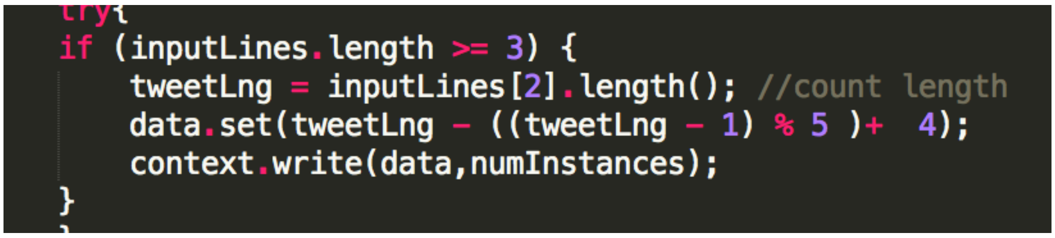
COURSEWORK 1 - TWITTER ANALYSIS WITH MAP REDUCE

Part A – Content Analysis

A.A - Histogram.

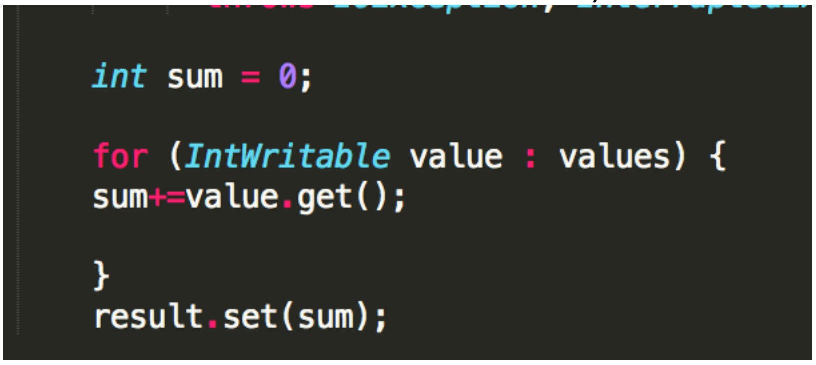
Approach

To aggregate the values into bins of five the approach were to use the modulus operator. This simple line of code aggregates on the actual length of each tweet then subtracts the modulus of the tweet from the tweet length and the adds four to make into a number dividable on five. Then the code writes the current tweet length bin and adds an increment of one.



```
try{
if (inputLines.length >= 3) {
    tweetLng = inputLines[2].length(); //count length
    data.set(tweetLng - ((tweetLng - 1) % 5 )+  4);
    context.write(data,numInstances);
}
```

The reducer adds each value based on the key which is the bin size.



```
int sum = 0;

for (IntWritable value : values) {
sum+=value.get();

}
result.set(sum);
```

Results

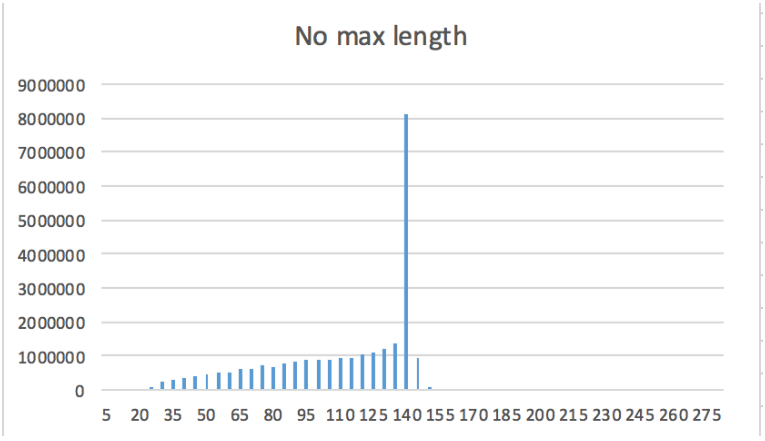
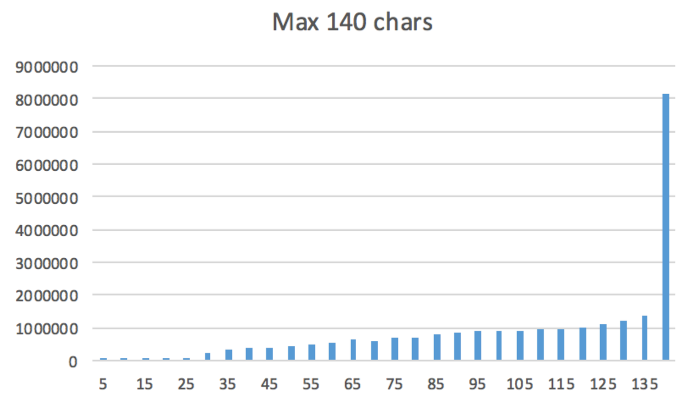


