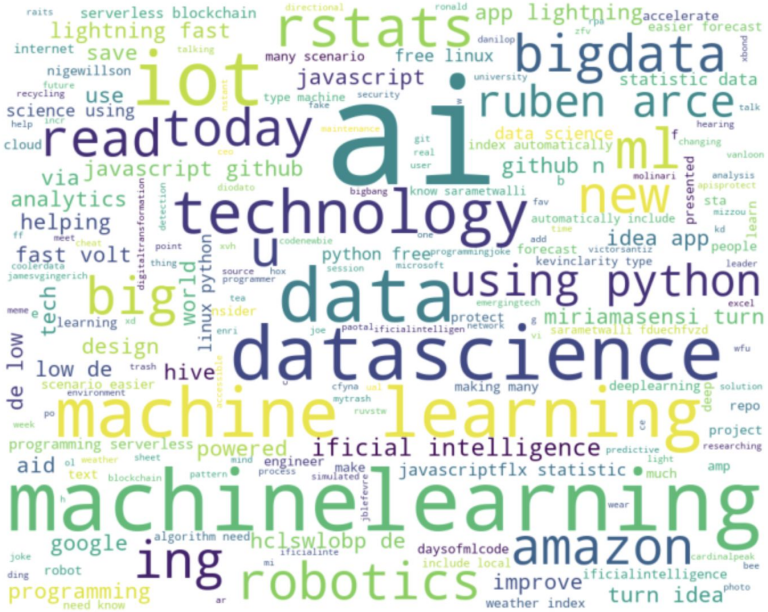
ETL Pipeline Using Twitter API, Mongo, & Python

Justin Greenberg, Irbaz Bin Riaz, Ramakrishna Devarakonda, Craig Fulgency



Project Idea

- Livestream Tweets using Twitter API to MongoDB
- Build ETL pipeline using Python, Pandas
  - Google API as second data source
  - Transformation included cleaning, renaming, merging, deleting duplicates and null values
  - Load transformed data back into MongoDB

API

mongoDB

Extract

- Livestream into Mongo
- Use **Tweepy** (Python wrapper for Twitter API)
- Data in JSON form
  - Credentials saved as external JSON file

Create a StreamListener Class to scrap twitter API

```python
class StreamListener(tweepy.StreamListener):
    # This is the class provided by tweepy to access twitter Streaming API

    def on_connect(self):
        # called initially to connect to the streaming API
        print("you are connected to the streaming API")


    def on_error(self,status_code):
        # on error - if an error occurs displays the error / status code
        print("An Error has occured: " + repr(status_code))
        return False

    def on_data(self,data):
        # this part is main script where we connect to MongoDB and stores the tweet
        try:
            client = MongoClient(MONGO_HOST)

            # use twitterdb database. if it doesn't existed, it will be created
            db= client.twitterdb

            # Decode the JSON response from Twitter
            datajson = json.loads(data)

            # grab the Created_at data from the tweet to use for display
            created_at = datajson['created_at']

            # print out the message on every successful tweet we collected
            print("Tweet collected at " + str(created_at))

            # drop db if its existed
            db.twitter_search.drop()

            # insert the data into the mongodb collection called twitter_search
            # it twitter_search is not existed, it will be created
            db.twitter_search.insert(datajson)

        except Exception as e:
            print(e)
```

```python
WORDS =['#bigdata','#AI','#datascience','#machinelearning','#ml','#iot']
```

Set up Twitter API tokens and consumer keys

```python
# load credentails from twitter_credentails.json
with open('twitter_crdentials.json','r') as file:
    creds = json.load(file)
```

```python
# authentification so we can access twitter
auth = tweepy.OAuthHandler(creds['CONSUMER_KEY'],creds['CONSUMER_SECRET']
auth.set_access_token(creds['ACCESS_KEY'], creds['ACCESS_SECRET'])
api =tweepy.API(auth, wait_on_rate_limit=True)

# create instance of Streamlistener
listener = StreamListener(api = api)
stream = tweepy.Stream(auth, listener = listener)

#track = ['golf', 'masters', 'reed', 'mcilroy', 'woods']
#track = ['nba', 'cavs', 'celtics', 'basketball']
# choose what we want to filter by
#print('Tracking: ' +str(WORDS))

# create a filter with query and targeted language
stream.filter(track = WORDS,languages=['en'])
```

