Mandate 3
Problem Statement :

(Abstractive Summarization) Given a set of tweets pertaining to a trending topic, create an abstractive prose summary of the tweets. Do not just string the tweets together to form the summary. The summary will need to paraphrase and/or say more than what is directly said in the tweets. Propose a rubric to evaluate the accuracy of your summarization.

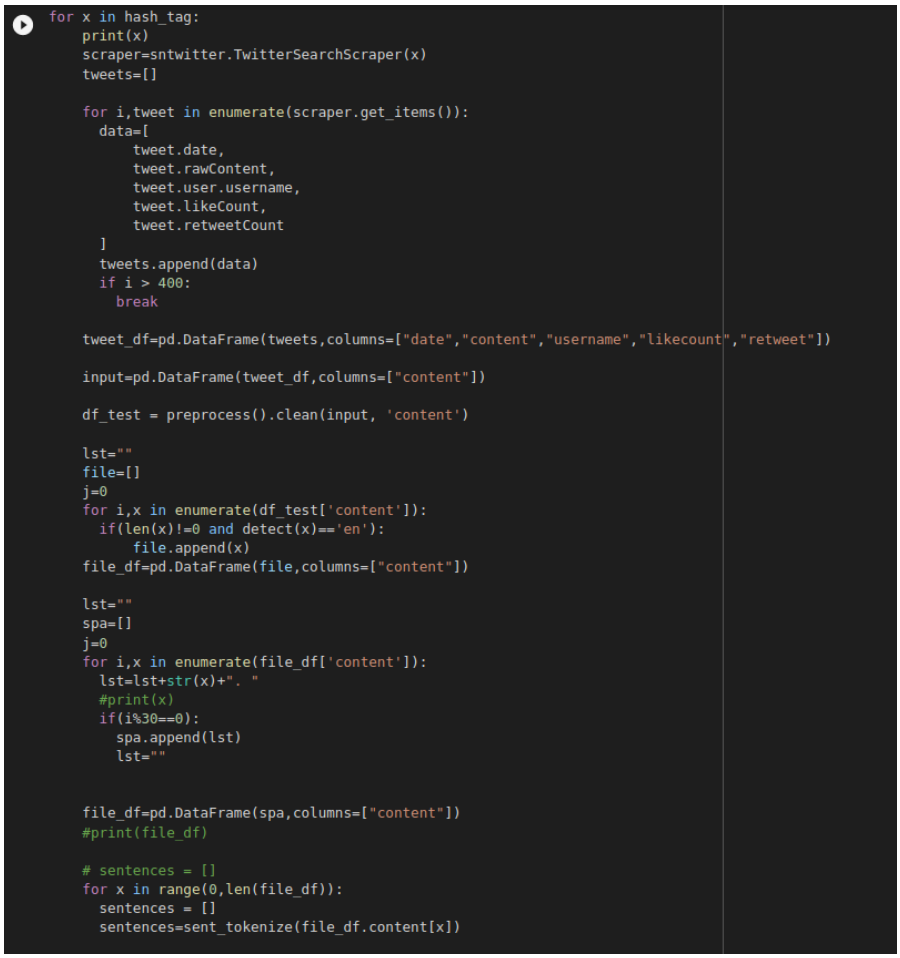
Mandate 2 Specific Task :-

- **Automation of data creation** with Text Rank just by giving terms
- Fine-tuning of Model
- Applying methods for summary creation on live twitter data

