

Data Analytics at Scale

Structured abstract

This report mainly analyzes the Twiiter data set(both hadoop 2014 and the data we fetched real time from twitter), finds out the hot activity and preference(like or hate) of the python’s 2 main web framework.

We sampled for one day in 2014 and the changes and comparisons of the sampled one day in 2021.

报告标题

2

Table of contents

Introduction ——————————————————————3

Dataset Analytics ———————————————————- 4

Discussion and conclusions of the analysis —————————- 12

Appendix ——————————————————————— 13

Introduction

Consider the professional attributes and your own preferences. I am more familiar with python and web development, so I want to study the changes of python-based web frameworks. Because twiiter's data set exceeds 2G in a single day(/data/ProjectDatasetTwitter ), it is necessary to use big data cross-machine frameworks such as hadoop and spark.

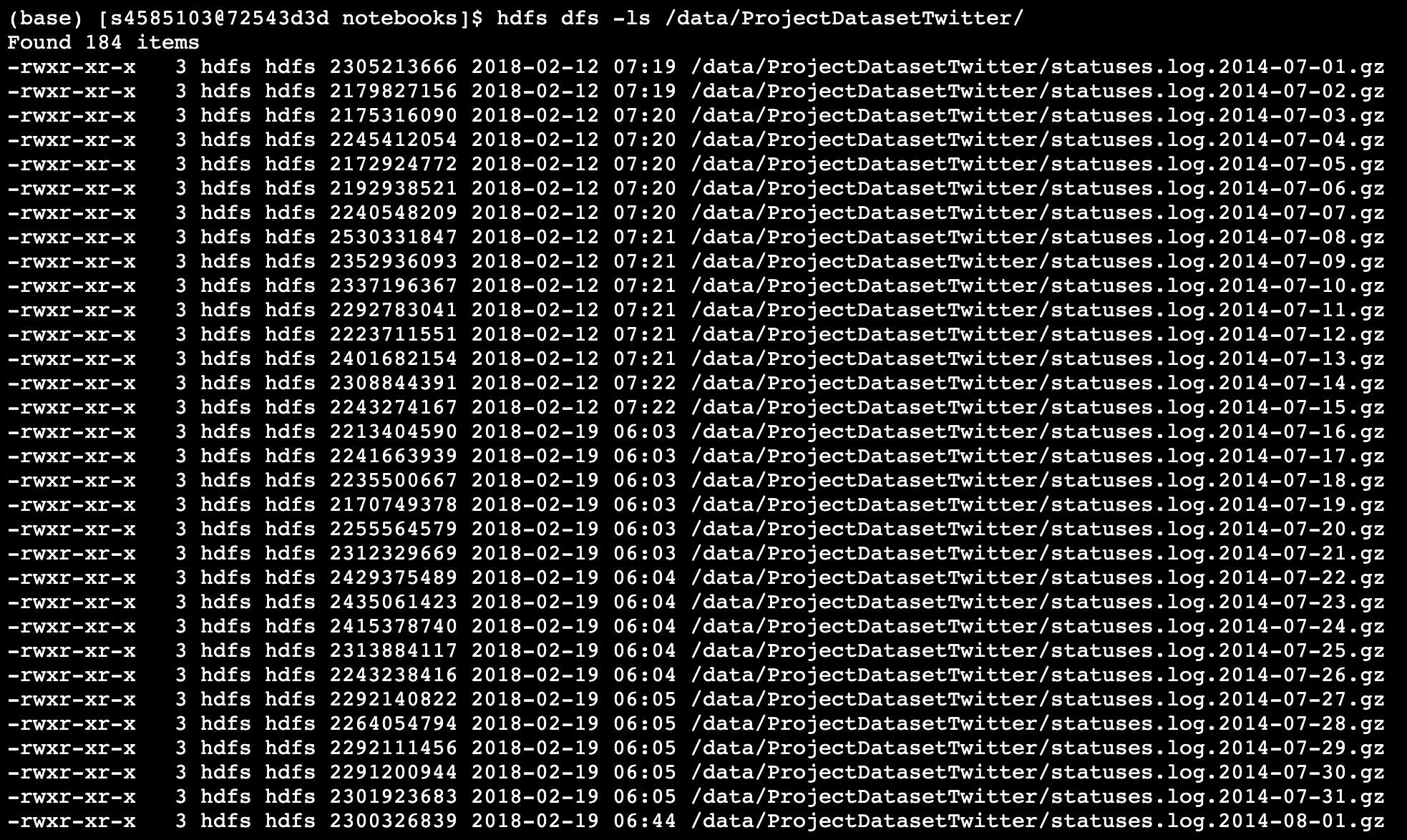
As the advantage of **hadoop** is that it is scalable, ie, any number of systems can be added at any point in time. It works on commodity hardware, so it is easy to keep costs low as compared to other databases. HDFS is mainly designed for large files, and it works on the concept of write once and read many times. In HDFS, individual files are broken into blocks of fixed size (typically 64MB) and stored across a cluster of nodes (not necessarily on the same machine). These files can be more than the size of an individual machine’s hard drive. The individual machines are called data nodes. So we use hdfs to store all ProjectDatasetTwitter dataset.

