memory usage: 91.7+ KB
```

- TABLE twitterarchive
  - DataTypes:
    - tweet_id convert to string
    - in_reply_to_status_id convert to string
    - in_reply_to_user_id convert to string
    - timestamp convert to datetime
    - retweeted_status_id convert to string
    - retweeted_status_user_id convert to string
    - retweeted_status_timestamp convert to datetime
- TABLE imageprediction
  - DataTypes:
    - tweet_id convert to string
    - p1_dog convert to boolean
    - p2_dog convert to boolean
    - p3_dog convert to boolean
- TABLE twitterapidata
  - DataTypes:
    - create_at convert to datetime
    - tweet_id convert to string

2.2.2 Duplicated Data

The code below does a check on the tweet id to ensure that the tweets are not being duplicated in the datasets at this point in time. For each of the tables there were no duplicates observed.

Observations with non empty retweet information must be removed. The SQL below identifies there are 181 observations that meet this criteria

SELECT *
FROM twitterarchive
WHERE retweeted_status_id is not null or
retweeted_status_user_id is not null or
retweeted_status_timestamp is not null

2.2.3 Completeness

- TABLE twitterarchive

> - in_reply_to_status_id 78 observations
> - in_reply_to_user_id 78 observations
> - retweeted_status_id 181 observations
> - retweeted_status_user_id 181 observations
> - retweeted_status_timestamp 181 observations
> - expanded_urls 2297 observations

The in_reply and retweeted variables makes sense to not always contain data, but the expanded_url could be enriched with another url from the text variable, which would produce similar output or image source. These variables are not critical to my analysis

2.2.4 Accuracy (valid entries, but wrong)

During my analysis in SQL I noticed that some observations contain a rating denominator > 10. In some instances the rating was just stripped incorrectly due to other numbers in the text that has the same pattern 'x/y'. These entries are valid, but has to be fixed to the correct rating. The SQL below shows how I identified these observations

SELECT *
FROM twitterarchive
WHERE text LIKE '%10%' --text contains a 10
AND rating_denominator <> 10 --the parsed value does not contain 10

- TABLE twitterarchive

> - source : strip the information for the http url instead of the tagged string
> - name : incorrectly extracted information from the pattern 'this is...'
> - some observations has more than one dogstage specified (most often it's multiple dogs on one image presented as one observation (tweet))
> - some observations has faulty ratings due to entries being parsed incorrectly
> - expanded_urls contains a list of similar urls, only one is required
> - rating_numberator and rating_denominator variables to be correctly extracted from the text variable

In the twitterarchive data set the 'source' variable has to be stripped for the https url string and stored as the source instead of a tagged string. The 'name' variable also had some issues and the reason for this is because it was extracted from the 'text' variable with pattern 'this is....', not all of the strings start with this pattern and it has resulted in names that does not make sense. Below code collects all these names and does a count on how many times each of these occur. Since these are only string values that would not impact on the analysis or predictions there is no need to clean this - it is a nice to have.

In [980]:

```
listofaccuracyissues = ["a","actually","all","an","by","getting","his","incredibly","in
furiating","just","life",
                        "light","mad","my","not","officially","old","one","quite","spac
e","such","the","this",
                        "unacceptable","very","None"]

twitterarchive_clean.query('name in @listofaccuracyissues')['name'].value_counts()
```

Out[980]:

```
None            745
a                55
the               8
an                7
very              5
one               4
quite             4
just              4
mad               2
not               2
actually          2
getting           2
light             1
by                1
unacceptable      1
space             1
all               1
my                1
infuriating       1
his               1
old               1
officially        1
such              1
incredibly        1
this              1
life              1
Name: name, dtype: int64
```

2.2.5 Validity (invalid entries)

During my analysis of the data in SQL, I noticed that some of the text string contains phrases that indicate that the tweets are not valid, since they do not relate to dogs. Below I show the SQL code used to get an idea of how many of these observation there are within the dataset.

```
SELECT Text
FROM twitterarchive
WHERE Text LIKE '%we only rate dogs%'
OR
Text LIKE '%we usually don''t%'
OR
Text LIKE '%we normally don''t%'
OR
Text LIKE '%we don''t rate%'
OR
Text LIKE '%please don''t%'
OR
Text LIKE '%only send%'
```

The SQL code below identifies the observations where one tweet relates to many dogs. It's true that these observations do relate to dogs, but its invalid, since each observation has to present the demographics of one dog

