Updated Problem Formulation

As stated in the previous problem statement, we planned on analyzing real-time tweets to predict earthquakes. Since we did not receive access to the Twitter Developer Portal, we have updated our problem formulation to get the historical data from Twitter and simulate a real-time stream of data with the help of Kafka Stream. This simulation will replicate the real-time data we would receive using the Twitter API.

Literature Review

[1] This paper discusses how Twitter can be used as a source of information and proposes a method for detecting real-time events using Convolution Neural Network (CNN). The CNN model is trained on past earthquake tweets labeled by crowdsourcing and used to predict whether a tweet containing the earthquake keyword is informative. The informative tweets are then used as input streaming data for the real-time event detection algorithm. This system can detect earthquakes with a tolerance level faster than official government disaster websites.

[2]This paper investigates using Twitter as a real-time interaction platform to detect events such as earthquakes. The authors propose an algorithm that uses a classifier to monitor tweets and detect target events based on features such as keywords and context. They also devise a probabilistic spatiotemporal model for the event and use Kalman filtering and particle filtering for location estimation. The proposed system is applied to construct an earthquake reporting system in Japan that detects earthquakes with high probability and sends emails to registered users much faster than official announcements by the Japan Meteorological Agency.

[3]The proposed system in this paper uses deep learning techniques such as RNN/LSTM to validate Twitter tweets and provide real-time detection of earthquakes. Machine learning models are trained from past labeled tweets related to earthquakes and used as classifiers to predict the validity of tweets. The system listens to particular keywords like "earthquake" using Twitter's API and converts tweets to word embeddings using BERT before inputting them into the model. The proposed system can detect earthquakes with a higher tolerance level than existing websites and provide earlier warnings to the public.

[4]This paper outlines two main approaches to tracking and analyzing hashtag-based Twitter activities during a crisis: using an open-source tool like yourTwapperkeeper and additional tools to process and visualize Twitter activities. The other requires custom-designed tracking and analysis tools like Gawk.

The Proposed Method

Our problem solution first requires us to extract the tweets from the historic data streamed as live data using kafka stream. The tweets will be extracted based on the word and hashtags like earthquake, stuck, help,#earthquake, and many more. These extracted tweets will become our dataset. The dataset will have Tweet, the user's location who has tweeted, as a major feature that can be used further.

In our project, we have used the BERT algorithm. BERT can be used to analyze the data and extract useful information such as an earthquake's location, magnitude, and depth. Here, we have extracted the location referred to in the tweet with the help of this algorithm.

After the application of BERT, we have two locations, first the user's location and second the location that we extract from the tweets.

Eventually, we will find out the location with the most occurrence from all the tweets and then compare it with the user's location. Comparison will happen as the tweet's most referred location will be found in the user's location, and if the same location has occurred in the user's location more than a threshold set up by us, we will be able to detect that location is likely to have had an earthquake.

Thereafter, we will create a system that will notify the nearby NGOs about the earthquake so that they can provide necessary aid immediately.

Updated Baseline Results

Create a named entity recognition pipeline using the BERT- base-cased pre-trained model from the Hugging Face Transformations library.

Import required python packages

```python
from transformers import pipeline

nlp = pipeline("ner", model="bert-base-cased")
```

```
Downloading (…)lve/main/config.json: 100% 570/570 [00:00<00:00, 13.2kB/s]
Downloading pytorch_model.bin: 100% 436M/436M [00:04<00:00, 103MB/s]
Some weights of the model checkpoint at bert-base-cased were not used when initializing BertForTokenClassification: ['cls.predictions.transform.dense.bias', 'cls.predictions.transform.
- This IS expected if you are initializing BertForTokenClassification from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSe
- This IS NOT expected if you are initializing BertForTokenClassification from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassifi
Some weights of BertForTokenClassification were not initialized from the model checkpoint at bert-base-cased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Downloading (…)okenizer_config.json: 100% 29.0/29.0 [00:00<00:00, 338B/s]
Downloading (…)solve/main/vocab.txt: 100% 213k/213k [00:00<00:00, 1.26MB/s]
Downloading (…)/main/tokenizer.json: 100% 436k/436k [00:00<00:00, 2.28MB/s]
```

```python
import spacy
import re
!python3 -m spacy download xx_ent_wiki_sm
```

```
/usr/local/lib/python3.9/dist-packages/torch/cuda/__init__.py:497: UserWarning: Can't initialize NVML
  warnings.warn("Can't initialize NVML")
2023-03-19 19:16:43.627024: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libnvinfer.so.7'; dlerror: libnvinfer.so.7: car
2023-03-19 19:16:43.627164: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libnvinfer_plugin.so.7'; dlerror: libnvinfer_pl
2023-03-19 19:16:43.627196: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Cannot dlopen some TensorRT libraries. If you would like to use Nvidia GPU with Tens
```

Previous Algorithm used to find locations

- The code loads necessary libraries, including spaCy, regular expression, and Geopy.
- It defines an example text that contains information about various cities and countries.
- It loads a pre-trained spaCy model for named entity recognition (NER) and extracts entities of type GPE (geo-political entity) from the text.