Hadoop was the first open source system that introduced us to the MapReduce paradigm of programming . But **Spark** is the system that made it faster, much much faster(100x).There used to be a lot of data movement in Hadoop as it used to write intermediate results to the file system.This affected the speed at which you could do analysis. Spark provided us with an in-memory model, so Spark doesn’t write too much to the disk while working. We use pyspark streaming to do our main analytics.

Then after we get result of our data exploration, we use **python**/js(**D3**)/**html5** to To integrate the results and display the data.

Dataset Analytics

step1: use hadoop command to observe:



we can see: all twitter dataset is stored in /data/ProjectDatsetTwitter and it’s distributed as many .gz compressed files. Each compressed files is more than 2Gb sizes, it should contain ’s datetime’s tweets from twitter apis.

step2: read twitter api docs to get to know the structure and description of the dataset:

All Twitter APIs that return Tweets provide that data encoded using JavaScript Object Notation (JSON). JSON is based on key-value pairs, with named attributes and associated values. These attributes, and their state are used to describe objects.

At Twitter we serve many objects as JSON, including Tweets and Users. These objects all encapsulate core attributes that describe the object. Each Tweet has an author, a message, a unique ID, a timestamp of when it was posted, and sometimes geo metadata shared by the user. Each User has a Twitter name, an ID, a number of followers, and most often an account bio.

With each Tweet we also generate "entity" objects, which are arrays of common Tweet contents such as hashtags, mentions, media, and links. If there are links, the JSON payload can also provide metadata such as the fully unwound URL and the webpage’s title and description.

So, in addition to the text content itself, a Tweet can have over 150 attributes associated with it.

So each single record should look like following JSON , which illustrates the structure for these objects and some of their attributes:

{

"created\_at": "Thu Apr 06 15:24:15 +0000 2017",

"id\_str": "850006245121695744",

"text": "1\/ Today we\u2019re sharing our vision for the future of the Twitter API platform!\nhttps:\/\/t.co\/XweGngmxlP",

"user": {

"id": 2244994945,

"name": "Twitter Dev",

"screen\_name": "TwitterDev",

"location": "Internet",

"url": "https:\/\/dev.twitter.com\/",

"description": "Your official source for Twitter Platform news, updates & events. Need technical help? Visit https:\/\/twittercommunity.com\/ \u2328\ufe0f #TapIntoTwitter"

},

"place": {

},

"entities": {

"hashtags": [

],

"urls": [

{

"url": "https:\/\/t.co\/XweGngmxlP",

"unwound": {

"url": "https:\/\/cards.twitter.com\/cards\/18ce53wgo4h\/3xo1c",

"title": "Building the Future of the Twitter API Platform"

}

}

],

"user\_mentions": [

]

}

}

step3: we choose random day’s giz file from hadoop, use pyspark to get the sample data to explore:

**inputFile = '/data/ProjectDatasetTwitter/statuses.log.2014-12-04.gz'**

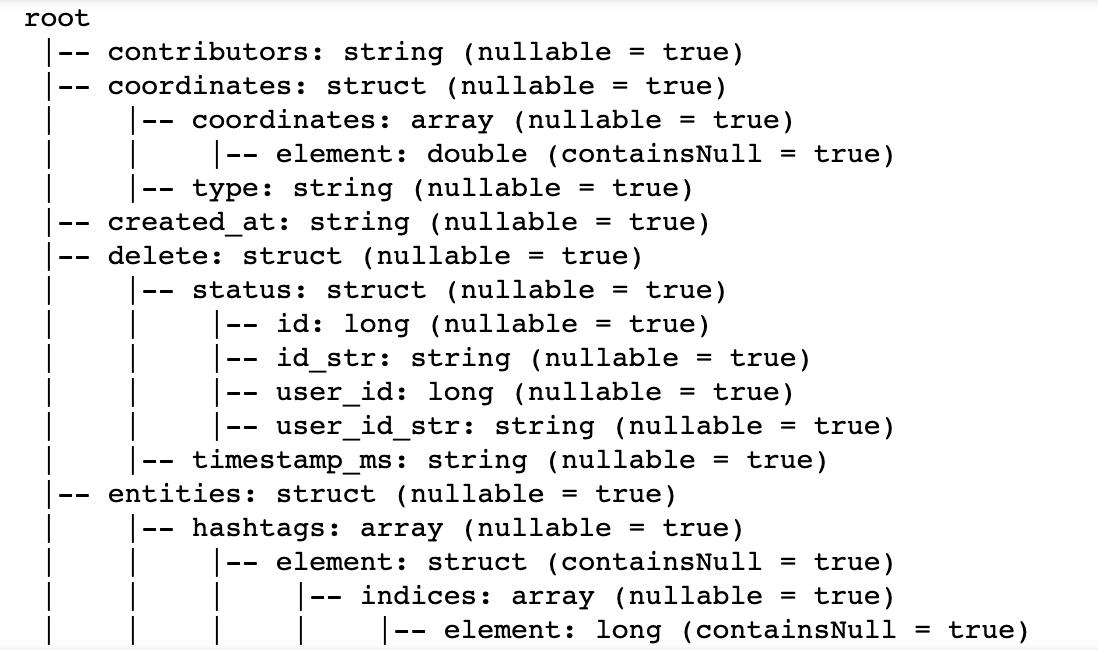
**conf = SparkConf().setAppName("SparkSQLTwitter")**

