|  |  |  |
| --- | --- | --- |
| **UMBC – CSEE Department** |  | Dr. Waleed Youssef |
| **Data Science Program** |  | youssef1@umbc.edu |
| **Spring 2023** |  |  |
| **DATA 603 – Big Data Platforms** | | |
|  |  |  |
| **Homework #10 – Spark Streaming** | | |
|  |  |  |

1. **[10 Points]** Describe Kafka? Provide its benefits and usage? What are its advantages and disadvantages in the BigData framework?

It is an Apache project initially developed by LinkedIn. It is an open-source stream-processing software platform written in Scala and Java.

**Benefits of Kafka:**

* **Scalability:** Kafka is designed to scale to handle large volumes of data. It can be easily scaled up or down to meet the needs of the application.
* **Reliability:** Kafka is highly reliable and fault-tolerant. It can continue to operate even if some of the nodes in the cluster fail.
* **Durability:** Kafka is designed to store data on disk, making it durable and ensuring that data is not lost in case of node failures**.**
* **Flexibility:** Kafka is very flexible and can be used for a variety of purposes. It can be used for real-time data processing, analytics, and logging**.**

**Usage Of Kafka:**

* **Real-time data processing**: Kafka can be used to process real-time data streams. This can be useful for applications such as fraud detection, real-time analytics, and event streaming.
* **Analytics**: Kafka can be used to store and analyze large amounts of data. This can be useful for applications such as business intelligence, machine learning, and data warehousing.
* **Logging**: Kafka can be used to collect and store logs from applications and systems. This can be useful for troubleshooting, auditing, and compliance.

**Advantages of Kafka:**

* **Processing Speed:**Kafka implements a data processing system with brokers, topics, and APIs that outperforms both SQL and NoSQL database storage with horizontal scalability of hardware resources in multi-node clusters that can be positioned across multiple data center locations.
* **Scalability**: Kafka is designed to scale to handle large volumes of data. This makes it a good choice for big data applications that need to process and analyze large amounts of data.
* **Pre-Built Integrators**:Kafka Connect offers more than 120 pre-built connectors from open source developers, partners, and ecosystem companies. Examples include integration with Amazon S3 storage, Google BigQuery, ElasticSearch, MongoDB, Redis, Azure Cosmos DB, AEP, SAP, Splunk, and DataDog. Programming teams can use the connector resources of Kafka Connect to accelerate application development with support for organizational requirements.
* **Real-time processing**: Kafka enables real-time processing of data, which is important for applications that require real-time data processing.

**Disadvantages Of Kafka:**

* **Do not have complete set of monitoring tools:** Apache Kafka does not contain a complete set of monitoring as well as managing tools. Thus, new startups or enterprises fear to work with Kafka.
* **Reduces Performance**: Brokers and consumers reduce the performance of Kafka by compressing and decompressing the data flow. This not only affects its performance but also affects its throughput.
* **Clumsy Behaviour**: Apache Kafka most often behaves a bit clumsy when the number of queues increases in the Kafka Cluster.

**Reference:** <https://www.altexsoft.com/blog/apache-kafka-pros-cons/>

<https://www.javatpoint.com/apache-kafka-advantages-and-disadvantages>

[Kafka Benefits and Use Cases (confluent.io)](https://www.confluent.io/learn/apache-kafka-benefits-and-use-cases/)

1. **[10 points]** Explain how does Spark Streaming works? Draw an architectural diagram showing your answer.

