

CS 242 Project

Part A & Part B

Project Members:

Net ID	Name
sgupt068	Sheenam Gupta
hsang006	Hanisha Sree Sangam
hsrin004	Harish Kanna Srinivasan
pvelm001	Ponmanikandan Velmurugan

A. COLLABORATION DETAILS:

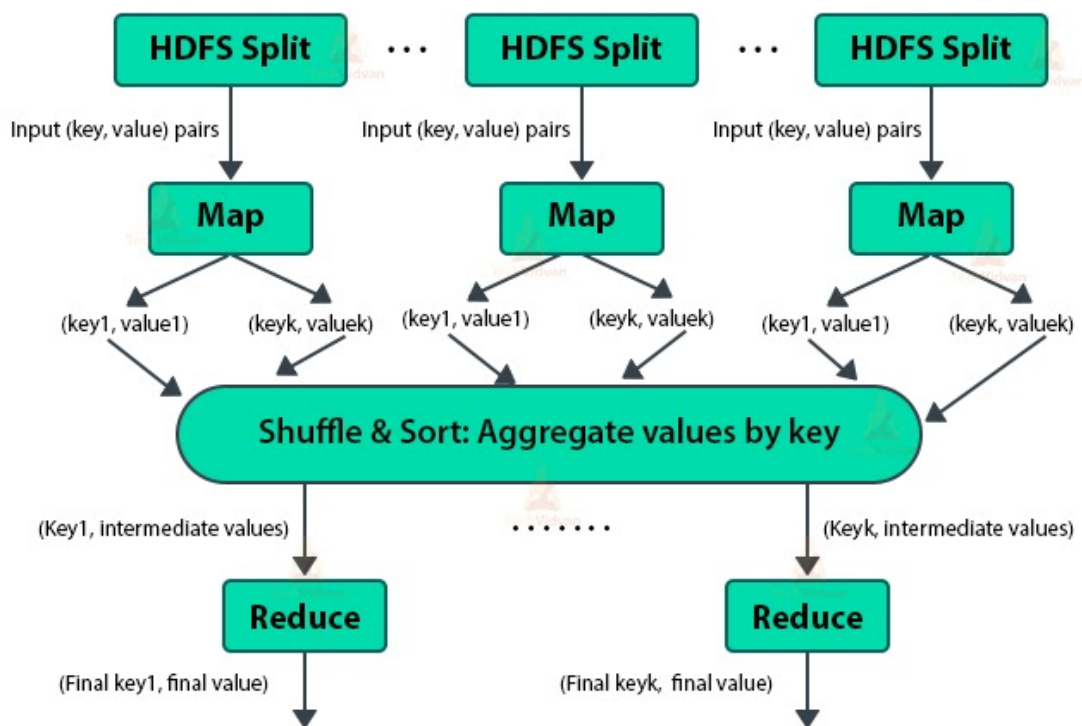
Harish and Ponmanikandan worked on Hadoop indexing and ranking. Sheenam and Hanisha focused on the creation of the web interface for the Search engine. Apart from this we all collaborated and helped each other in resolving issues in our project.

B. OVERVIEW OF SYSTEM

In this part of the project we integrated Lucene (indexed) and Hadoop (Indexed) data using Spring.io and AngularJS for presenting in the web interface. Our web interface is a simple search engine that allows the user to choose between lucene and hadoop for desired output. User has to enter the query in the search box and click search, the interface will show the top 10 searches.

I. ARCHITECTURE

Hadoop MapReduce



II. DETAILS OF HOW HADOOP IS USED (KEY, VALUES DEFINITIONS AND DATA)

We are using the **Hadoop Streaming jar** to execute MapReduce operations on our crawled data. Hadoop streaming is a utility that comes with the Hadoop distribution. The utility allows you to create and run Map/Reduce jobs with any executable or script as the mapper and/or the reducer. Using this utility we are able to write our Map and Reduce functions in Python.

```
$HADOOP_HOME/bin/hadoop jar $HADOOP_HOME/hadoop-streaming.jar \
-input myInputDirs \
-output myOutputDir \
-mapper /bin/cat \
-reducer /bin/wc
```

Both the mapper and the reducer are executables that read the input from **stdin (line by line)** and emit the output to **stdout**. The utility will create a Map/Reduce job, submit the job to an appropriate cluster, and monitor the progress of the job until it completes.