- It uses Geopy to get location information for each identified city, including its country.
- The extracted city-country pairs are stored in a dictionary and printed out at the end. If no cities are found in the text, it prints a message indicating that.

```python
# Load necessary libraries
import spacy
import re
from geopy.geocoders import Nominatim

# Load pre-trained BERT model and tokenizer
nlp = spacy.load('en_core_web_sm')

# Define example text
text = "I have lived in Delhi united States,Delhi united States, Germany and Berlin,Delhi united States, Connecticut. Also visited Paris, Rome and Mumbai."

# Extract entities using spacy NER
geolocator = Nominatim(user_agent="my_app")
city_country_pairs = {}
doc = nlp(text)
for ent in doc.ents:
    if ent.label_ == 'GPE':
        city = ent.text
        if city in city_country_pairs:
            continue
        try:
            # Get location information of city using geopy
            location = geolocator.geocode(city)
            country = location.raw['display_name'].split(",")[-1].strip()
        except:
            country = 'unknown'
        city_country_pairs[city] = country

# Print extracted city-country pairs
if city_country_pairs:
    print("Cities and their countries:")
    for city, country in city_country_pairs.items():
        print(f"{city}: {country}")
else:
    print("No cities found in text.")
```

```
Cities and their countries:
Delhi united States: United States
Germany: Deutschland
Berlin: Deutschland
Connecticut: United States
Paris: France
Rome: Italia
Mumbai: India
```

```python
[ ] import spacy
    from spacy import displacy
    !python3 -m spacy download xx_ent_wiki_sm
    !pip install IPython
```

```
2023-03-19 19:17:11.850977: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libnvinfer_plugin.so.7'; dlerror: libnvinfer_plugin.so.7: cannot open shared object file: No
2023-03-19 19:17:11.851001: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Cannot dlopen some TensorRT libraries. If you would like to use Nvidia GPU with TensorRT, please make sure the missing libraries m
2023-03-19 19:17:13.665847: E tensorflow/compiler/xla/stream_executor/cuda/cuda_driver.cc:267] failed call to cuInit: CUDA_ERROR_NO_DEVICE: no CUDA-capable device is detected
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting xx-ent-wiki-sm==3.5.0
  Using cached https://github.com/explosion/spacy-models/releases/download/xx_ent_wiki_sm-3.5.0/xx_ent_wiki_sm-3.5.0-py3-none-any.whl (11.1 MB)
Requirement already satisfied: spacy<3.6.0,>=3.5.0 in /usr/local/lib/python3.9/dist-packages (from xx-ent-wiki-sm==3.5.0) (3.5.1)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (3.3.0)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (2.0.8)
Requirement already satisfied: pathy>=0.10.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (0.10.1)
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (1.22.4)
Requirement already satisfied: typer<0.8.0,>=0.3.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (0.7.0)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (2.27.1)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (2.0.7)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (2.4.6)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<1.11.0,>=1.7.4 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (1.10.6)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (3.1.2)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (1.0.9)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (23.0)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (1.1.1)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (3.0.12)
Requirement already satisfied: setuptools in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (63.4.3)
Requirement already satisfied: thinc<8.2.0,>=8.1.8 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (8.1.9)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (1.0.4)
Requirement already satisfied: smart-open<7.0.0,>=5.2.1 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (6.3.0)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (4.65.0)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.9/dist-packages (from spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (3.0.8)
Requirement already satisfied: typing-extensions>=4.2.0 in /usr/local/lib/python3.9/dist-packages (from pydantic!=1.8,!=1.8.1,<1.11.0,>=1.7.4->spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (4.5.0)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (2.0.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (3.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (2022.12.7)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.6.0,>=3.5.0->xx-ent-wiki-sm==3.5.0) (1.26.15)
```

Dropping the rows containing missing values(i.e., NaN values) from the dataframe.
Reset the index of a pandas Dataframes.

```
[ ] dataf=dataf.dropna()
```

```
dataf=dataf.reset_index()
dataf
```

	index	Date	User	Tweet	Location
0	0	2023-03-19 19:11:01+00:00	LastQuake	Help us keep doing what we've always done: inf...	based in Paris, works globally
1	2	2023-03-19 12:45:41+00:00	Benefit_Mankind	This #Ramadhan, help us rebuild lives for the ...	UK
2	3	2023-03-19 12:32:51+00:00	WorldHelpTsunam	#earthquake	World
3	5	2023-03-19 07:24:10+00:00	msi_press	This Caturday's post shares information from S...	Hollister, California
4	6	2023-03-19 03:54:46+00:00	therealishfaq	â□¡#ATENCIÃ□N| AsÃ se siento el Terremoto de ...	Lucknow, India
...	...	...	...	...	...
395	493	2023-02-26 13:17:53+00:00	BarzaniCF	#BCF quickly responded to the #earthquake that...	Iraq
396	494	2023-02-26 13:11:10+00:00	UNHCRinSYRIA	The #earthquake caused not only material damag...	Syria
397	495	2023-02-26 12:57:05+00:00	withfahimkhan	Emergency Earthquake Relief for Turkey ð□□'ð□□...	Earth
398	497	2023-02-26 12:26:52+00:00	basmehzeitooneh	20 days after the devastating #earthquake disa...	Beirut, Lebanon
399	499	2023-02-26 11:28:46+00:00	bawali_patrkar	Just imagine this situation, where loved one i...	Noida, India

