Elena Barrio 252060
Laura Juan 254094
Rocio Gilabert 251217

# REPORT PART 1

## Text Processing and Data Exploratory Analysis

**Github LINK and TAG: IRWA-2024-part-1**

## 1. Functions

In this part of the report we will briefly explain which are the functions we have defined to be able to filter all the data from the .json and .csv files.

- `def build_terms(line)`
  This function takes a line (later will be applied to the content of each tweet and processes it converting it into lowercase; removing URLs, punctuation marks, emojis, symbols, etc.; and tokenizing the text into individual words. This function also deletes some stop words ('the', 'is'...) and performs stemming, reducing every word to its root form.

- `def extract_hashtags(content)`
  This function is used to find all hashtags in a text, it finds every word that starts with '#' symbol and adds it to a list.

- `def getCleanTweets(tweet)`
  This function creates an empty dictionary to store the desired information from the json file, we will store the ID of the tweet, the content of the tweet, the processed content of the tweet, the date of the tweet, the number of likes of the tweet, the number of retweets, the URL of the tweet, the username and also a list with the hashtags.

## 2. Load and filter the data

In this part of the code we have loaded the data from the json file and the csv file into a list of dictionaries. Then we have created a function called `def process_tweets` that will process all the tweets of the json file using the function.
The `def process_tweets function` tries to match the tweets IDs between the two files, json and csv files. It cleans the matched tweets and stores them into a dictionary. It first converts the csv list of dictionaries into a single dictionary where the tweets IDs are the keys and the values are the document ids. Then for each tweet id in the json file it finds the corresponding tweet id in the csv dictionary, and when they match the tweet is processed using the `def getCleanTweets` function and then adding it into the dictionary with the document id as the new key.

Elena Barrio 252060
Laura Juan 254094
Rocio Gilabert 251217

## 3. Exploratory data analysis

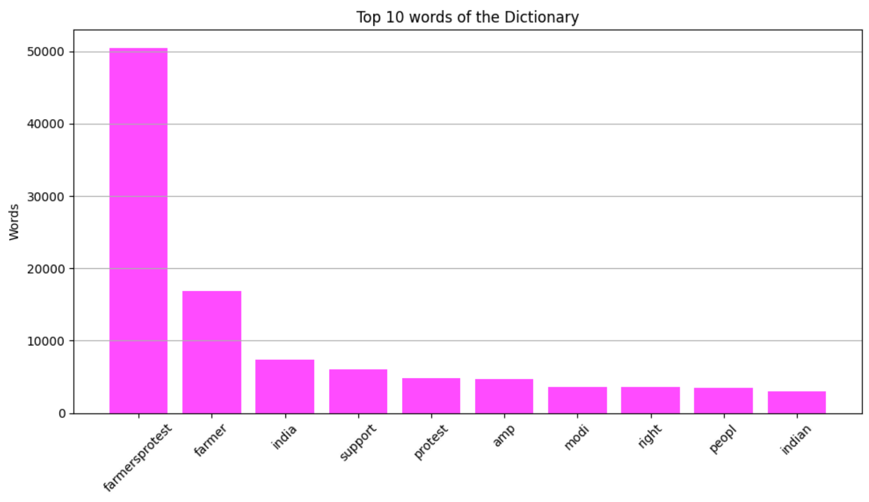
Fo the exploratory data analysis section, we have decided to study the following aspects:

### a. Top Words & Top Hashtags

Here we have created two different functions `def word_count(dic)` and `def hashtag_count(dic)` to know which are the top 5 most repeated words and hashtags in the tweets. To do so we have counted the different words in the processed tweets and the number of different hashtags in the hashtags list of the tweet and then we have ordered them in descending order.
Also we have created a function to get the top 10 tweets regarding the parameter you decided to be able to have a plot of the top 10 words and the top 10 hashtags.