Transform: Loading the data into pandas df

Read data from mongoDB

```
import pymongo
import pandas as pd
from pymongo import MongoClient
```

```
client = MongoClient("mongodb://localhost:27017")
```

```
db = client['twitter_db']
```

```
#db.tweets.drop()
```

```
tweets = db['twitter_search']
```

```
df = pd.DataFrame(list(tweets.find()))
```

twitter_db.twitter_search

DOCUMENTS 118.5k

Documents | Aggregations | Schema | Explain Plan | Indexes

FILTER

ADD DATA | VIEW | Display


```
_id: ObjectId("5fd512ae1c612b9b42dd53f9")
created_at: "Sat Dec 12 18:57:45 +0000 2020"
id: 1337834038262829056
id_str: "1337834038262829056"
text: "RT @SourabhSKatoch: Daily Life as a Data Scientist: Work From Home Edi..."
source: "<a href="https://mobile.twitter.com" rel="nofollow">Twitter Web App</a..."
truncated: false
in_reply_to_status_id: null
in_reply_to_status_id_…: null
in_reply_to_user_id: null
in_reply_to_user_id_str: null
in_reply_to_screen_name: null
> user: Object
geo: null
coordinates: null
place: null
contributors: null
> retweeted_status: Object
is_quote_status: false
quote_count: 0
reply_count: 0
retweet_count: 0
favorite_count: 0
> entities: Object
favorited: false
```


SHOW 5 MORE FIELDS

Over 90 columns in Twitter response JSON!

Transform: Cleaning, Filtering

Identify Duplicate rows

```
duplicates = df[df.duplicated('id')]
duplicates
```

	_id	created_at	id	id_str	text	source	truncated	in_reply_to_status_id	in_reply_to_status_id_str	in_reply_to_user_id	...	filter_level	lang	timestamp_ms

0 rows × 37 columns

```
# prepare a list to Drop unwanted columns
list_unwanted_col = ['_id','id','truncated','in_reply_to_status_id','in_reply_to_status_id_str','in_reply_to_user_id','in_reply_to_user_id_str',\
                     'in_reply_to_screen_name','geo','coordinates','retweet_count','place','contributors','is_quote_status','extended_tweet','quote_count','reply_count',\
                     'favorite_count','entities','favorited','retweeted','possibly_sensitive','filter_level','timestamp_ms','quoted_status_id',\
                     'quoted_status_id_str','quoted_status','quoted_status_permalink','retweeted_status','display_text_range','extended_entities','created_at', 'source']

# create a new data frame with clean data
new_df = df.drop(list_unwanted_col,axis=1)
```

```
new_df.head()
```

	id_str	text	user	lang
0	1337464014419480576	Interesting... Machine Learning and AI - What ...	{'id': 22631958, 'id_str': '22631958', 'name':...	en
1	1337464027719499777	Take a look at these open positions in applied...	{'id': 1079825507737202688, 'id_str': '1079825...	en
2	1337464032564027392	RT @Xbond49: Honored & humbled to be in th...	{'id': 716658880508510208, 'id_str': '71665888...	en
3	1337464037680955392	@sciencebase add #IoT data/log with current lo...	{'id': 15081182, 'id_str': '15081182', 'name':...	en
4	1337464049794076672	iShares Robotics And Artificial Intelligence M...	{'id': 856240505826496513, 'id_str': '85624050...	en

