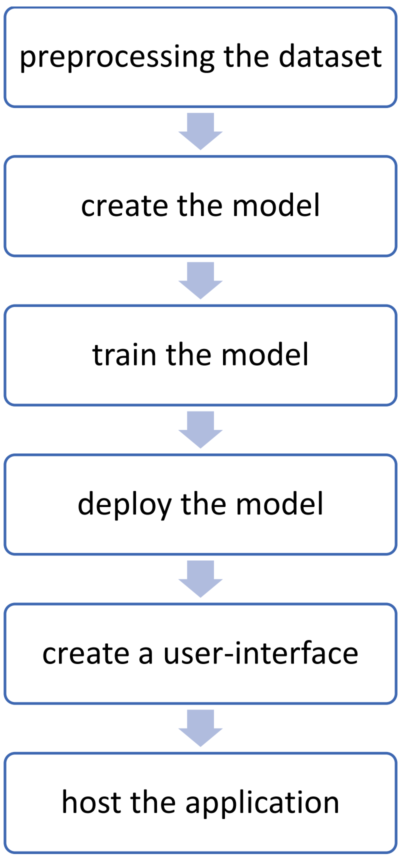
In this report We will go through the solution for the problem of predicting which tweets are related to real disasters and which aren't:

## The process

To solve this problem, we will go through the process as the following:



## the specifications of the problem

To solve this problem:

1- We will be using the **dataset** provided for the Kaggle competition https://www.kaggle.com/competitions/nlp-getting-started/data
2- We will also be using the Kaggle notebook to preprocessing and training code.
3- We will be using Visual Studio Code to create the python files for the deployment and the user interface.
4- We will be hosting the application on the huggingface platform.

## The requirements

Let's go through the libraries that we will be using:

1- We will use a transformer model than fine tune it to get the best results, the model that we will be using is the one published on the huggingface platform: https://huggingface.co/sacculifer/dimbat_disaster_distilbert
so we will use the transformers library.

2- To build the model we will use the Tensorflow library.
3- will also be using streamlit library to create the user-interface.
4- Matplotlib will be used to create graphs.

5- Pandas will be used to deal with the CSV datafiles.