When an executable is specified for mappers, each mapper task will launch the executable as a separate process when the mapper is initialized. As the mapper task runs, it converts its inputs into lines and feed the lines to the stdin of the process. In the meantime, the mapper collects the line oriented outputs from the stdout of the process and converts each line into a key/value pair, which is collected as the output of the mapper. By default, the *prefix of a line up to the first tab(**'t'**) character* is the **key** and the rest of the line (excluding the tab character) will be the **value**. If there is no tab character in the line, then entire line is considered as key and the value is null. However, this can be customized.

In our program where we have the (word,tweet_id) as the key we concatenate the two strings by separating them with a (**','**) and then concatenate them into one string that is separated by a tab character.

For example:

If tweet_id = 123455

word = apple

Then the output of the Map for the getting the word count will be

123455,apple\t1

When an executable is specified for reducers, each reducer task will launch the executable as a separate process then the reducer is initialized. As the reducer task runs, it converts its input key/values pairs into lines and feeds the lines to the stdin of the process. In the meantime, the reducer collects the line oriented outputs from the stdout of the process, converts each line into a key/value pair, which is collected as the output of the reducer. By default, the prefix of a line up to the first tab character is the key and the rest of the line (excluding the tab character) is the value. However, this can be customized, as discussed later.

This is the basis for the communication protocol between the Map/Reduce framework and the streaming mapper/reducer.

Streaming supports streaming command options as well as generic command options. The general command line syntax is shown below.

`bin/hadoop command [genericOptions] [streamingOptions]`

The Hadoop streaming command that we use in our project are:

Parameter	Optional/Required	Description
-input directory name or filename	Required	Input location for mapper
-output directory name	Required	Output location for reducer
-mapper executable or JavaClassName	Required	Mapper executable .py file in our case.
-reducer executable or JavaClassName	Required	Reducer executable .py in our case
-file filename	Optional	Make the mapper, reducer, or combiner executable available locally on the compute nodes

III. INDEXES BUILT BY HADOOP:

To calculate the TF-IDF indexing we have implemented a three phase MapReduce using the Hadoop Stream jar. The output generated by the Map and Reduce function in each of the three phases is given below

Phase 1:

Mapper: ((word, tweet_id), 1)

Reducer: ((word, tweet_id), word_count_in_tweet)

Phase 2:

Mapper: (tweet_id, (word, word_count_in_tweet))

Reducer: ((word, tweet_id), (word_count_in_tweet, words_in_tweet))

Phase 3:

Mapper: (word, (tweet_id, word_count_in_tweet, words_in_tweet, 1))

Reducer: ((word, tweet_id), tf-idf)

IV. HADOOP-CREATED INDEX TO DO THE RANKING

The created TF-IDF scores are used to rank the tweets in the following way,

- We look for each Tweet_id.
- For each tweet ID we calculate a TF-IDF score

$$\text{TF-IDF}_{\text{tweet_id},Q} = \text{tf-idf}_{q_1,\text{tweet_id}} + \text{tf-idf}_{q_2,\text{tweet_id}} + \dots + \text{tf-idf}_{q_n,\text{tweet_id}}$$

- $Q = q_1, q_2, \dots, q_n$ are the terms in the query.
- Now the top 10 tweet_id with the highest value for TF-IDF is selected from the values calculated for each tweet_id.

V. LUCENE INDEXING AND RANKING

We are performing lucene indexing separately on User data and Tweet data. This is to reduce the space occupied by indexes. This is achieved by using Hash-map to eliminate the duplicate user details that come along with the tweets. The result of this implementation is presented below.

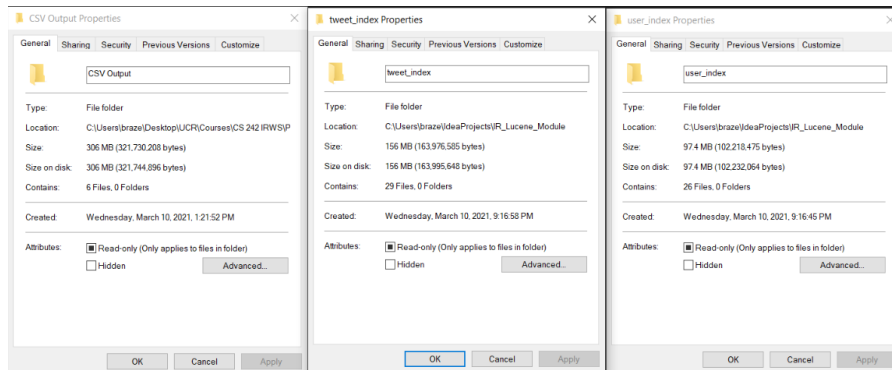
Before Hashing		After Hashing	
Parameter	Size	Parameter	Size
Users	861247	Users	521028
Tweets	861247	Tweets	861247

This above table shows the reduction in user count that is indexed by lucene. This improves two more parameters. They are indexing time and index size. The effect that hashing has on the size of the indices is presented below. This is because we are indexing a reduced number of users. The number of duplicate users eliminated in this case is 340219.