‘user’ column populated by many subcolumns

Transform: Extracting user information

```
user_df = pd.DataFrame.from_dict([new_df['user'][i] for i in range(len(new_df['user']))])
```

```
user_df.columns
```

```
Index(['id', 'id_str', 'name', 'screen_name', 'location', 'url', 'description',
       'translator_type', 'protected', 'verified', 'followers_count',
       'friends_count', 'listed_count', 'favourites_count', 'statuses_count',
       'created_at', 'utc_offset', 'time_zone', 'geo_enabled', 'lang',
       'contributors_enabled', 'is_translator', 'profile_background_color',
       'profile_background_image_url', 'profile_background_image_url_https',
       'profile_background_tile', 'profile_link_color',
       'profile_sidebar_border_color', 'profile_sidebar_fill_color',
       'profile_text_color', 'profile_use_background_image',
       'profile_image_url', 'profile_image_url_https', 'profile_banner_url',
       'default_profile', 'default_profile_image', 'following',
       'follow_request_sent', 'notifications'],
      dtype='object')
```

In order to merge later, it is necessary to add a common column

```
user_df['id_str'] = new_df['id_str']
```

Making sure to drop duplicates which resulted from running code many times

Identify Duplicate rows from user_df

```
user_df.loc[user_df['id'] == 1331034000404647936 ]
```

```
duplicates = user_df[user_df.duplicated('id')]
```

Drop Duplicate rows from user_df

```
user_df=user_df.drop_duplicates(subset=['id'])
user_df
```

4195 rows × 39 columns

Transform: Cleaning

Remove unwanted columms from user_df

```
user_unwanted_col = ['id', 'url','translator_type','protected','verified','utc_offset','time_zone','geo_enabled',\
                     'lang','contributors_enabled','is_translator','profile_background_color','profile_background_image_url',\
                      'profile_background_image_url_https','profile_background_tile', 'profile_link_color',\
                      'profile_sidebar_border_color', 'profile_sidebar_fill_color','profile_text_color', 'profile_use_background_image',\
                      'profile_banner_url','default_profile', 'default_profile_image', 'following','follow_request_sent', 'notification
s','profile_image_url','profile_image_url_https']
```

```
clean_user_df = user_df.drop(user_unwanted_col,axis=1)
clean_user_df
```

Plan on calling to Google API, so drop null location values

```
clean_user_df=clean_user_df.dropna(subset=['location'])
```

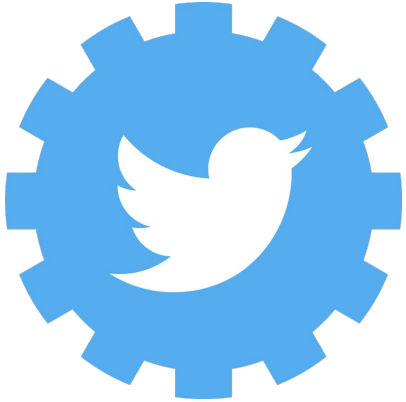
Transform: Merge

```
# Merge new_df clean_user_df
combined_df = clean_user_df.merge(new_df,on='id_str')
combined_df.columns
```

```
Index(['id_str', 'name', 'screen_name', 'location', 'description',
       'followers_count', 'friends_count', 'listed_count', 'favourites_count',
       'statuses_count', 'created_at', 'text', 'user', 'lang'],
      dtype='object')
```

```
combined_df=combined_df.drop(columns=['user'])
```

3210 rows × 13 columns



Extracting Second Data Source: Google API

- Call to API and store latitude and longitude in data frame
  - Must remove space between city and state
- Add ‘id_str’ to new dataframe in order to merge later
- Google API call limits impeded data collection (1000 max)

```
user_locations = pd.DataFrame(combined_df['location'].str.replace(' ',''))
user_locations['id_str']=pd.DataFrame(combined_df['id_str'])
user_locations
```

	location	id_str
0	Harrisburg,PA	1337464014419480576
1	Baltimore,MD	1337464027719499777
2	Online	1337464032564027392
3	LasVegas,NV	1337464037680955392
4	London,UK	1337464049794076672
...	...	...
3205	Athens,Greece	1338932428140916737
3206	UnitedStates	1338932465193390083
3207	UnitedStates	1338932468779536385
3208	GloballyMonitoredDataFusion	1338932488945737728
3209	Atlanta	1338941910916993024