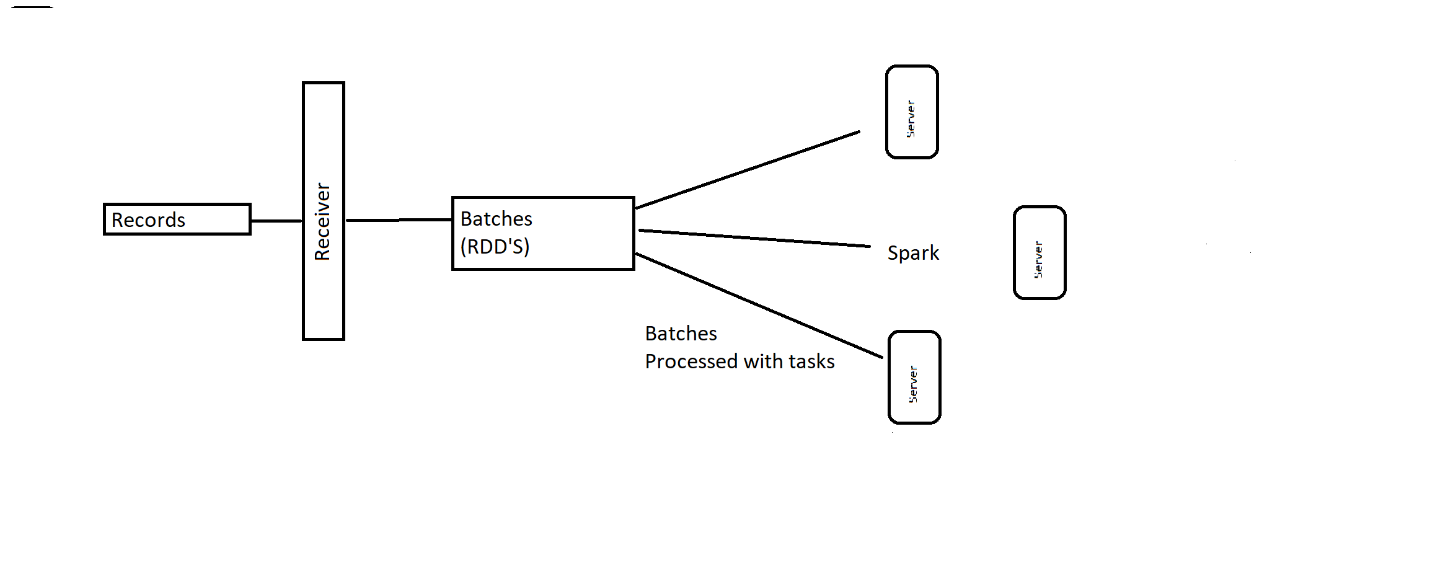
Spark Streaming was launched in 2013 to enable data engineers and data scientists to process real-time data from SQL databases, Flume, Amazon Kinesis, etc. Discretized Streams, or DStreams, are fundamental abstractions here, as they represent streams of data divided into small chunks or batches. DStreams can be created either from input data streams from sources such as Kafka, and Kinesis, or by applying high-level operations on other DStreams. Internally, a DStream is represented as a sequence of RDDs.

Data is ingested from various sources (e.g. Kafka, Flume, Twitter) into Spark Streaming. The data is divided into small batches, with each batch representing a small interval of time (e.g. a few seconds).

The data is then processed by Spark Streaming using high-level operations such as map, reduce, filter, join, and window(Actions and Transformations). These operations are applied to each batch of data in parallel, using the same programming abstractions as batch processing in Spark.

The processed data can then be output to various destinations (e.g. HDFS, databases, dashboards) in real-time or near real-time.

**Architecture:**



Spark Streaming discretizes the data streaming data into tiny, sub-second micro batches called DStreams. Spark Sreaming Receivers accept the data in parallel and buffer it in memory of Spark’s workers nodes. Then the latency optimized Spark engine runs short tasks to process the batches and output the result to other systems. Spark tasks are assignes dynamically to workers bsed in the locality of the data and available resources. Thus enabling better load balancing and faster fault recovery.