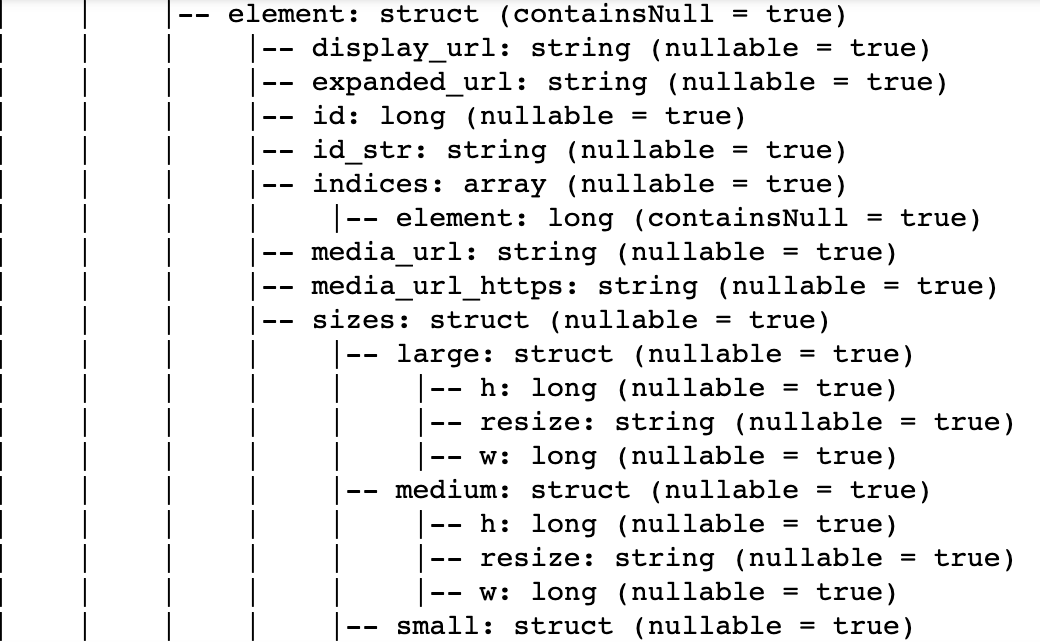
**sc = SparkContext()**

**input = hiveCtx.read.json(inputFile)**

**input.show(n=20, truncate=False)**

**input.printSchema()**

we get real schema from data:



sample record is:

row= Row(contributors=None, coordinates=None, created\_at=None, delete=Row(status=Row(id=56428245058457600, id\_str='56428245058457600', user\_id=276700097, user\_id\_str='276700097'), timestamp\_ms='1417647600084'), entities=None, extended\_entities=None, favorite\_count=None, favorited=None, filter\_level=None, geo=None, id=None, id\_str=None, in\_reply\_to\_screen\_name=None, in\_reply\_to\_status\_id=None, in\_reply\_to\_status\_id\_str=None, in\_reply\_to\_user\_id=None, in\_reply\_to\_user\_id\_str=None, lang=None, place=None, possibly\_sensitive=None, retweet\_count=None, retweeted=None, retweeted\_status=None, scopes=None, source=None, text=None, timestamp\_ms=None, truncated=None, user=None, withheld\_in\_countries=None)

step4: data clean

we want to see tweet with content , and related retweet\_count, fovorite\_count is not null.

So wo have to remove useless records from datatset:

input.na.drop(subset=[“text"])

input.na.drop(subset=[“retweet\_count"])

input.na.drop.na.drop(subset=[“entities”])

step5: data explore

df.registerTempTable("tweets")

t = sqlContext.sql("SELECT distinct id,entities.hashtags FROM tweets").rdd

related\_hashtags= t.map(lambda t: map(lambda t0: t0[2].lower(), t[1])) \

.map(lambda t: list(itertools.combinations(t, 2))) \

.flatMap(lambda t: t) \

.map(lambda t: sorted(t)) \

.map(lambda x: '\t'.join(unicode(i) for i in x)) \

.map(lambda t: (t, 1)) \

.reduceByKey(lambda x,y: x+y) \

.filter(lambda t: t[1]>=min\_occurs) \

.sortByKey(False) \

.map(lambda x: '\t'.join(unicode(i) for i in x)) \

.repartition(1)

print('here', related\_hashtags)