3210 rows × 2 columns

check if there is any duplicate rows with same id's

```
user_locations[user_locations.duplicated()]
```

	location	id_str

Google API Call & store latitude, longitude

```python
from config import gkey
import requests
lat=[]
lng=[]
id_str=[]
for index,location in user_locations.iterrows():

    #Build and end point URL
    target_url = ('https://maps.googleapis.com/maps/api/geocode/json?'
    'address={0}&key={1}').format(location['location'], gkey)

    print(target_url)
    id_str.append(location['id_str'])
    # Run a request to endpoint and convert result to json
    geo_data = requests.get(target_url).json()

    try:
        # Extract latitude and longitude
        lat.append(geo_data["results"][0]["geometry"]["location"]["lat"])
        lng.append(geo_data["results"][0]["geometry"]["location"]["lng"])
    except:
        print(location)
        pass
```

```
https://maps.googleapis.com/maps/api/geocode/json?address=Harrisburg,PA&key=AIzaSyAuQLcV7VJEMolYoEVym53T8m6B86UOKaI
https://maps.googleapis.com/maps/api/geocode/json?address=Baltimore,MD&key=AIzaSyAuQLcV7VJEMolYoEVym53T8m6B86UOKaI
https://maps.googleapis.com/maps/api/geocode/json?address=Online&key=AIzaSyAuQLcV7VJEMolYoEVym53T8m6B86UOKaI
https://maps.googleapis.com/maps/api/geocode/json?address=LasVegas,NV&key=AIzaSyAuQLcV7VJEMolYoEVym53T8m6B86UOKaI
https://maps.googleapis.com/maps/api/geocode/json?address=London,UK&key=AIzaSyAuQLcV7VJEMolYoEVym53T8m6B86UOKaI
https://maps.googleapis.com/maps/api/geocode/json?address=Wesupportworldwide&key=AIzaSyAuQLcV7VJEMolYoEVym53T8m6B86UOKaI
https://maps.googleapis.com/maps/api/geocode/json?address=UNITEDSTATESOFEUROPE&key=AIzaSyAuQLcV7VJEMolYoEVym53T8m6B86UOKaI
https://maps.googleapis.com/maps/api/geocode/json?address=JustwereIhavetobe.&key=AIzaSyAuQLcV7VJEMolYoEVym53T8m6B86UOKaI
location    JustwereIhavetobe
```

create a dataframe with lat & lng

```python
lat_lng_df=pd.DataFrame()
lat_lng_df['id_str']=[]
lat_lng_df['lat']=[]
lat_lng_df['lng']=[]
lat_lng_df['id_str']=id_str
lat_lng_df['lat']=pd.DataFrame(lat)
lat_lng_df['lng']=pd.DataFrame(lng)
```

```python
lat_lng_df
```

	id_str	lat	lng
0	1337464014419480576	40.273191	-76.886701
1	1337464027719499777	39.290385	-76.612189
2	1337464032564027392	36.416750	-94.222944
3	1337464037680955392	36.169941	-115.139830
4	1337464049794076672	51.507351	-0.127758
...	...	...	...
1143	1338875103921270785	NaN	NaN
1144	1338875122208415745	NaN	NaN
1145	1338875127904280576	NaN	NaN
1146	1338875150993862658	NaN	NaN
1147	1338875160183627782	NaN	NaN

1148 rows × 3 columns

Transform: Merging to create final df of original, user, lat_lng dfs

```
final_df=combined_df.merge(lat_lng_df, on='id_str')
final_df
```