400 rows × 5 columns

Replacing non-word characters and underscores with spaces in the 'Tweet' and 'Location' columns of a DataFrame.

```
[ ] j=0
    for i in dataf['User']:
      dataf['Tweet'][j]= re.sub(r'[^\w]',' ',dataf['Tweet'][j] ).replace("_"," ")
      dataf['Location'][j]= re.sub(r'[^\w]',' ',dataf['Location'][j]).replace("_"," ")
      j=j+1
    dataf.head()
```

```
<ipython-input-41-871a5a5c753b>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataf['Tweet'][j]= re.sub(r'[^\w]',' ',dataf['Tweet'][j] ).replace("_"," ")
<ipython-input-41-871a5a5c753b>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataf['Location'][j]= re.sub(r'[^\w]',' ',dataf['Location'][j]).replace("_"," ")
```

	index	Date	User	Tweet	Location
0	0	2023-03-19 19:11:01+00:00	LastQuake	Help us keep doing what we ve always done inf...	based in Paris works globally
1	2	2023-03-19 12:45:41+00:00	Benefit_Mankind	This Ramadhan help us rebuild lives for the ...	UK
2	3	2023-03-19 12:32:51+00:00	WorldHelpTsunam	earthquake	World
3	5	2023-03-19 07:24:10+00:00	msi_press	This Caturday s post shares information from S...	Hollister California
4	6	2023-03-19 03:54:46+00:00	therealishfaq	â ATENCIÃ N AsÃ se siento el Terremoto de ...	Lucknow India

Defining a function punctuation_removal to remove all punctuation from the 'Tweet' and 'Location' columns of a DataFrame and prints the first few rows of the modified columns. The function is called on the DataFrame dataf. Importing the necessary libraries to remove stop words.

```
[ ] def punctuation_removal(df):
        df['Tweet'] = df['Tweet'].str.replace('[^\w\s]','')
        df['Location'] = df['Location'].str.replace('[^\w\s]','')
        print(df['Tweet'].head())
        print(df['Location'].head())
    punctuation_removal(dataf)
```

```
0    Help us keep doing what we ve always done  inf...
1    This  Ramadhan  help us rebuild lives for the ...
2                                           earthquake
3    This Caturday s post shares information from S...
4    â   ATENCIÃ N  AsÃ  se siento el Terremoto de ...
Name: Tweet, dtype: object
0    based in Paris  works globally
1                                UK
2                             World
3              Hollister  California
4                     Lucknow  India
Name: Location, dtype: object
<ipython-input-42-b0335ce412a8>:2: FutureWarning: The default value of regex will change from True to False in a future version.
  df['Tweet'] = df['Tweet'].str.replace('[^\w\s]','')
<ipython-input-42-b0335ce412a8>:3: FutureWarning: The default value of regex will change from True to False in a future version.
  df['Location'] = df['Location'].str.replace('[^\w\s]','')
```

```
[ ] #number of stop words
    import nltk
    nltk.download('stopwords')
    from nltk.corpus import stopwords
    stop = stopwords.words('english')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Removing the stop words from the ‘Tweet’ and ‘Location’ columns using the lambda function and printing the sample data.

```
[ ] #removing stop words
    def stop_words_removal(df):
        df['Tweet'] = df['Tweet'].apply(lambda x: " ".join(x for x in x.split() if x not in stop))
        df['Location'] = df['Location'].apply(lambda x: " ".join(x for x in x.split() if x not in stop))
        print(df['Tweet'].head())
        print(df['Location'].head())
    stop_words_removal(dataf)
```

```
0    Help us keep always done inform reassure citiz...
1    This Ramadhan help us rebuild lives families l...
2                                           earthquake
3    This Caturday post shares information Syria af...
4    â ATENCIÃ N AsÃ se siento el Terremoto de 6 5 ...
Name: Tweet, dtype: object
0    based Paris works globally
1                            UK
2                         World
3          Hollister California
4                 Lucknow India
Name: Location, dtype: object
```

