Mandate 2
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 :-
- Logic for Extraction and preparaing dataset for model.
- PLM and fine-tuning task
- How to evaluate the result

Logic for Extraction and preparaing dataset : Dataset for training the model is collected from various sources since it denotes the ground truth which must be solid. Available sources :
- Tweetsumm, A dataset focused on summarization of dialogs, which represents the rich domain of Twitter customer care conversations, which can be found in **https://github.com/guyfe/Tweetsumm.git**.
- External content Summarizers.
- Extractive Summaries using scrapping data from live twitter feed.

For the above mentioned 2nd and 3rd point we need data from live twitter feed which is generally vague data, so we need to pre-process the data to feed the model.

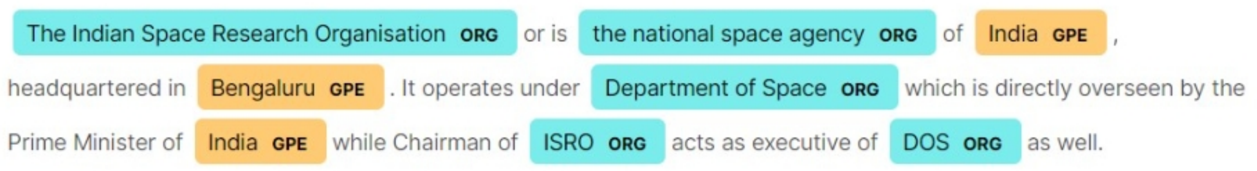
Scraping and Pre-preprocess:
First step is using Python library "*snscrape*" to scrape the twitter data which has specific module of twitter. **After that we put the data into list and send for pre-processing.** Another thing to note that when we are scraping data using hashtags we need to sure that **provided hashtags contains information** not dummy one.

Steps includes in pre-processing :
- Remove hashtag's, url's, html tags.
- Remove all non-alpha characters such that it should only contains english letters.
- Stop words removals and lemmetization.
- Pattern removal like if something like this occur *"zzzzzzzzzz"* .
- Remove words with number and decontracted the apostrophe(').
- Use as language detector such that only "EN" is used.

Again we can add more functions to pre-process but they also have a downside like spell-checker if there is name like "*Anna Hazare*" it will replace "*Hazare*" with "*Hazard*", so we need to be cautious.

Selecting Useful Tweets:
For creating the summaries one thing is very clear that **not all tweets are important**, let's take an example,
Tweet1: "*Nigerian stock market has crashed again for the third time in a row 2015, 2019 and 2023 !!!*"
Tweets2: "*Birds are looking into stock markets*"
Above tweets shows that Tweet1 contains some info which is useful in creating summaries while tweets like **Tweet2 needs to be discarded**, so we need some content words or class of terms like noun, numericals, money etc and the words which fell into these categories called as content words. We can define classes which contains important terms.

SpaCy(a natural language processing library, which automatically analyzes and extracts information from text) is a very useful tool to identify content words, when SpaCy tokenizes text, it adds a lot of additional information to the tokens, such as whether it is an entity (and if it is, what type of entity it is), what its part of speech is (i.e. is it a noun? a verb? a numeral?), or even the token's sentiment.


The Indian Space Research Organisation ORG or is the national space agency ORG of India GPE , headquartered in Bengaluru GPE . It operates under Department of Space ORG which is directly overseen by the Prime Minister of India GPE while Chairman of ISRO ORG acts as executive of DOS ORG as well.

I used SpaCy to tokenize the tweets. This meant breaking the tweets down into their components (primarily words, but also punctuation and numbers), and turning these components into(tokens). The power of SpaCy is that these tokens are full of additional information; for example, I can find out the part of speech of all of these tokens:

```
[u'PUNCT', u'ADP', u'NUM', u'VERB', u'ADP', u'NOUN', u'PUNCT',
u'PROPN', u'PROPN', u'PROPN', u'PUNCT', u'PROPN', u'PUNCT']
```

By this way we can categorize tweets based on content words and sort them, but there is catch in this which is content words fails to capture any information about the words themselves, all words weight the same. So we need to attach weights to the words to categorise how important that words and It is possible to introduce such a weighting, using term frequency — inverse document frequency (tf-idf) scores.
Main logic behind this is "*Translated into plain English, importance of a term is high when it occurs a lot in a given document and rarely in others. In short, commonality within a document measured by TF is balanced by rarity between documents measured by IDF. The resulting TF-IDF score reflects the importance of a term for a document in the corpus.*"

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF
Term x within document y

$tf_{x,y}$ = frequency of x in y
df_x = number of documents containing x
N = total number of documents

Basically we want the term like "1 Billion dollar" weigh higher than term like "can".
Now, we need to pick sentences which is full of content words.