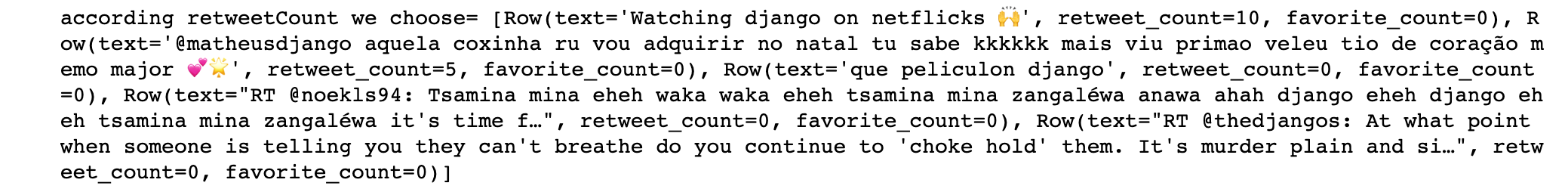
When the processing is finished , then we save some result from tweets to hdfs temperally

related\_hashtags.saveAsTextFile("%s/%s" % ("/tmp/tests/00", “related\_hashtags”))

we explore the top 20 tweets by all angles: like favorite\_count / retweet\_count , something like that:

topTweets = hiveCtx.sql("SELECT text, retweet\_count,favorite\_count FROM tweets where text IS NOT NULL and \

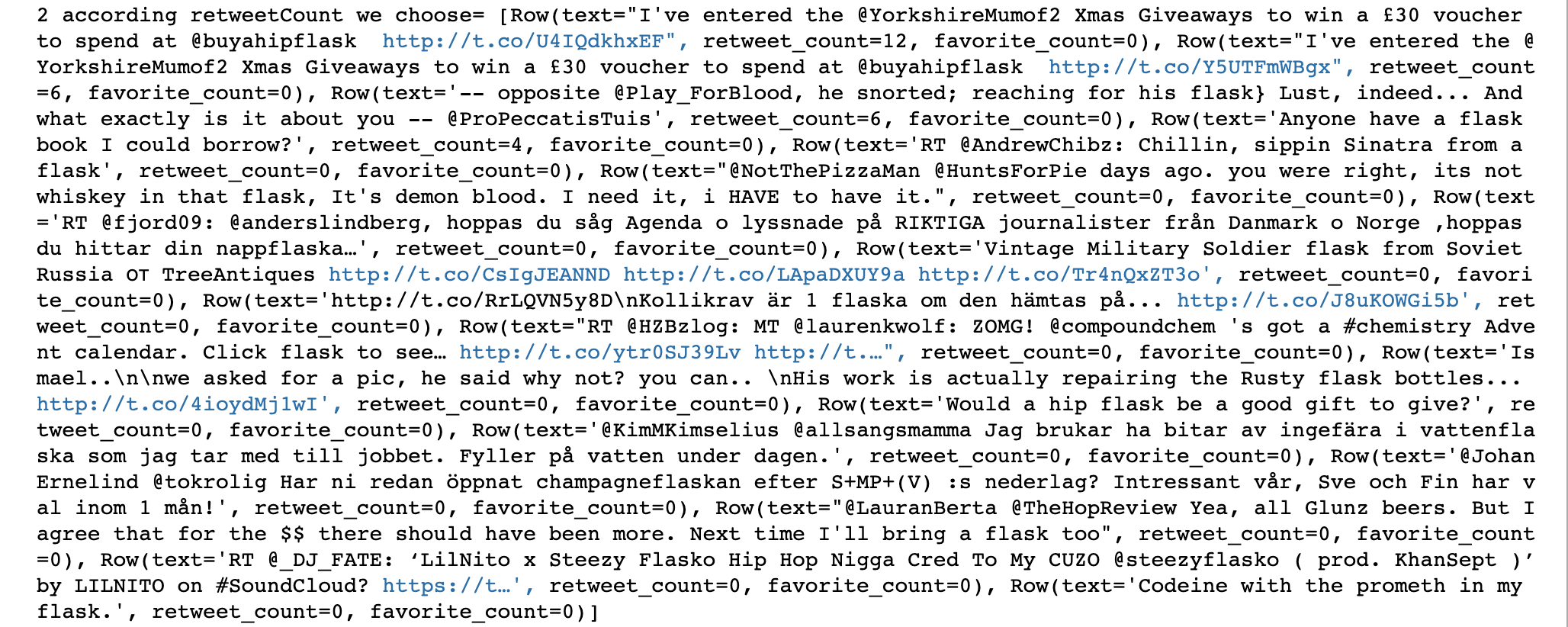
text like '%django%' LIMIT 20")

print('according retweetCount we choose=', topTweets.collect() )

