Principles of Big Data-SP 2016

Twitter Tweets Analysis using Apache Spark

Project Phase-2 Report

Submitted on : 7th April, 2016

<u>Submitted By:</u>

Justin Baraboo(jjbkb4)
Santhosh Kumar Gattu (sg6n6)
David Rodgers (daryt7)

Introduction:

The project was divided into 3 phases. In phase 1, the assigned task was to collect twitter tweets using any twitter streaming API and running word count program on the collected tweets using Apache Spark.

For phase 2, the assigned task was to store the collected tweets and writing at least 8 analytical queries using Spark SQL and visualize the query results using any tool/ technology. In this phase, we have collected tweets about different topics like #WT20, #SuperTuesday during the Iowa Primary, and tweets collected from Kansas City during the night (8PM to 4 AM mostly) using Java's Twitter4j and Python's Tweepy API's respectively.

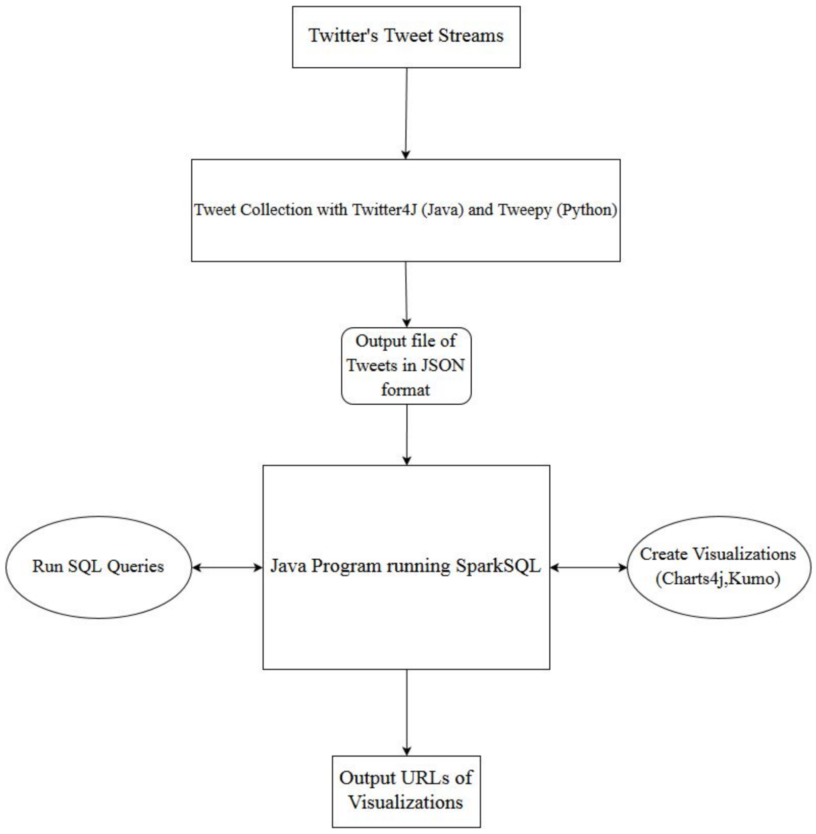
The collected tweets were stored in JSON format in the local machine. With the help of Eclipse and the Spark SQL context, analytical queries were written and executed using both Java and Scala programming languages. Our report and submission is limited to the Java code.

The visualizations were created with Charts4j and Kumo to create bar graphs, pie charts, and word cloud graphs. The graphics created from Charts4j are web queries that generate the image. The graphics created from Kumo are png files.

Technologies Used:

1. For collecting tweets: Java's Twitter4j and Python's Tweepy API.
2. For executing Spark SQL queries: Eclipse, IntelliJ
3. Languages: Java, Python, Scala,SQL
4. For visualization:
    - WordCloud graph - Kumo (open source Java library)
    - All other graphs - Charts4j (google's open source visualization Java library)

Architecture Diagram:


Twitter's Tweet Streams
Tweet Collection with Twitter4J (Java) and Tweepy (Python)
Output file of Tweets in JSON format
Run SQL Queries
Java Program running SparkSQL
Create Visualizations (Charts4j,Kumo)
Output URLs of Visualizations

Our architecture depicted graphically.

Queries in SQL form

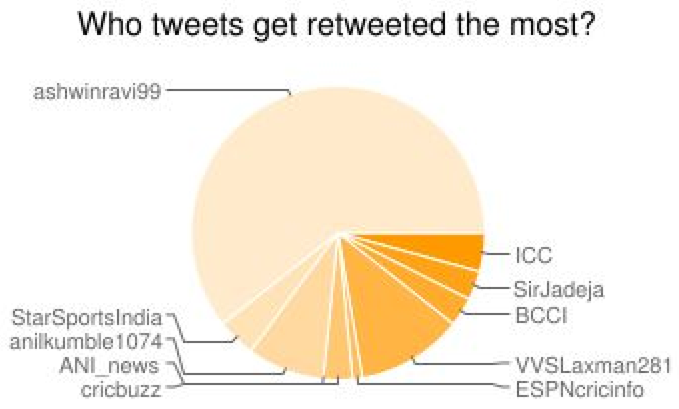
#1 QUERY:

```
DataFrame famous = sqlContext.sql("SELECT a.name, a.location, count(a.text) AS total_tweets,
sum(a.retweets) AS total_retweets "+
              "FROM (SELECT " +
                    " retweeted_status.user.screen_name as name,"+
                    " retweeted_status.user.location as location,"+
                    " retweeted_status.text as text,"+
                    " max(retweeted_status.retweet_count) as retweets"+
                    " FROM Tweets"+
                    " GROUP BY
retweeted_status.user.screen_name,retweeted_status.user.location,retweeted_status.text)a"+
                    " GROUP BY a.name, a.location"+
                    " ORDER BY total_retweets DESC LIMIT 10 ");
```