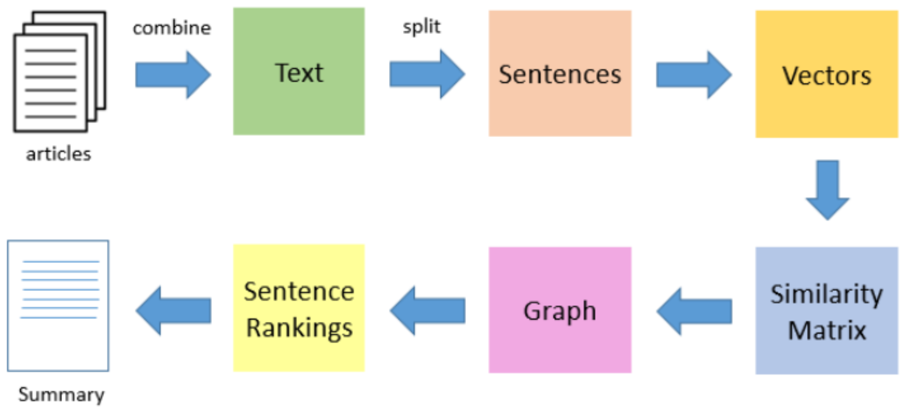
```
WORD:taliban -- tf-idf SCORE:5.269697449699962
WORD:this -- tf-idf SCORE:6.955875960943814
WORD:timesofindia -- tf-idf SCORE:4.864232341591798
WORD:today -- tf-idf SCORE:4.576550269140016
WORD:tuesday -- tf-idf SCORE:5.269697449699962
```

Now to pick the important words which got high tf-idf score we need to define some mathematical equation and contraints :-
Equation: Maximize the total score of the content words in my summary.

Constraint:

- The summary must be shorter than 100 words.
- If I pick a content word to be in my summary, then I must pick some tweet which contains that content word to be in my summary.
- If I pick some tweet to be in my summary, then all the content words in that tweet must be included.
- When the equation is solved we will get our collection of important tweets which contains useful information.

Applying TextRank for extractive summary:
Yet another way of generating extractive summary since tf-idf vectorization doesn't always work good as it **ignore the order of the words.**
TEXT RANK :- Textrank is a graph-based ranking algorithm like Google's PageRank algorithm which has been successfully implemented in citation analysis. We use text rank often for keyword extraction, automated text summarization and phrase ranking. Basically, in the text rank algorithm, we measure the relationship between two or more words.



Following are the steps required for the text rank :

- The first step would be to concatenate all the text contained in the articles.
- Then split the text into individual sentences.
- In the next step, we will find vector representation (word embeddings) for each and every sentence.
- Similarities between sentence vectors are then calculated and stored in a matrix.
- The similarity matrix is then converted into a graph, with sentences as vertices and similarity scores as edges, for sentence rank calculation.
- Finally, a certain number of **top-ranked sentences** form the final summary.

For the word embedding we will use both GloVe and Word2Vec. Glove generally performs better than word2vec since, Word2Vec takes texts as training data for a neural network. The resulting embedding captures whether words appear in similar contexts. **GloVe focuses on words co-occurrences over the whole corpus**. Its embeddings relate to the probabilities that two words appear together.

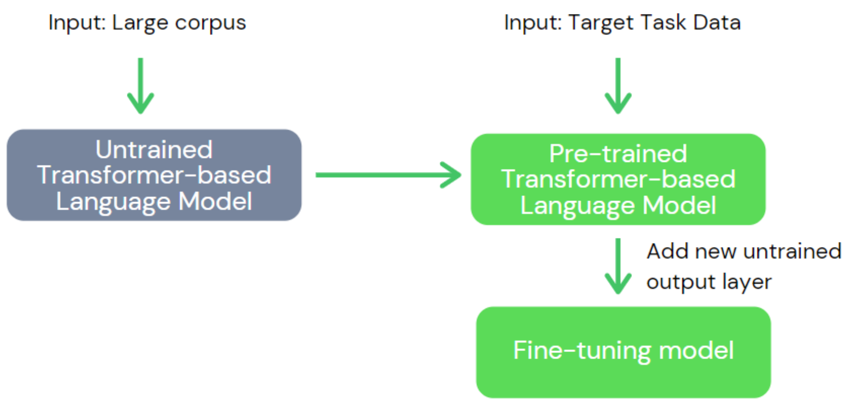
After completing all the previous steps we have our Extractive summary which can be given to model as input.