topTweets = hiveCtx.sql("SELECT text, retweet\_count,favorite\_count FROM tweets where text IS NOT NULL and \

text like '%flask%' LIMIT 20")

print('2 according retweetCount we choose=', topTweets.collect() )



step6: We check the result , Exclude tweets with non-frame information with the same name (determined based on the context of the tweet sentence)，from anlysing we can see: in a random day in 2014, the django/flask Heat comparison of events：**15: 4**

step7: We use scrapy/python to download

athering tweets URL by searching through hashtags

For searching for tweets we will be using the legacy Twitter website. Let’s try searching for :

mobile.twitter.com/hashtag/flask

mobile.twitter.com/hashtag/django

We want to use this legacy version for gathering tweets URLs as it loads the data without using javascript which makes our job easy.

The first parse() function bears the brunt of the task. First, items are introduced as lists, so that they can be used in the function.

Then, using the response.xpath() method from before to collect links, we collect every link on the page.

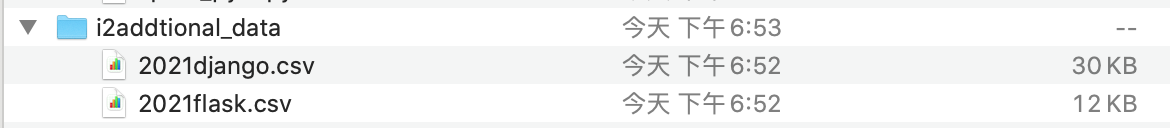
We then filter those links into twitterlink, which collects links that include “twitter.com,” which can be truncated into handles, and domainlink, or links from that website, which we use to crawl the website for further handles.

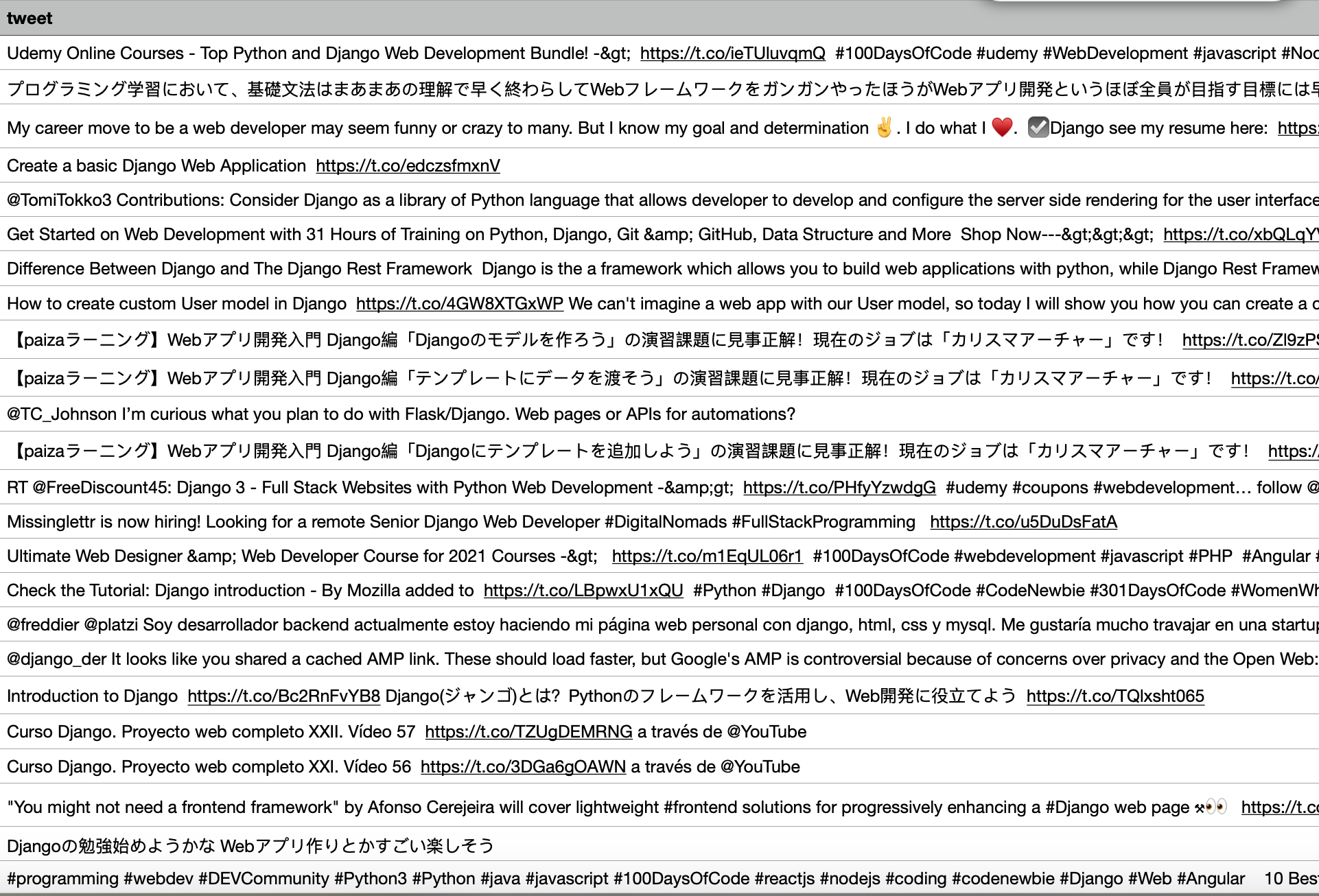
Then, as a way of keeping a running list throughout the spider, we add twitterlink to twitterlinku, a cumulative list of Twitter links.