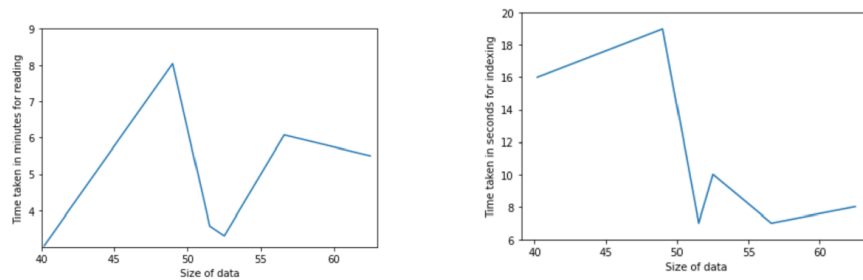
File Name	Size
Crawled Input	306 MB
tweet_index	156 MB
user_index	97.4 MB

The above table shows the reduction in user_index size. We can see a drastic ~50 MB reduction in size. The index also depends on the number of values we wish to store. But in

this case, We are storing almost equal numbers of variables in both tweet index and user index.



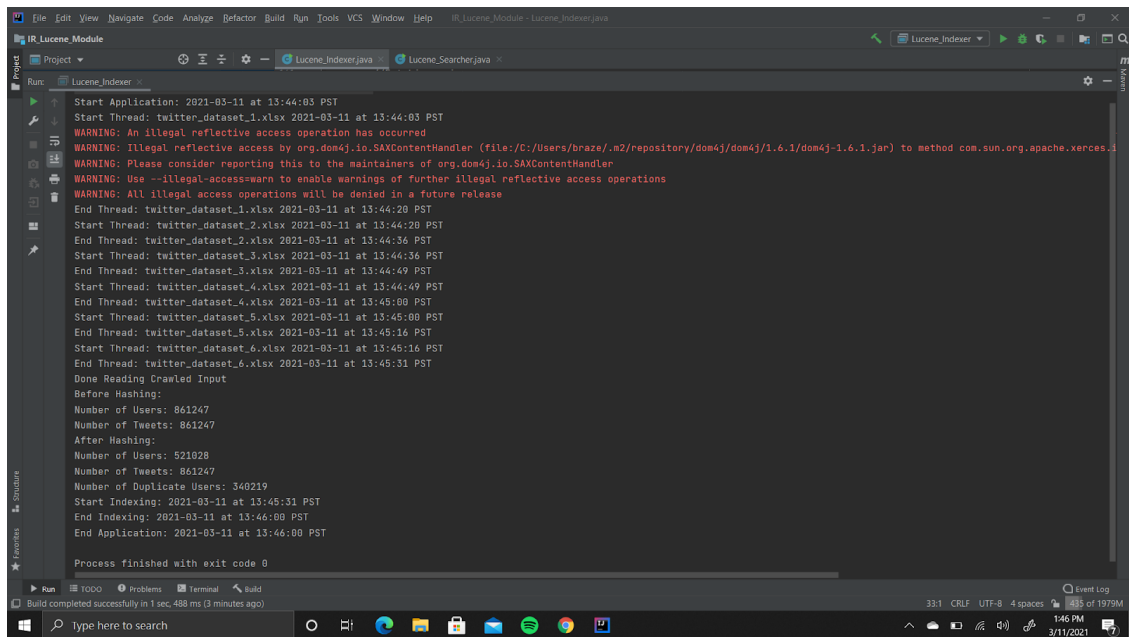
In addition to this the lucene indexing is done in a single-threaded fashion to reduce the time complexity. In Part A of the project, the reported run time of lucene indexing was ~ 30 mins, but by the use of threading we have drastically reduced the time taken to index the tweets by just ~ 2 mins of around 300 MB of data. The result of this implementation is presented below



The above graphs show Time taken to read the data and index the data.

Inference: Reading the data is taking more time than indexing the data.