Automation of data creation with Text Rank just by giving terms:-

Steps :-

- Scrapping the live data from twitter by providing **key terms which is in loop** i.e. we will pass an array of key terms and after picking one terms it will do the following,
- first clean the data and applying pre-processing steps
- after cleaning there is high chances that we will pick lots of waste tweets so we need to remove them, so **we will use text rank to get only top tweets**
- after getting top tweets we will combine them
- after doing this for one key terms we will repeat the same for other terms
- at last we **will have only top tweets** from the terms provided
- We will feed this to openAI api to get the summaries for our as ground truth must be of high quality.



```
for x in hash_tag:
    print(x)
    scraper=sntwitter.TwitterSearchScraper(x)
    tweets=[]

    for i,tweet in enumerate(scraper.get_items()):
      data=[
          tweet.date,
          tweet.rawContent,
          tweet.user.username,
          tweet.likeCount,
          tweet.retweetCount
      ]
      tweets.append(data)
      if i > 400:
        break

    tweet_df=pd.DataFrame(tweets,columns=["date","content","username","likecount","retweet"])

    input=pd.DataFrame(tweet_df,columns=["content"])

    df_test = preprocess().clean(input, 'content')

    lst=""
    file=[]
    j=0
    for i,x in enumerate(df_test['content']):
      if(len(x)!=0 and detect(x)=='en'):
          file.append(x)
    file_df=pd.DataFrame(file,columns=["content"])

    lst=""
    spa=[]
    j=0
    for i,x in enumerate(file_df['content']):
      lst=lst+str(x)+". "
      #print(x)
      if(i%30==0):
        spa.append(lst)
        lst=""


    file_df=pd.DataFrame(spa,columns=["content"])
    #print(file_df)

    # sentences = []
    for x in range(0,len(file_df)):
      sentences = []
      sentences=sent_tokenize(file_df.content[x])
```

Fine-tuning of Model:-

For Fine tuning we will use two models (as they are kind of best and fast)

1. Pegasus
2. Simple T5

Pegasus :- PEGASUS proposes a transformer-based model for abstractive summarization. It uses a special self-supervised pre-training objective called gap-sentences generation (GSG) that's designed to perform well on summarization-related downstream tasks. In PEGASUS pre-training, several whole sentences are removed from documents and the model is tasked with recovering them. An example input for pre-training is a document with missing sentences, while the output consists of the missing sentences concatenated together. This is an incredibly difficult task that may seem impossible, even for people, and we don't expect the model to solve it perfectly.
For our training we pick "transformersbook/pegasus-samsum" which is a variant and train on samsun dataset.
Before training we Freeze token embeddings and positional embeddings for pegasus.

Simple T5:- The T5 model was presented in Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. T5 is an encoder-decoder model pre-trained on a multi-task mixture of unsupervised and supervised tasks and for which each task is converted into a text-to-text format. T5 works well on a variety of tasks out-of-the-box by prepending a different prefix to the input corresponding to each task.

Parameter Setting:-

- MAX length set to 128 as we don't want very long summaries(also trained other model with less lenght).
- Learning Rate is set 2e-4
- Weight decay is 0.01
- Total epochs set to 1
- Used Adafactor optimizer as it saves lot of RAM else we will have memory out of space error.

For Scoring we used ROUGE(proxy metric) and BERT_SCORE, following are the score:-
rougeLsum : 44.1302 (for large summarization)

```
out.metrics
```

```
{'test_loss': 1.4925072193145752,
 'test_rouge1': 48.4559,
 'test_rouge2': 26.3418,
 'test_rougeL': 39.7548,
 'test_rougeLsum': 44.1302,
 'test_gen_len': 64.61,
 'test_runtime': 155.4315,
 'test_samples_per_second': 0.643,
 'test_steps_per_second': 0.322}
```

Rouge is bad evaluation method for scoring these models. Will explain in next mandate.

bert_score:-

```
System level F1 score: 0.894
System level precision score: 0.896
System level recall score: 0.893
```

Applying methods for summary creation on live twitter data :-
Once our model is trained and getting fine results we will upload the model to “hugging-face” and will use api of our model for abstract summarization.

After this for creating summaries again we will scrape the data, pre-process it. Use **POS tagging** to get top sentences(they ain’t that good), so we will use **GloVE & word2Vec for applying text rank** since we need to get Top tweets/data for feeding to our model and at last we will use model api for summarization.