EXPLANATION: This query is designed to find the most famous person with the help of retweets and tweet count. The ratio of how many retweets you achieve against how many tweets you send out was utilized as the metric of being “famous.”

Visually, we used a pie chart to determine who, out of the top 10 retweeted people was the most "famous." The visualization is dynamic though, in that we could take in the top X and it would be able to scale the pie chart accordingly.

http://chart.apis.google.com/chart?cht=p&chs=500x200&chts=000000,16&chd=e:CuCDCDHfAsCDAAFdCumz&chtt=Who+tweets+get+retweeted+the+most%3F&chl=ICC|SirJadeja|BCCI|VVSLaxman281|ESPNcricinfo|ANI_news|cricbuzz|anilkumble1074|StarSportsIndia|ashwinravi99



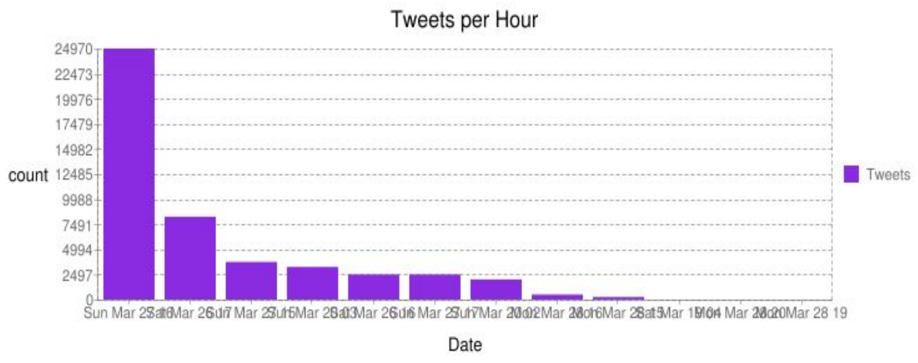
#2 QUERY:

```
DataFrame tweetsperhour = sqlContext.sql(" SELECT a.date, count(text) as cnt FROM"+
              " ( SELECT text, SUBSTRING(created_at,0,13) AS date"+
                      " FROM Tweets where text!= '')a"+
                      " GROUP BY a.date ORDER BY cnt desc");
```

EXPLANATION: This query is designed to find the number of tweets generated per hour on the day tweets were collected. By using this query we can determine during which hours the tweet count was peak.

This visualization looks at the raw count instead of a percentage count via the bar chart.

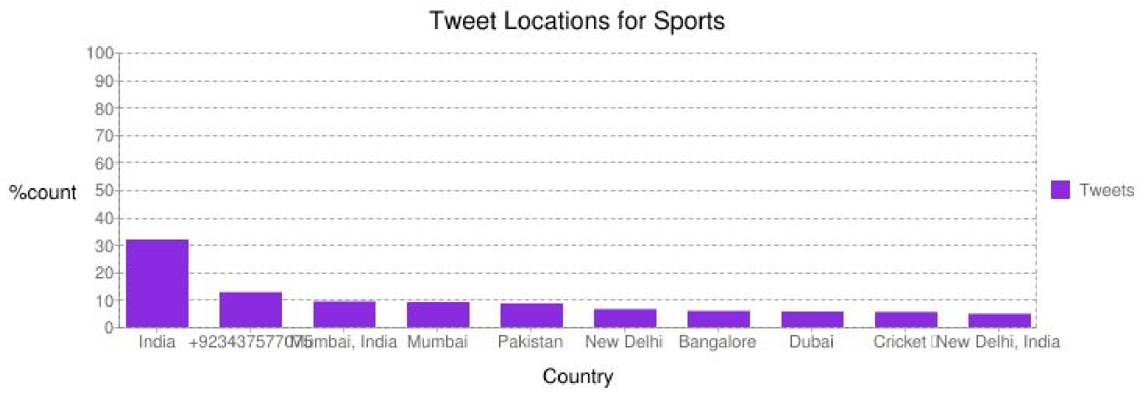
http://chart.apis.google.com/chart?cht=bvg&chco=8A2BE2&chdl=Tweets&chbh=41,20,8&chxt=y,y,x,x&chs=1000x250&chts=000000,16&chxp=1,50.0|3,50.0&chg=100.0,10.0,3,2&chtt=Tweets+per+Hour&chd=e:..VHJmIUGaGaFIBSApAAAAAA&chxs=1,000000,13,0|3,000000,13,0&chxr=0,0.0,24977.0,2497.0|1,0.0,100.0|3,0.0,100.0&chxl=1:|count|2:|Sun+Mar+27+16|Sat+Mar+26+17|Sun+Mar+27+15|Sun+Mar+20+03|Sat+Mar+26+16|Sun+Mar+27+17|Sun+Mar+20+02|Mon+Mar+28+16|Mon+Mar+28+15|Sat+Mar+19+04|Mon+Mar+28+20|Mon+Mar+28+19|3:|Date