## The steps

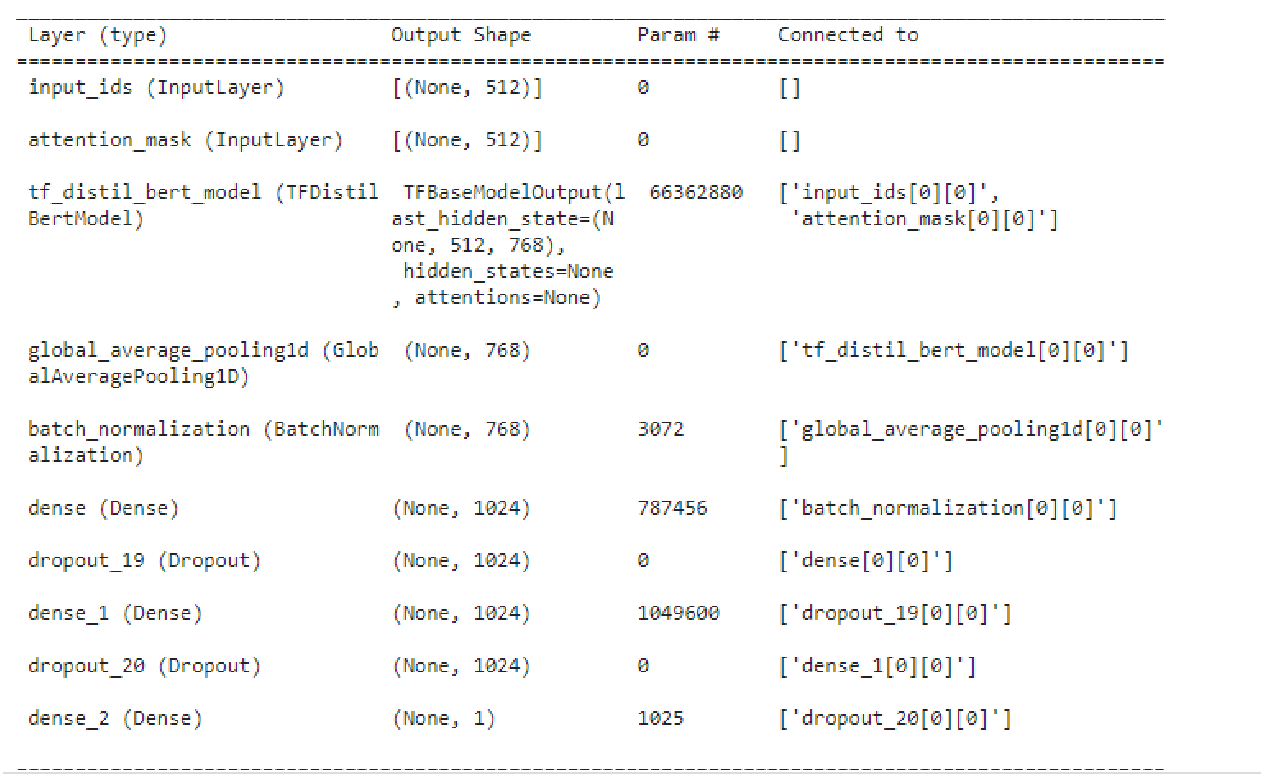
### preprocessing the dataset

1- In the used dataset there are null values so we will be dropping the columns that has a lot of null values since they don't provide useful information.
2- We will also remove some useless information from the texts of the dataset like URLs and user tags also "#" symbol and other unreadable symbols like.
 We will also tokenize the new dataset after the previous modifications, and to do so I will use the tokenizer provided by:
https://huggingface.co/sacculifer/dimbat_disaster_distilbert
3- Then we will be creating a new dataset using the tokens from the previous step and transform the dataset to a dataset of tensors.

Now we have our dataset ready to be fed to a model.

### Create the model

1- As we said before we will use the pretrained weights from the dimbat_disaster_distilbert model, so we will start by getting them.
2- Then we will create a tensorflow model.



```
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_ids (InputLayer)         [(None, 512)]        0           []

 attention_mask (InputLayer)    [(None, 512)]        0           []

 tf_distil_bert_model (TFDistil  TFBaseModelOutput(l  66362880   ['input_ids[0][0]',
 BertModel)                     ast_hidden_state=(N               'attention_mask[0][0]']
                                one, 512, 768),
                                 hidden_states=None
                                , attentions=None)

 global_average_pooling1d (Glob  (None, 768)         0           ['tf_distil_bert_model[0][0]']
 alAveragePooling1D)

 batch_normalization (BatchNorm  (None, 768)         3072        ['global_average_pooling1d[0][0]'
 alization)                                                      ]

 dense (Dense)                  (None, 1024)         787456      ['batch_normalization[0][0]']

 dropout_19 (Dropout)           (None, 1024)         0           ['dense[0][0]']

 dense_1 (Dense)                (None, 1024)         1049600     ['dropout_19[0][0]']

 dropout_20 (Dropout)           (None, 1024)         0           ['dense_1[0][0]']

 dense_2 (Dense)                (None, 1)            1025        ['dropout_20[0][0]']

--------------------------------------------------------------------------------------------------
```

### train the model

1- We will use "Adam" optimizer and "cross entropy" loss function also "accuracy" metric to judge the performance.

2- We will train for 8 epochs and we reached an accuracy of 0.9779 and a loss of: 0.07, and these results are really amazing.

3- Let’s save the new weights that we got, and move to the testing step.

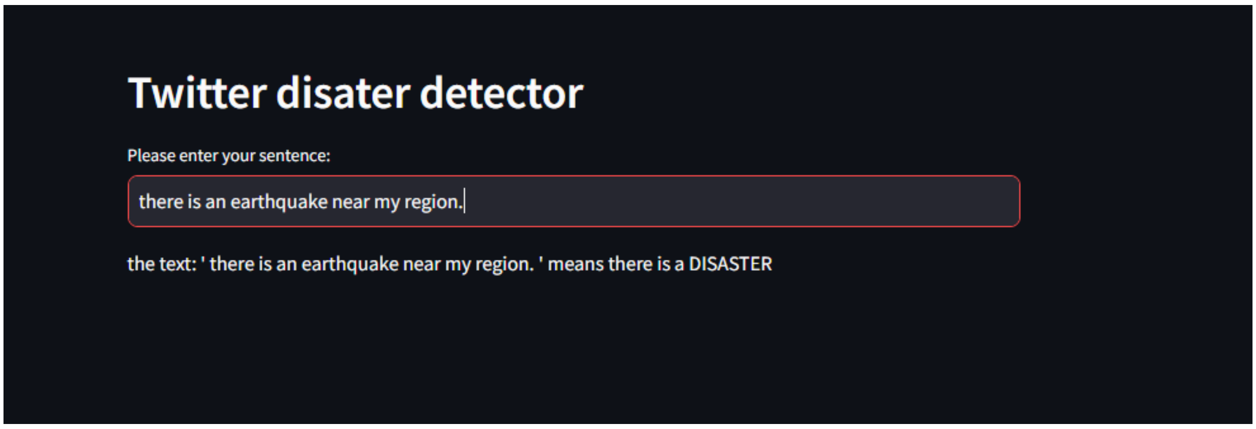
## deploy the model

we will start by creating the twitter_model class to wrap the code for the deployment, this class will:

- __init()__: create an instance of the model and load the training weights.
- Predict(): It will also prepare the input text by tokenizing it and converting it to tensors to make it ready to be fed to the model. Then it will feed the new input to the test_model and output the prediction.

## Create the user-interface

We mentioned that we will be using the streamlit library to create a simple user interface.


Twitter disater detector

Please enter your sentence:

there is an earthquake near my region.

the text: ' there is an earthquake near my region. ' means there is a DISASTER

## Host the application

The most continent way for me to host this application is to use the huggingface platform.