```
[ ] punctuation_removal(dataf)
```

```
<ipython-input-42-b0335ce412a8>:2: FutureWarning: The default value of regex will change from True to False in a future version.
  df['Tweet'] = df['Tweet'].str.replace('[^\w\s]','')
0    Help us keep always done inform reassure citiz...
1    This Ramadhan help us rebuild lives families l...
2                                           earthquake
3    This Caturday post shares information Syria af...
4    â ATENCIÃ N AsÃ se siento el Terremoto de 6 5 ...
Name: Tweet, dtype: object
0    based Paris works globally
1                            UK
2                         World
3          Hollister California
4                 Lucknow India
Name: Location, dtype: object
<ipython-input-42-b0335ce412a8>:3: FutureWarning: The default value of regex will change from True to False in a future version.
  df['Location'] = df['Location'].str.replace('[^\w\s]','')
```

- Importing the NLTK module and defining a function perform_tokenization that tokenizes a string and appends the tokens to a list tk.
- This function is applied to the 'Tweet' column of a DataFrame dataf using a loop that iterates through each row of the DataFrame.

```
[ ] tk=[]
    import nltk
    #nltk.download('punkt')
    from nltk.tokenize import word_tokenize
    def perform_tokenization(word):
        tokens=list(set(word_tokenize(word)))
        tk.extend(tokens)
        return tokens
    for i in range(0,j-1):
      dataf.iloc[i]['Tweet']=perform_tokenization(dataf.iloc[i]['Tweet'])
```

```
<ipython-input-46-c54995858912>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataf.iloc[i]['Tweet']=perform_tokenization(dataf.iloc[i]['Tweet'])
```

```
[ ] dataf.iloc[1]['Tweet']
```

```
'This Ramadhan help us rebuild lives families lost everything recent earthquake claimed 50 000 lives unfortunately Support https co 09JiWvqJCY https co aigFpBQmqp'
```

```
[ ] df=df.dropna()
    df
```

	Location	address
0	Syria	(سوريا, (34.6401861, 39.0494106))
1	Terremoto	(Las Hiedritas, Badiraguato, Sinaloa, México, ...
2	Turkiye	(Türkiye, (38.9597594, 34.9249653))
3	India	(India, (22.3511148, 78.6677428))
4	Turkey	(Türkiye, (38.9597594, 34.9249653))
...	...	...
376	Turkey	(Türkiye, (38.9597594, 34.9249653))
377	Turkey	(Türkiye, (38.9597594, 34.9249653))
378	Turkey	(Türkiye, (38.9597594, 34.9249653))
379	Syria	(سوريا, (34.6401861, 39.0494106))
380	Turkey	(Türkiye, (38.9597594, 34.9249653))

334 rows × 2 columns

```
[ ] df=df.reset_index(drop=True)
    df
```

	Location	address
0	Syria	(سوريا, (34.6401861, 39.0494106))
1	Terremoto	(Las Hiedritas, Badiraguato, Sinaloa, México, ...
2	Turkiye	(Türkiye, (38.9597594, 34.9249653))
3	India	(India, (22.3511148, 78.6677428))
4	Turkey	(Türkiye, (38.9597594, 34.9249653))
...	...	...

Downloading a spaCy model 'xx_ent_wiki_sm,' loading it that sets the maximum text length. It then tokenizes a list of strings, joins them into a single string, creates a spaCy Doc object, extracts named entities from the Doc object, and prints them with their entity labels.

```
!python -m spacy download xx_ent_wiki_sm
import spacy

nlp = spacy.load('xx_ent_wiki_sm')
nlp.max_length = 2000000

# Define a list of tokens
tokens=tk

# Join the tokens into a string
text = " ".join(tokens)

# Create a spaCy Doc object from the string
doc = nlp(text)

# Extract named entities from the Doc object
entities = list(doc.ents)

# Print the named entities
for entity in entities:
    print(entity.text, entity.label_)
```

```
✓ Download and installation successful
You can now load the package via spacy.load('xx_ent_wiki_sm')
If keep HYs89Kcupv MISC
Thank PER
Help MISC
This claimed Ramadhan MISC
This volunteers Ernesto MISC
Syria LOC
Sanctuary Cats MISC
Caturday Dzmaj1LsHc MISC
Syrian MISC
maÃ PER
Really PER
```

Making a list named ‘Locations’ to store the extracted tokens with entity labeled LOC.

```
[ ]
locations=[]
locations.extend([[entity.text] for entity in entities if entity.label_ in ['LOC']])
df = pd.DataFrame(locations, columns=['Location'])
df
```

	Location
0	Syria
1	Terremoto
2	Turkiye
3	India
4	Turkey
...	...
376	Turkey
377	Turkey
378	Turkey
379	Syria
380	Turkey

381 rows × 1 columns

```
[ ]
pip install geopandas
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting geopandas
  Downloading geopandas-0.12.2-py3-none-any.whl (1.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 14.7 MB/s eta 0:00:00
Collecting fiona>=1.8
  Downloading Fiona-1.9.1-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.0/16.0 MB 49.7 MB/s eta 0:00:00
Collecting pyproj>=2.6.1.post1
  Downloading pyproj-3.4.1-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (7.7 MB)
```

Dropping the rows containing missing values(NaN Values)

```
[ ] df=df.dropna()
```

```
[ ] df
```

	Location
0	Syria
1	Terremoto
2	Turkiye
3	India
4	Turkey
...	...
376	Turkey
377	Turkey
378	Turkey
379	Syria
380	Turkey

381 rows × 1 columns

Location extracting & Mapmaking

Using the geopy and pandas module, we are extracting the location coordinates for the dataset we collected previously. Through this we get the longitude latitude of the location. We are feeding longitude and latitude to the folium module, and it is making a map out of it .