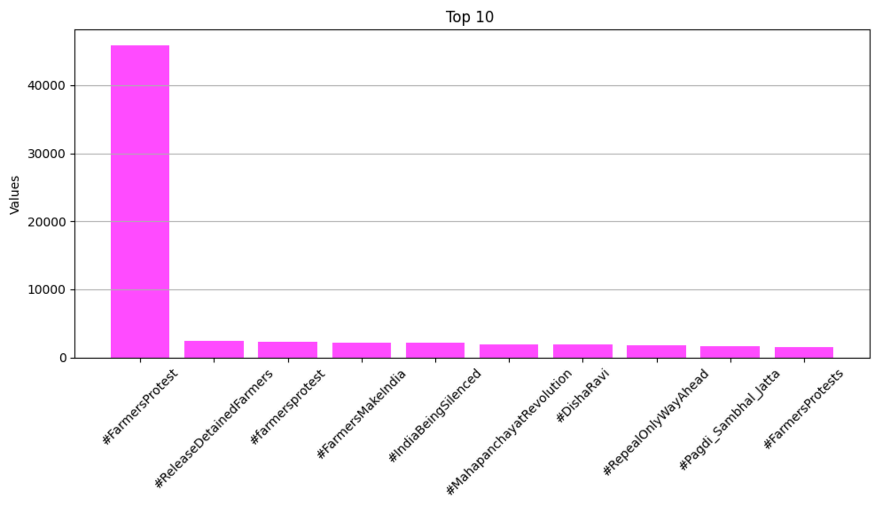
Words:

```
Top 1: farmersprotest --> 50454
Top 2: farmer --> 16877
Top 3: india --> 7332
Top 4: support --> 6028
Top 5: protest --> 4802
```



Hashtags:

```
Top 1: #FarmersProtest --> 45856
Top 2: #ReleaseDetainedFarmers --> 2430
Top 3: #farmersprotest --> 2305
Top 4: #FarmersMakeIndia --> 2129
Top 5: #IndiaBeingSilenced --> 2128
```



### b. Average Sentence Length

Then in this part we have defined a function `def avg_length(dic)` that calculates the average length of processed tweets stored in the dictionary. The function iterates over each tweet in the dictionary, accessing the processed tweet key to measure the length of each tweet and accumulate these values. After summing the lengths, the function returns the average by dividing the total sum by the number of tweets in the dictionary.

Importantly, we have removed URL's (i.e., 'https') when displaying the processed tweets, and the average is calculated based on the tokens, excluding spaces and emojis.

```
The average length of the tweets is 14.699498234528898
```

### c. Most retweeted tweets

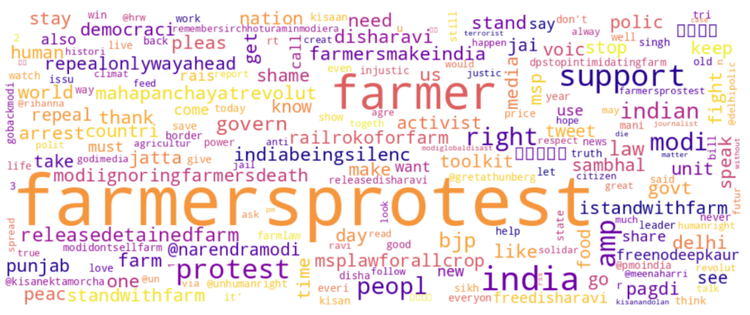
Here we have defined a function `def most_retweeted(dic)` that identifies and ranks tweets based on the number of retweets. The function creates a dictionary 'retweeted' where we first extract the content of the tweet related to each doc_id and use it as keys. These keys are then associated with their respective retweet counts. This dictionary is then sorted in descending order based on the number of retweets. The result is returned as a dictionary, sorted by popularity and only the top 5 are printed, displaying both the number of retweets and the tweet content.

```
Nº of retweets:  6164
 Content:  There's a #FarmersProtest happening in Germany.
```

