Data Wrangling Report

Introduction

This project report is an internal document I created to showcase data wrangling efforts while analysing Twitter WeRateGogs Community .

This project has given me an opportunity to learn and utilize different data wrangling specifically data gathering techniques taught under Udacity nanodegree program such as gather and access data from API , download from Internet and also from simple file systems.

I will elaborate all these methods one by one in below sections of paragraphs.

Project Details

The Data wrangling break down into 3 different stages:-
1) Gather Data
2)Access Data
3)Clean Data

Gathering Data

Data for this project obtained from 3 different places :-

- **Twitter Archive file-** This tweeter Archive file provided by Udacity itself which had all details regarding WeRateDogs Community This archive contains basic tweet data (tweet ID, timestamp, text, etc.) for all 5000+ of their tweets as they stood on August 1, 2017

```
#Access twitter archive file just to have alook
df_archive = pd.read_csv('twitter-archive-enhanced.csv')
df_archive.head()
```

- **Dogs Image Prediction file-** This file received by my Mentor which has Dog image prediction results containing urls, names, different algorithm's confidence figures which has confidence % numbers i.e. 95% etc that shows how correctly this algorithm guessed the dog's breed correctly from the image he received and Mentor got this data after sending dogs images to neural network. This file I downloaded from the Udacity server with the help of Python's requests library.

  ```
  #downloading image predicton file programatically from udacity server
  ```

```
response =
requests.get("https://d17h27t6h515a5.cloudfront.net/topher/2017/August/599fd2ad
_image-predictions/image-predictions.tsv")
with open('image-predictions.tsv', 'wb') as file:
    file.write(response.content)

df_image = pd.read_csv('image-predictions.tsv' , sep = '\t')
 #access and analyz this file

df_image.head()
```

- **Twitter API and JSON-** This file took the maximum of the efforts and was so much fun to interact with Twitter API to access data by authenticating with the help of token and Access keys provided by Twitter developer's forum by requesting access to them and later with the help of Python's Tweepy library access data from twitter api for almost 2000+ tweets by looping around these tweet ids which we already got from archive file.
  Twitter API is queried for each tweet's json data using **Tweepy** and later sore this json data important points likes retweets number, favorites or likes data into json .txt file after reading line by line.

```
#entering secret keys and tokens to access twitter api
consumer_key = ''
consumer_secret = ''
access_token = ''
access_secret = ''
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_secret)
```

```
api = tweepy.API(auth , wait_on_rate_limit=True,
wait_on_rate_limit_notify=True)
#including handling parameters for timeouts

# Testing functionality
tweet = api.get_status(892177421306343426, wait_on_rate_limit=True,
wait_on_rate_limit_notify=True)
tweet



# creating a list for tweets with exceptions/errors
tweets_error_list = []
# List of tweets
df_tweets = []
start = time.time()

# For loop which will add each available tweet json to df_list
for tweet_id in df_tweet_ids:
    try:
        tweet = api.get_status(tweet_id, tweet_mode= 'extended')._json

        favorites = tweet['favorite_count'] # number of favorites for the
tweet
        retweets = tweet['retweet_count'] # number of retweets
        user_followers = tweet['user']['followers_count'] # number of
followers of the user who tweeted
        user_favourites = tweet['user']['favourites_count'] # number of
favourites for the user who tweeted
```

```
        date_time = tweet['created_at'] # the timestamp i.e. date and time
of creation of the tweet
        df_tweets.append({'tweet_id': int(tweet_id),
                          'favorites': int(favorites),
                          'retweets': int(retweets),
                          'user_followers': int(user_followers),
                          'user_favourites': int(user_favourites),
                          'date_time': pd.to_datetime(date_time)})
    except Exception as e:
        print(str(tweet_id)+ " _ " + str(e))
        tweets_error_list.append(tweet_id)

# end time for excution
end = time.time()

#printing time for execution
print("Total time taken for execution", end - start)

# creating DataFrames
df_tweets_json = pd.DataFrame(df_tweets, columns = ['tweet_id',
'favorites', 'retweets',
                                        'user_favourites', 'date_time'])
# saving the dataFrame to file
df_tweets_json.to_csv('tweet_json.txt', encoding = 'utf-8', index=False)

# Read the saved tweet_json.txt file into a dataframe
df_tweetsdata = pd.read_csv('tweet_json.txt', encoding = 'utf-8')
```

```
df_tweetsdata.head()
```

END : GATHER DATA

The 3 dataframes are:- df_archive - contains data read from provided csv
df_image - contains data from tsv file downloaded from udacity server
df_tweetsdata - contains data after authenticating the twitter api by using tweepy library

Assessing Data

Data Assessment is done fully programmatically using Pythons Pandas library and later created the list for all the Quality and Tidiness issues identified which needs to be cleaned.

