**BUAN 6346 Big Data Analytics**

**(Hive – Twitter Data Analysis)**

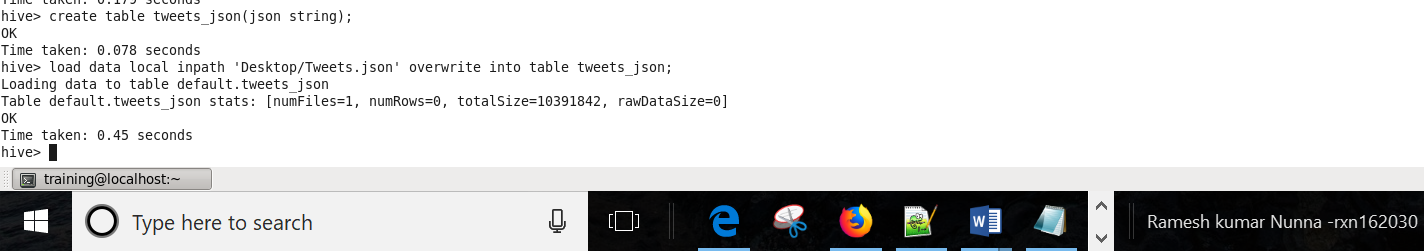
1. What are the hashtags used and how many times each hashtag is used?

**Step 1: To import JSON file from Desktop into Hive:**

**Query :**

create table tweets\_json(json string);

load data local inpath 'Desktop/Tweets.json' overwrite into table tweets\_json;

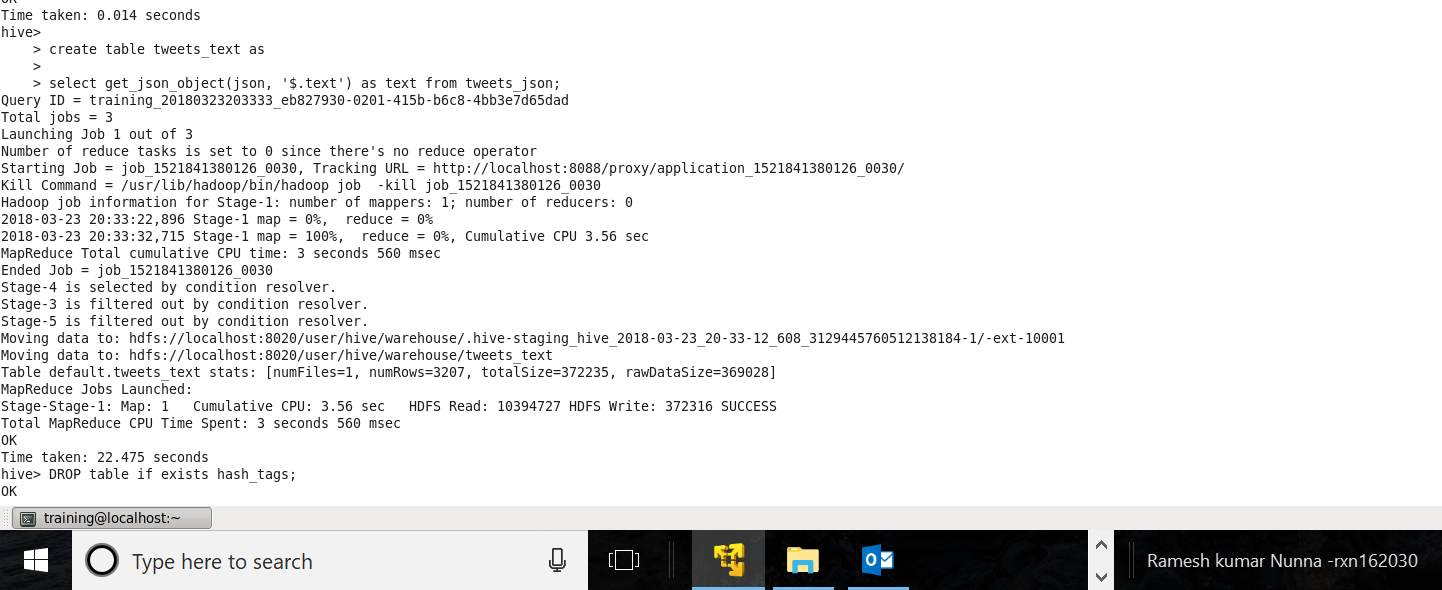


**Step 2: Reading the Json data(.text object) into tweets\_text table**

**Query:**

create table tweets\_text as

select get\_json\_object(json, '$.text') as text from tweets\_json;

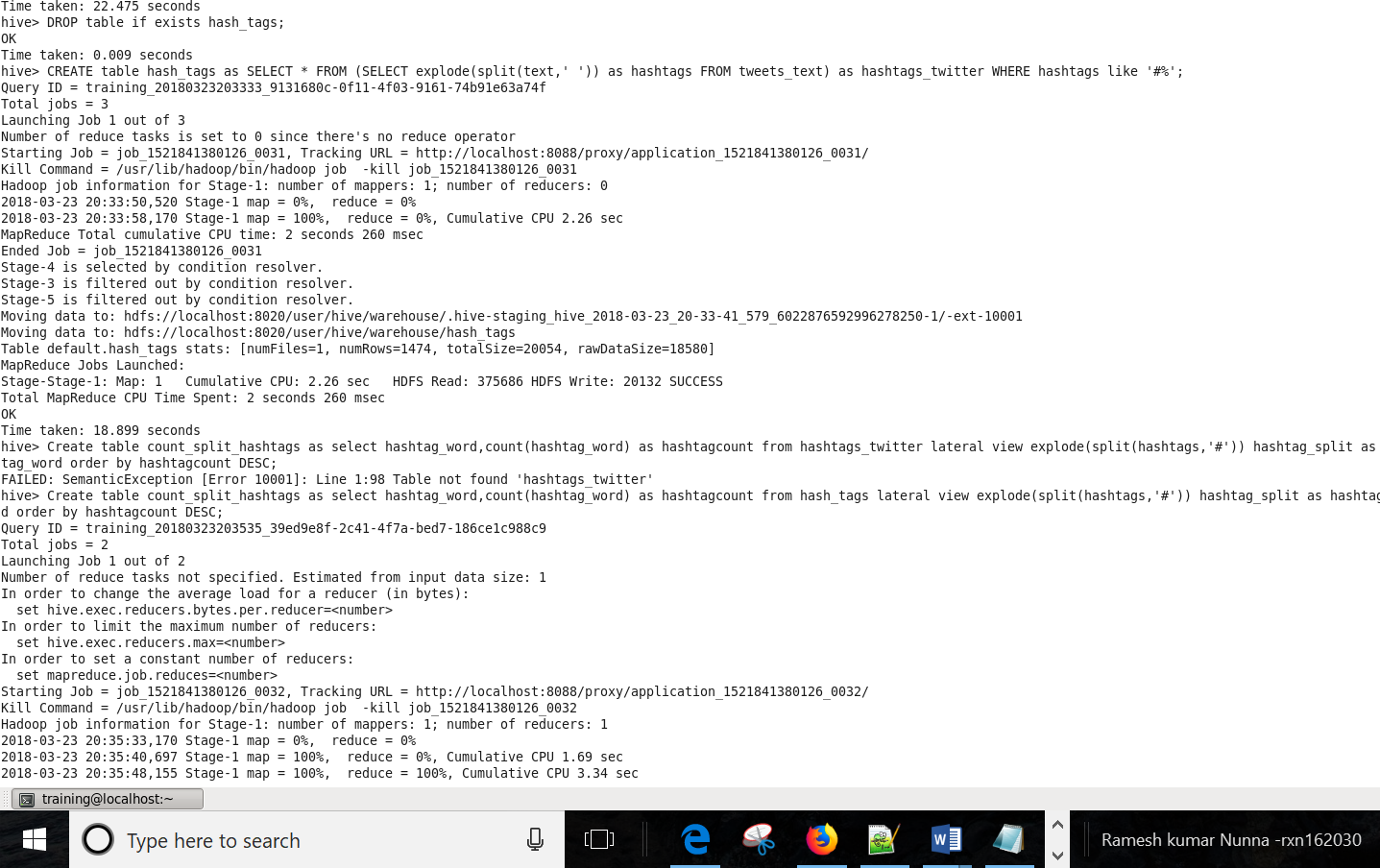


**Step 3: Splitting the text by space and filtering out the hashtags**

**Query:**

DROP table if exists hash\_tags;