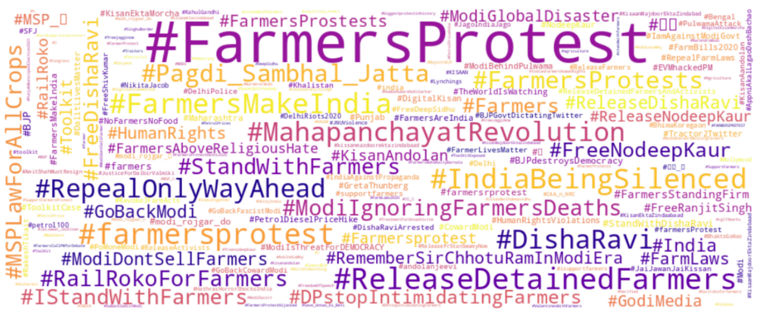
### d. Word Cloud

Then in this part we have made the code that generates two distinct word clouds using the library 'WordCloud', which visually represent the most frequent words and hashtags in the dataset. The word clouds are created by initializing the WordCloud object with specified parameters. The first one is generated from the dictionary of word frequencies 'frequency_words', and the second using hashtag frequencies 'frequency_hashtags'. These visualizations help identify the most distinguished words and hashtags.

**Frequency Words**



**Frequency Hashtags**



**e. Vocabulary Size**

To perform this analysis what we have done is use the dictionary returned by the function `def word_count(dic)` as it returns a dictionary with each unique word that is used in the tweets. Having this dictionary of frequencies, what we have done next is return the length of this dictionary which gives us the number of unique words that are being used. The results obtained are that the number of distinct words in the dataset is 38922 words having that we are analyzing pre-processed tweets. We can use this information to know the average of unique words per tweet dividing the total number of unique words by the total number of tweets and we get 0.804. We can compare this information to the average number of words per tweet computed before where we obtained 14.69. Therefore, we can observe and deduce that the number of repeated words among tweets is very high and this seems reasonable since they are all related to the Farmers Protests 2021 and they use all the same words.

**f. Top 5 most active users**

Here we have created a function called `def most_active_users(dic)` to know how many tweets has each user posted. To do so we have created a dictionary with the form `Username: number of tweets`. First, we iterate through all the tweets in the dataset and we take the username from the Username field of the dictionary. We then check if the username has already appeared and compute the frequency. Then, by ordering the dictionary we are able to take the top 5 most active users and the amount of tweets they have posted and we get the following result:

```
Top 1: @jot__b --> posted 679 tweets
Top 2: @shells_n_petals --> posted 489 tweets
Top 3: @KaurDosanjh1979 --> posted 423 tweets
Top 4: @DigitalKisanBot --> posted 368 tweets
Top 5: @ish_kayy --> posted 366 tweets
```

**g. Most used emojis**

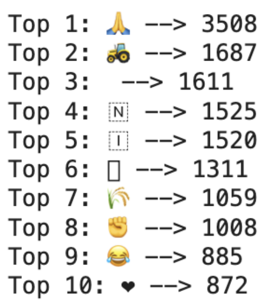
We thought that analyzing the most used emojis could also give us relevant information. To do this, we created a function called `def emoji_count(dic)` to know the frequency of each emoji. First, we compile the same emoji patterns that we used in the `build_terms` function using re.compiler. Then, for each tweet in the dataset, we have taken the tweet content and found all the emojis that appear in the tweet (using the compiler and findall function) and saved them in a list. We have observed that some emojis appeared together in a tweet without a space between them. For example:

Elena Barrio 252060
Laura Juan 254094
Rocio Gilabert 251217

Farmers constantly distroying crops throughout India. Really, it's hearts breaking...we care about our crops like our children. And govt. agriculture minister is laughing on us🚜🌾WE WILL WIN💪

So, we had to take care of this aspect. To do so, we checked if there were more than one emoji by observing the length of each item of the list of emojis. Then, we proceed the same way as in `word_count` and `hashtag_count` functions, checking the frequency of each emoji found.

When we had the dictionary in the form `Emoji:Counter` we have ordered the frequencies by descending order and taken the top 10 most used emojis, obtaining the following results



```
Top 1: 🙏 --> 3508
Top 2: 🚜 --> 1687
Top 3:  --> 1611
Top 4: 🇳 --> 1525
Top 5: 🇮 --> 1520
Top 6: ▯ --> 1311
Top 7: 🌾 --> 1059
Top 8: ✊ --> 1008
Top 9: 😂 --> 885
Top 10: ❤ --> 872
```

We can observe that the top 2 most used emojis can be related to the main protest. The first one is a ‘please’ emoji and may act as a protest symbol and the second one is a tractor emoji which seems reasonable as it is a Farmer Protest.