Fig 1: Phase 1 Lucene Indexing Runtime



```
Start Application: 2021-03-11 at 13:44:03 PST
Start Thread: twitter_dataset_1.xlsx 2021-03-11 at 13:44:03 PST
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.dom4j.io.SAXContentHandler (file:/C:/Users/braze/n2/repository/dom4j/dom4j-1.6.1/dom4j-1.6.1.jar) to method com.sun.org.apache.xerces.internal.impl.xpath.XPathNavigatorImpl.getNodeValue()
WARNING: Please consider reporting this to the maintainers of org.dom4j.io.SAXContentHandler
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
End Thread: twitter_dataset_1.xlsx 2021-03-11 at 13:44:20 PST
Start Thread: twitter_dataset_2.xlsx 2021-03-11 at 13:44:20 PST
End Thread: twitter_dataset_2.xlsx 2021-03-11 at 13:44:36 PST
Start Thread: twitter_dataset_3.xlsx 2021-03-11 at 13:44:36 PST
End Thread: twitter_dataset_3.xlsx 2021-03-11 at 13:44:49 PST
Start Thread: twitter_dataset_4.xlsx 2021-03-11 at 13:44:49 PST
End Thread: twitter_dataset_4.xlsx 2021-03-11 at 13:45:00 PST
Start Thread: twitter_dataset_5.xlsx 2021-03-11 at 13:45:00 PST
End Thread: twitter_dataset_5.xlsx 2021-03-11 at 13:45:16 PST
Start Thread: twitter_dataset_6.xlsx 2021-03-11 at 13:45:16 PST
End Thread: twitter_dataset_6.xlsx 2021-03-11 at 13:45:31 PST
Done Reading Crawled Input
Before Hashing:
Number of Users: 861247
Number of Tweets: 861247
After Hashing:
Number of Users: 521028
Number of Tweets: 861247
Number of Duplicate Users: 340219
Start Indexing: 2021-03-11 at 13:45:31 PST
End Indexing: 2021-03-11 at 13:46:00 PST
End Application: 2021-03-11 at 13:46:00 PST
Process finished with exit code 0
```

Fig 2: Phase 2 Lucene Indexing Runtime

We are taking the lucene index as a database and get all the information from them after ranking. For this, We are analyzing and storing all the variables of the tweets and users using **keyword analyzer**. We are analyzing and storing the user description and tweet description using **english analyzer** because it provides us with stemming and stop words removal.

For Ranking, we are following the same structure for finding results from both user index and tweet index. We are creating Index Searchers for both indices and perform ranking with them. First, we will parse the input query with the help of english analyzer. Then, we use the parsed query to rank the documents using tweetIndexSearcher object. This will return a top n results by using **BM25 Ranking algorithm**. For the query parser, we will be sending the fields that are going to be ranked, the analyzer choice and the field weights. The fields which we used are Tweet description, Tweet hashtags which are analyzed using english analyzer and keyword analyzer respectively. They are given a weights of 0.7 and 1.0 respectively because in tweets hashtags carry more relevance to the content of the tweet.

Then, for each tweet we will be retrieving the user details by using the **Boolean Ranking algorithm** on user_id which is present on both the indices. user_id acts as a primary key in this case. We then combine the tweet information along with the user information and store it in an arraylist. This arraylist is later looked up to displaying the results in the web interface.

VI. RUN TIME OF THE HADOOP INDEX CONSTRUCTION

Information of sampling	Average Map time(s)	Average shuffle time(s)	Average merge time(s)	Average reduce time(s)	Average total time(s)
Phase 1	18	16.2	7.3	1.6	43.3
Phase 2	24.5	22	7.9	10.31	58.2
Phase 3	32.3	30.9	10.21	17.07	90.48

C. LIMITATIONS OF SYSTEM.

- ★ **Lucene Indexing:** The sole limitation we found during indexing is that only one instance of index writer can be active at a time.
- ★ **Lucene Ranking:** We did not come across any limitations while working with lucene ranking.

D. OBSTACLES AND SOLUTIONS

- ★ **Obstacle in Lucene Indexing:** Initially we tried to use a CSV(comma separated file) but it was memory bound. Since the size of the dataset got from the crawling is large, It is not possible to provide a large enough buffer to read the dataset in one go.
- ★ **Solution:** In order to address this issue, we came up with threading, in which the dataset is broken into multiple small dataset and read by the use of threads. This also improved the time taken to read the data considerably.