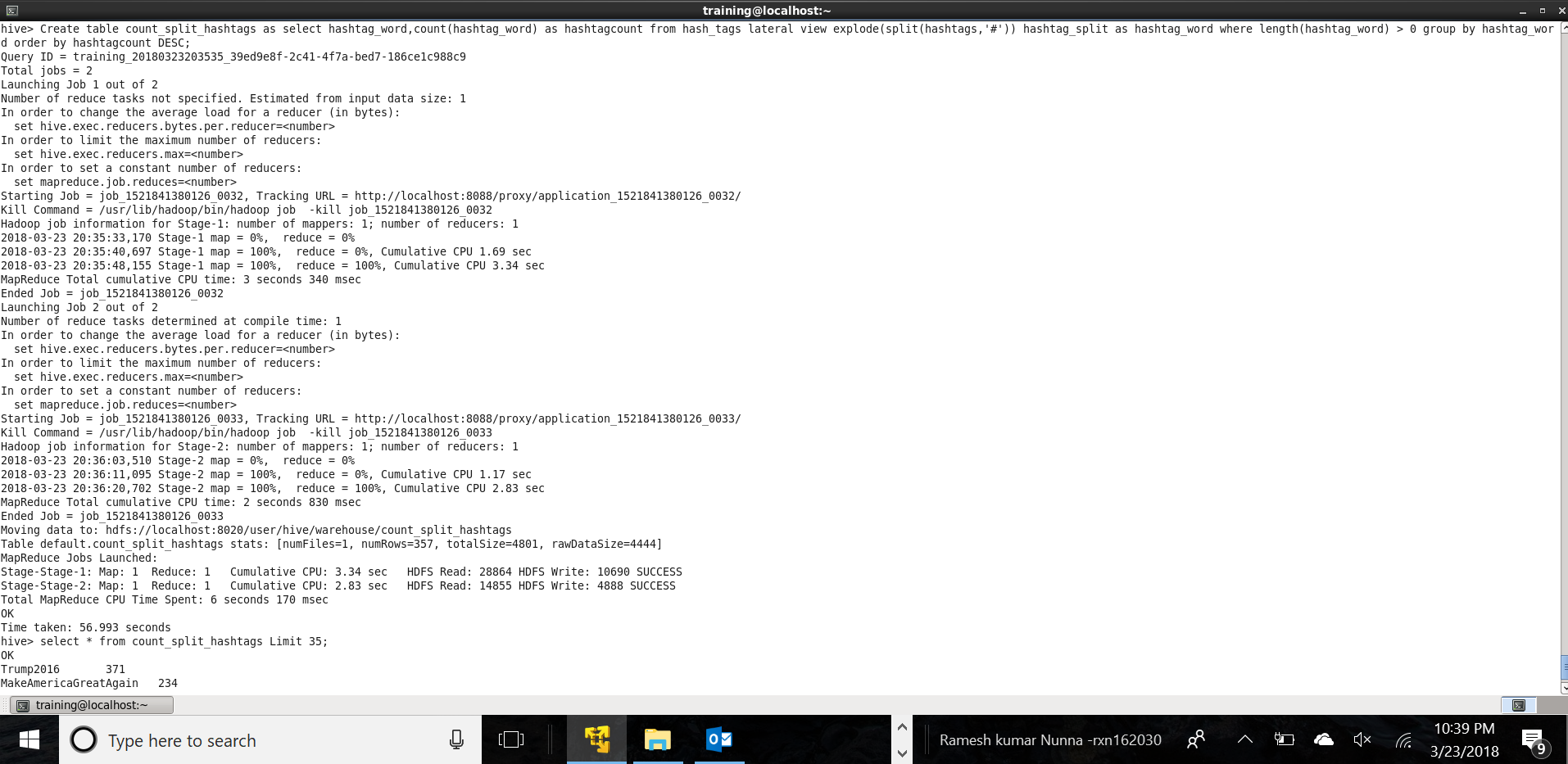
CREATE table hash\_tags as SELECT \* FROM (SELECT explode(split(text,' ')) as hashtags FROM tweets\_text) as hashtags\_twitter WHERE hashtags like '#%';



**Step 4: Again, splitting the words around # (so that continuous hashtags will get split)**

**Query:**

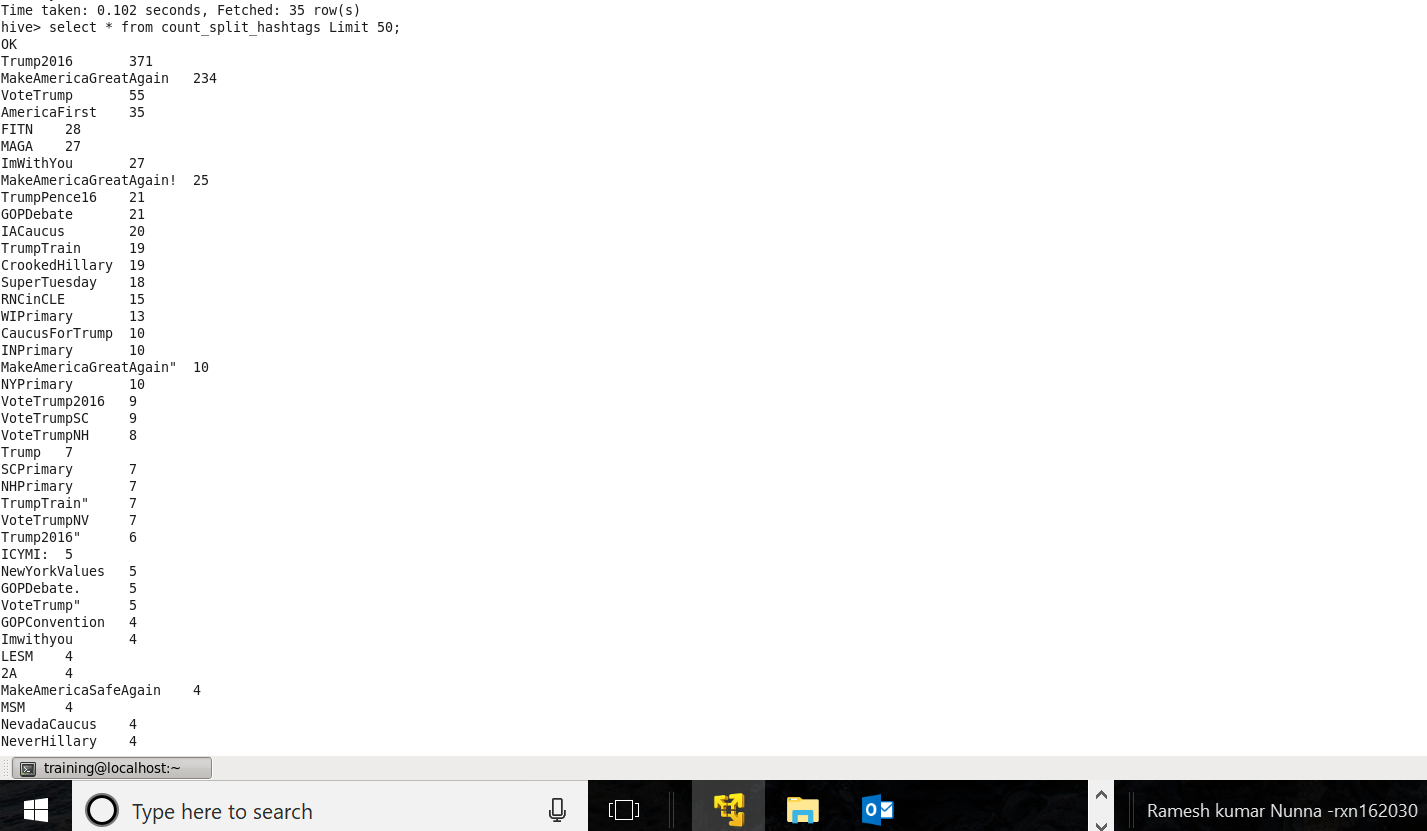
Create table count\_split\_hashtags as select hashtag\_word,count(hashtag\_word) as hashtagcount from hash\_tags lateral view explode(split(hashtags,'#')) hashtag\_split as hashtag\_word where length(hashtag\_word) > 0 group by hashtag\_word order by hashtagcount DESC;



**Step 4: Showing top 35 trending hashtags from current table**

**Query:**

select \* from count\_split\_hashtags Limit 35;

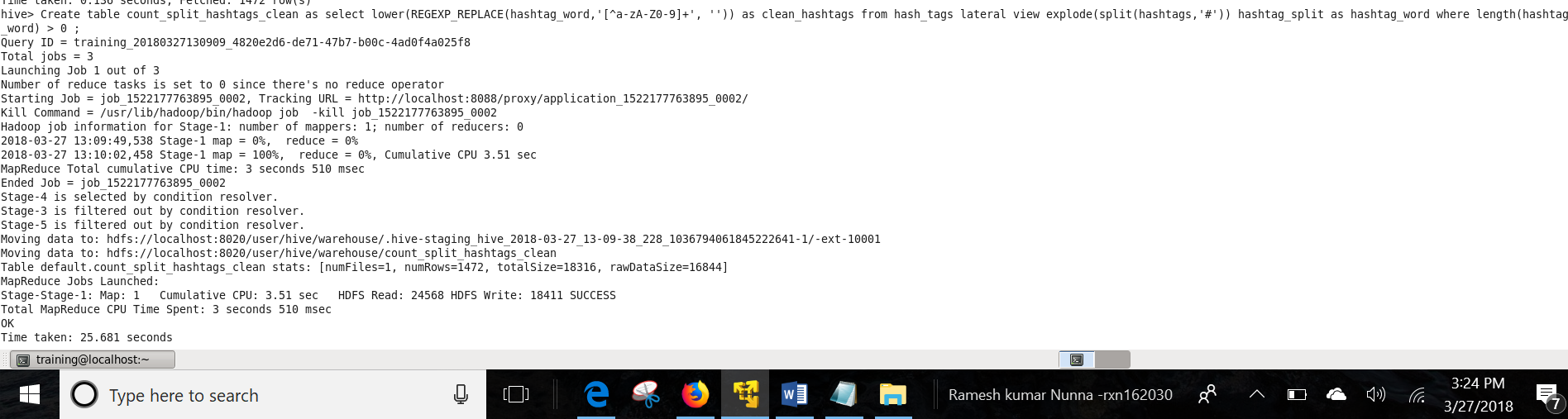


*Optimization: Assuming we should handle case, remove punctuation and extra characters from hashtags*