At this point, we change gears from scraping to crawling. Since in many cases scraping an entire website will take too much time and memory, we’ll only do this to a secondary level: only the first page and any page that the first page links to will be scraped. This is useful for most instances where there is a list of sites that include Twitter links.

The method scrapy.Request() allows us to call parse2(). However, before we yield this request, we use .meta to ensure that the lists of items are maintained as the same throughout the process.

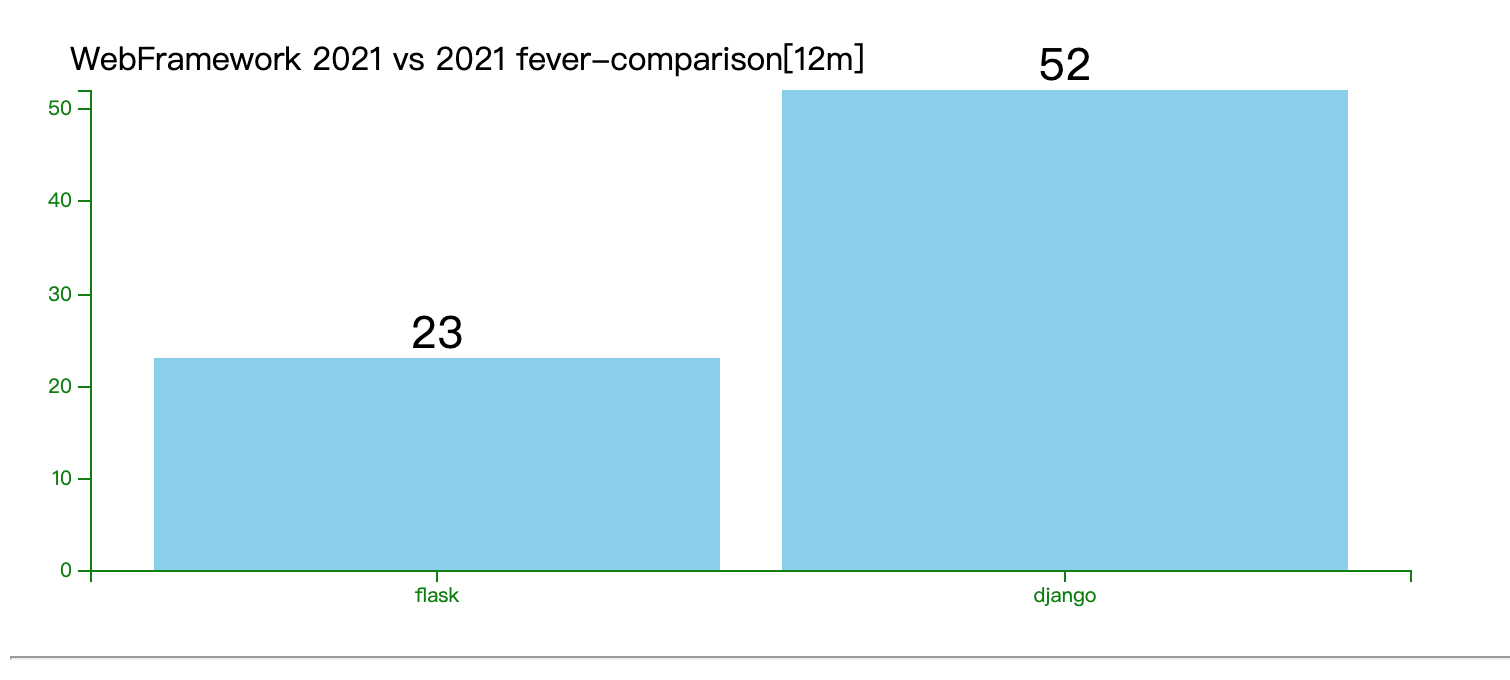
After all downloading is done, we got today’s all tweets about “Flask/Django”, we save into 2 seperate csvs.