```
[ ]
import pandas as pd

import geopandas as gpd
import geopy
import matplotlib.pyplot as plt
from geopy.extra.rate_limiter import RateLimiter
locator = geopy.geocoders.Nominatim(user_agent="mygeocoder")
geocode = RateLimiter(locator.geocode, min_delay_seconds=0.00001)
df["address"] = df["Location"].apply(geocode)
```

```
WARNING:geopy:RateLimiter caught an error, retrying (0/2 tries). Called with (*('Â',), **{}).
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-packages/geopy/geocoders/base.py", line 344, in _call_geocoder
    page = requester(req, timeout=timeout, **kwargs)
  File "/usr/lib/python3.9/urllib/request.py", line 517, in open
    response = self._open(req, data)
  File "/usr/lib/python3.9/urllib/request.py", line 534, in _open
    result = self._call_chain(self.handle_open, protocol, protocol +
  File "/usr/lib/python3.9/urllib/request.py", line 494, in _call_chain
    result = func(*args)
  File "/usr/lib/python3.9/urllib/request.py", line 1389, in https_open
    return self.do_open(http.client.HTTPSConnection, req,
  File "/usr/lib/python3.9/urllib/request.py", line 1350, in do_open
    r = h.getresponse()
  File "/usr/lib/python3.9/http/client.py", line 1377, in getresponse
    response.begin()
  File "/usr/lib/python3.9/http/client.py", line 320, in begin
    version, status, reason = self._read_status()
  File "/usr/lib/python3.9/http/client.py", line 281, in _read_status
    line = str(self.fp.readline(_MAXLINE + 1), "iso-8859-1")
  File "/usr/lib/python3.9/socket.py", line 704, in readinto
    return self._sock.recv_into(b)
  File "/usr/lib/python3.9/ssl.py", line 1242, in recv_into
    return self.read(nbytes, buffer)
  File "/usr/lib/python3.9/ssl.py", line 1100, in read
    return self._sslobj.read(len, buffer)
socket.timeout: The read operation timed out

During handling of the above exception, another exception occurred:
```

```
df['coordinates'] = df['address'].apply(lambda loc: tuple(loc.point) if loc else None)
df[['latitude', 'longitude', 'altitude']] = pd.DataFrame(df['coordinates'].tolist(), index=df.index)
df.latitude.isnull().sum()
df = df[pd.notnull(df["latitude"])]
```

+ Code + Text

```
[ ]
import folium
from folium.plugins import FastMarkerCluster
folium_map = folium.Map(location=[59.338315,18.089960],
 zoom_start=2,tiles='CartoDB dark_matter')
FastMarkerCluster(data=list(zip(df['latitude'].values, df['longitude'].values))).add_to(folium_map)
folium.LayerControl().add_to(folium_map)
folium_map
```

Here we are just taking location data from users' location in twitter accounts and preprocessing it like tokenization , removing punctuation etc.

After that We applied nlp to find out words which contain location in it . and collecting those words in the location list and making a dataframe from it. From the location words folium module is creating a map out of it.

```
[ ] tk2=[]
    import nltk
    #nltk.download('punkt')
    from nltk.tokenize import word_tokenize
    def perform_tokenization(word):
        tokens=list(set(word_tokenize(word)))
        tk2.extend(tokens)
        return tokens
    for i in range(0,j-1):
      dataf.iloc[i]['Location']=perform_tokenization(dataf.iloc[i]['Location'])
```

```
<ipython-input-60-a236bd4a7ad4>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataf.iloc[i]['Location']=perform_tokenization(dataf.iloc[i]['Location'])
```

```
[ ] nlp= spacy.load('xx_ent_wiki_sm')

    # Define a list of tokens
    tokens=tk2

    # Join the tokens into a string
    text = " ".join(tokens)

    # Create a spaCy Doc object from the string
    doc = nlp(text)

    # Extract named entities from the Doc object
    entities = list(doc.ents)

    # Print the named entities
    for entity in entities:
        print(entity.text, entity.label_)
```

```
Paris LOC
UK World California MISC
India Lucknow PER
Bengaluru India LOC
Bengaluru India LOC
```

```
[ ] locations2=[]
    locations2.extend([[entity.text] for entity in entities if entity.label_ in ['LOC']])
    df = pd.DataFrame(locations2, columns=['Location'])
    df.head()
```

	Location
0	Paris
1	Bengaluru India
2	Bengaluru India
3	Syria
4	Afghanistan