1. **[10 Points]** Describe the difference between Amazon EMR and Cloudera?

|  |  |  |
| --- | --- | --- |
|  | **AWS EMR** | **Cloudera** |
| Auto Scaling | EMR segregates slave nodes into two subtypes – Core Nodes and Task nodes. This results in high scalability and low cost by using the spot instance for task node. | Cloudera does not categorize slave nodes into core and task nodes. So if a node is removed/lost then there is increase the risk of losing HDFS data. |
| Access to Amazon S3 | You can access data on S3 from EMR directly or through Hive Tables. EMR is highly tuned for working with data on S3. | Cloudera uses Apache libraries (s3a) to access data on S3 .But EMR uses AWS proprietary code to have faster access to S3. |
| Availablity | EMR Service monitors the slave nodes and replaces any unhealthy node with a new node. | Unlike EMR, Cloudera does not categorize slave nodes into core and task nodes. This increases the risk of losing HDFS data in case a node is removed/lost. |
| Ease of Use | AWS manages EMR Hadoop service as well as underlying AWS infrastructure. So you can quickly start a new Hadoop cluster quickly and start processing the data. | Cloudera is comparatively more difficult to learn and configure.But once you have it setup, it’s far more flexible than EMR, and there’s no extra infrastructure cost. Cloudera Manager has an easy to use web GUI. This helps manage and monitor Hadoop services, cluster, and physical host hardware. |
| Hadoop Management Console | AWS does not provide any management console like Apache’s Ambari or Cloudera Manager, for EMR. This makes it difficult to manage and track various Hadoop services on a running cluster. | Cloudera also provides Cloudera Director to enable self-service for using CDH in the cloud. It provides an administration experience for central IT to reduce costs and deliver agility. There is interface for end-users provisioning and scaling clusters. |
| On-Premise and Cloud Options | AWS does not provide the on-premise option and rely on the other Amazon services. | Cloudera offers both on-premise and on-cloud options. This helps reuse the on-premise expertise – experience, human resources, and learnings. |

1. **[30 points]** Programming Assignment:

Write a Spark Stream program to stream and read data from Twitter and then display recent tweets related to hashtag a specific hashtag **#soccer**

import pandas as pd

import tweepy

import datetime

import matplotlib.pyplot as plt

def printtweetdata(n, ith\_tweet):

    print()

    print(f"Tweet {n}:")

    print(f"Username:{ith\_tweet[0]}")

    print(f"Description:{ith\_tweet[1]}")

    print(f"Location:{ith\_tweet[2]}")

    print(f"Total Tweets:{ith\_tweet[3]}")

    print(f"Tweet Text:{ith\_tweet[4]}")

    print(f"Hashtags Used:{ith\_tweet[5]}")

def scrape(words, date\_since, numtweet):

    df = pd.DataFrame(columns=['username', 'description', 'location', 'totaltweets', 'text', 'hashtags'])

    tweets = tweepy.Cursor(api.search\_tweets, words, lang="en", since\_id=date\_since, tweet\_mode='extended').items(numtweet)

    list\_tweets = [tweet for tweet in tweets]

    i = 1

    for tweet in list\_tweets:

        username = tweet.user.screen\_name

        description = tweet.user.description

        location = tweet.user.location

        totaltweets = tweet.user.statuses\_count

        hashtags = tweet.entities['hashtags']

        try:

            text = tweet.retweeted\_status.full\_text

        except AttributeError:

            text = tweet.full\_text

        hashtext = list()

        for j in range(0, len(hashtags)):

            hashtext.append(hashtags[j]['text'])

        ith\_tweet = [username, description, location, totaltweets, text, hashtext]

        df.loc[len(df)] = ith\_tweet

        if i<=5:

          printtweetdata(i, ith\_tweet)

        i = i+1

    country\_counts = df['location'].value\_counts()

    fig, ax = plt.subplots(figsize=(10,6))

    country\_counts.plot(kind='bar', ax=ax)

    ax.set\_xlabel('Country')

    ax.set\_ylabel('Number of users')

    ax.set\_title('Total number of countries users are in')

    plt.show()

    df['num\_hashtags'] = df['hashtags'].apply(lambda x: len(x))

    hashtag\_counts = df['num\_hashtags'].value\_counts()

# Create a bar plot to show the counts

    plt.bar(hashtag\_counts.index, hashtag\_counts.values)

    plt.xlabel('Number of hashtags')

    plt.ylabel('Count')

    plt.title('Distribution of hashtag counts in tweets')

    plt.show()