#3 QUERY:

```
 DataFrame loclang = sqlContext.sql("SELECT user.location, user.lang, COUNT(text)cnt FROM "
                                + "SportTweets WHERE user.location IS NOT NULL GROUP BY user.location,
user.lang ORDER BY cnt desc limit 10"
                                  );
```

EXPLANATION: This query is designed to get the distribution of tweets across locations durintg WT20 and the user language respectively. The user language wasn't visualized as all queries came up English, but could have been resolved as a pie chart. The locations and tweet charts have been visualized with a bar chart below: http://chart.apis.google.com/chart?cht=bvs&chco=8A2BE2&chdl=Tweets&chs=800x250&chxt=y,y,x,x&chbh=40,20,8&chts=000000,16&chxp=1,50.0|3,50.0&chg=100.0,10.0,3,2&chd=e:UXIGF.F0FgEODyDoDiDH&chtt=Tweet+Locations+for+Sports&chxs=1,000000,13,0|3,000000,13,0&chxr=0,0.0,100.0|1,0.0,100.0|3,0.0,100.0&chxl=1:|%25count|2:|India|%2B923437577075|Mumbai%2C+India|Mumbai|Pakistan|New+Delhi|Bangalore|Dubai|Cricket+%E2%9C%94|New+Delhi%2C+India|3:|Country



#4 QUERY:

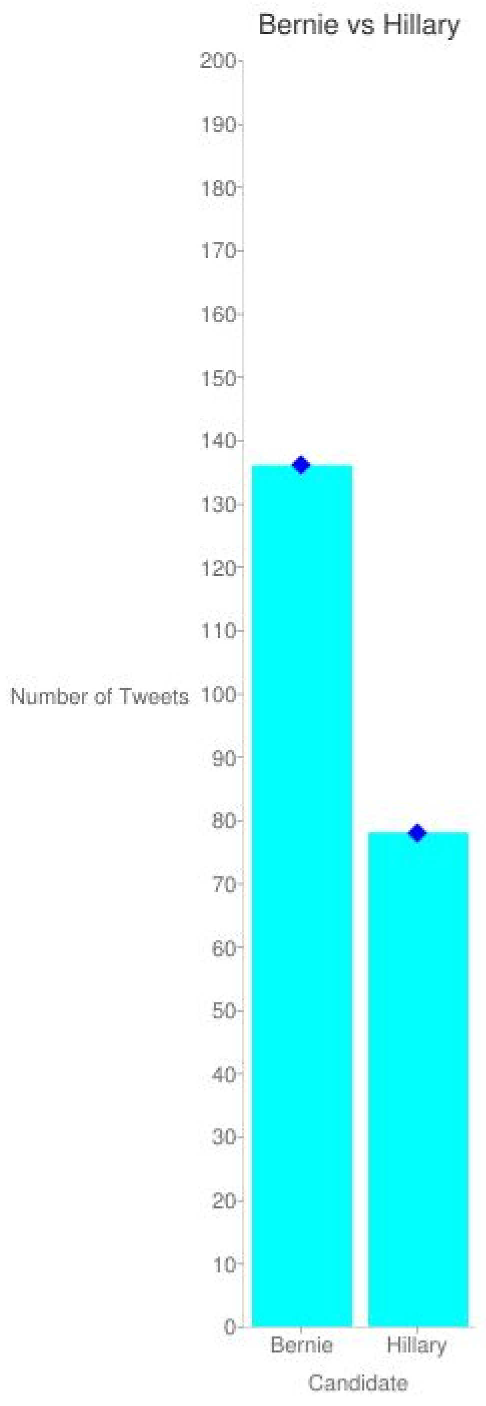
```
	DataFrame bern = sqlContext.sql("SELECT * FROM GoodTweets WHERE UPPER(text) LIKE
UPPER('%bern%')");

	DataFrame hillary = sqlContext.sql("SELECT * FROM GoodTweets WHERE UPPER(text) LIKE
UPPER('%hillary%')");
```

EXPLANATION:

This query is analyzing tweets to see which Democratic candidate (Bernie Sanders or Hillary Clinton) is tweeted about more. This query was chosen as the query itself was interesting as to which Democratic candidate was generating more online interest.