Figure 1 Without filtering out non-english characters

When plotting the result without filtering on non-English characters the values exceed the maximum tweet length. Unfortunately, characters such as emojis, 'æøå' and others often does not play well with the charset which resulted for some tweets to exceed the maximum length of a tweet. Additionally, Twitter treats links as maximum 23 characters regardless of length (Twitter , 2016). That means that in the dataset a tweet might contain a link that makes the actual character count to exceed the 140 characters' limit.

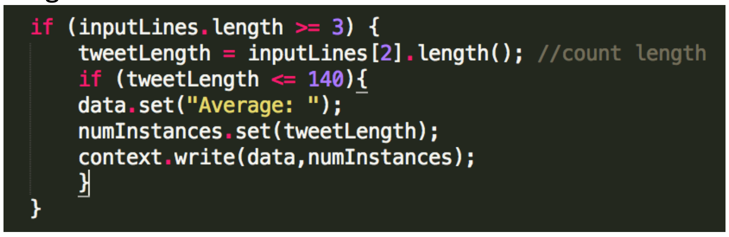
When only plotting data with less than 140 characters (The max tweet-length at the time of the Rio Olympics) the results were more realistic. This could also have been fixed directly in Hadoop by cleaning each tweet from a non-English character with java regular expression which might have resulted in slightly different numbers, but most likely a similar distribution. On the other hand, by cleaning each tweet information such as emoji's would be lost and the actual tweet lengths would, therefore, be shorter than what the case is. When just leaving out tweets with values above the 140 char limit we get a more plausible distribution.



A.B - Average length

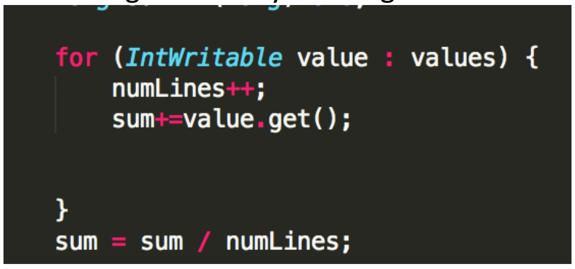
Approach

The mapper gets the length of the current tweet, and then write it to the context with the key "Average: ". By using the same key the job in the reducer will be a simple divide job and leave the combination job for the reducer. When calculating the average all tweets that are longer than 140 chars are discarded so that tweets with character encoding gets passed on.



```
if (inputLines.length >= 3) {
    tweetLength = inputLines[2].length(); //count length
    if (tweetLength <= 140){
    data.set("Average: ");
    numInstances.set(tweetLength);
    context.write(data,numInstances);
    }
}
```

The container takes all values, sums them together and keep track of how many lines/tweets there are. Then a simple operation to get the average is done, and the result is outputted by combining on the key "Average: ".



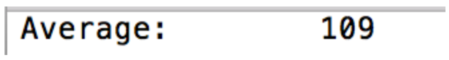
```
for (IntWritable value : values) {
    numLines++;
    sum+=value.get();


}
sum = sum / numLines;
```

The final result is then outputted with the key and the value. The key “Average: ” were chosen due to better readability in the final output.

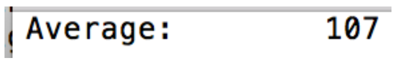
Results

The final result of 107 is lower than the maximum tweet length of 140, and thus a sensible result. When allowing tweets with more than 140 character the result was 109 which isn’t significally higher.



```
Average:        109
```

Figure 3 No maximum tweeth length limitation



```
Average:        107
```

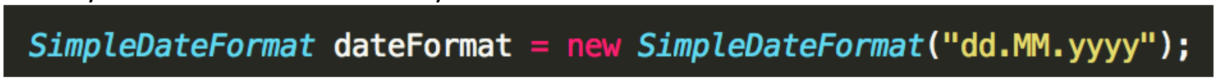
Figure 2 Figure 3 Maximum tweet length of 140

Part B – Time Analysis

A - Time Series Diagram

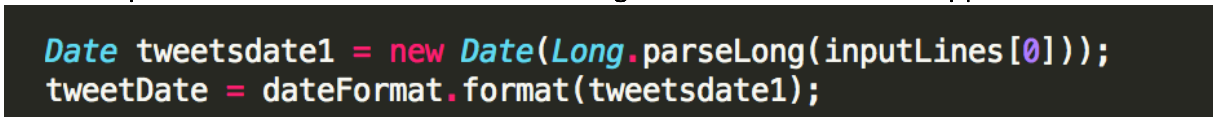
Approach

To create a time series, the timestamp from each tweet had to be extracted. When this was done the actual date format had to be transformed into a data format that could be used as the key. The clue here was to only use date and leave out hours and minutes.