#twitter developer credentials

consumer\_key = "KBJ8WOib3Q4PKoF5q9URSOkx4"

consumer\_secret = "Aabrmwk6li7hAOdEsgLhplnwIemKCewiXevBquzNedCpvY9hHw"

access\_key = "950062185874563072-ghAd9nj7Oc8dCe2g3g1PCPgHImIfhrd"

access\_secret = "zoRfo4OisEloiqotqdvGC9gsVAj5Nftq0nm4glefaTIYK"

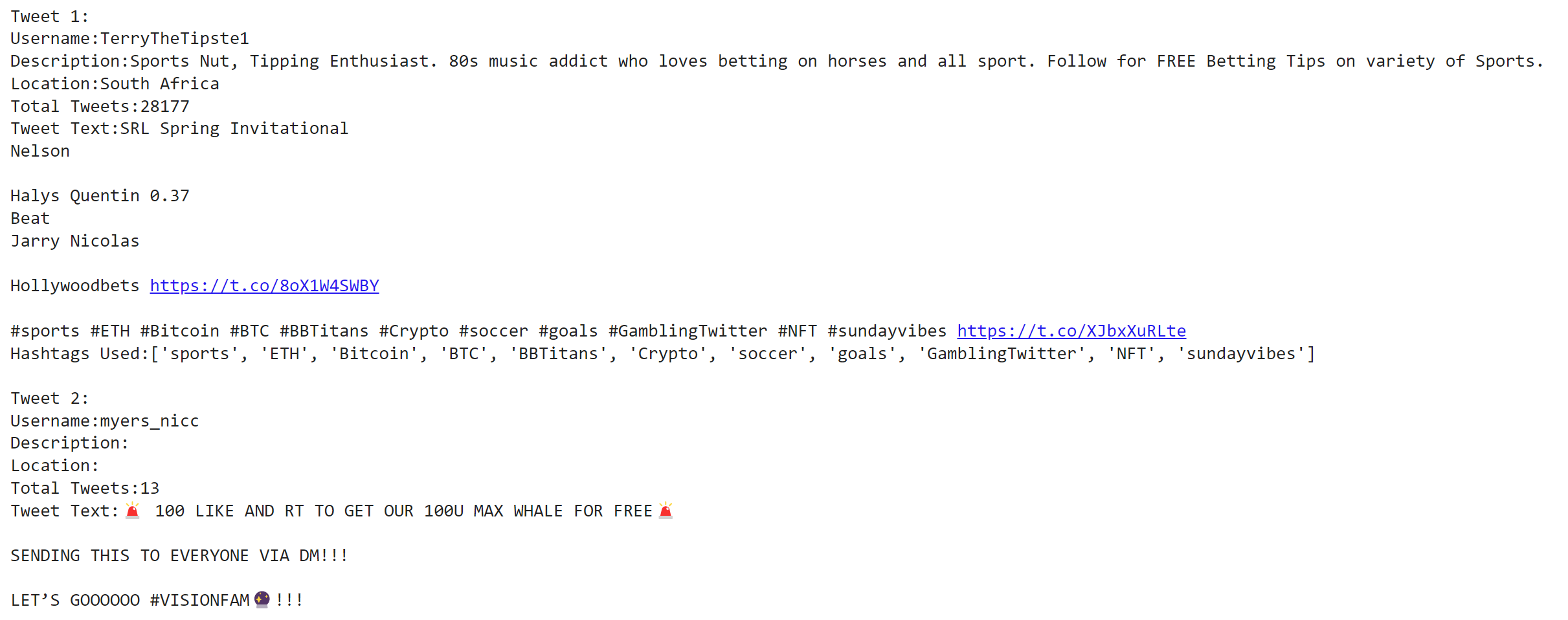
auth = tweepy.OAuthHandler(consumer\_key, consumer\_secret)

auth.set\_access\_token(access\_key, access\_secret)

api = tweepy.API(auth)