PLM and Fine-tuning: The main logic behind pre-trained language models is to create a black box which understands the language and can do tasks related to that language. The idea is to create the **machine equivalent of a 'well-read' human being** which is also called as '**fuzzy logic**'. They are generally trained on very large corpus like Wikipedia and that's how they learn the usage of words and semantics in language.
For the given task we will select two PLM model

1. Pegasus :
    - Pegasus' pretraining task is intentionally similar to summarization: **important sentences are removed/masked from an input document and are generated together as one output sequence** from the remaining sentences, similar to an extractive summary.
    - Pegasus achieves SOTA summarization performance on all 12 downstream tasks, as measured by ROUGE and human eval.
2. T5 :
    - T5 is an encoder-decoder model pre-trained on a multi-task mixture of unsupervised and supervised tasks and for which each task is c**onverted 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, e.g., for translation: translate English to German: …, for summarization: summarize.

These two models are a State-of-the-Art Model for Abstractive Text Summarization that's why I prefer them.

Need of Fine tuning : The main issue with PLM is that the word representation/weight they used are **very generalized** and **do not represent task-specific information**. Like **word "cool" have different meaning on different context**. This is the main reason fine-tuning is used so that we can modify the weights in pre-trained langauge to suits our context. This is what makes the fine-tuning approach so attractive. If we can find a trained model that already does one task well, and that task is similar to ours in at least some remote way, then we can take advantage of everything the model has already learned and apply it to our specific task.

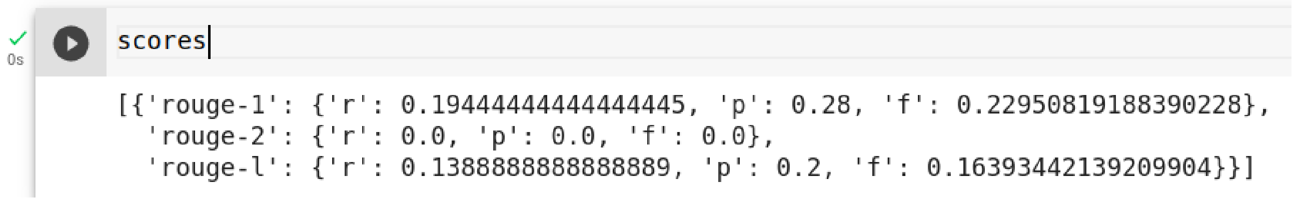

Input: Large corpus
Input: Target Task Data
Untrained Transformer-based Language Model
Pre-trained Transformer-based Language Model
Add new untrained output layer
Fine-tuning model

How to Fine-Tune : Since we already have a PLM that does create summaries from wiki corpus and we need to tune it such that it can do that for twitter data. Fine tuning can be done by many ways like we can also **remove some layers and add our's** but in our case we don't need to remove anything we can simply attach one more layer after the last layer of our PLM model. While tuning our model we need to keep one thing in mind that we **don't update weights of previous layers** because we just want to **update weights in last layer** which been added by us. After we do this, all that's left is just to train the model on our new data.
Few Imp params:-

- **vocab_size** (int, optional, defaults to 50265) — Vocabulary size of the PEGASUS model. Defines the number of different tokens that can be represented by the inputs_ids passed when calling PegasusModel or TFPegasusModel.
- **encoder_layers** (int, optional, defaults to 12) — Number of encoder layers.
- **decoder_layers** (int, optional, defaults to 12) — Number of decoder layers.
- **activation_function** (str or function, optional, defaults to "gelu") — The non-linear activation function (function or string) in the encoder and pooler. If string, "gelu", "relu", "silu" and "gelu_new" are supported.
- **dropout** (float, optional, defaults to 0.1) — The dropout probability for all fully connected layers in the embeddings, encoder, and pooler.
- **use_cache** (bool, optional, defaults to True) — Whether or not the model should return the last key/values attentions (not used by all models)

How to evaluate the result : There are several ways to evaluate the performance of the Pegasus model, which is a state-of-the-art pre-trained sequence-to-sequence model for natural language processing tasks such as summarization, paraphrasing, and text generation. There are several automatic evaluation metrics that can be used to evaluate the performance of the Pegasus model, such as ROUGE (Recall-Oriented Understudy for Gisting Evaluation) and BLEU (Bilingual Evaluation Understudy).

In this project we will use ROUGE.
So we are getting below rouge score in PLM :



```
scores
```

```
[{'rouge-1': {'r': 0.19444444444444445, 'p': 0.28, 'f': 0.22950819188390228},
  'rouge-2': {'r': 0.0, 'p': 0.0, 'f': 0.0},
  'rouge-l': {'r': 0.1388888888888889, 'p': 0.2, 'f': 0.16393442139209904}}]
```

which is not good enough so we need to fine tune our model and as mention above we need to add a layer above the PLM so that our ROUGE score would improve.

Again there are few things we need to implement like replacing emoticons with words, implementing encoder/decoder, fine tune by freezing /not freezing weights etc. By going foward will try to implement more functions and make model robust.