```
[ ] locator = geopy.geocoders.Nominatim(user_agent="mygeocoder")
    geocode = RateLimiter(locator.geocode, min_delay_seconds=0.00001)
    df["address"] = df["Location"].apply(geocode)
```

```
[ ] df['coordinates'] = df['address'].apply(lambda loc: tuple(loc.point) if loc else None)
    df[['latitude', 'longitude', 'altitude']] = pd.DataFrame(df['coordinates'].tolist(), index=df.index)
    df.latitude.isnull().sum()
    df = df[pd.notnull(df["latitude"])]
```

```
[ ] import folium
    from folium.plugins import FastMarkerCluster
    folium_map = folium.Map(location=[59.338315,18.089960],
     zoom_start=2,tiles='CartoDB dark_matter')
    FastMarkerCluster(data=list(zip(df['latitude'].values, df['longitude'].values))).add_to(folium_map)
    folium.LayerControl().add_to(folium_map)
    folium_map
```

Here we are taking the intersection of the locations which came from twitter text data and users location and making the map using those locations.

```
[ ] location3=[]
    intersection = [x for x in locations if x in locations2]
    for i in intersection:
      for j in i:
        location3.append(j)
    print(location3)
```

```
['Syria', 'India', 'Turkey', 'Syria', 'Turkey', 'Syria', 'Syria', 'Turkey', 'Afghanistan', 'India', 'Turkey', 'Turkey', 'India', 'Turkey', 'Syria', 'Syria', 'Turkey', 'UK', 'India', 'Syria', 'Turkey', 'Turkey', 'Syria', 'Syria', 'Turk
```

```
df4 = pd.DataFrame(location3, columns=['Location'])
df4
```

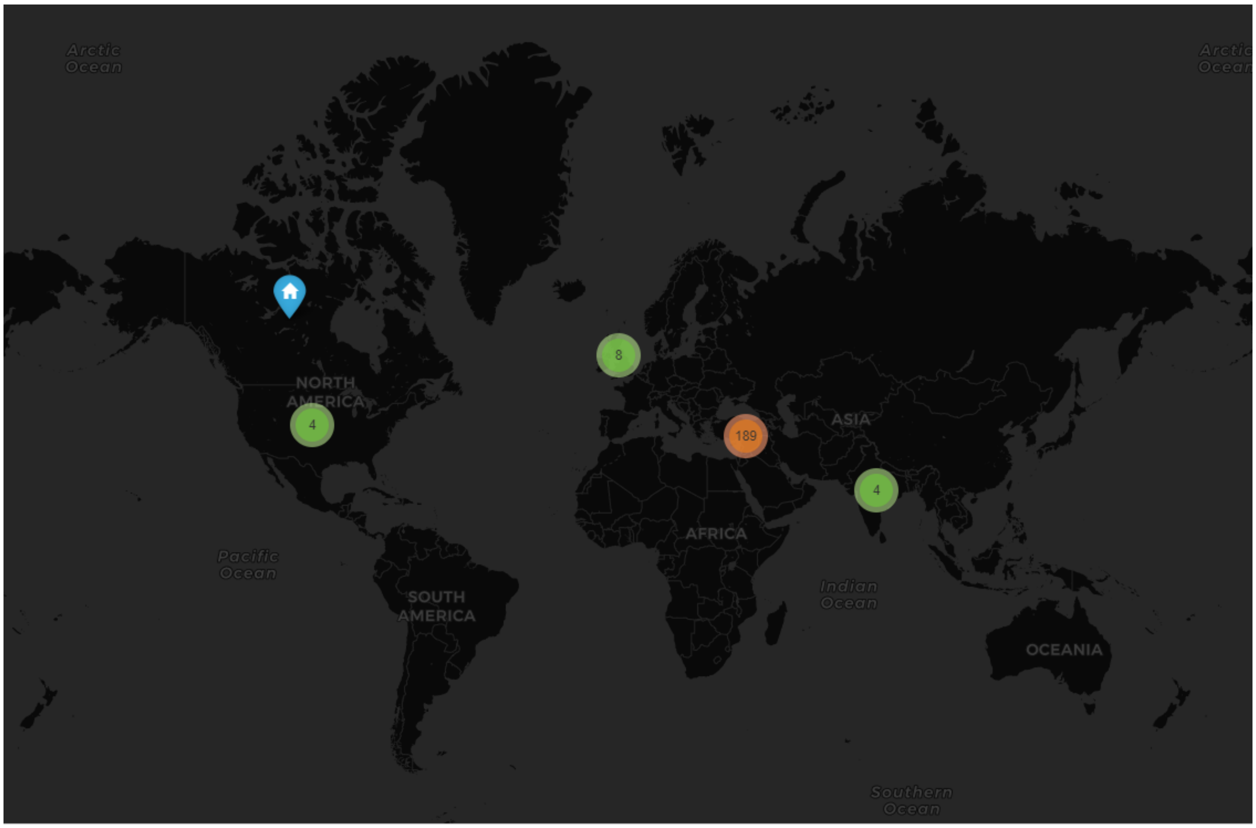
	Location
0	Syria
1	India
2	Turkey
3	Syria
4	Turkey
...	...
201	Turkey
202	Turkey
203	Turkey
204	Syria
205	Turkey

206 rows × 1 columns