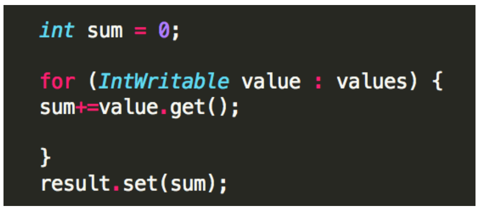
```java
SimpleDateFormat dateFormat = new SimpleDateFormat("dd.MM.yyyy");
```

By using the simple date format the custom date could be created. Thereafter the actual timestamp from the tweet is extracted as a long and the date format is applied.



```java
Date tweetsdate1 = new Date(Long.parseLong(inputLines[0]));
tweetDate = dateFormat.format(tweetsdate1);
```

The date is now written to the context as the key, together with a count of one. Thereafter the reducer combines all the values with the same date and increments the count by one.

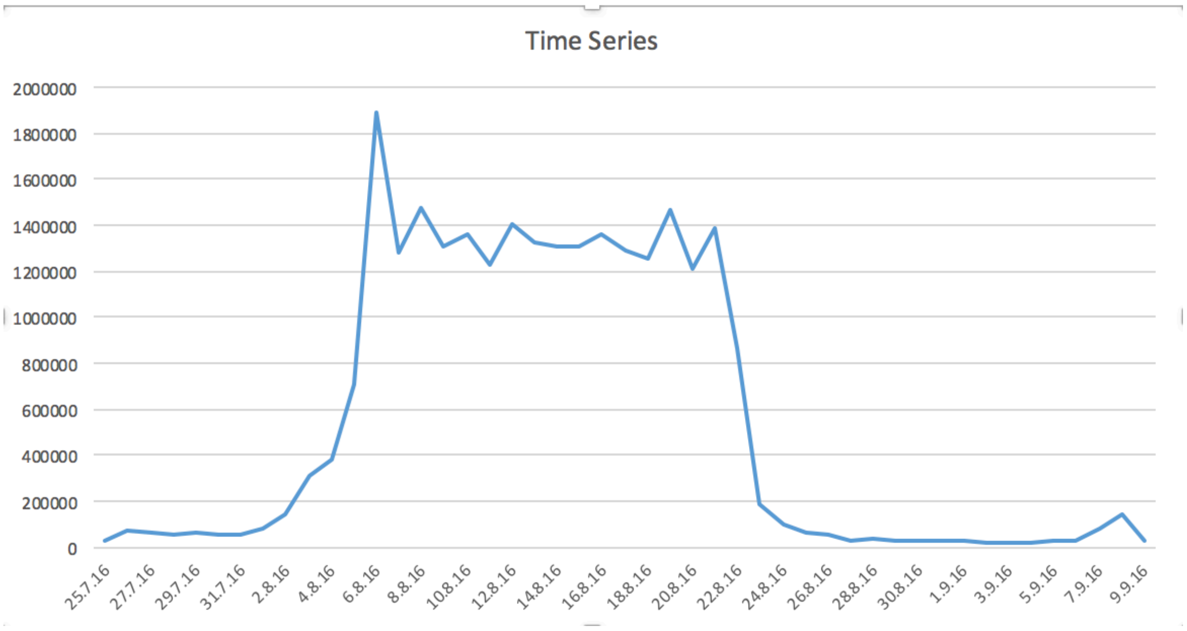


```java
int sum = 0;

for (IntWritable value : values) {
sum+=value.get();

}
result.set(sum);
```

Results



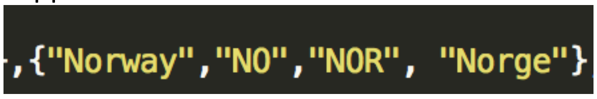
The Time-series shows that there was more activity during the initial days before the opening ceremony of 5th of august 2016 and around the final day the 21st august 2016 (Wikipedia, 2016). Right before, and shortly after the peaks were lower than during the actual games. This fits well whit what one would expect, and based on intuition the time-series graphs seem to illustrate a possible scenario of tweet activity.