**Step 5: Optimizing code (Removing punctuation, extra characters and handling case)**

**Query:**

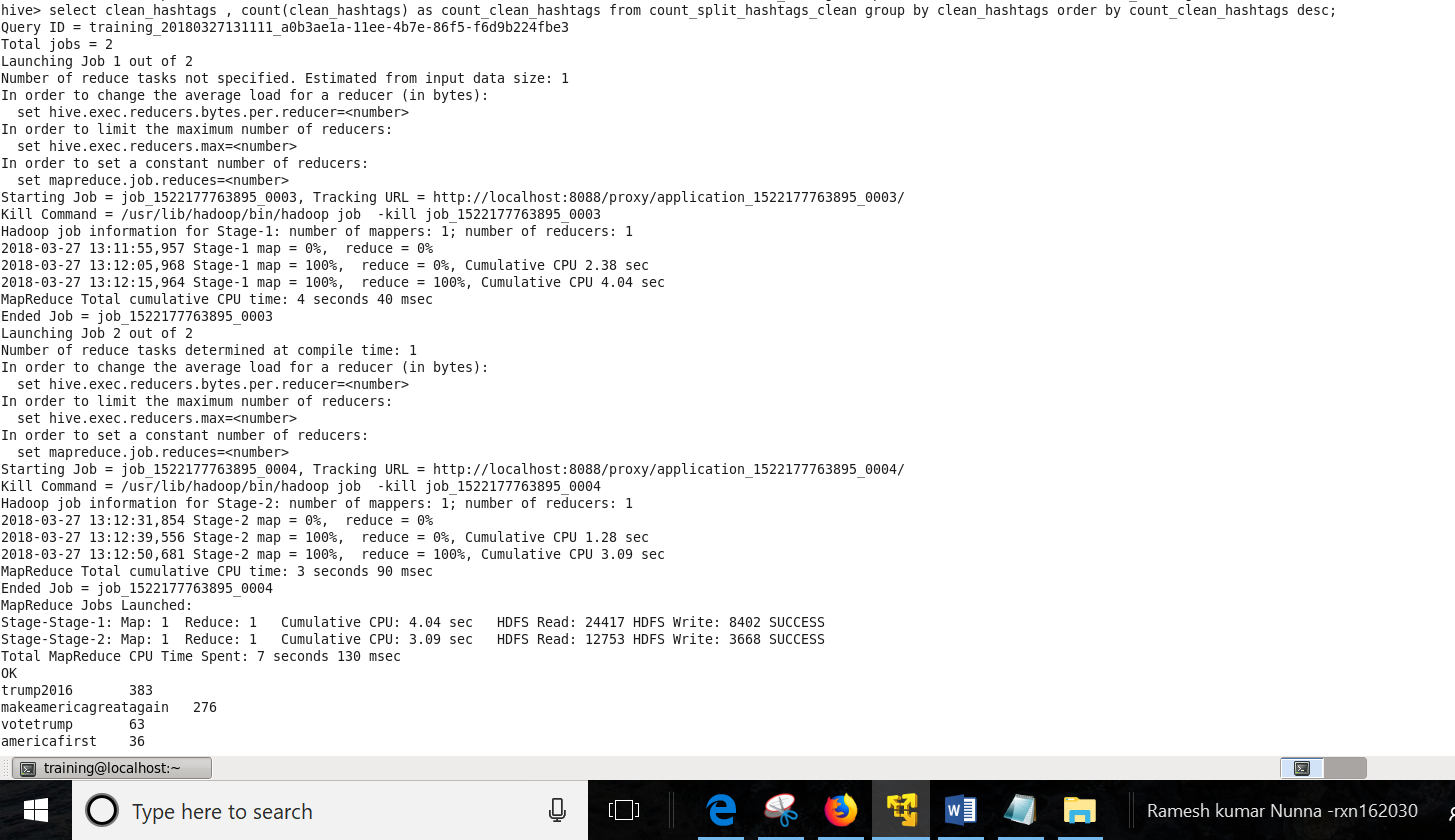
Create table count\_split\_hashtags\_clean as select lower(REGEXP\_REPLACE(hashtag\_word,'[^a-zA-Z0-9]+', '')) as clean\_hashtags from hash\_tags lateral view explode(split(hashtags,'#')) hashtag\_split as hashtag\_word where length(hashtag\_word) > 0