We chose a bar chart for this visualization as seeing the difference between the candidates is the easiest. A pie chart could have also been employed.

http://chart.apis.google.com/chart?cht=bvg&chd=e:rhY9&chco=00FFFF&chbh=50,4,8&chs=400x700&chxt=y,y,x,x&chxp=1,50.0|3,50.0&chtt=Bernie+vs+Hillary&chm=d,0000FF,0,-1,10,0&chxr=0,0.0,200.0,10.0|1,0.0,100.0|3,0.0,100.0&chxl=1:|Number+of+Tweets|2:|Bernie|Hillary|3:|Candidate


Bernie vs Hillary
Number of Tweets
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
Bernie
Hillary
Candidate

#5 QUERY:

```
DataFrame taco = sqlContext.sql("SELECT createdAt FROM GoodTweets WHERE UPPER(text) LIKE
UPPER('%taco%')"
                        + " OR UPPER(text) LIKE UPPER('%mexican%food') OR UPPER(text) LIKE
                        + "UPPER('%Chipotle%')");
DataFrame burger = sqlContext.sql("SELECT createdAt FROM GoodTweets WHERE UPPER(text) LIKE
UPPER('%burger%')"
                                + " OR UPPER(text) LIKE UPPER('%McDonalds%') OR UPPER(text) LIKE
                                + "UPPER('%Wend%')");
DataFrame pizza = sqlContext.sql("SELECT createdAt FROM GoodTweets WHERE UPPER(text) LIKE
                                + "UPPER('%pizza%')"
                                + " OR UPPER(text) LIKE UPPER('%Papa John%') OR UPPER(text) LIKE
                                + " UPPER('%Domino%')");
```

EXPLANATION:

This query was designed to look at late night food cravings for people in the Kansas City Area, primarily from 8PM to 3 AM.

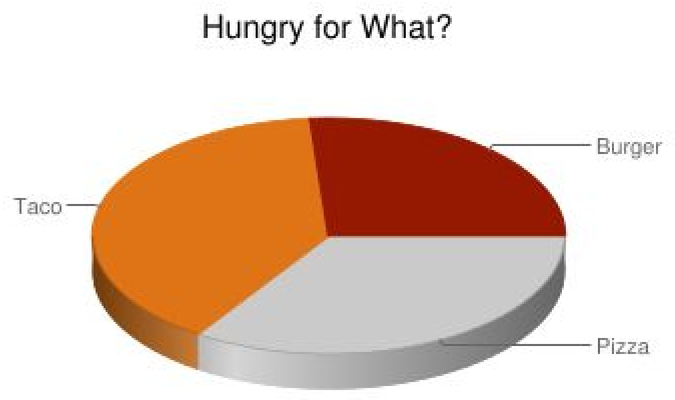
Computationally, this query was interesting to transform into usuable data as it required a union of the hours (so if Pizza was tweeted about at 1 AM but Burger and Taco weren't, they won't have a 1 AM count while Pizza does, so they would need to have a count of 0 put in for 1 AM). These was achieved by the following map reduction:

```
JavaPairRDD<String,Integer> hours = sqlContext.sql("SELECT SUBSTRING(createdAt, 12,2) as hour FROM
GoodNightTweets").distinct()
                          .toJavaRDD().mapToPair(row -> new
Tuple2<String,Integer>(row.getString(0),0));
List<Tuple2<String,Integer>> pizzaList = pizza.toJavaRDD().mapToPair(row -> new
Tuple2<String,Integer>(row.getString(0),1))
                                  .union(hours)
                                  .reduceByKey((int1,int2) -> int1 + int2).collect();
```

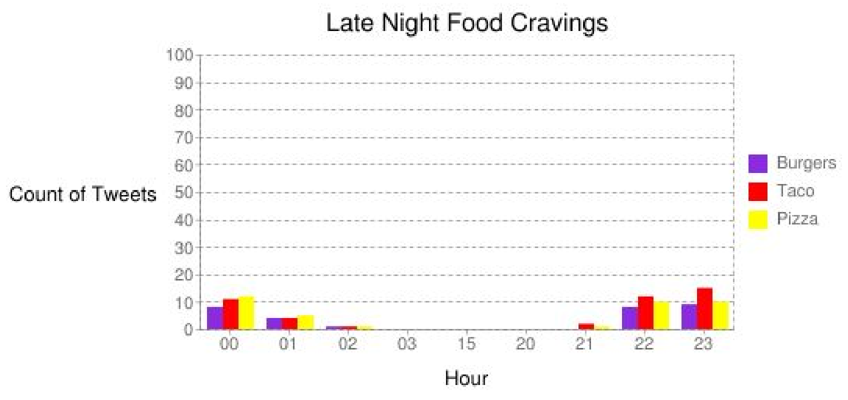
So while this query, on the surface, appears similar to an earlier query about tweets during an hour, it requires a much higher degree of processing to get it to that point.