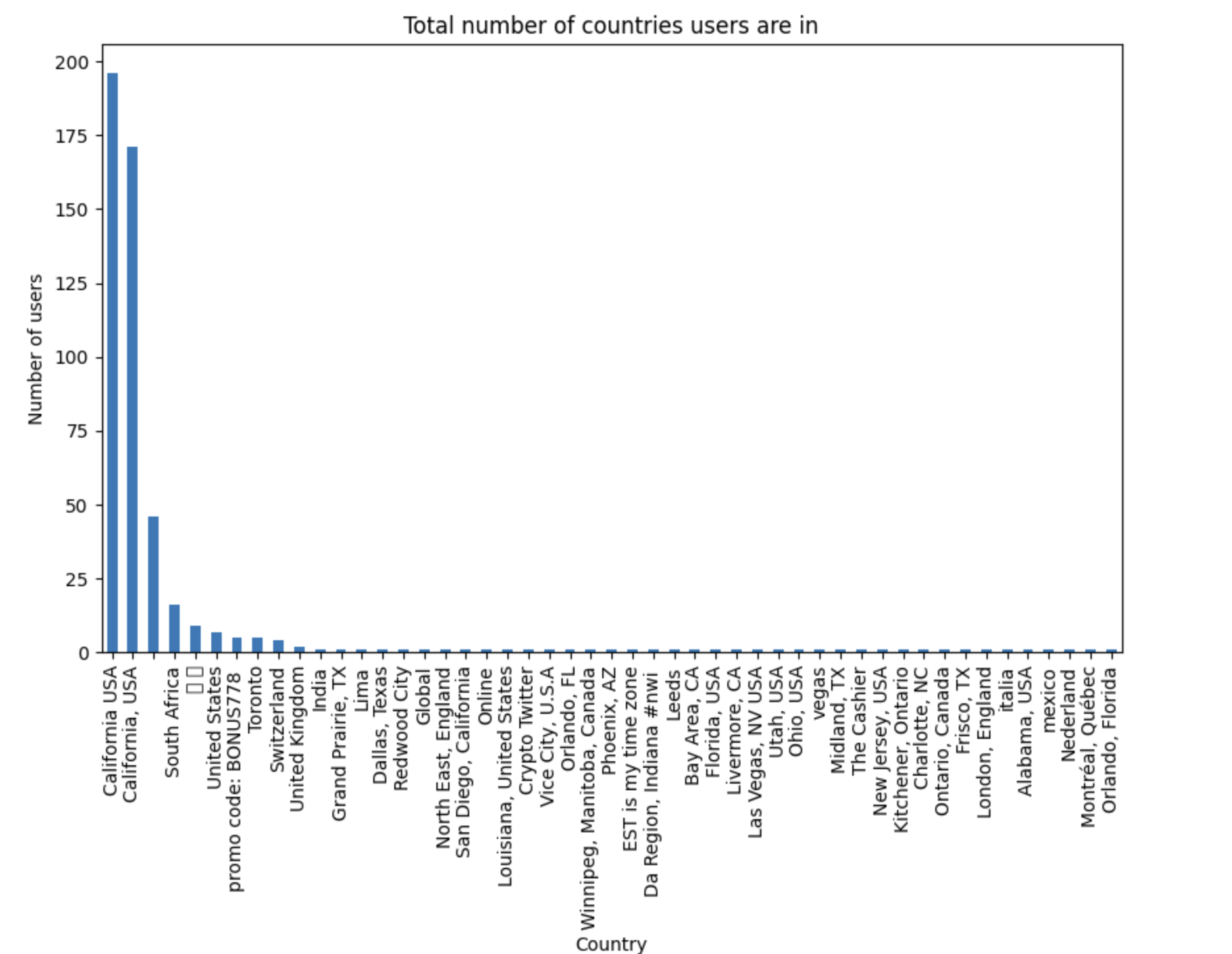
numtweet = 500

scrape("#soccer", datetime.date.today(), numtweet)