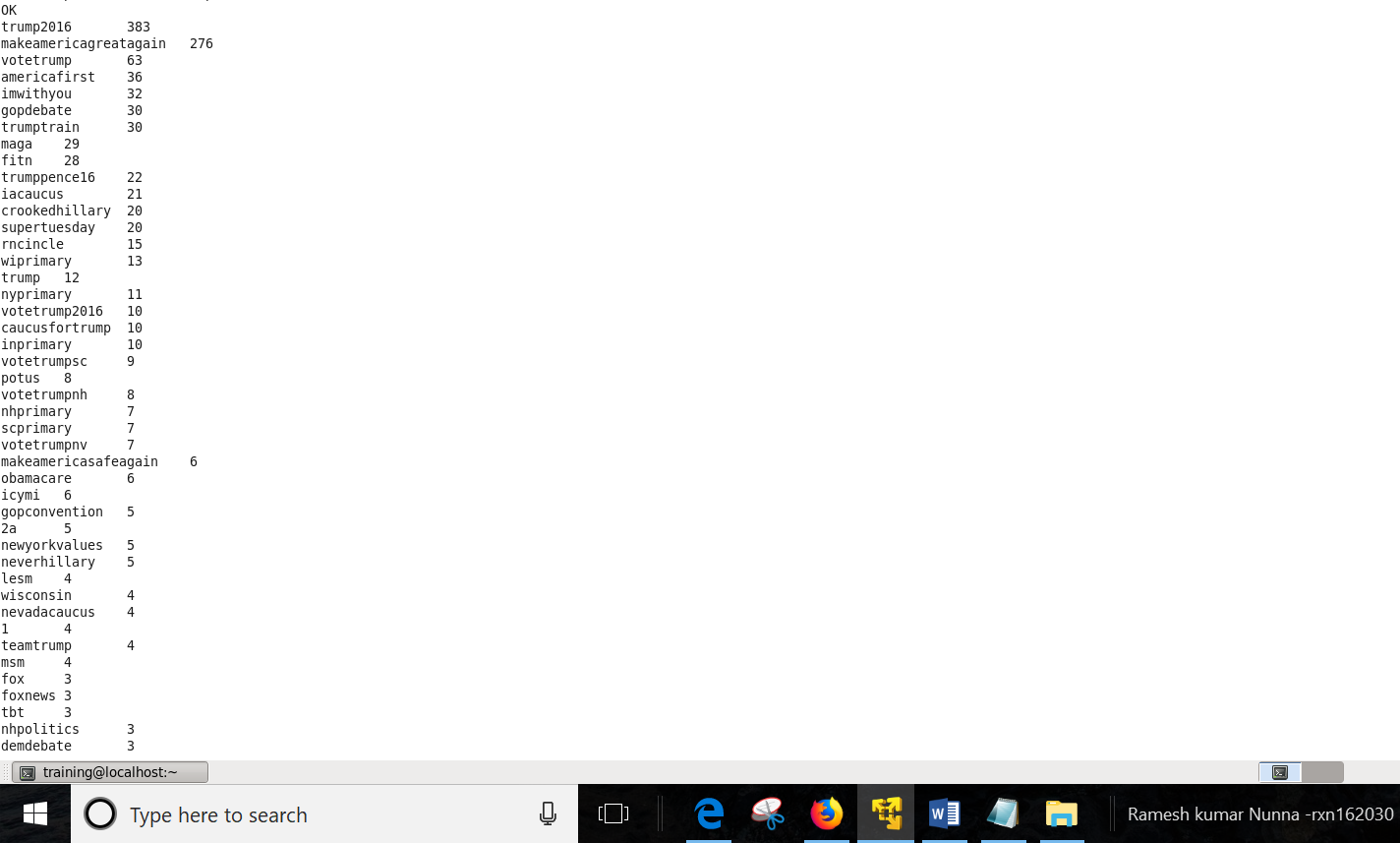
**Step 6: output**

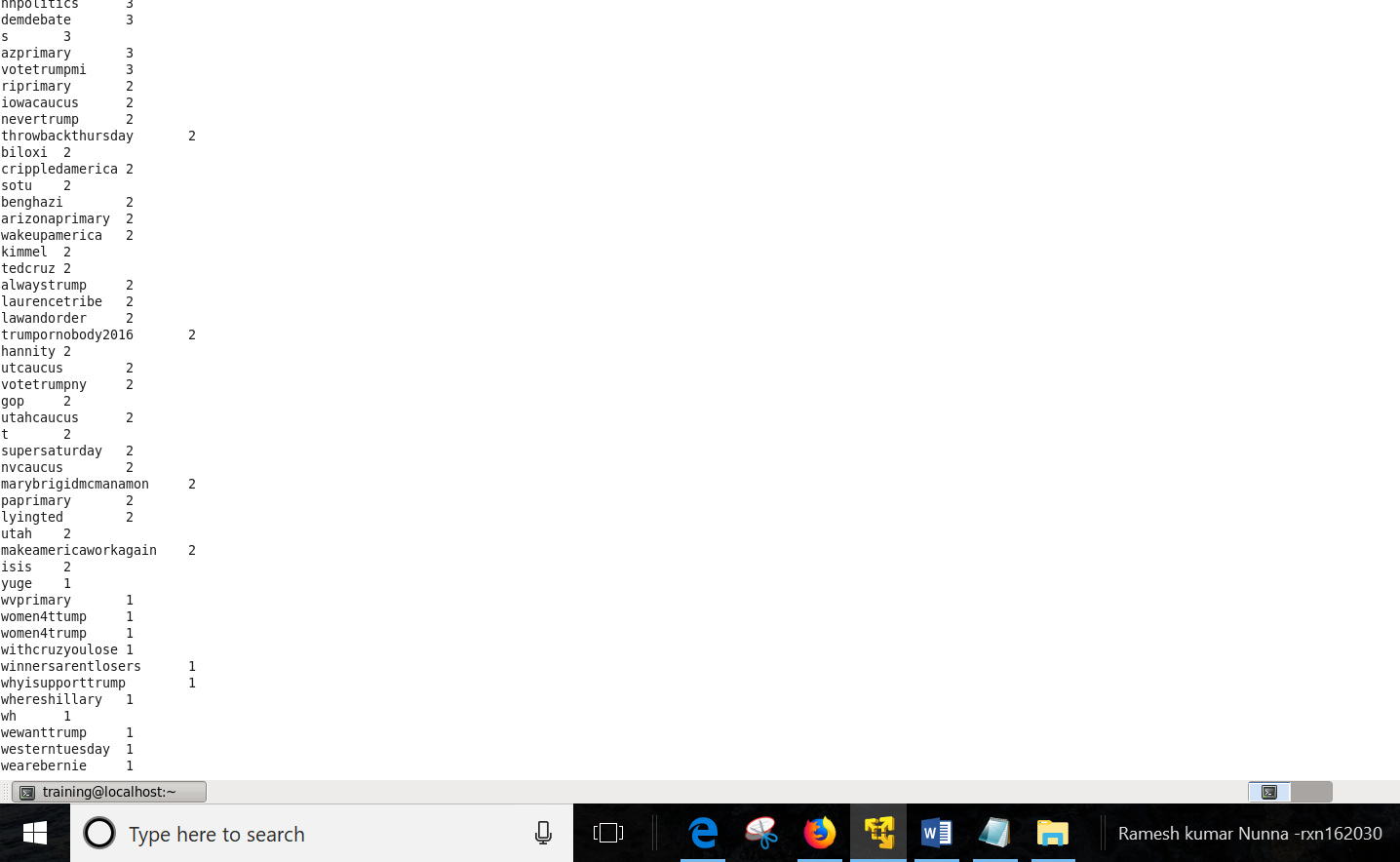
**Query:**

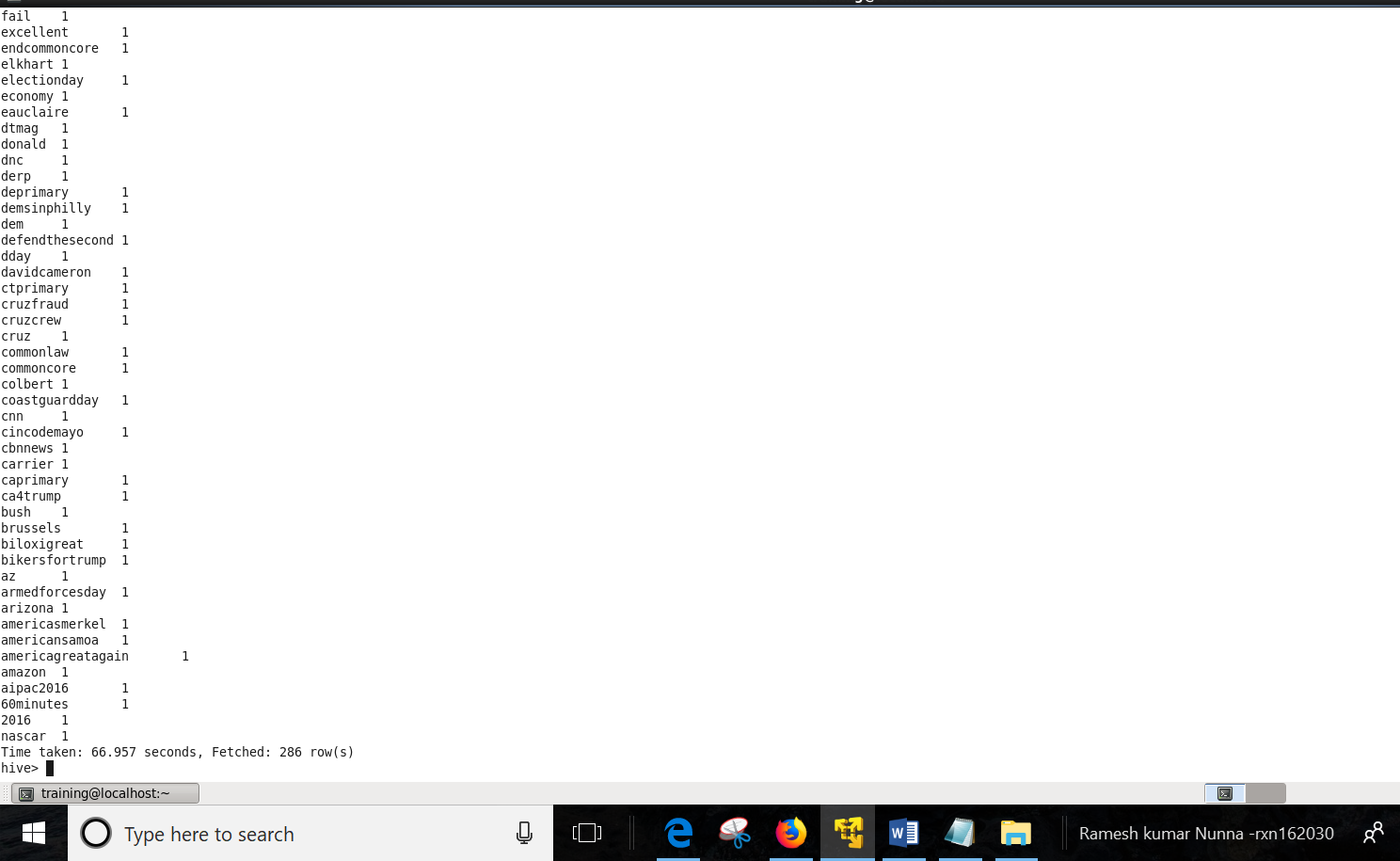
select clean\_hashtags , count(clean\_hashtags) as count\_clean\_hashtags from count\_split\_hashtags\_clean group by clean\_hashtags order by count\_clean\_hashtags desc;



**Showing few hashtags:**







1b) Which State have the most active users and how many tweets are posted by State?

1. ***For questions 1b, 1c assuming '$.user.id' is the user tweeting and '$.user.location’ is the location of the user tweeting.***

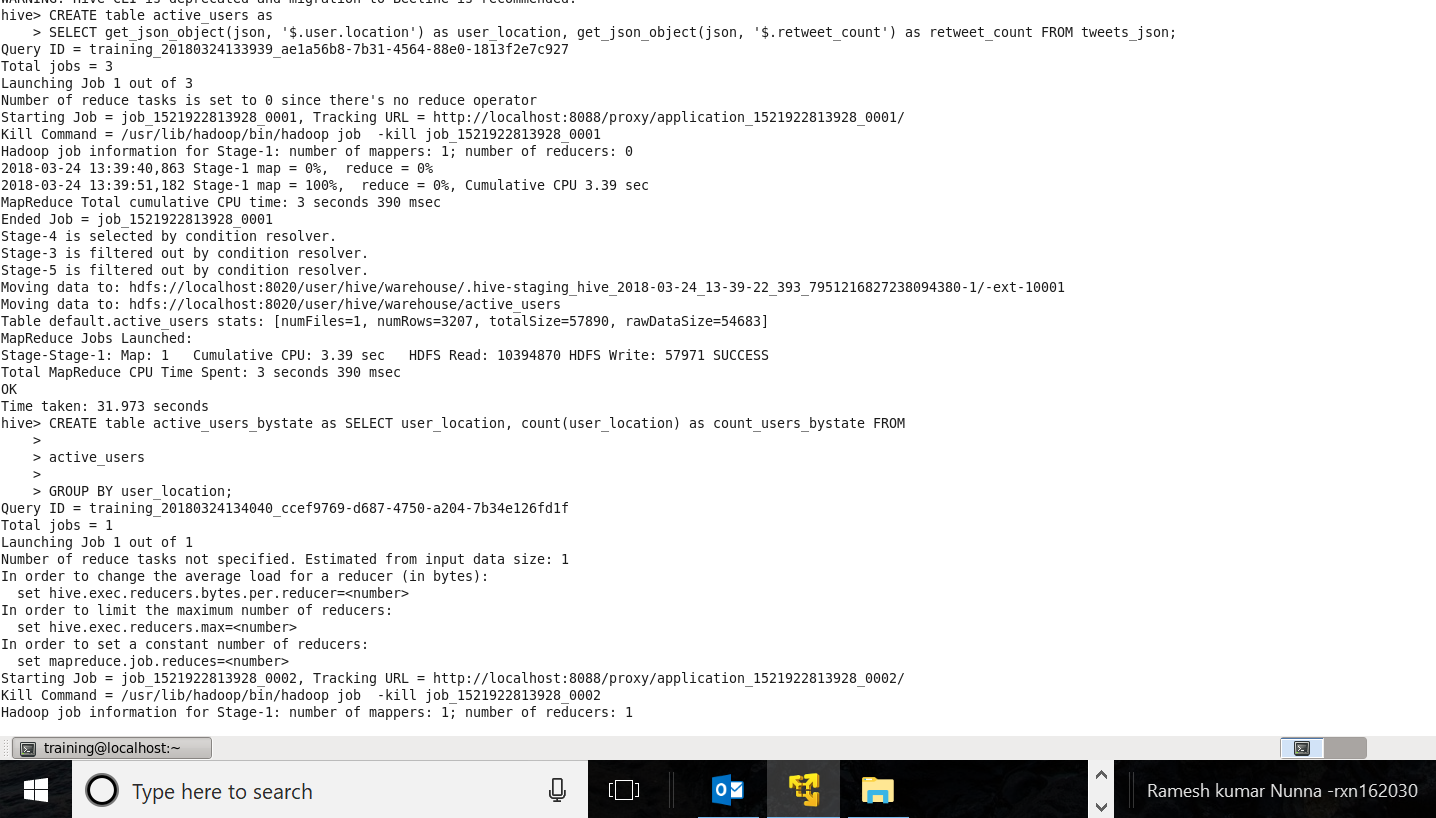
**Assuming '$.user.location’ is the location of the user(s) tweeting**

**Step 1: Reading JSON data**

**Query:**

CREATE table active\_users as

SELECT get\_json\_object(json, '$.user.location') as user\_location, get\_json\_object(json, '$.retweet\_count') as retweet\_count FROM tweets\_json;



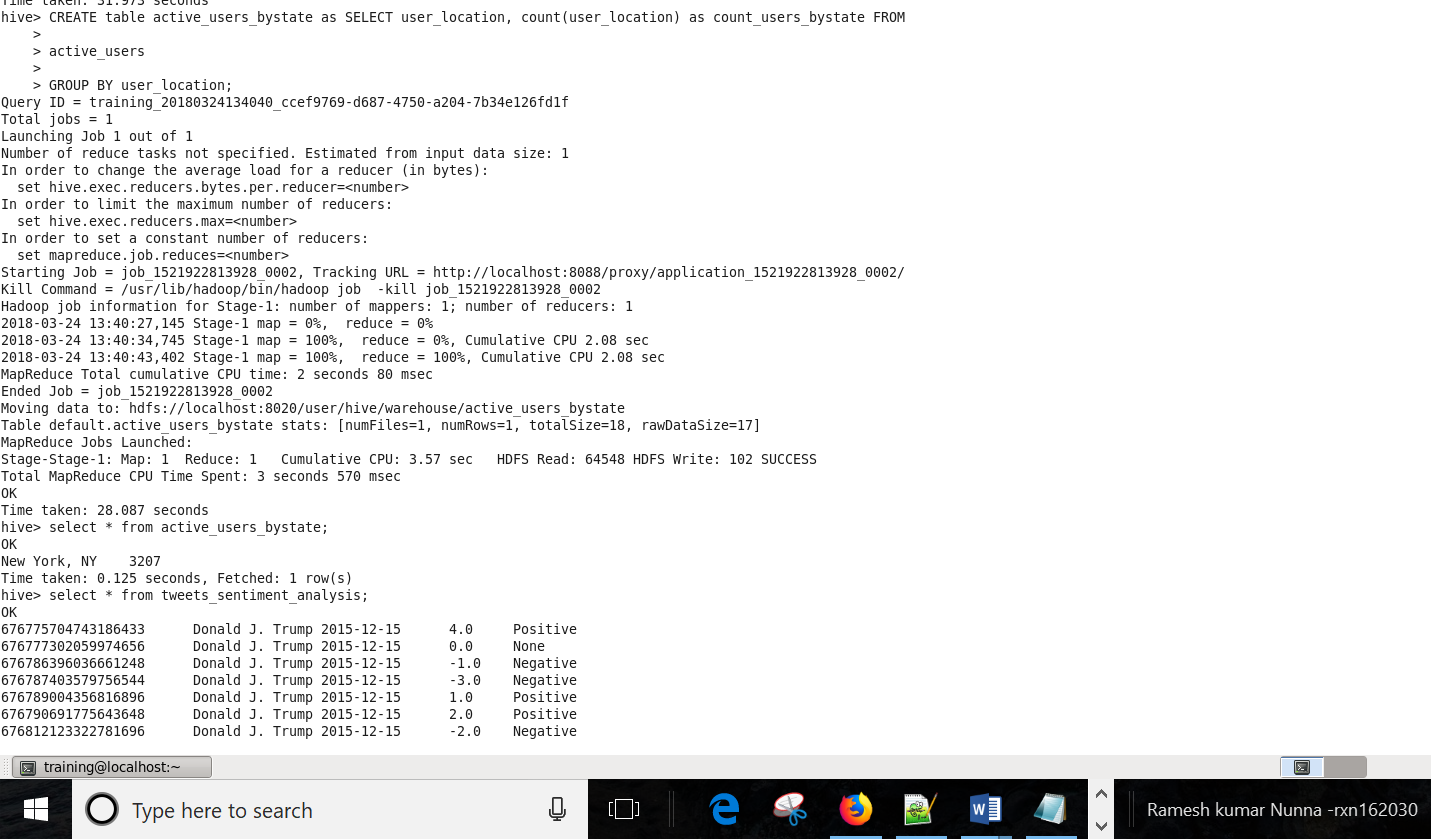
**Step 2: Aggregating users by location**