```
locator = geopy.geocoders.Nominatim(user_agent="mygeocoder")
geocode = RateLimiter(locator.geocode, min_delay_seconds=0.00001)
df4["address"] = df4["Location"].apply(geocode)
```

```
df4['coordinates'] = df4['address'].apply(lambda loc: tuple(loc.point) if loc else None)
df4[['latitude', 'longitude', 'altitude']] = pd.DataFrame(df4['coordinates'].tolist(), index=df4.index)
df4.latitude.isnull().sum()
df4 = df4[pd.notnull(df4["latitude"])]
```

```
import folium
from folium.plugins import FastMarkerCluster
folium_map = folium.Map(location=[59.338315,18.089960],
 zoom_start=2,tiles='CartoDB dark_matter')
FastMarkerCluster(data=list(zip(df4['latitude'].values, df4['longitude'].values))).add_to(folium_map)
folium.LayerControl().add_to(folium_map)
folium_map
```



After setting up threshold values on locations we are finding the potential location where earthquakes might occur and visualizing those areas/location on the map.

```
[ ] dict_location={}
    for i in location3:
      if i in dict_location:
        dict_location[i]+=1
      else:
        dict_location[i]=1
```

```
[ ] dict_location
```

```
{'Syria': 68,
 'India': 4,
 'Turkey': 117,
 'Afghanistan': 2,
 'UK': 8,
 'Pakistan': 2,
 'USA': 4,
 'Canada': 1}
```

```
[ ] threshold=25
    Final_result=[]
    for i in dict_location:
      if dict_location[i]>=threshold:
        Final_result.append(i)
```

```
[ ] Final_result
```

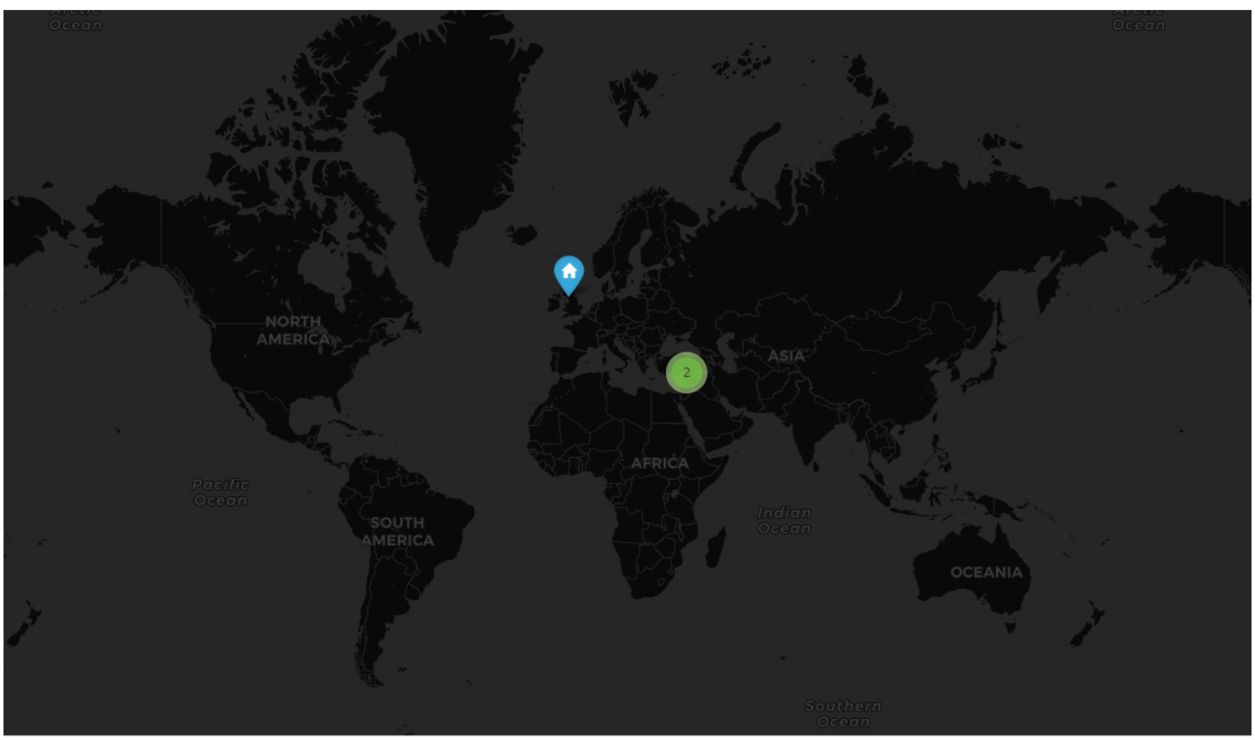
```
['Syria', 'Turkey', 'UK']
```