The data was visualized with a pie chart to show the relative proportion of the overall tweets for Pizza, Tacos, and Burgers. Then a bar chart was employed to show the temporal relationship between the tweets and the categories. It's interesting to note that while Tacos beat out the other two overall, Pizza is most tweeted after midnight.

http://chart.apis.google.com/chart?cht=p3&chs=500x200&chd=e:V-ZNQz&chts=000000,16&chl=Pizza|Taco|Burger&chtt=Hungry+for+What%3F&chco=CACACA,DF7417,951800



http://chart.apis.google.com/chart?cht=bvg&chbh=10,0,8&chs=800x250&chxt=y,y,x,x&chts=000000,16&chxp=1,50.0|3,50.0&chg=100.0,10.0,3,2&chdl=Burgers|Taco|Pizza&chco=8A2BE2,FF0000,FFFF00&chtt=Late+Night+Food+Cravings&chxs=1,000000,13,0|3,000000,13,0&chxr=0,0.0,100.0|1,0.0,100.0|3,0.0,100.0&chxl=1:|Count+of+Tweets|2:|00|01|02|03|15|20|21|22|23|3:|Hour&chd=e:FlCkApAAAAAAAAFlFx,HCCkApAAAAAABSHrJm,HrDNApAAAAAAApGaGa



#6 QUERY:

```
 DataFrame lenTweet = sqlContext.sql("SELECT text, createdAt FROM GoodTweets");
         lenTweet.show();
```

Extra Java code to format output:

```
         JavaPairRDD<String,Integer> thing = lenTweet.javaRDD().mapToPair(
                         row -> new Tuple2<String, Integer>( row.getString(1).substring(11, 13),
row.getString(0).length())
                         );

         for(Tuple2<String,Integer> tuple : thing.reduceByKey( (num1, num2) -> num1 + num2 ).collect()
){
                 System.out.println(tuple._1() + "," + tuple._2());
         }

         Map<String, Object> count = thing.countByKey();
         ArrayList<String> hours = new ArrayList<String>();
         ArrayList<Double> averageTL = new ArrayList<Double>();
         for(Tuple2<String,Integer> tuple : thing.reduceByKey( (num1, num2) -> num1 + num2 ).collect()
){
                         System.out.println(tuple._1() + "," + tuple._2()/((Long)
count.get(tuple._1())).doubleValue() );
                         hours.add(tuple._1());
                         averageTL.add(tuple._2()/((Long) count.get(tuple._1())).doubleValue());
            }
```

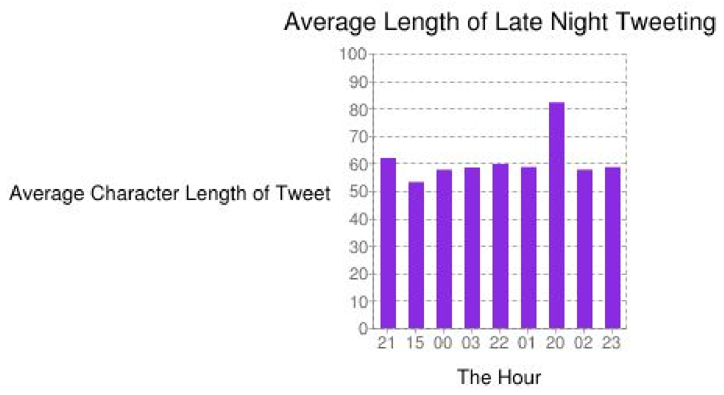
EXPLANATION:

This query is designed to grab the hour marker of the createdAt field and analyze the length of tweets (in the text field) as a function of the hour of the day (such as tweets collected during the night). We are finding the average length of a tweet for the hour.

This is computationally interesting as sparkSql doesn't have a len() function to determine the length of a tweet, so we make up for this by applying the length() function while we map reduce. We also have to actually compute the average by finding how many tweets occured during each hour.

We chose a bar chart for this visualization to express the temporal relationship of the hours with the average length of the tweets. It's interesting to note that 8 PM has a large spike of length.

http://chart.apis.google.com/chart?cht=bvg&chco=8A2BE2&chbh=10,0,8&chs=800x250&chxt=y,y,x,x&chts=000000,16&chxp=1,50.0|3,50.0&chg=100.0,10.0,3,2&chd=e:noiDk7IcmMIk0ok8Ij&chxs=1,000000,13,0|3,000000,13,0&chxr=0,0.0,100.0|1,0.0,100.0|3,0.0,100.0&chtt=Average+Length+of+Late+Night+Tweeting&chxl=1:|Average+Character+Length+of+Tweet|2:|21|15|00|03|22|01|20|02|23|3:|The+Hour



#7 QUERY:

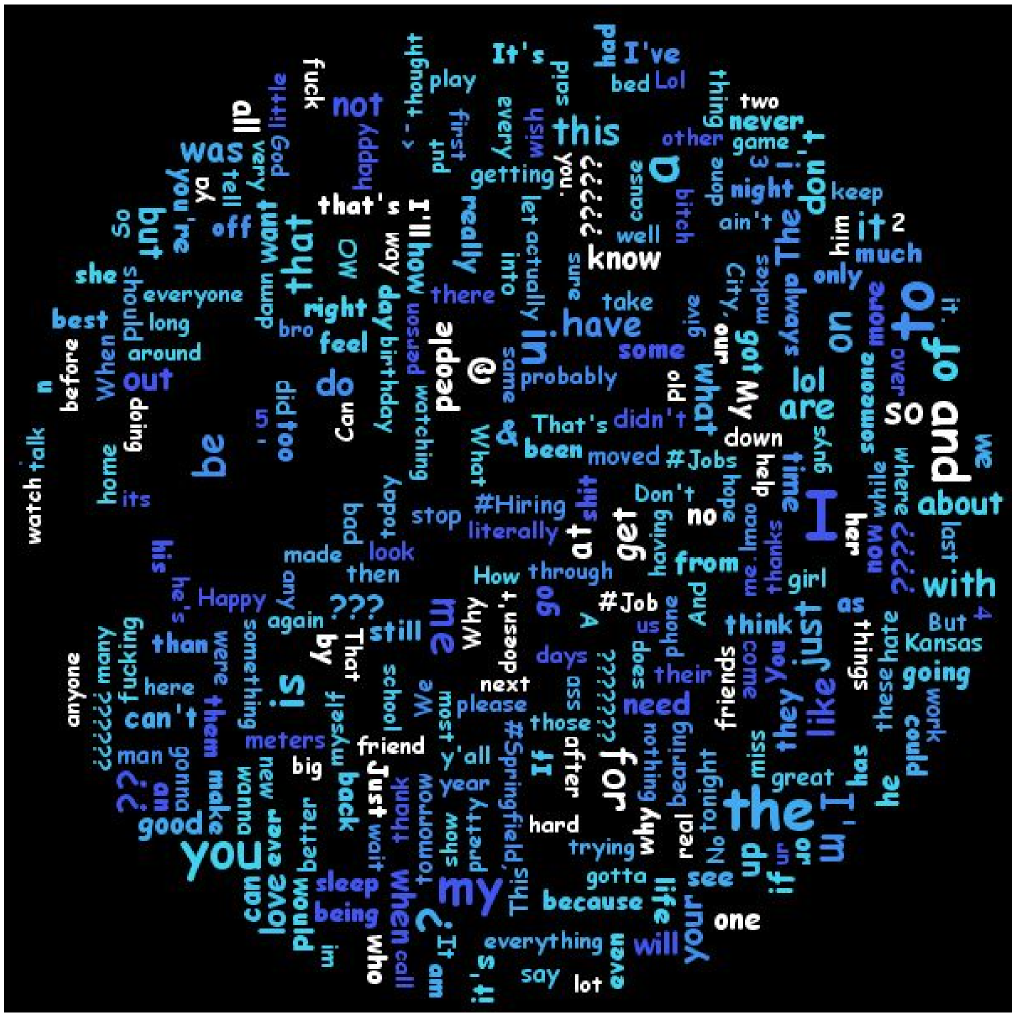
```
 DataFrame wordsMostTweeted = sqlContext.sql("SELECT text FROM GoodNightTweets");
```

EXPLANATION:

This looks at all tweets and then with this:

```
ArrayList<Tuple2<String, Integer>> words = new ArrayList<Tuple2<String,Integer>>( wordsMostTweeted.javaRDD().flatMap(
                                row -> Arrays.asList(row.get(0).toString().split(" "))
                                ).mapToPair(
                                                    word -> new Tuple2<String, Integer>( word, 1)
                                                    ).reduceByKey(
                                                                        (count1, count2) -> count1 + count2
                                                                        ).collect());



            Collections.sort(words,
                                new Comparator<Tuple2<String, Integer>>(){

                                                @Override
                                                public int compare(Tuple2<String, Integer> o1, Tuple2<String, Integer> o2) {
                                                          // TODO Auto-generated method stub
                                                          return o1._2() > o2._2() ? -1 : o1._2() < o2._2() ? 1 : 0;
                                                }
                                      }
      );
```

The map reduction performs a word count on the text from the tweets but then needs an extra step of sorting them so we can graph the most frequent words.