**Query:**

CREATE table active\_users\_bystate as SELECT user\_location, count(user\_location) as count\_users\_bystate FROM

active\_users

GROUP BY user\_location



**Step 3: Output**

**Query**

Select \* from active\_users\_bystate



***"New York, NY" has the most active users and all the tweets 3207 are posted by that state.***

1 c) Based on the user’s followers count, who are the top ten users who have tweeted?

1. ***Assuming all the tweets are from only user(s) under json object ‘$.user.id'***

**Assuming ‘$.user.id' is the user who is tweeting**

**Step1: Reading Json data**

**Query:**

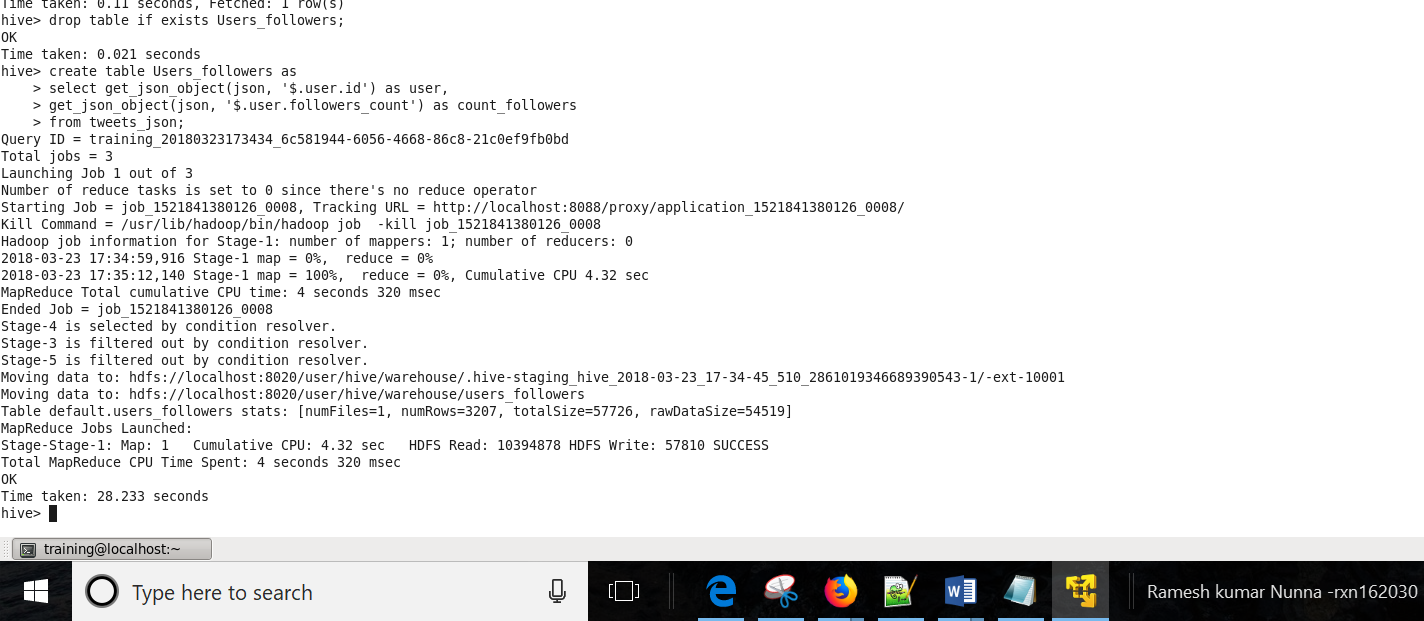
*drop table if exists Users\_followers;*

*create table Users\_followers as*

*select get\_json\_object(json, '$.user.id') as user,*

*get\_json\_object(json, '$.user.followers\_count') as count\_followers*

*from tweets\_json;*



**Step 2: Creating the table and sorting users by Users\_followers\_count**

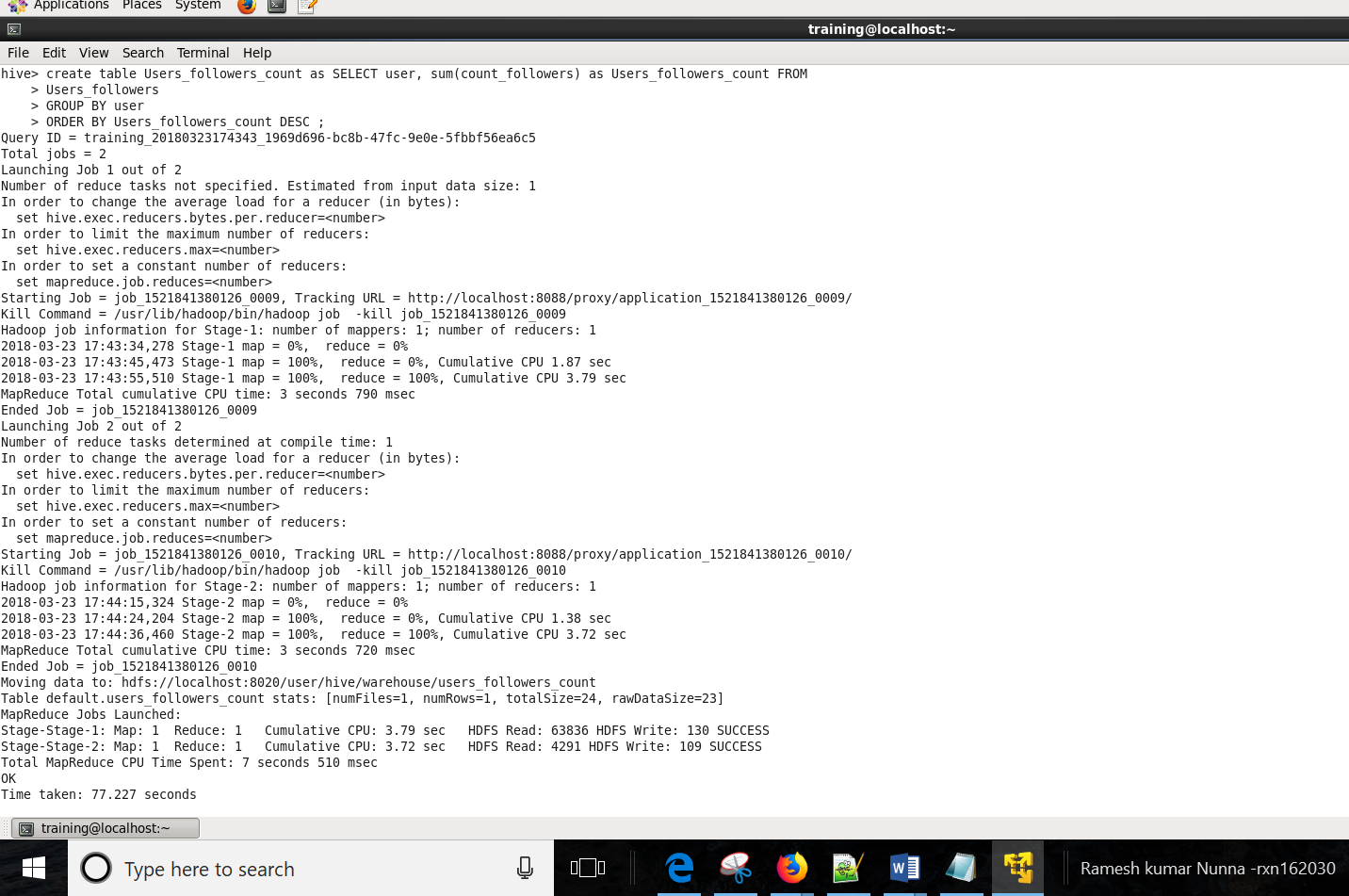
**Query:**

*create table Users\_followers\_count as SELECT user, sum(count\_followers) as Users\_followers\_count FROM*

*Users\_followers*

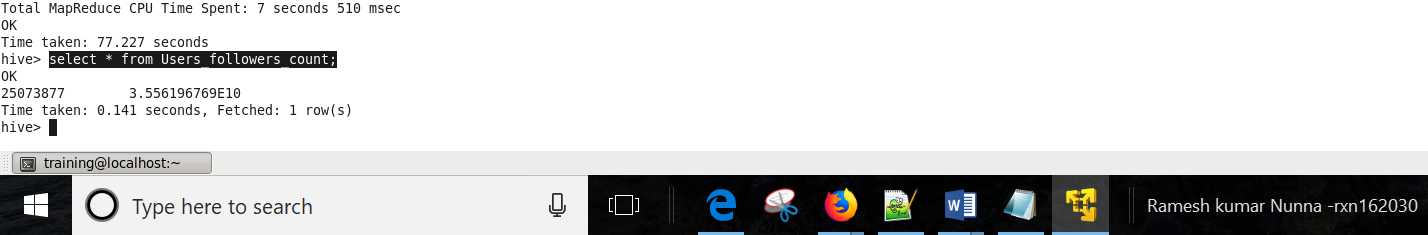
*GROUP BY user*

*ORDER BY Users\_followers\_count DESC ;*



**Step 3:**

*select \* from Users\_followers\_count;*



***"Donald J. Trump"(user id:* *25073877) is the only user who tweeted and he is the top user based on followers\_count***