Part C – Hashtag Analysis

A – Getting number of tweets supporting each country

Approach

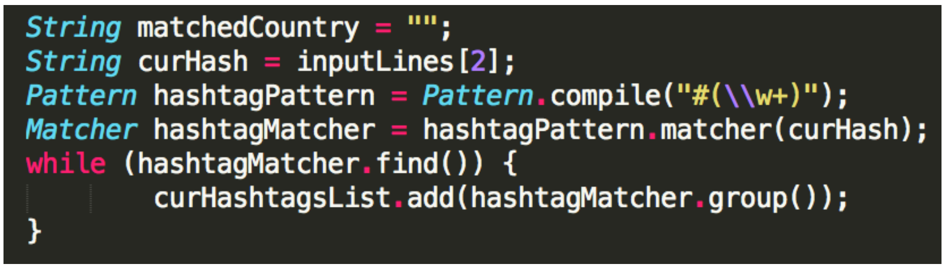
To estimate what countries got the highest support the approach was first to create a huge multidimensional array of all counties in the world, and their three letter codes. The list was fetched from https://countrycode.org and cleaned in Excel to form an array.
The array can then be extended to contain common ways of writing the country, additional ways can be added per request as the code is highly dynamical. For example, for tweets supporting Norway, a common hashtag would be the local translation of the country name e.g. “Norge” which very simply can be added to the array. Additionally, after the first exception some nations three letter code had to be removed as they were the same as general every day word which resulted in some very small nations getting a high number of support.



```
,{"Norway","NO","NOR", "Norge"}
```

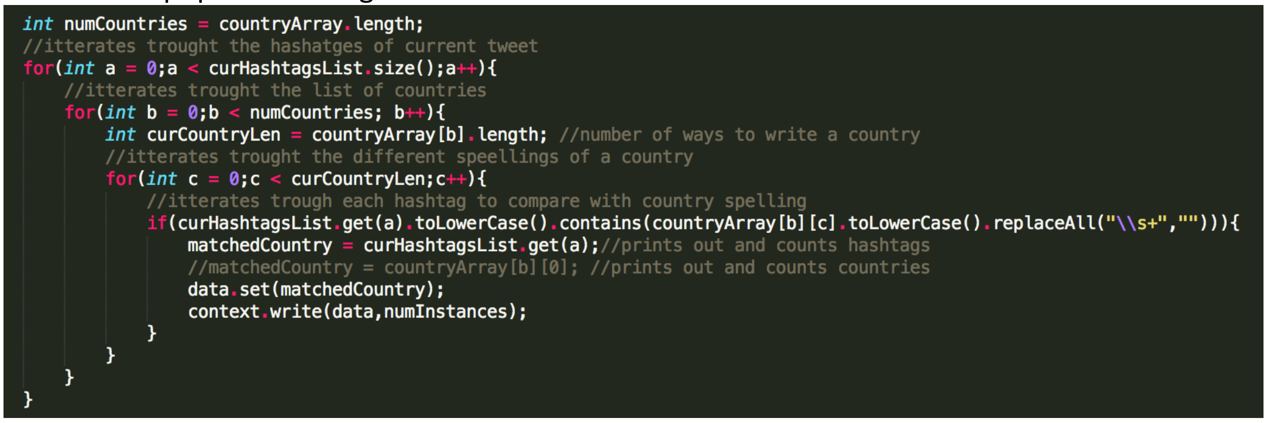
Figure 4 Example of adding addtionally words to a country

The mapper fetches all tweets and their hashtags to compare them against the country array. The first part of the array splits the input e.g. the tweet and extracts the actual tweet text. Then a list is created for all words that start with an ‘#’.