E. INSTRUCTIONS ON HOW TO DEPLOY THE SYSTEM.

```
$HADOOP_HOME/bin/hadoop jar $HADOOP_HOME/hadoop-streaming.jar \  
-input input/*.csv \  
-output myOutputDir \  
-mapper MapperPhaseOne.py \  
-reducer ReducerPhaseOne.py
```


II. WEB-APPLICATION (E.G. WEB ARCHIVE) THAT IS DEPLOYED TO A WEB SERVER LIKE TOMCAT.

We used Angular CLI and Spring IO to develop and publish our web application. We used the Spring framework as the backend and Angular framework as the front end.

Web application parts:

- ★ Spring Framework
- ★ Angular Framework
- ★ Application Frontend

We will explain each part of the web application on how it works!

→ Setting up the Spring Framework:

- 1) **Initialization:** We created and initialized the project from the website <https://start.spring.io/>. We provided information like project type, language, spring boot, metadata and java version. We will get a zip file of the project with all the dependencies. We can use this as the base for our project.
- 2) **Initial Run:** We installed IntelliJ to run maven projects. We can open the project through the pom.xml file and run the main class for the spring to boot up. After booting up we can visit where the spring has hosted our project. Usually it will be localhost.com/8080. This can be modified depending on our requirements.
- 3) **Spring Modification:** This part of the project can be broken down into three parts.
 - a) **Tweet Class:** We create a template in the form of a tweet class and use it all across the spring framework to pass the data.
 - b) **Lucene Searcher:** We can make use of the tweet_index and user_index that we got from the lucene indexer. We have to modify the ranking function to make it work alongside spring. We did this by creating a getResults() function inside a lucene searcher class and calling it from the tweet controller.
 - c) **Tweet Controller:** The tweet controller is responsible for mapping our project in the localhost link. We can make use of get and post api in the tweet controller. We used RequestParam to get the query from the URL and pass it to the lucene searcher class by creating an object and calling the getResults() function along with the query that is got from the URL. After getting the result from the class we just post this information on the mapped path in localhost.

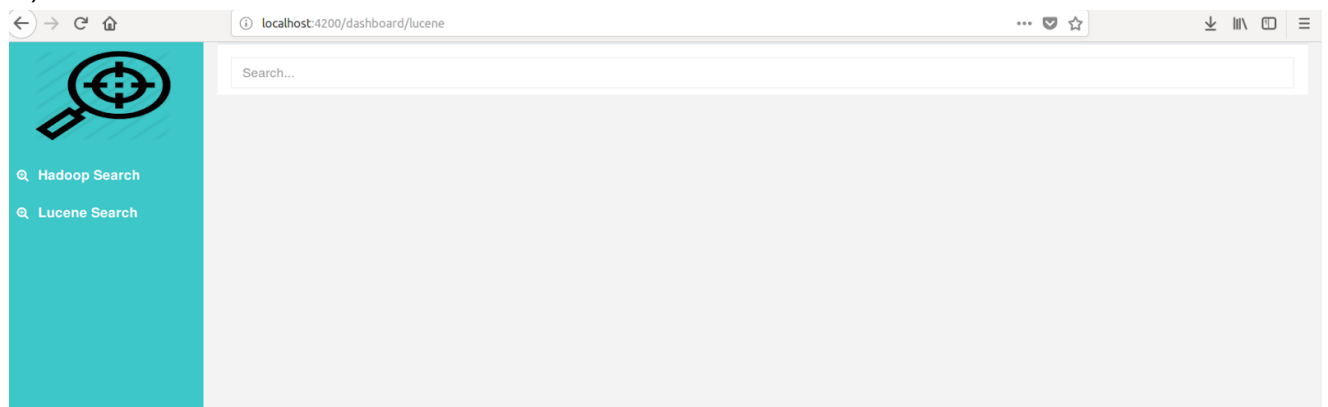
Thus we set up the spring to get the ranked tweets from the lucene searcher and post it in the mapped localhost link. Now, we need to set up the angular part to get the result from the spring framework and show it in the front end using angular framework.