Drop null values and rename columns

```
final_df=final_df.dropna(subset=['lat', 'lng'])
```

```
final_df=final_df.rename(columns={'created_at_y': "account_creation_date", 'lang': 'language'})
```

Load

```
db.tweet_analysis.drop()
```

```
#connect to mongo
MONGO_HOST = 'mongodb://localhost/twitteranalysis'
client = MongoClient(MONGO_HOST)
# use twitteranalysis database. if it doesn't exist, it will be created
db= client['twitterdb']
#collection name
twitter_collection = db.tweet_analysis
```

```
#insert data into targeted database
final_df.reset_index(inplace=True)
data_dict = final_df.to_dict('records')
db.tweet_analysis.insert_many(data_dict)
```

Now we have a database of tweets related to data science, machine learning, AI, etc. ready to be analyzed

MongoDB Compass - localhost:27017/twitterdb.tweet_analysis

Local

11 DBS 12 COLLECTIONS

FAVORITE

HOST
localhost:27017

CLUSTER
Standalone

EDITION
MongoDB 4.4.0 Community

Filter your data

- ClassDB
- Dumpster_DB
- admin
- config
- craigslist_app
- fruits_db
- local
- team_db
- travel_db
- twitterdb
  - tweet_analysis
  - tweets
  - twitter_search
- weather_app