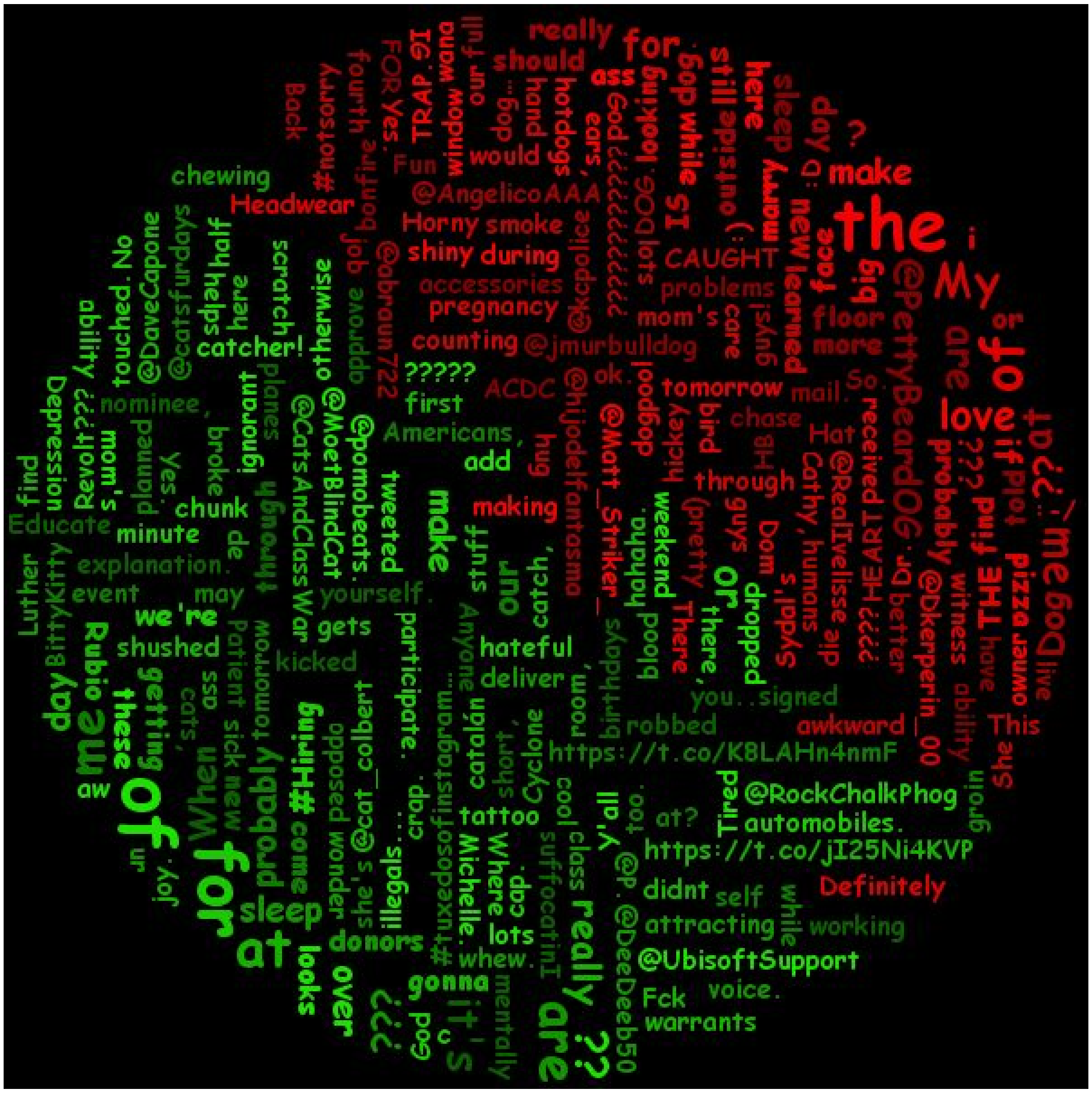


the
I
to
and
of
you
you
my
me
is
be
that
this
in
have
a
for
get
at
it
on
are
so
lol
with
about
like
just
they
not
was
all
but
that's
know
people
@
do
out
feel
right
want
off
she
best
everyone
long
around
before
When
should
doing
watch
talk
home
its
his
he's
Happy
than
can't
them
make
good
can
love
ever
would
sleep
being
who
when
it's
am
call
everything
say
lot
even
will
your
one
see
life
because
gotta
trying
hard
why
real
No
tonight
up
if
I'm
has
he
could
work
going
Kansas
But
these
hate
things
as
think
friends
come
great
miss
need
nothing
bearing
their
phone
us
does
#Job
A
days
ass
after
If
those
please
next
doesn't
Why
How
through
from
girl
thanks
me.
lmao
her
now
while
where
last
we
guys
time
help
hope
down
Don't
having
no
shit
#Hiring
literally
stop
today
look
then
made
bad
any
again
???
still
by
That
school
We
most
y'all
year
pretty
This
#Springfield
friend
Just
myself
big
meters
new
wanna
gonna
man
here
fucking
anyone
many
were
something
didn't
That's
been
moved
#Jobs
&
What
watching
same
probably
birthday
day
Can
person
way
I'll
how
really
there
into
actually
let
sure
take
some
old
give
our
what
My
got
makes
City,
always
The
don't
night
ain't
keep
him
2
much
only
more
someone
over
it.
game
never
two
thing
done
bitch
other
bed
Lol
I've
had
said
It's
play
thought
first
put
every
wish
getting
you.
cause
well
fuck
little
God
very
tell
ya
So
happy
damn
bro
did
too

#8 QUERY:
```
 DataFrame catTweeted = sqlContext.sql("SELECT text FROM GoodNightTweets WHERE UPPER(text) LIKE UPPER('%cat%')");
            DataFrame dogTweeted = sqlContext.sql("SELECT text FROM GoodNightTweets WHERE UPPER(text) LIKE UPPER('%dog%')");

            ArrayList<Tuple2<String, Integer>> words1 = new ArrayList<Tuple2<String,Integer>>( catTweeted.javaRDD().flatMap(
                    row -> Arrays.asList(row.get(0).toString().split(" "))
                    ).mapToPair(
                            word -> new Tuple2<String, Integer>( word, 1)
                            ).reduceByKey(
                                    (count1, count2) -> count1 + count2
                                    ).collect());

            ArrayList<Tuple2<String, Integer>> words2 = new ArrayList<Tuple2<String,Integer>>( dogTweeted.javaRDD().flatMap(
                    row -> Arrays.asList(row.get(0).toString().split(" "))
                    ).mapToPair(
                            word -> new Tuple2<String, Integer>( word, 1)
                            ).reduceByKey(
                                    (count1, count2) -> count1 + count2
                                    ).collect());
```

EXPLANATION:
This query counts the text of two sets of tweets and outputs the most frequent words within those tweets.