```
SELECT *
FROM twitterarchive
WHERE rating_denominator <> 10
AND tweet_id NOT IN (SELECT tweet_id
FROM twitterarchive
WHERE text LIKE '%10%'
AND rating_denominator <> 10)
```

- TABLE twitterarchive

> - text
> - Multiple dogs in an image form part of one observation. These entries has to be removed

In the twitterarchive dataset some of the observations do not relate to dogs and because of this it would make the whole observation invalid. The 'text' variable containing the extracts below, samples some of these invalid observations:

> > - ....we only rate dogs....
> > - ...we usually don't... (this is subjective but I still think its invalid)
> > - ...we normally don't... (this is subjective but I still think its invalid)
> > - ...we don't rate...
> > - ...please don't...
> > - ....only send...

2.2.6 Consistancy (multiple ways of referring to the same thing)

- TABLE imageprediction

> - p1 and p2 and p3

In the imageprediction dataset the three different prediction variables 'p1','p2' and 'p3' is not consistent in the way it presents a prediction.

As example

- shopping_cart vs shopping_basket
- golden_retriever vs Labrador_retriever

This is how the algorithm predicted and I did not want to change the outcome of the predictions, since that will mean I interfere with the outcome. I consider consistancy issues rather as something that someone manually records incorrectly

3 CLEAN

Now that the data has been assessed and I have developed a better understanding of what to do. The tasks involded in cleaning the data can be completed. This process consists of three steps:

- 3.1 Define
- 3.2 Code
- 3.3 Test

TABLE Twitterarchive

Define

Table twitterarchive convert variables doggo, floofer, pupper and puppo into one variable dogstage using pandas melt function

Code

In [981]:

```
#First I use the melt function to pivot the variables:
tidy = twitterarchive_clean.melt(id_vars = [ 'tweet_id',
                                              'in_reply_to_status_id',
                                              'in_reply_to_user_id',
                                              'timestamp',
                                              'source',
                                              'text',
                                              'retweeted_status_id',
                                              'retweeted_status_user_id',
                                              'retweeted_status_timestamp',
                                              'expanded_urls',
                                              'rating_numerator',
                                              'rating_denominator',
                                              'name'],
                                 var_name = 'dogstage')
#Overwrite newly created variable:
tidy['dogstage'] = tidy['value']
#This dataframe will hold all the twitter_id's for which none of the dogstage categorie
s were given:
tidy1 = tidy.groupby(['tweet_id','dogstage'], as_index=False).count().sort_values(['twe
et_id','dogstage']).query('value == 4').loc[:,['tweet_id','dogstage']]
#This dataframe will hold all the twitter_id's for which the dogstage categories were g
iven:
tidy2 = tidy.groupby(['tweet_id','dogstage'], as_index=False).count().sort_values(['twe
et_id','dogstage']).query('dogstage != "None"').loc[:,['tweet_id','dogstage']]
#Drop the value column as it's no longer required:
tidy.drop('value', axis = 1, inplace = True)
```

In [982]:

```
#This set limits the tidy data set to only those obeservations that had none of the dog
stage categories specified:
set1 = pd.merge(tidy, tidy1, how='inner', on='tweet_id')
#Keep only one of the observations:
set1.drop_duplicates(inplace=True)
```

In [983]:

```
#This set limits the tidy data set to only those observations that had at least one dog
stage category specified:
set2 = pd.merge(tidy, tidy2, how='inner', on=['tweet_id','dogstage'])
```

In [984]:

```
#Only select the columns as per the original and update the two sets:
col = list(tidy) #get the column (variable) names
set1 = set1.loc[:,col]
set2 = set2.loc[:,col]
```

```
C:\Users\Byron\Applications\DataScienceToolkit\Anaconda\envs\DataAnalystna
nodegreeterm2\lib\site-packages\ipykernel_launcher.py:3: FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-re
index-listlike
  This is separate from the ipykernel package so we can avoid doing import
s until
C:\Users\Byron\Applications\DataScienceToolkit\Anaconda\envs\DataAnalystna
nodegreeterm2\lib\site-packages\pandas\core\indexing.py:1367: FutureWarnin
g:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-re
index-listlike
  return self._getitem_tuple(key)
```

In [985]:

```
#Append the two data sets:
combine = pd.concat([set1,set2])
```

Test

In [986]:

```
#Note that the dogstage variable should now be available in the list of variables
list(combine)
```

Out[986]:

```
['tweet_id',
 'in_reply_to_status_id',
 'in_reply_to_user_id',
 'timestamp',
 'source',
 'text',
 'retweeted_status_id',
 'retweeted_status_user_id',
 'retweeted_status_timestamp',
 'expanded_urls',
 'rating_numerator',
 'rating_denominator',
 'name',
 'dogstage']
```

Define

Convert variables tweet_id, in_reply_to_status_id, in_reply_to_user_id, retweeted_status_id, retweeted_status_user_id to string.
Convert variables timestamp and retweeted_status_timestamp to datetime.
Convert variables rating_numberator and rating_denominator to float using pandas astype function

Code

In [987]:

```
#Convert datatypes to string:
combine = combine.astype({ 'tweet_id': str,
                           'in_reply_to_status_id': str,
                           'in_reply_to_user_id': str,
                           'retweeted_status_id': str,
                           'retweeted_status_user_id': str
                         })
```

In [988]:

```
#Convert datatypes to datetime:
combine['timestamp'] = pd.to_datetime(combine['timestamp'])
combine['retweeted_status_timestamp'] = pd.to_datetime(combine['retweeted_status_timest
amp'])
```

In [989]:

```
#Convert datatypes to float:
combine = combine.astype({ 'rating_numerator': float,
                           'rating_denominator': float
                        })
```

Test

In [990]:

```
#Check to see if the data types updated correctly:
combine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2370 entries, 0 to 393
Data columns (total 14 columns):
tweet_id                      2370 non-null object
in_reply_to_status_id         2370 non-null object
in_reply_to_user_id           2370 non-null object
timestamp                     2370 non-null datetime64[ns]
source                        2370 non-null object
text                          2370 non-null object
retweeted_status_id           2370 non-null object
retweeted_status_user_id      2370 non-null object
retweeted_status_timestamp    183 non-null datetime64[ns]
expanded_urls                 2311 non-null object
rating_numerator              2370 non-null float64
rating_denominator            2370 non-null float64
name                          2370 non-null object
dogstage                      394 non-null object
dtypes: datetime64[ns](2), float64(2), object(10)
memory usage: 277.7+ KB
```

Define

Strip the https url from the source variable using pandas extract function

Code

In [991]:

```
#Create a function that will parse the html:
def getsource(x):
    soup = bs(x,'html.parser') #create soup object
    return(soup.a.get('href')) #strip url

#Update resource variable:
combine['source'] = combine['source'].map(getsource) #apply function accross the whole
 series
```

Test

In [992]:

```
#Check the source variable to ensure the url has been parsed correctly:
combine['source'].unique()
```

Out[992]:

```
array(['http://twitter.com/download/iphone', 'http://twitter.com',
       'http://vine.co', 'https://about.twitter.com/products/tweetdeck'],
      dtype=object)
```

Define

Remove list like expanded_urls using python split method

Code

In [993]:

```
#Create a function to extract the first url
def getfirsturl(x):
    if x is not None: #if the data is not of type none or null
        if x.find(',') >= 0: #if a comma is present within the string
            return(x.split(',')[0]) #return the first element of the list
        else:
            return(x) # return the current entry
    else:
        return('') #none type can be made blank

combine['expanded_urls'] = combine['expanded_urls'].map(getfirsturl)
```

Test

In [994]:

```
#Check the expanded_urls variable to ensure the url has been parsed correctly:
combine['expanded_urls'].unique()
```

Out[994]:

```
array(['https://twitter.com/dog_rates/status/892420643555336193/photo/1',
       'https://twitter.com/dog_rates/status/892177421306343426/photo/1',
       'https://twitter.com/dog_rates/status/891815181378084864/photo/1',
       ...,
       'https://twitter.com/dog_rates/status/744995568523612160/photo/1',
       'https://twitter.com/dog_rates/status/743253157753532416/photo/1',
       'https://twitter.com/dog_rates/status/738537504001953792/photo/1'],
      dtype=object)
```

Define

Remove records that have more than one dogstage using pandas drop function

Code

The code below shows that there are actually a total of 28 observations (14 'duplicates') left over. What I've noticed here is that for each of these tweet_id's there are more than one category of dogstage. These should be seperated out as different observations. What is happening here is that for these tweets two or more dogs are being viewed in one observation. This forms part of the accuracy issues and I will remove these entries.

In [995]:

```
listofduplicates = combine[combine['tweet_id'].duplicated() == True]['tweet_id']
combine.query('tweet_id in @listofduplicates').loc[:,['tweet_id','dogstage']].sort_valu
es('tweet_id')
```

Out[995]:

	tweet_id	dogstage
91	733109485275860992	doggo
202	733109485275860992	pupper
87	741067306818797568	doggo
195	741067306818797568	pupper
179	751583847268179968	pupper
78	751583847268179968	doggo
70	759793422261743616	doggo
174	759793422261743616	pupper
64	770093767776997377	doggo
169	770093767776997377	pupper
59	775898661951791106	doggo
164	775898661951791106	pupper
56	781308096455073793	doggo
157	781308096455073793	pupper
52	785639753186217984	doggo
156	785639753186217984	pupper
145	801115127852503040	pupper
45	801115127852503040	doggo
43	802265048156610565	doggo
144	802265048156610565	pupper
42	808106460588765185	doggo
141	808106460588765185	pupper
134	817777686764523521	pupper
37	817777686764523521	doggo
8	854010172552949760	doggo
98	854010172552949760	floofer
7	855851453814013952	doggo
370	855851453814013952	puppo

In [996]:

```
#This code removes the 14 tweets that contained more than 1 dogstage category:
combine.drop(combine.query('tweet_id in @listofduplicates').index, inplace=True)
```

Test

In [997]:

```
#There are no more duplicates left in the set
combine['tweet_id'].duplicated().value_counts()
```

Out[997]:

```
False    2335
Name: tweet_id, dtype: int64
```

Define

Remove observations with non empty retweet information using the python drop function

Code

In [998]:

```
listoftweetstoremove = combine.query('retweeted_status_id != "nan"')['tweet_id']
```

In [999]:

```
#This code removes the tweets with retweet information:
combine.drop(combine.query('tweet_id in @listoftweetstoremove').index, axis=0, inplace=
True)
```

Test

In [1000]:

```
#The records should now be on 0:
combine.query('tweet_id in @listoftweetstoremove').count()
```

Out[1000]:

```
tweet_id                      0
in_reply_to_status_id         0
in_reply_to_user_id           0
timestamp                     0
source                        0
text                          0
retweeted_status_id           0
retweeted_status_user_id      0
retweeted_status_timestamp    0
expanded_urls                 0
rating_numerator              0
rating_denominator            0
name                          0
dogstage                      0
dtype: int64
```

Define

Remove the observations that do not relate to dogs

Code

In [1001]:

```
#Remove invalid observations:
#These are some (hopefully all) of the observations that do not relate to dogs:
#This code removes white space and makes all text lower case before looking for the pat
tern
#The code then looks for every return that is not (-1) in other words the pattern has b
een found
#6 arrays are then created containing the tweet_id's
r1 = combine[combine['text'].str.replace(' ','').str.lower().str.find('weonlyratedogs')
!= -1]['tweet_id']
r2 = combine[combine['text'].str.replace(' ','').str.lower().str.find('weusuallydon\'t'
) != -1]['tweet_id']
r3 = combine[combine['text'].str.replace(' ','').str.lower().str.find('wenormallydon\'
t') != -1]['tweet_id']
r4 = combine[combine['text'].str.replace(' ','').str.lower().str.find('wedon\'trate') !
= -1]['tweet_id']
r5 = combine[combine['text'].str.replace(' ','').str.lower().str.find('pleasedon\'t') !
= -1]['tweet_id']
r6 = combine[combine['text'].str.replace(' ','').str.lower().str.find('onlysend') != -1
]['tweet_id']
```

In [1002]:

```
#Append all the tweet_ids and remove any duplication:
remove = pd.concat([r1, r2, r3, r4, r5, r6])
remove = remove.unique() #all the unique tweet_ids that has to be removed
```

In [1003]:

```
#Remove the invalid observations:
combine.drop(combine.query('tweet_id in @remove').index, axis=0, inplace=True)
```

Test

In [1004]:

```
#This code checks to see if there are still any invalid observations:
r1 = combine[combine['text'].str.replace(' ','').str.lower().str.find('weonlyratedogs')
!= -1]['tweet_id']
r2 = combine[combine['text'].str.replace(' ','').str.lower().str.find('weusuallydon\'t'
) != -1]['tweet_id']
r3 = combine[combine['text'].str.replace(' ','').str.lower().str.find('wenormallydon\'
t') != -1]['tweet_id']
r4 = combine[combine['text'].str.replace(' ','').str.lower().str.find('wedon\'trate') !
= -1]['tweet_id']
r5 = combine[combine['text'].str.replace(' ','').str.lower().str.find('pleasedon\'t') !
= -1]['tweet_id']
r6 = combine[combine['text'].str.replace(' ','').str.lower().str.find('onlysend') != -1
]['tweet_id']
assert len(pd.concat([r1, r2, r3, r4, r5, r6])) == 0
```

Define

Fix the rating values by extracting the correct information from the text variable - fix the decimal ratings

Code

In [1005]:

```
decimals = combine['text'].str.extract(r'(\d+\.\d+)\/(\d+)').rename(columns={0:'rating_
numerator', 1:'rating_denominator'})
```

```
C:\Users\Byron\Applications\DataScienceToolkit\Anaconda\envs\DataAnalystna
nodegreeterm2\lib\site-packages\ipykernel_launcher.py:1: FutureWarning: cu
rrently extract(expand=None) means expand=False (return Index/Series/DataF
rame) but in a future version of pandas this will be changed to expand=Tru
e (return DataFrame)
  """Entry point for launching an IPython kernel.
```

In [1006]:

```
decimals = decimals[~decimals['rating_numerator'].isnull()]
```

In [1007]:

```
#update the observation ratings:
for i in decimals.index:
    combine.loc[combine.index==i,'rating_numerator'] = decimals['rating_numerator'][i]
    combine.loc[combine.index==i,'rating_denominator'] = decimals['rating_denominator']
[i]
```

Test

In [1008]:

```
assert combine[combine.text.str.contains(r'(\d+(\.\d+))\/(\d+)')]['text'].count() == co
mbine[combine.text.str.contains(r'(\d+(\.\d+))\/(\d+)')]['rating_numerator'].count()
```

```
C:\Users\Byron\Applications\DataScienceToolkit\Anaconda\envs\DataAnalystna
nodegreeterm2\lib\site-packages\ipykernel_launcher.py:1: UserWarning: This
pattern has match groups. To actually get the groups, use str.extract.
  """Entry point for launching an IPython kernel.
```

Define

Fix the ratings that are invalid

Code

In [1009]:

```
#Any observation with incorrect rating and no rating out of 10 found within the text va
riable:
obs = combine.query('rating_denominator != 10.0').loc[:,['text']]['text'].str.extract(
'(10)',expand=True) != '10'

#Rename column:
obs.columns = ['invalid']

#Filter for the invalid records by index:
multipleobservations = obs.query('invalid==True').index

#Remove the observations
multipleobs = []
for i in multipleobservations:
    multipleobs.append(combine.loc[combine.index==i,'tweet_id'].item())
combine.drop(combine.query('tweet_id in @multipleobs').index, axis=0, inplace=True)
```

In [1010]:

```
#Filter for valid records to fix:
fixobservations = obs.query('invalid==False')

#merge the observations to fix:
fix = pd.merge(combine,
               fixobservations,
               how="inner",
               left_index=True,
               right_index=True)['text'].str.extractall('([0-9]+\S+[0-9]+)')
fix = fix.query('match == 1')
fix[0].str.split('/',expand=True)
```

Out[1010]:

		0	1
	match		
160	**1**	948	None
	1	948	None
1092	**1**	13	10
3472	**1**	14	10
3768	**1**	13	10
3888	**1**	11	10
5320	**1**	1zfnTJLt55	None
5396	**1**	10	10
5564	**1**	2S6p9	None
7820	**1**	9	10

In the output above I notice index 5320 does not seem right. The reason for this entry is because my SQL code looked at %10% which will also collect the ratings of 100, 110.....This particular entry also had multiple dogs linked to one observation and for this reason I will remove it.

In [1011]:

```
#Remove the entry:
combine.drop(5320, inplace=True)
```

The entry should now be removed. Now I'll redefine the set to fix

In [1012]:

```
#Any observation with incorrect rating and no rating out of 10 found within the text va
riable:
obs = combine.query('rating_denominator != 10.0').loc[:,['text']]['text'].str.extract(
'(10)',expand=True) != '10'

#Rename column:
obs.columns = ['invalid']

#Filter for valid records to fix:
fixobservations = obs.query('invalid==False')

#merge the observations to fix:
fix = pd.merge(combine,
               fixobservations,
               how="inner",
               left_index=True,
               right_index=True)['text'].str.extractall('([0-9]+\S+[0-9]+)')
fix = fix.query('match == 1')
fix = fix[0].str.split('/',expand=True)
fix = pd.DataFrame(fix.to_records())
fix.drop('match', axis=1, inplace=True)
fix.columns = ['index','numerator','denominator']
fix
```

Out[1012]:

	index	numerator	denominator
0	160	948	None
1	160	948	None
2	1092	13	10
3	3472	14	10
4	3768	13	10
5	3888	11	10
6	5396	10	10
7	5564	2S6p9	None
8	7820	9	10

In [1013]:

```
#update the observation ratings:
combine.loc[combine.index==1092,'rating_numerator'] = float(fix.query('index == 1092')[
'numerator'])
combine.loc[combine.index==1092,'rating_denominator'] = float(fix.query('index == 1092'
)['denominator'])

combine.loc[combine.index==3472,'rating_numerator'] = float(fix.query('index == 3472')[
'numerator'])
combine.loc[combine.index==3472,'rating_denominator'] = float(fix.query('index == 3472'
)['denominator'])

combine.loc[combine.index==3768,'rating_numerator'] = float(fix.query('index == 3768')[
'numerator'])
combine.loc[combine.index==3768,'rating_denominator'] = float(fix.query('index == 3768'
)['denominator'])

combine.loc[combine.index==3888,'rating_numerator'] = float(fix.query('index == 3888')[
'numerator'])
combine.loc[combine.index==3888,'rating_denominator'] = float(fix.query('index == 3888'
)['denominator'])

combine.loc[combine.index==5396,'rating_numerator'] = float(fix.query('index == 5396')[
'numerator'])
combine.loc[combine.index==5396,'rating_denominator'] = float(fix.query('index == 5396'
)['denominator'])

combine.loc[combine.index==7820,'rating_numerator'] = float(fix.query('index == 7820')[
'numerator'])
combine.loc[combine.index==7820,'rating_denominator'] = float(fix.query('index == 7820'
)['denominator'])
```

Test

In [1014]:

```
#Any observation with incorrect rating and no rating out of 10 found within the text va
riable:
test = combine.query('rating_denominator != 10.0').loc[:,['text']]['text'].str.extract(
'(10)',expand=True) != '10'

#Rename column:
test.columns = ['invalid']

#Filter for the invalid records:
#The assert statement should pass (index array should have length of 0)
assert len(test.query('invalid==True').index) == 0
```

In [1015]:

```
combine[combine.index == 1092].loc[:,['rating_numerator','rating_denominator']]
```

Out[1015]:

	rating_numerator	rating_denominator
1092	13	10

In [1016]:

```
combine[combine.index == 3472].loc[:,['rating_numerator','rating_denominator']]
```

Out[1016]:

	rating_numerator	rating_denominator
3472	14	10

In [1017]:

```
combine[combine.index == 3768].loc[:,['rating_numerator','rating_denominator']]
```

Out[1017]:

	rating_numerator	rating_denominator
3768	13	10

In [1018]:

```
combine[combine.index == 3888].loc[:,['rating_numerator','rating_denominator']]
```

Out[1018]:

	rating_numerator	rating_denominator
3888	11	10

In [1019]:

```
combine[combine.index == 5396].loc[:,['rating_numerator','rating_denominator']]
```

Out[1019]:

	rating_numerator	rating_denominator
5396	10	10

In [1020]:

```
combine[combine.index == 7820].loc[:,['rating_numerator','rating_denominator']]
```

Out[1020]:

	rating_numerator	rating_denominator
7820	9	10

In [1021]:

```
#there are no duplicates within this dataset:
sum(combine.duplicated())
```

Out[1021]:

```
0
```

In [1022]:

```
#Update the copy frame:
twitterarchive_clean = combine
```

In [1023]:

```
#Random sample:
twitterarchive_clean.sample(5)
```

Out[1023]:

	tweet_id	in_reply_to_status_id	in_reply_to_user_id	timestamp	
5580	680161097740095489	nan	nan	2015-12-24 23:00:17	ht
7804	666353288456101888	nan	nan	2015-11-16 20:32:58	ht
2512	778624900596654080	nan	nan	2016-09-21 16:00:17	ht
7012	670073503555706880	nan	nan	2015-11-27 02:55:47	ht
7416	668190681446379520	nan	nan	2015-11-21 22:14:07	ht

TABLE Imageprediction

Define

Convert variables p1, p2 and p3 into one variable prediction using pandas melt function
Convert variables p1_conf, p2_conf and p3_conf into one variable confidencerating using pandas melt function
Convert variables p1_dog, p2_dog and p3_dog into one variable classification using pandas melt function

Code

In [1024]:

```
#For this table we need to melt three new variables:
#New variable prediction
f1 = imageprediction_clean.melt(id_vars=['tweet_id','jpg_url'],
                                value_vars=['p1','p2','p3'],
                                value_name='prediction').sort_values('tweet_id')
#New variable confidencerating
f2 = imageprediction_clean.melt(id_vars=['tweet_id','jpg_url'],
                                value_vars=['p1_conf', 'p2_conf', 'p3_conf'],
                                value_name='confidencerating').sort_values('tweet_id')
#New variable classification
f3 = imageprediction_clean.melt(id_vars=['tweet_id','jpg_url'],
                                value_vars=['p1_dog', 'p2_dog', 'p3_dog'],
                                value_name='classification').sort_values('tweet_id')
```

In [1025]:

```
#Only get the columns required:
f1 = f1.loc[:,['tweet_id','prediction']]
f2 = f2.loc[:,['tweet_id','confidencerating']]
f3 = f3.loc[:,['tweet_id','classification']]
```

In [1026]:

```
#What does a sample of f1 look like
f1.head(2)
```

Out[1026]:

	tweet_id	prediction
0	666020888022790149	Welsh_springer_spaniel
4150	666020888022790149	Shetland_sheepdog

In [1027]:

```
#What does a sample of f2 look like
f2.head(2)
```

Out[1027]:

	tweet_id	confidencerating
0	666020888022790149	0.465074
4150	666020888022790149	0.061428

In [1028]:

```
#What does a sample of f3 look like
f3.head(2)
```

Out[1028]:

	tweet_id	classification
0	666020888022790149	1
4150	666020888022790149	1

In [1029]:

```
#Join the three datasets together:
f12 = pd.merge(f1, f2, how='inner', left_index=True, right_index=True)
f123 = pd.merge(f12, f3, how='inner', left_index=True, right_index=True)
f123 = f123.loc[:,['tweet_id','prediction','confidencerating','classification']]
```

Test

In [1030]:

```
#After the pivot is complete there should be the original 2075 records x 3.
#No error should be thrown
expected = 2075*3
assert f123.loc[:,['tweet_id','prediction','confidencerating','classification']].shape[
0] == expected
```

In [1031]:

```
f123.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6225 entries, 0 to 6224
Data columns (total 4 columns):
tweet_id            6225 non-null int64
prediction          6225 non-null object
confidencerating    6225 non-null float64
classification      6225 non-null int64
dtypes: float64(1), int64(2), object(1)
memory usage: 403.2+ KB
```

In [1032]:

```
#no duplicates within this dataset
sum(f123.duplicated())
```

Out[1032]:

```
0
```

Define

Convert variable tweet_id to string and variable clasification to boolean using pandas astype function

Code

In [1033]:

```
#Update the datatypes on the dataset
f123 = f123.astype({'tweet_id': str,
                    'classification': bool})
```

Test

In [1034]:

```
f123.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6225 entries, 0 to 6224
Data columns (total 4 columns):
tweet_id            6225 non-null object
prediction          6225 non-null object
confidencerating    6225 non-null float64
classification      6225 non-null bool
dtypes: bool(1), float64(1), object(2)
memory usage: 360.6+ KB
```

In [1035]:

```
#Update the copy frame:
imageprediction_clean = f123
```

In [1036]:

```
#Random sample:
imageprediction_clean.sample(5)
```

Out[1036]:

	tweet_id	prediction	confidencerating	classifica
2764	684188786104872960	American_Staffordshire_terrier	0.082953	True
423	674053186244734976	Cardigan	0.984725	True
1096	720059472081784833	Mexican_hairless	0.451852	True
1920	856282028240666624	Chihuahua	0.876543	True
1363	761334018830917632	Norwegian_elkhound	0.822936	True

TABLE Twitterapidata

Define

Convert variable tweet_id to string and variable created_at to datetime using pandas astype function

Code

In [1037]:

```
#Update the datatypes on the dataset:
twitterapidata_clean = twitterapidata_clean.astype({'tweet_id': str})
twitterapidata_clean['created_at'] = pd.to_datetime(twitterapidata_clean['created_at'])
```

Test

In [1038]:

```
twitterapidata_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2346 entries, 0 to 2345
Data columns (total 5 columns):
Unnamed: 0        2346 non-null int64
created_at        2346 non-null datetime64[ns]
favorite_count    2346 non-null int64
retweet_count     2346 non-null int64
tweet_id          2346 non-null object
dtypes: datetime64[ns](1), int64(3), object(1)
memory usage: 91.7+ KB
```

Define

Remove the unwanted column due to the index written to the file. I have amended the code that creates the file to set index = False.

Code

In [1039]:

```
if 'Unnamed: 0' in list(twitterapidata_clean):
    twitterapidata_clean.drop('Unnamed: 0', axis=1, inplace=True)
```

Test

In [1040]:

```
#The column should now be removed:
twitterapidata_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2346 entries, 0 to 2345
Data columns (total 4 columns):
created_at        2346 non-null datetime64[ns]
favorite_count    2346 non-null int64
retweet_count     2346 non-null int64
tweet_id          2346 non-null object
dtypes: datetime64[ns](1), int64(2), object(1)
memory usage: 73.4+ KB
```

In [1041]:

```
#no duplicates within this dataset
sum(twitterapidata_clean.duplicated())
```

Out[1041]:

0

In [1042]:

```
twitterapidata_clean.sample(5)
```

Out[1042]:

	created_at	favorite_count	retweet_count	tweet_id
1686	2015-12-27 22:37:04	1611	609	681242418453299201
1971	2015-12-05 02:46:02	994	380	672970152493887488
543	2016-12-02 00:02:45	6773	2298	804475857670639616
511	2016-12-18 00:43:57	39016	13034	810284430598270976
2009	2015-12-02 18:48:47	2517	1224	672125275208069120

Now that the data has mostly been cleaned I write it back to SQL for more investigation and potentially repeating the process of cleaning and refining the data. The code below creates the 'clean' tables is SQL

In [1043]:

```
#Initiate the variables:
dblocation = os.path.join(os.getcwd(), r'WrangleAndAnalyzeProjectDB.db') #creates the d
atabase file location
dbconnection = 'sqlite:///' + dblocation #creates the connection object in case the dat
abase does not exist

#Create clean SQL objects to store in db:
twitterarchive_clean.to_sql('twitterarchive_clean', db, if_exists='replace', index=Fals
e)
imageprediction_clean.to_sql('imageprediction_clean', db, if_exists='replace', index=Fa
lse)
twitterapidata_clean.to_sql('twitterapidata_clean', db, if_exists = 'replace', index=Fa
lse)
```

Now that the clean datasets has been written to the SQL database, I will create a view in SQL. The code below shows the SQL script used for creating the view. I've also provided the SQL script in a file called 'Create_View_Script.sql' as part of the submission.

--Create view statement
CREATE VIEW Finaldataset AS

SELECT
tac.tweet_id,
tac.timestamp,
tac.rating_numerator,
tac.rating_denominator,
tapi.favorite_count,
tapi.retweet_count,
bip.confidencerating,
bip.prediction,
tac.dogstage,
CASE WHEN bip.classification = 1 THEN 'True' ELSE 'False' END AS classification
FROM twitterarchive_clean tac --this is the master table
LEFT JOIN (SELECT
tweet_id,
created_at,
favorite_count,
retweet_count
FROM twitterapidata_clean) tapi
ON tac.tweet_id = tapi.tweet_id --this left joins the api data to the master
LEFT JOIN (SELECT
A.tweet_id,
A.prediction,
A.confidencerating,
A.classification
FROM imageprediction_clean A
INNER JOIN (SELECT
tweet_id,
MAX(confidencerating) AS confidencerating
FROM imageprediction_clean
GROUP BY tweet_id) B
ON A.tweet_id = B.tweet_id
AND A.confidencerating = B.confidencerating) bip
ON tac.tweet_id = bip.tweet_id --this gets the most confident rating and links it to the master
WHERE (tapi.tweet_id AND bip.tweet_id) is not null --this excluded any tweet's that did not bring in any addisional information

In [1044]:

```
#Initiate the variables:
dblocation = os.path.join(os.getcwd(), r'WrangleAndAnalyzeProjectDB.db') #creates the d
atabase file location
dbconnection = 'sqlite:///' + dblocation #creates the connection object

#This code reads the view in from SQL:
finaldataset = pd.read_sql(sql='select * from Finaldataset',con=dbconnection)
```

In [1045]:

```
finaldataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1874 entries, 0 to 1873
Data columns (total 10 columns):
tweet_id              1874 non-null object
timestamp             1874 non-null object
rating_numerator      1874 non-null object
rating_denominator    1874 non-null object
favorite_count        1874 non-null int64
retweet_count         1874 non-null int64
confidencerating      1874 non-null float64
prediction            1874 non-null object
dogstage              283 non-null object
classification        1874 non-null object
dtypes: float64(1), int64(2), object(7)
memory usage: 146.5+ KB
```

The output above shows the general information for the final dataset. This set will now be written to a csv file and used for generating plots and insights within the next section

In [1046]:

```
#Write data to file:
finaldataset.to_csv('twitter_archive_master.csv')
```

In [1047]:

```
#write data to R working directory:
rfilelocation = os.path.join(os.getcwd(), r'AnalysisReport\twitter_archive_master.csv')
finaldataset.to_csv(rfilelocation)
```

In [1048]:

```
finaldataset.sample(10)
```

Out[1048]:

	tweet_id	timestamp	rating_numerator	rating_denominator	favori
975	684926975086034944	2016-01-07 02:38:10	11.0	10.0	3786
431	773670353721753600	2016-09-07 23:52:41	10.0	10.0	5847
1796	681891461017812993	2015-12-29 17:36:07	10.0	10.0	2653
1722	720340705894408192	2016-04-13 19:59:42	10.0	10.0	3075
1510	667455448082227200	2015-11-19 21:32:34	7.0	10.0	198
1771	687818504314159109	2016-01-15 02:08:05	12.0	10.0	2705
773	707021089608753152	2016-03-08 01:52:18	12.0	10.0	4366
1366	670361874861563904	2015-11-27 22:01:40	9.0	10.0	339
1859	825026590719483904	2017-01-27 17:04:02	12.0	10.0	6918
1142	675432746517426176	2015-12-11 21:51:30	12.0	10.0	1602

4 INSIGHTS

For the plots and insights I perform an analysis within a R markdown file called 'act_report.rdm' to practice the techniques learned as part of the nano degree. I provide this R project as part of my submission under the folder 'AnalysisReport'. To view the full report please see file act_report.html. For the report that briefly describes the wrangling efforts please view the wrangle_report.html file.