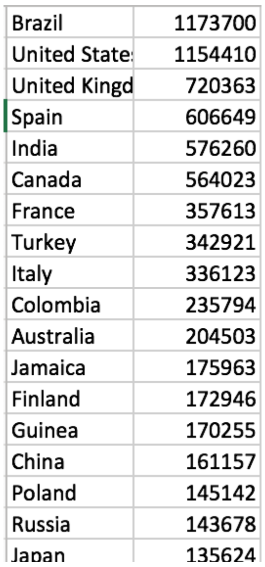
```java
String matchedCountry = "";
String curHash = inputLines[2];
Pattern hashtagPattern = Pattern.compile("#(\\w+)");
Matcher hashtagMatcher = hashtagPattern.matcher(curHash);
while (hashtagMatcher.find()) {
        curHashtagsList.add(hashtagMatcher.group());
}
```

The next step is a nested for-loop that first takes the number of hashtags in the current tweet, then it fetches the number of countries in the array. The next step is to get each way the current country is spelt in the array. The current tweet is then compared against the current way of spelling a country. If there is a match the name of the country and a count of one is written to the context. Inside the final, if the "matchedCountry" string has two different options, it can output either hashtag and the count of each hashtags occurrence, or each country and its occurrence. This makes it simple to find the most supported countries and the most popular hashtags.



```java
int numCountries = countryArray.length;
//itterates trought the hashatges of current tweet
for(int a = 0;a < curHashtagsList.size();a++){
    //itterates trought the list of countries
    for(int b = 0;b < numCountries; b++){
        int curCountryLen = countryArray[b].length; //number of ways to write a country
        //itterates trought the different speellings of a country
        for(int c = 0;c < curCountryLen;c++){
            //itterates trough each hashtag to compare with country spelling
            if(curHashtagsList.get(a).toLowerCase().contains(countryArray[b][c].toLowerCase().replaceAll("\\s+",""))){
                matchedCountry = curHashtagsList.get(a);//prints out and counts hashtags
                //matchedCountry = countryArray[b][0]; //prints out and counts countries
                data.set(matchedCountry);
                context.write(data,numInstances);
            }
        }
    }
}
```

Results



Brazil	1173700
United State	1154410
United Kingd	720363
Spain	606649
India	576260
Canada	564023
France	357613
Turkey	342921
Italy	336123
Colombia	235794
Australia	204503
Jamaica	175963
Finland	172946
Guinea	170255
China	161157
Poland	145142
Russia	143678
Japan	135624

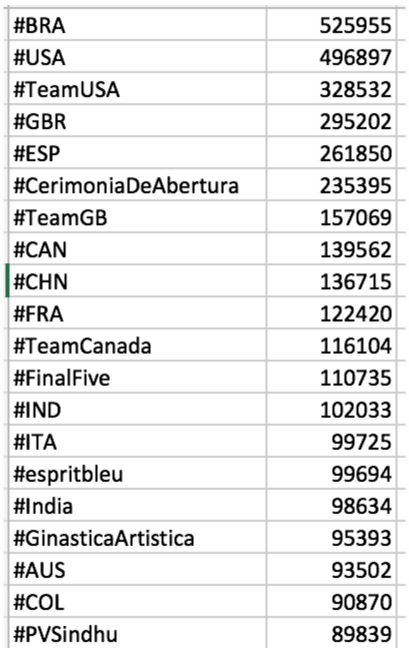
Figure 5 Selection of the 20 most supported countries

The most supported countries

The final result sorted in Excel shows that Brazil was the most supported country, which makes sense as they were the host. The second most supported country was the US and thereafter the UK. The screenshot shows the twenty most supported countries, but in total, there are 240 countries in the final result, which is more countries than the 206 that actually participated (Wikipedia, 2016) in the games. This has happened as some countries might match with a tweet and thus they get incremented. To avoid this all non-participating countries will need to be removed from the array, but in the final result such countries did not make it to the top list and a manual cleaning was not necessary as they do not affect the most supported countries.

The most popular hashtags

When sorting hashtags by their occurrences using excel the four top hashtags are #BRA, #USA, #TeamUSA and #GBR which matches well with the most supported countries.



#BRA	525955
#USA	496897
#TeamUSA	328532
#GBR	295202
#ESP	261850
#CerimoniaDeAbertura	235395
#TeamGB	157069
#CAN	139562
#CHN	136715
#FRA	122420
#TeamCanada	116104
#FinalFive	110735
#IND	102033
#ITA	99725
#espritbleu	99694
#India	98634
#GinasticaArtistica	95393
#AUS	93502
#COL	90870
#PVSindhu	89839

Figure 6 Selection of the 20 most popular hashtags

Bibliography

Twitter , 2016. *Twitter.* [Online]
Available at: https://blog.twitter.com/2016/doing-more-with-140-characters
[Accessed 08 11 2016].
Wikipedia, 2016. *2016 Summer Olympics.* [Online]
Available at: https://en.wikipedia.org/wiki/2016_Summer_Olympics
[Accessed 8 11 2016].