This query is fairly similar to the last, but we wanted to showcase how the visualization could display two different sets as being different.



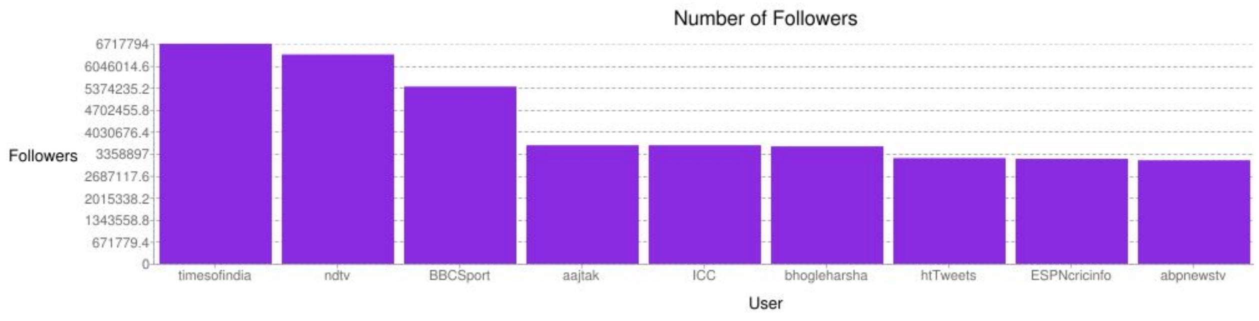
the
My
of
love
are
if
make
@PettyBeardOG
face
big
floor
more
really
for
should
ass
still
here
sleep
Headwear
chewing
Horny
smoke
shiny
during
accessories
pregnancy
counting
@jmurbulldog
@AngelicoAAA
CAUGHT
problems
mom's
tomorrow
through
making
first
add
Americans,
?????
ACDC
ok.
dogpool
chase
Hat
mail.
So.
probably
pizza
owner
THE
witness
better
Definitely
awkward
This
She
@RockChalkPhog
automobiles.
https://t.co/jI25Ni4KVP
https://t.co/K8LAHn4nmF
@UbisoftSupport
attracting
working
voice.
warrants
Fck
didnt
self
tattoo
lots
hateful
deliver
robbed
you...signed
stuff
catch,
our
make
tweeted
yourself.
gets
kicked
we're
shushed
event
explanation.
minute
chunk
Educate
Luther
find
nominee,
catcher!
@DaveCapone
@catsfurdays
touched. No
ability
Depression
Revolt???
mom's
planned
ignorant
otherwise
@MoetBlindCat
@CatsAndClassWar
@pomobeats.
day
me
of
for
at
sleep
these
getting
When
probably
#Hiring
come
looks
over
donors
gonna
really
are
Where
Michelle.
Cyclone
class
y'all
@DeeDee50

#9 QUERY:

```
DataFrame followers = sqlContext.sql("SELECT user.screen_name, max(user.followers_count) FROM tweets
WHERE user.screen_name is NOT NULL GROUP BY user.screen_name ORDER BY max(user.followers_count) DESC
limit 10")
```

EXPLANATION: This query is designed to display top ten user screen names which is displayed on the user account and respective maximum followers count. (as some of our queries may look similar on what they grab from the database, we made another query).

http://chart.apis.google.com/chart?cht=bvg&chco=8A2BE2&chbh=90,0,8&chxt=y,y,x,x&chs=1000x250&chts=000000,16&chxp=1,50.0|3,50.0&chg=100.0,10.0,3,2&chtt=Number+of+Followers&chd=e:..84ziieieiJeveheHZ3&chxs=1,000000,13,0|3,000000,13,0&chxr=0,0.0,6717794.0,671779.4|1,0.0,100.0|3,0.0,100.0&chxl=1:|Followers|2:|timesofindia|ndtv|BBCSport|aajtak|ICC|bhogleharsha|htTweets|ESPNcricinfo|abpnewstv|ayushmannk|3:|User



Google drive/ dropbox link for collected tweets:
https://drive.google.com/drive/u/0/folders/0B3KtUUlhHoLfLWJfT1F6REZRLTA - tweets collected in Python
https://drive.google.com/drive/u/0/folders/0ByvYLh8H7jyLWXZFbVl3dGtMYmc - tweets collected in Java

Github Link which has entire project's resources: https://github.com/DRinKC/bigdatabigtweets

References:

1. **http://stats.seandolinar.com/collecting-twitter-data-using-a-python-stream-listener/**
2. **https://developer.ibm.com/hadoop/blog/2015/12/07/configuring-intellij-for-spark/**
3. **http://www.jsoneditoronline.org/**
4. **https://github.com/julienchastang/charts4j for Charts4j**
5. **https://github.com/kennycason/kumo for Kumo**
6. **http://twitter4j.org/en/code-examples.html for Twitter4J**