→ Setting up the Angular framework:

- 1) **Initialization:** We need to install nodeJS and angular cli from <https://nodejs.org/en/> and <https://cli.angular.io/> respectively. We are using nodeJS version 6.14.11 and Angular version 11.2.3 for our project.
- 2) **Initial Run:** We can run the angular projects using the command `ng --serve` which will host our project at `localhost.com/4200`. This is the default path but this can be customized based on the requirements.
- 3) **Running the provided code :-** To run our final angular code, you need to first download spring code and host it with intell j, it will host it by default on `localhost:8080`, now download the angular code and run `ng serve` in it, it will host it by default on `localhost:4200`. We have included the `localhost:8080` link in our angular to host the apis. The code will run on `localhost:4200/dashboard/lucene`(for lucene) and on `localhost:4200/dashboard/hadoop`(for hadoop). You can also switch to hadoop or lucene from the sidebar panel. I have removed the node packages folder as the project size was becoming heavy, so you need to run **npm install** after downloading the code.
- 4) **Code structure :-** We have tried optimising the code and have created separate components for lucene and hadoop, so lucene have its own html,css,ts files and hadoop has its own files of html,ts and css. Both these components are in the layout folder of the project. I also have a shared folder of sidebar which is used for both lucene and hadoop.

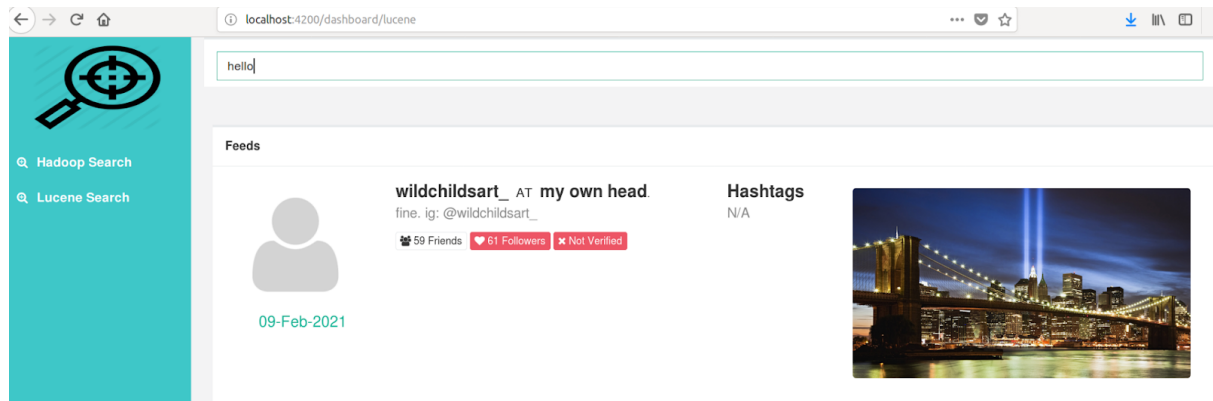
F. Screenshots showing system in action with a complete explanation :-

1.)



This is the frontpage of our Search Engine. You can either query by “lucene” indexing or “hadoop” indexing. There is search bar at the top for searching which will give results based on the search query. In “Lucene” it will return results according to lucene ranking and in hadoop tab it will return results according to the hadoop ranking. **We also added a icon for our Search Engine on the top left corner.**

2.)




When we query, it shows results like this as shown in the above screenshot. **We have done something different with the search query, instead of giving search results on the click of a button, we are returning search results as soon as the user is entering in the search box.** So let's say the user started from the letter "h" it will start giving results which contain the letter "h" in it. Search works on the user description, name and hashtags. So whatever will match according to the search query, it will return it as an output.

Parameters shown in the result :-

- We have shown the user profile pic, in case the user does not have any image or the profile pic parameter is empty we are displaying the dummy placeholder.
- Timestamp of the post creation is shown below the profile pic icon.
- User name is shown in bold in the 2nd column.
- Beside the username is the user location in the second column.
- Below it is the user bio or description or the tweet content.
- Below that is the Friends count, followers count in colourful boxes.
- We have done something different with the "Verified" column. See screenshot below for this.






-
- As you can see "Verified " button is sometimes Green and sometimes red, it is red when the user twitter account it "not verified" and green when the user twitter account is "verified". So for all verified account users it will show a green button with a tickbox and for all non verified users, it will show a cross with a red button.

- 


yanskifilms AT **chicago**

stop letting white boys in film school | they/she | 21
 | @juipiterising dY@âcâı.âcðY'.âcðY'@ðY'

 692 Friends
  1482 Followers
  Not Verified

Hashtags

Boomer, Kolo



- Next in the last column is the tweet image, if user posted any image, it will show the image in this last column otherwise it will not show any image.
- We have also given a hyperlink for the tweet, so some tweets were not having the urls returned while scraping. So we linked the tweets with their urls for which we had the urls. In case it will have the url it will be clicked and redirected to its link otherwise it will not be clicked. We have shown a blue colour on hover if there is a link attached to a tweet.

Now attached is the screenshot.