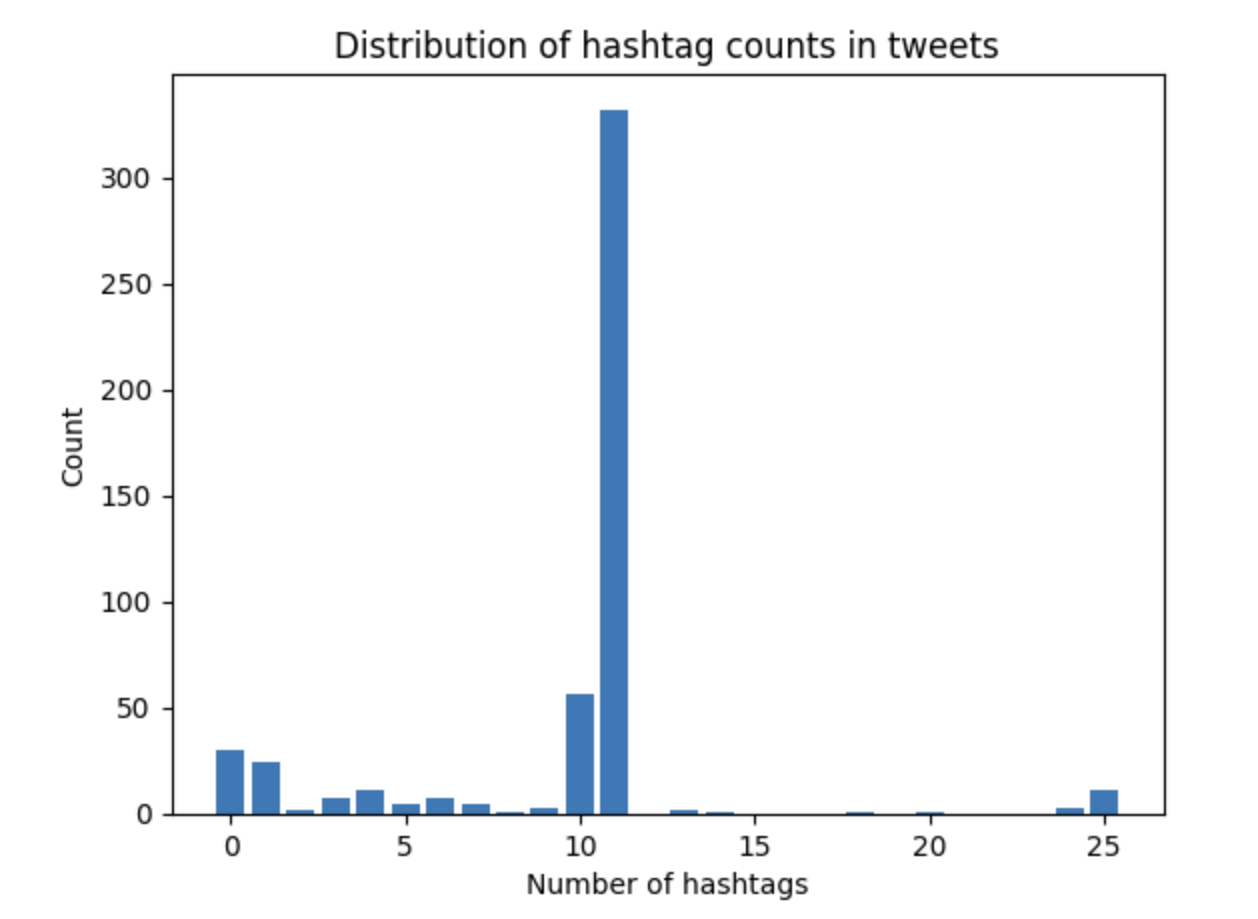
Tweets are being printed as below:  


Visulization:

Country in which users are located:



Number of Hashtags used in tweets having #soccer



**Reference:** <https://www.geeksforgeeks.org/extracting-tweets-containing-a-particular-hashtag-using-python/>

***Hints:***

1. Spark has a package called Twitter.utils. This package contains all the functions to stream data from Twitter
   * Usage of this package is optional, other packages can be used as well.
2. There is also the Tweepy library, which can be used as well
   * Use either one of these packages
3. You may use the Structured Streaming concepts, as discussed
4. If you prefer to use a different Spark package when dealing with Twitter, then that should be OK too.
5. Java, Scala, or Python are all accepted.
6. Results visualization is important, so think about ways to make the results clear.