***All the tweets in the data are “Donald J. Trump” tweets and the location is "New York, NY"***

1d) What is the polarity score for each tweet that was posted? Does the tweet have a positive or negative sentiment?

a)

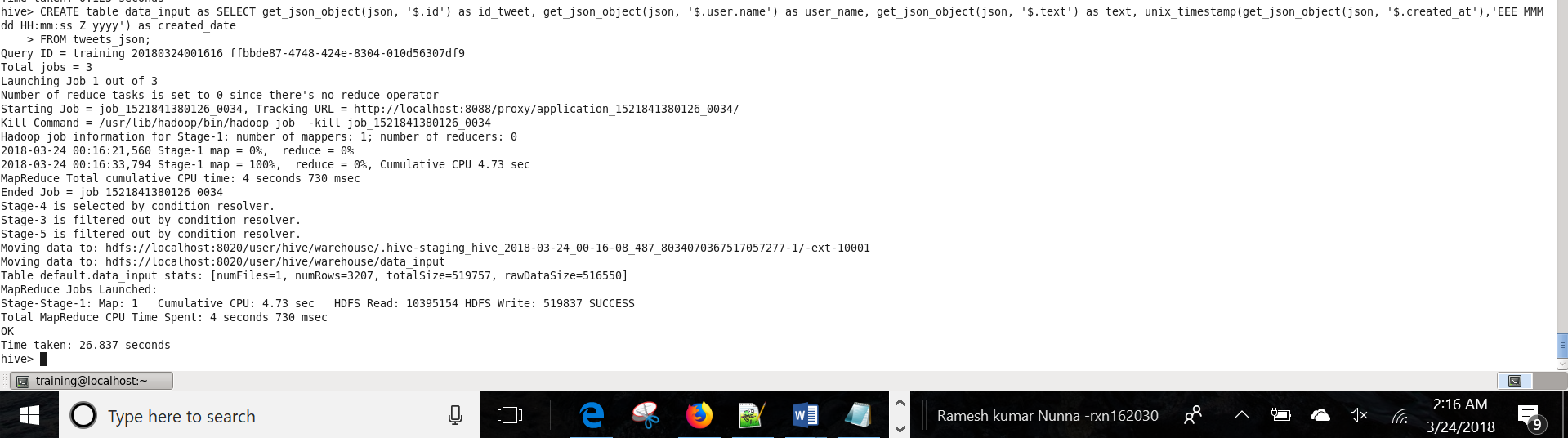
**Step 1: Reading JSON data**

**Query:**

*DROP table if exists data\_input;*

*CREATE table data\_input as SELECT get\_json\_object(json, '$.id') as id\_tweet, get\_json\_object(json, '$.user.name') as user\_name, get\_json\_object(json, '$.text') as text, unix\_timestamp(get\_json\_object(json, '$.created\_at'),'EEE MMM dd HH:mm:ss Z yyyy') as created\_date*

*FROM tweets\_json;*

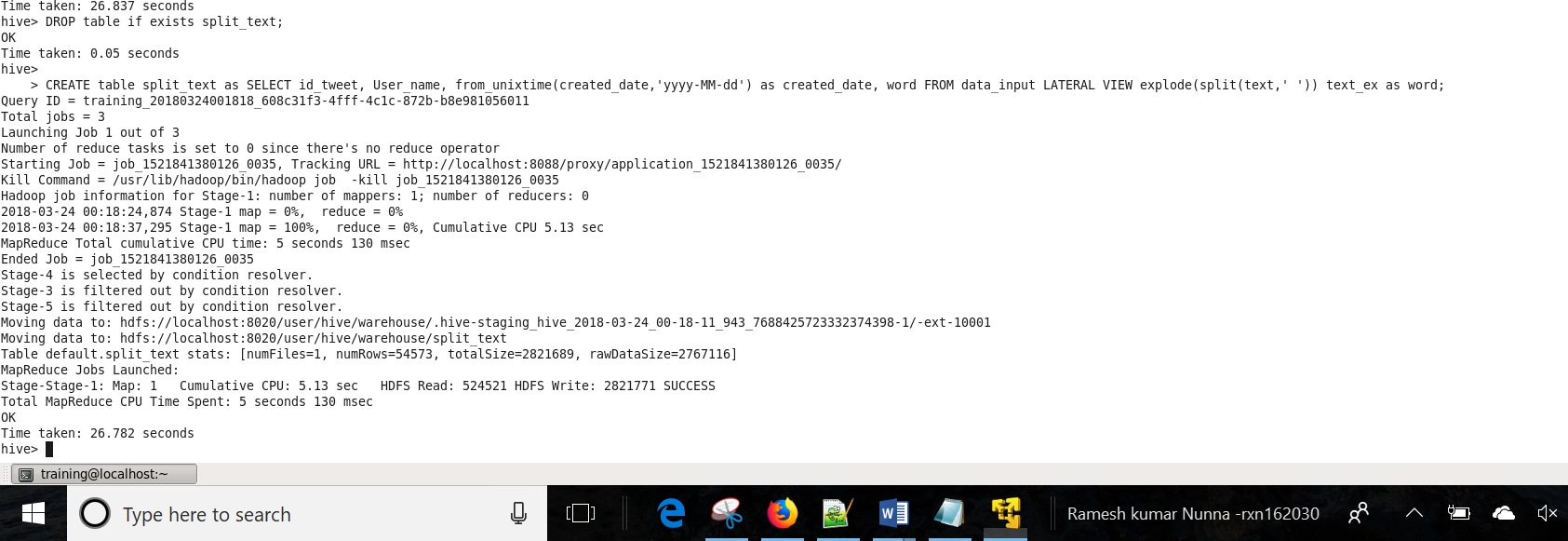


**Step 2: Splitting each tweet into words**

**Query:**

*DROP table if exists split\_text;*

*CREATE table split\_text as SELECT id\_tweet, User\_name, from\_unixtime(created\_date,'yyyy-MM-dd') as created\_date, word FROM data\_input LATERAL VIEW explode(split(text,' ')) text\_ex as word;*



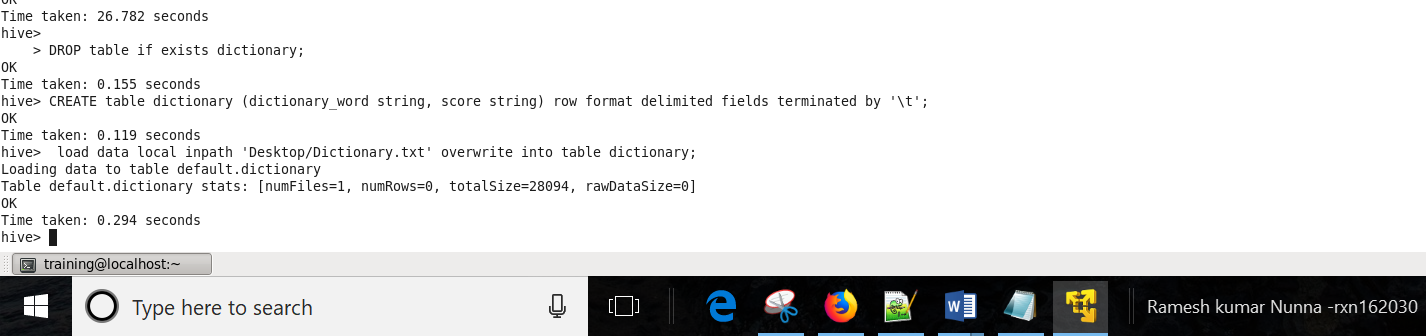
**Step 3: Importing dictionary into hive**

**Query:**

*DROP table if exists dictionary;*

*CREATE table dictionary (dictionary\_word string, score string) row format delimited fields terminated by '\t';*

*load data local inpath 'Desktop/Dictionary.txt' overwrite into table dictionary;*

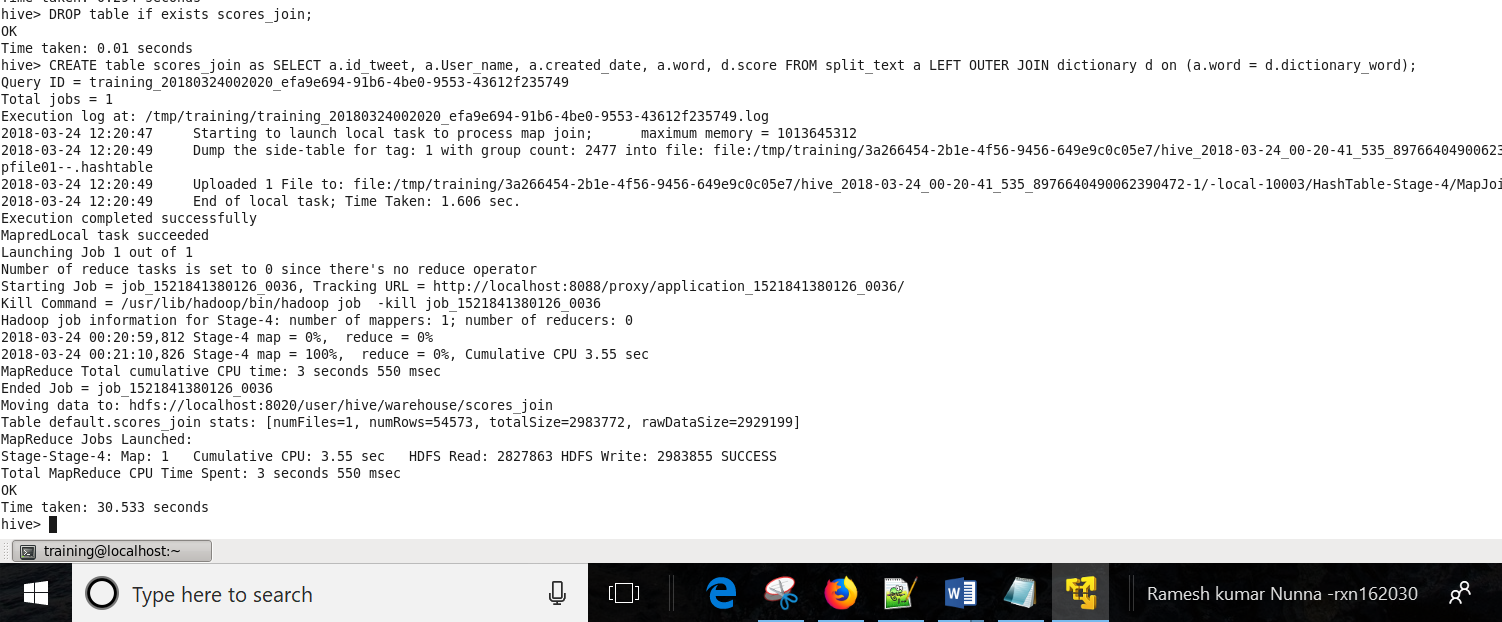


**Step 4: Joining scores from dictionary to twitter data**

**Query:**

*DROP table if exists scores\_join;*

*CREATE table scores\_join as SELECT a.id\_tweet, a.User\_name, a.created\_date, a.word, d.score FROM split\_text a LEFT OUTER JOIN dictionary d on (a.word = d.dictionary\_word);*