```
[ ] df6 = pd.DataFrame(Final_result, columns=['Location'])
    df6
```

	Location
0	Syria
1	Turkey
2	UK

```
[ ] locator = geopy.geocoders.Nominatim(user_agent="mygeocoder")
    geocode = RateLimiter(locator.geocode, min_delay_seconds=0.00001)
    df6["address"] = df6["Location"].apply(geocode)
    df6['coordinates'] = df6['address'].apply(lambda loc: tuple(loc.point) if loc else None)
    df6[['latitude', 'longitude', 'altitude']] = pd.DataFrame(df6['coordinates'].tolist(), index=df6.index)
    df6.latitude.isnull().sum()
    df6 = df6[pd.notnull(df6["latitude"])]
```

```
[ ]
    import folium
    from folium.plugins import FastMarkerCluster
    folium_map = folium.Map(location=[59.338315,18.089960],
     zoom_start=2,tiles='CartoDB dark_matter')
    FastMarkerCluster(data=list(zip(df6['latitude'].values, df6['longitude'].values))).add_to(folium_map)
    folium.LayerControl().add_to(folium_map)
    folium_map
```



Evaluation Methods

First Method

We used a sample of 500 tweets extracted from Twitter and used our model to identify the locations within the tweets. We used the model to label the locations as LOC. We calculated the Precision and Recall for the given dataset.

```
Psychological MISC
Syria Northwest PER
One MISC
ºEU LOC
Psychological MISC
Mouttaleb Krikor PER
Al MxB10jtWH5 PER
Ashnaklian Maidan PER
Aleppo LOC
Syriaâ Shelters PER
Caritas Qad MISC
Office MISC
Ronaldo PER
Syria CR7 MISC
LoiEqyCJtZ Cristiano MISC
Turkey LOC
CristianoRonaldo ORG
Help gG2B1ejb5V Tuckerbox Victims hKsS8aVup1 Earthquake MISC
Turkiye Syria PER
Join PER
Visit PER
Trapped Turkey PER
AdorablePaws ORG
MURDERED Sameh MISC
Arabic 4jr7JB0xA5 Nablus MISC
Palestinian MISC
Turkey LOC
Aqtash LOC
Turkey LOC
OUR We co medical MISC
Syria LOC
HelpNow MISC
Earthquake PER
Turkey LOC
Turn Ev3W98PJeO MISC
Ramadan Ramadan2023 MISC
```

The values we got are
True Positives = 254
True Negatives = 909
False Positives = 69
False Negatives=64

To calculate the Precision, we used

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Precision = 254/254+69 = 254/323 = 0.78
Or
78.6%

To calculate the Recall, we used

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Recall = 254/254+64 = 254/318 = 0.79
Or
79.8%

To get accuracy, we used

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + FP + TN + FN)}$$

Accuracy = 254+909/254+69+909+64 = 1163/1296 = 0.89
Or
89.7%

So far, we have achieved 78.6% precision, 79.8% recall, and 89.7% accuracy for location detection out of tweets.

Second Method

We took a sample of 10 tweets extracted from Twitter and manually identified all the locations from it.
Next, we took the same sample of tweets and extracted the locations within the tweets using our model.
We made a list of the locations that have been extracted using both methods.
Further, we compared the similarity between the manual locations and locations generated from our model. We were able to achieve **52.6%** accuracy.

```
[ ]  accuracy = (len(lst3)/len(y))*100
     print('Accuracy is ',accuracy)

     Accuracy is  52.63157894736842
```

Research Paper/References

[1] Van Quan, N., Yang, H. J., Kim, K., & Oh, A. R. (2017, April). Real-time earthquake detection using convolutional neural networks and social data. In 2017 IEEE Third International Conference on Multimedia Big Data (BigMM) (pp. 154–157). IEEE.CNN model trained from the tweet related to an earthquake is used for classification.
https://ieeexplore.ieee.org/abstract/document/7966735

[2] Sakaki, T., Okazaki, M., & Matsuo, Y. (2010, April). Earthquake shakes Twitter users: real-time event detection by social sensors. In Proceedings of the 19th international conference on the World wide web (pp. 851–860).
https://dl.acm.org/doi/abs/10.1145/1772690.1772777

[3] Real-time earthquake detection using Twitter tweets
AIP Conference Proceedings 2520, 030014 (2022);
https://doi.org/10.1063/5.0102903 Eldho Ittan Georgea) and Cerene Mariam Abrahamb)

[4] Alex Burns, Yuxian Eugene Liang "Tools and methods for capturing Twitter data during natural disasters," FirstMonday, Volume 17, Number 4 - 2 April 2012
https://firstmonday.org/article/view/3937/3193