twitterdb.tweet_analysis
Documents

twitterdb.tweet_analysis

DOCUMENTS 949 TOTAL SIZE 585.7KB AVG. SIZE 632B

INDEXES 1 TOTAL SIZE 24.0KB AVG. SIZE 24.0KB

Documents | Aggregations | Schema | Explain Plan | Indexes | Validation

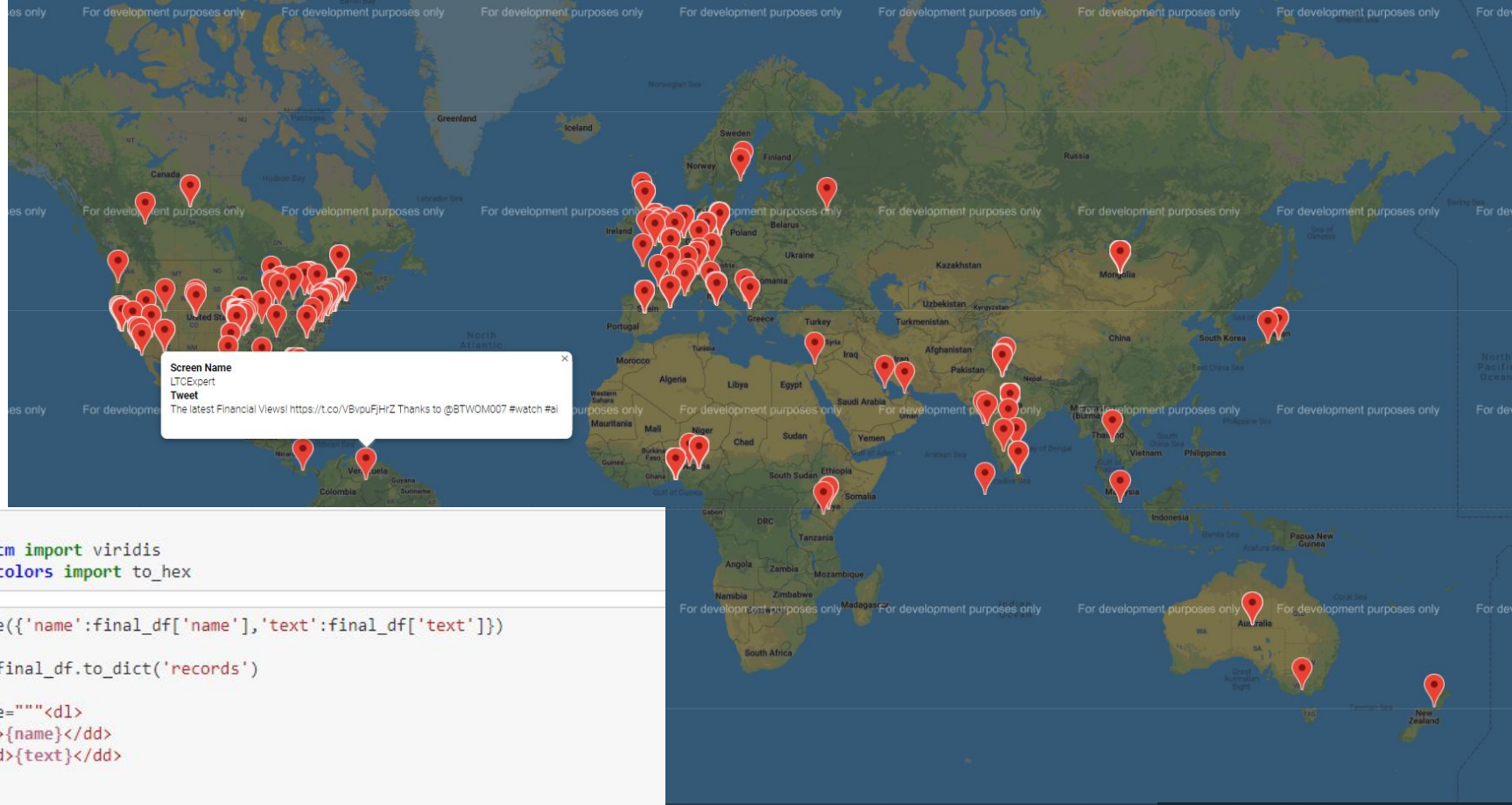
FILTER | OPTIONS | FIND | RESET

ADD DATA | VIEW | Displaying documents 1 - 20 of 949 | REFRESH


```
_id: ObjectId("5fda24977c1bd6113ea752ee")
index: 0
id_str: "1337464014419480576"
name: "Chuck Russell"
screen_name: "cichuck"
location: "Harrisburg, PA"
description: "Founder Collective Intelligence  #TheFutureOfWork #BMGI #TheBigShift #..."
followers_count: 9351
friends_count: 7702
listed_count: 601
favourites_count: 4516
statuses_count: 67874
created_at: "Tue Mar 03 15:05:32 +0000 2009"
text: "Interesting... Machine Learning and AI – What Does The Future Hold? – ..."
language: "en"
lat: 40.2731911
lng: -76.8867008
```

```
_id: ObjectId("5fda24977c1bd6113ea752ef")
index: 1
id_str: "1337464027719499777"
name: "Alliance for Artificial Intelligence in Healthcare"
screen_name: "theaaih"
location: "Baltimore, MD"
description: "Global organization to educate and advocate for AI in healthcare. Prov..."
followers_count: 1041
friends_count: 455
listed_count: 41
favourites_count: 858
statuses_count: 745
created_at: "Mon Dec 31 19:43:58 +0000 2018"
```


> _MongoSH Beta




```python
import gmaps
from matplotlib.cm import viridis
from matplotlib.colors import to_hex
```

```python
df = pd.DataFrame({'name':final_df['name'],'text':final_df['text']})
#df.fillna(0)
user_location = final_df.to_dict('records')

info_box_template="""<dl>
<dt>Name</dt><dd>{name}</dd>
<dt>Tweet</dt><dd>{text}</dd>
</dl>
"""
latlng = final_df[['lat', 'lng']]
location_info =[info_box_template.format(**location) for location in user_location]
marker_layer = gmaps.marker_layer(latlng, info_box_content=location_info)
fig = gmaps.figure()
fig.add_layer(marker_layer)
fig
```

Additional analysis on tech tweets resulted in this **Gmap**

Developer