Issues found after Assessment:-

Quality Issues -

df_archive dataframe

1. Remove columns with missing data no longer needed missing data :- retweeted_status_id ,retweeted_status_user_id , retweeted_status_timestamp , in_reply_to_status_id,in_reply_to_user_id
2. timestamp should be datetime datatype
3. Clean the content of source column, make it more readable.
4. Nan and Null values for all columns removal
5. Fix Dogs name with single character or null values

df_tweetsdata dataframe

6. datetime should be of datatime datatype instead of string
7. user_favourites value is same for all rows and we can delete this column or keep it but its of no use
8. p1, p2 and p3 should be categoral datatype

Tidiness Issues -

df_archive dataframe

1. Combine dog stages into one column

df_image dataframe

2. p1_conf, p2_conf and p3_conf and p1_dog,p2_dog,p3_dog columns merged for ,meaningful insights and df_image should only have jpg_url and tweet_id, breed and image no info from p1,p2,p3 columns

All clean dataframes should be merged into one in the last

3. df_archive_clean df should be joined to df_image_clean and df_tweetsdata_clean

Cleaning Data

Data cleaning done exclusively using Python's pandas library as follows using Mentor provided template Define, Code, Test where we are defining each issue identified above, and code to fix it and later using simple access methods like info , values_counts etc accesing every fix to reconfirm.

There is one new thing I like and newly learned is while gathering all pandas dataframe dog's life stages column into one using melt method .

Below is template and cleaning process inshort.

Define

Tidy 1)Combine dog stages into one column life_stage

Code

In [39]:

```
# melt columns and merge into one stages column


melt_coumns = ['doggo', 'floofer', 'pupper', 'puppo']
stay_columns = [x for x in df_archive_clean.columns.tolist() if x not in melt_coumns]

# Melt the the columns into values
```

```
df_archive_clean = pd.melt(df_archive_clean, id_vars = stay_columns, value_vars = melt_coumns,
                var_name = 'stages', value_name = 'life_stage')
```

In [40]:

```
# Delete column 'stages'
df_archive_clean = df_archive_clean.drop('stages', 1)
```

Test

In [41]:

```
#check all columns again
df_archive_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9424 entries, 0 to 9423
Data columns (total 9 columns):
tweet_id              9424 non-null int64
timestamp             9424 non-null datetime64[ns]
source                9424 non-null object
text                  9424 non-null object
expanded_urls         9188 non-null object
rating_numerator      9424 non-null int64
rating_denominator    9424 non-null int64
name                  6008 non-null object
life_stage            411 non-null object
dtypes: datetime64[ns](1), int64(3), object(5)
memory usage: 662.7+ KB
```

In [42]:

```
df_archive_clean.life_stage.value_counts()
```

Out[42]:

```
pupper     272
doggo       98
puppo       37
floofer      4
Name: life_stage, dtype: int64
```

Define

Tidy 2)Creates a predicted dog breed column, based on the the confidence level of minimum 20% and 'p1_dog', 'p2_dog' and 'p3_dog' statements

Code

In [43]:

```
#image sample
```

df_image_clean.sample()

Out[43]:

	tweet_id	jpg_url	img_num	p1	p1_conf	p1_dog	p2	p2_conf	p2_dog	p3	p3_conf	p3_dog
156	668826086256599040	https://pbs.twimg.com/media/CUglxbFXAAA5O0d.jpg	1	malinois	0.640185	True	Irish_terrier	0.1537	True	Rhodesian_ridgeback	0.068457	True

In [44]:

```
df_image_clean['pred_breed'] = [df['p1'] if df['p1_dog'] == True and df['p1_conf'] > 0.2
            else df['p2'] if df['p2_dog'] == True and df['p2_conf'] > 0.2
            else df['p3'] if df['p3_dog'] == True and df['p3_conf'] > 0.2
            else np.nan for index, df in df_image_clean.iterrows()]
```

In [45]:

```
# Drop 'p1', 'p1_dog', 'p1_conf','p2', 'p2_dog', 'p2_conf','p3', 'p3_dog', 'p3_conf' columns
df_image_clean.drop(['p1', 'p1_dog', 'p1_conf','p2', 'p2_dog', 'p2_conf','p3', 'p3_dog', 'p3_conf'], axis = 1, inplace=True)
```

In [46]:

df_image_clean.head()

Out[46]:

	tweet_id	jpg_url	img_num	pred_breed
0	666020888022790149	https://pbs.twimg.com/media/CT4udn0WwAA0aMy.jpg	1	Welsh_springer_spaniel
1	666029285002620928	https://pbs.twimg.com/media/CT42GRgUYAA5iDo.jpg	1	redbone
2	666033412701032449	https://pbs.twimg.com/media/CT4521TWwAEvMyu.jpg	1	German_shepherd
3	666044226329800704	https://pbs.twimg.com/media/CT5Dr8HUEAA-lEu.jpg	1	Rhodesian_ridgeback
4	666049248165822465	https://pbs.twimg.com/media/CT5IQmsXIAAKY4A.jpg	1	miniature_pinscher

Test

In [47]:

```
#check all columns again
df_image_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2075 entries, 0 to 2074
Data columns (total 4 columns):
tweet_id      2075 non-null int64
jpg_url       2075 non-null object
img_num       2075 non-null int64
pred_breed    1472 non-null object
dtypes: int64(2), object(2)
memory usage: 64.9+ KB
```

Conclusion

Data Wrangling is an iterative process meaning you will or have to visit every step after some time again inorder to fully analyze the data thoroughly.

I have used python's pandas library exclusively while gathering, assessing and cleaning data
Python programming is definitely way more advantageous than excel for data wrangling is proved now, hence it might be one of the reason why its more popular in today's data community.
Handling , cleaning , visualizing data is way more easier in python programming, its scalable and reusable.