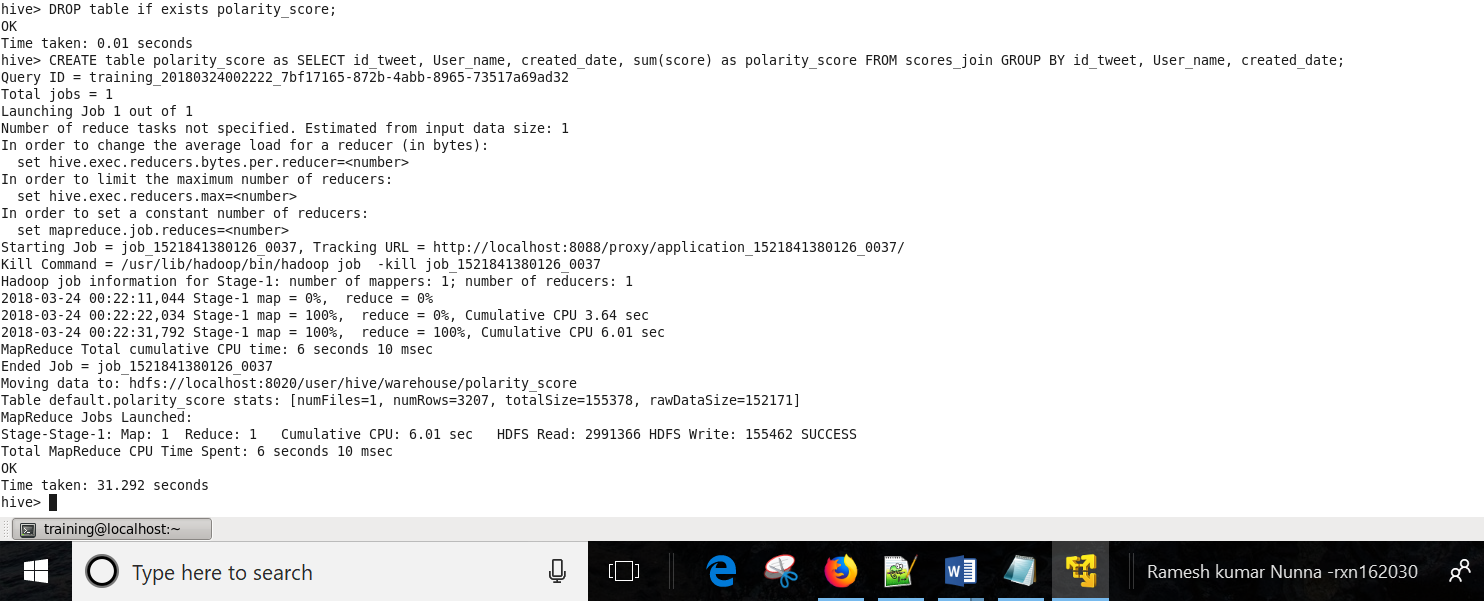


**Step 5: calculating polarity score for each tweet\_id:**

**Query:**

*DROP table if exists polarity\_score;*

*CREATE table polarity\_score as SELECT id\_tweet, User\_name, created\_date, sum(score) as polarity\_score FROM scores\_join GROUP BY id\_tweet, User\_name, created\_date;*

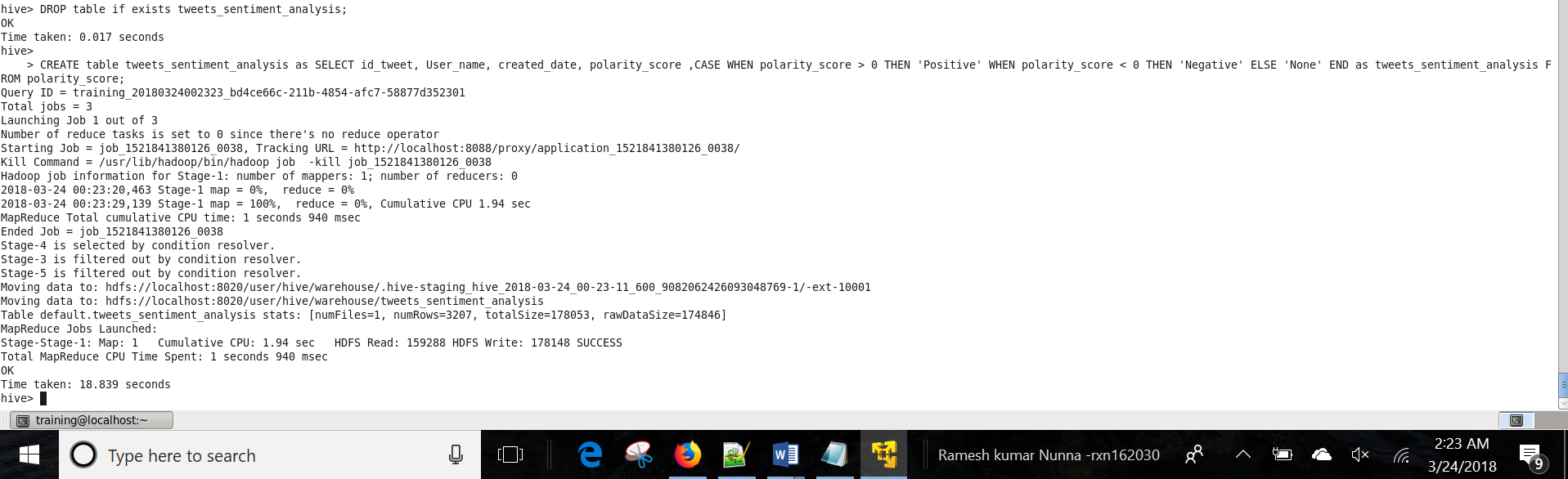


**Step 5: calculating sentiment for each tweet:**

**Query:**

*DROP table if exists tweets\_sentiment\_analysis;*

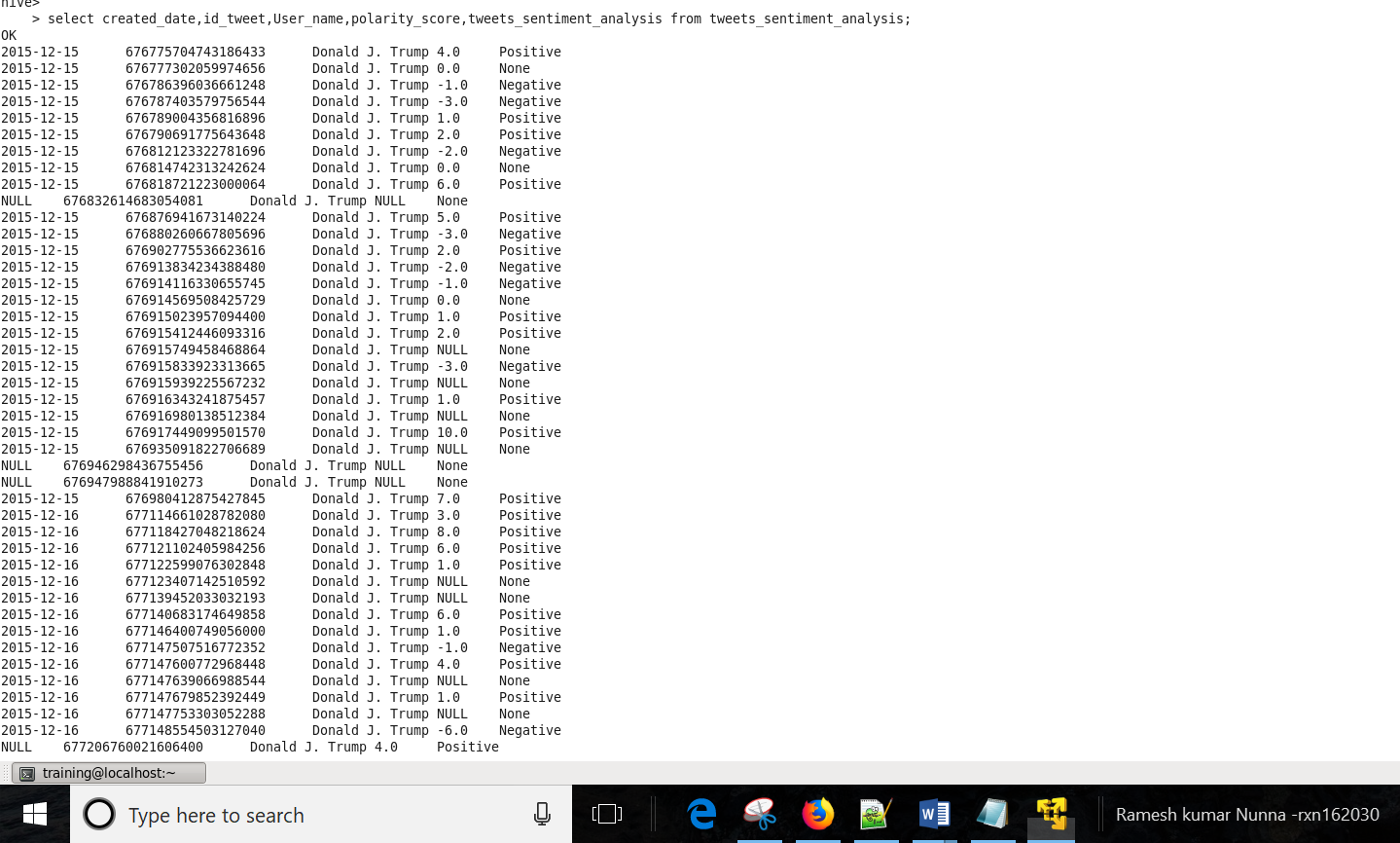
*CREATE table tweets\_sentiment\_analysis as SELECT id\_tweet, User\_name, created\_date, polarity\_score ,CASE WHEN polarity\_score > 0 THEN 'Positive' WHEN polarity\_score < 0 THEN 'Negative' ELSE 'None' END as tweets\_sentiment\_analysis FROM polarity\_score;*