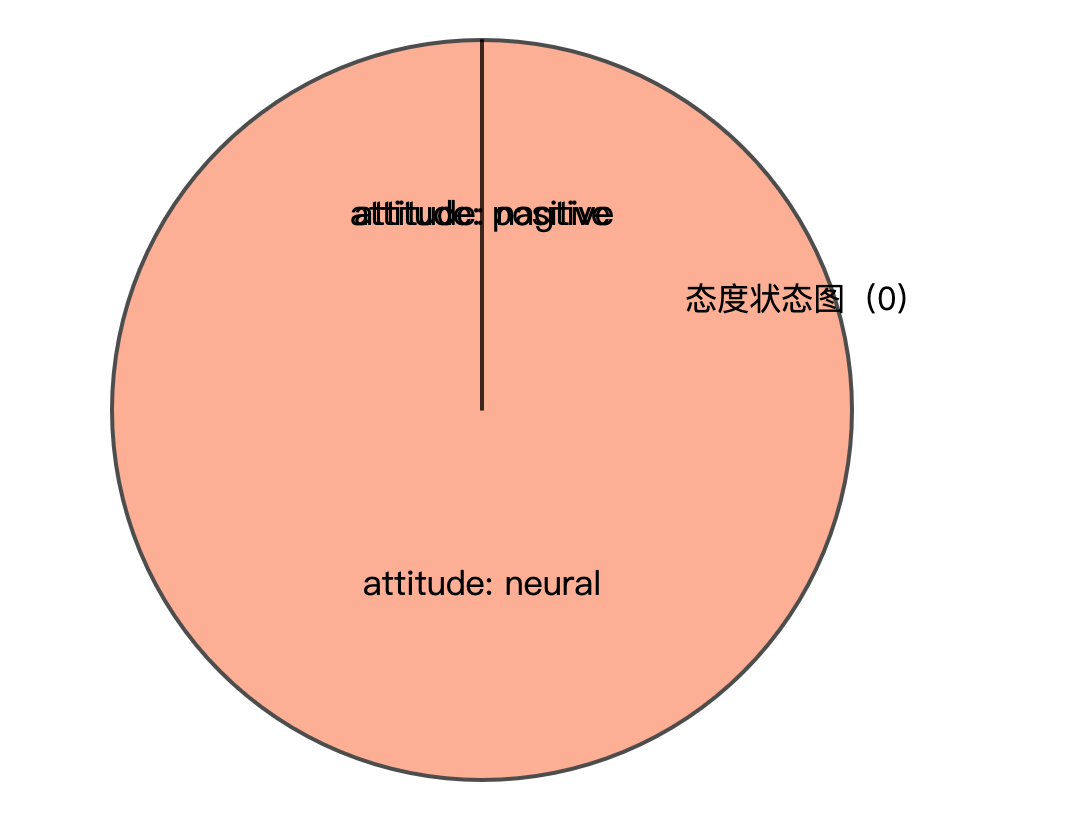


step8: We compare the downloaded yesterday’s data heat comparation, and we use d3 to visualize it:

in 2021’s ramdom day , the django/flask hot comparation: **23:52**



we all use maching learning to get to know people’s attitude about these web framework:



most of twitter use is neural about the web framework.

Then we use python scripts to explore the related words:

**a1\_sorted\_keys = sorted(wordfreq, key=wordfreq.get, reverse=True)**

**for r in a1\_sorted\_keys:**

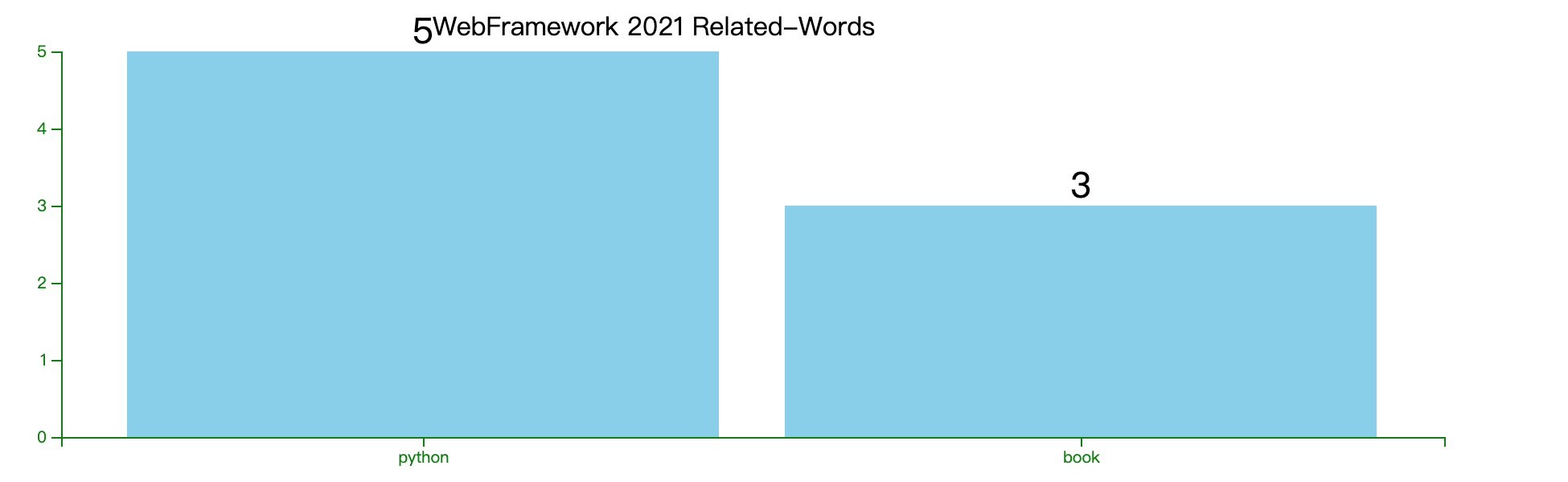
**if wordfreq[r] > 1:**

**print(r, wordfreq[r])**

**if index < 10:**

**js\_txt += "'" + r + "':" + str(wordfreq[r]) + ','**

**index += 1**

****

We can see: the most people tweet about the 2 framework is when you also tweet python or books .

We then use python scripts to check the most frequently tweets such subjects users, find out the most active twitter users in this field, and find out who are most concerned about flask/django keywords:

**a2\_sorted\_keys = sorted(usernamefreq, key=usernamefreq.get, reverse=True)**

**for r in a2\_sorted\_keys:**

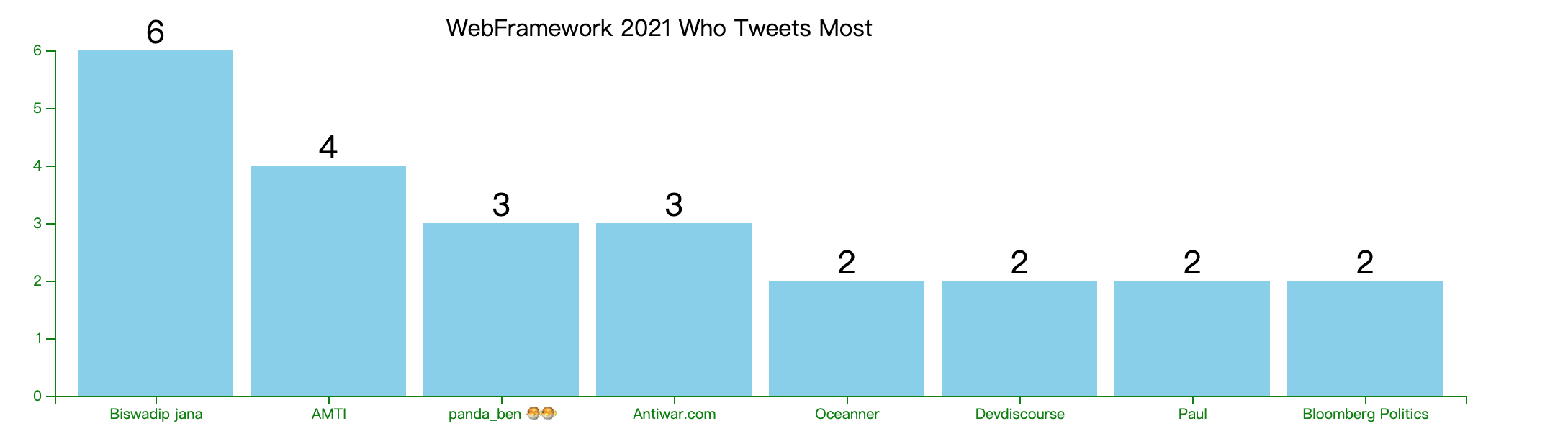
**if usernamefreq[r] > 1:**

**print(r, usernamefreq[r])**

**if index < 10:**

**js\_txt += "'" + r + "':" + str(usernamefreq[r]) + ','**

**index += 1**

****

Discussion and conclusions of the analysis

From the aboving , we can see:

1. the flask/django’s absolute popularity has risen. No matter which framework it is, the absolute number of people and news discussed in 2021 will increase compared with 2014. More people care about the python web framework. The total num from 19 jump to 75 in a random days.
2. The relative gap between flask and django has been greatly reduced, and the ratio has been reduced from “15: 4” to “52：23” I digged the possible reasons: some of world-famous companies such as Airbnb and Reddit use Flask. Flask gives you more control over your project, since you can choose which components to use and how you interact with them. Also, you can plug in any extension you need.
3. If the purpose is very clear, the demand for data processing capabilities can be greatly reduced. But in the exploratory stage, when the purpose is not clear, it is more advantageous to have the ability to process big data. In our case, the day of tweets is 2Gb, but tweets with ‘flask/django’ hastags in it, it’s just less than 1Mb.
4. Use hadoop or local file system to store intermediate calculation results, which can be very convenient for subsequent visualization and participation of machine learning libraries. In our case , we use hadoop to save template hashtags , and use local csvs to save 2021’s tweets data which compain ‘flask/django hashtags, it helps the processing procedure.

Appendix