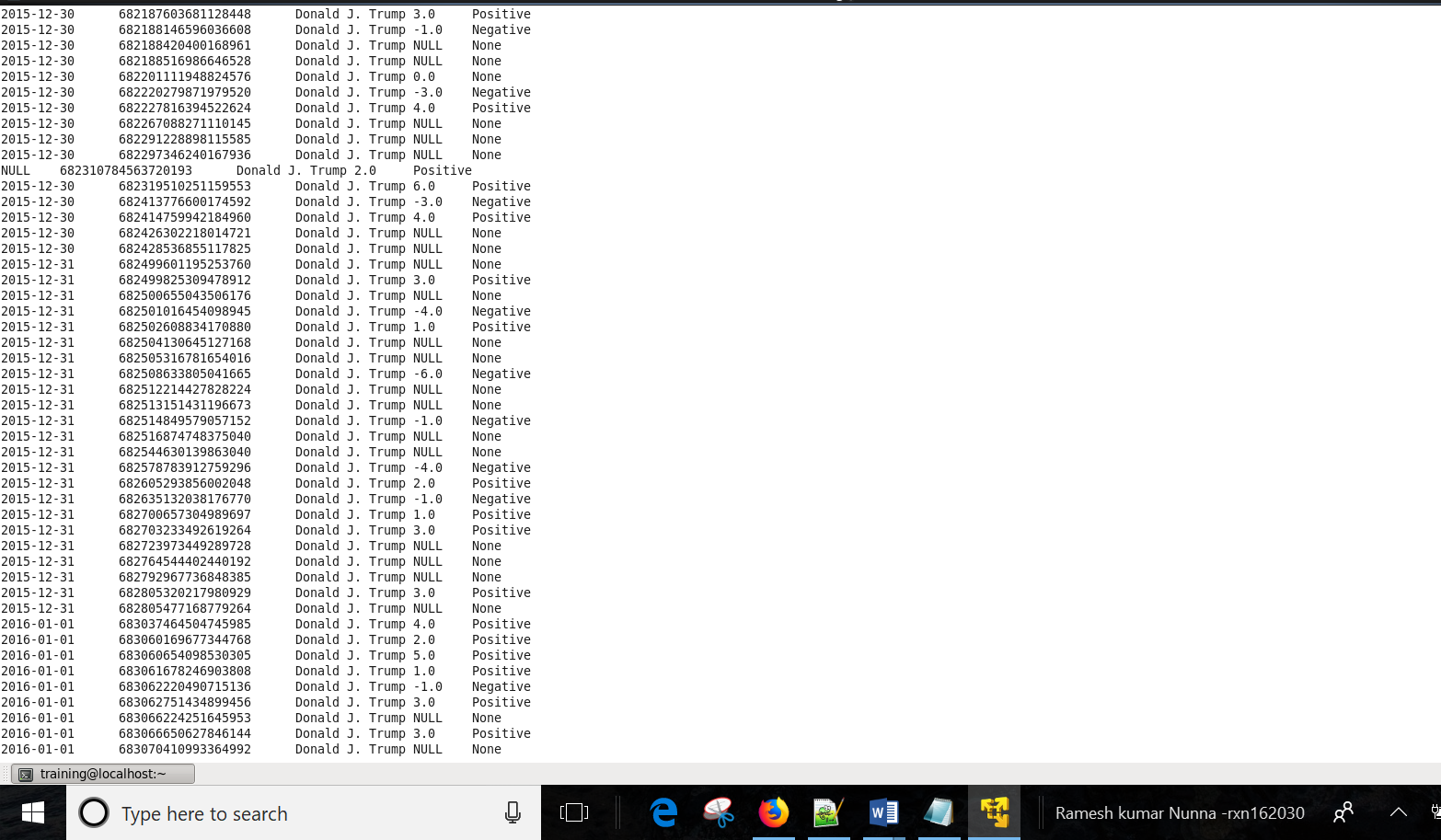


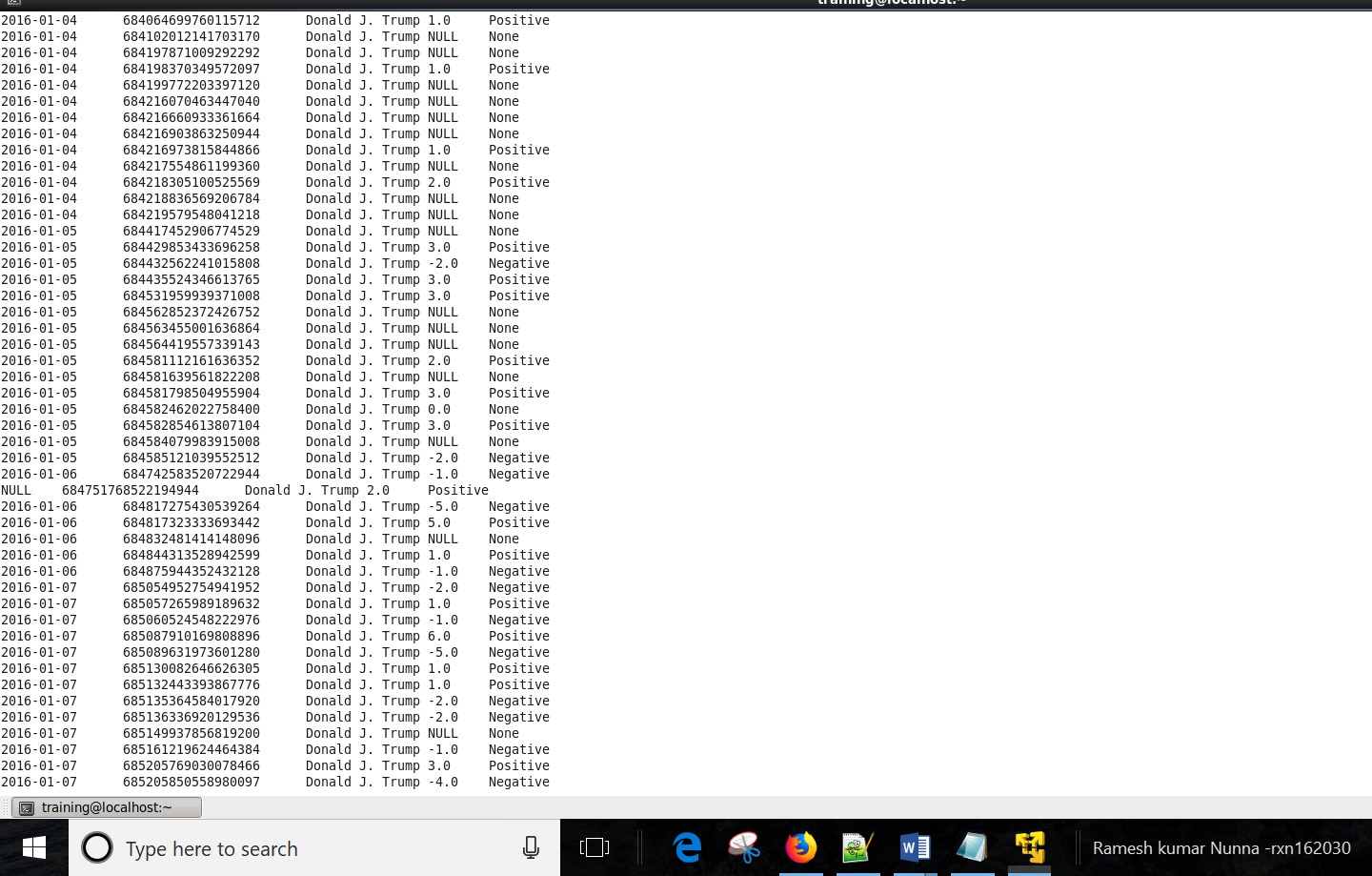
**Output: From total of 3207 tweets, the sentiment is shown for few tweets below due to space constraint.**

**Query:**

*select created\_date, id\_tweet,User\_name,polarity\_score,tweets\_sentiment\_analysis from tweets\_sentiment\_analysis;*







1. Do you find any problem in the way sentiment analysis was performed in the previous question? If so, how will you improve it?
2. **Problems:**
3. The analysis above is capturing sentiment based on individual words in the tweet and misses out on the overall context of the tweet (also words will have different meaning when used in phrases)
4. With sentiment analysis the tone of the writer cannot be properly analyzed whether the user tweeted in normal or in sarcastic tone.
5. The foremost problem I faced is lack of words in dictionary. Few tweets have no sentiment associated with it (null values) as they do not have corresponding words in the dictionary. Hence, the sentiment captured might not be accurate
6. The dictionary scores are very subjective. Words like compelled, bold are given positive ratings where as there can be negativity associated with it. Due to this the reason, sentiment of the tweet can be altered
7. No information on the rating scale is given (how the +5, -5 scores are decided)

**Improvements:**

1. By taking tweet context into account and using complex sentiment scoring algorithms, the process can be improved.
2. The dictionary scores should be assigned based on the connotative meaning of the word.
3. Dictionary should be made exhaustive.
4. The dictionary should be made dynamic as opposed to static nature. Dictionary scores should be updated as on the tweets generated. (based on how a word is used in the tweet context most of the times (+ve/-ve sentiment))