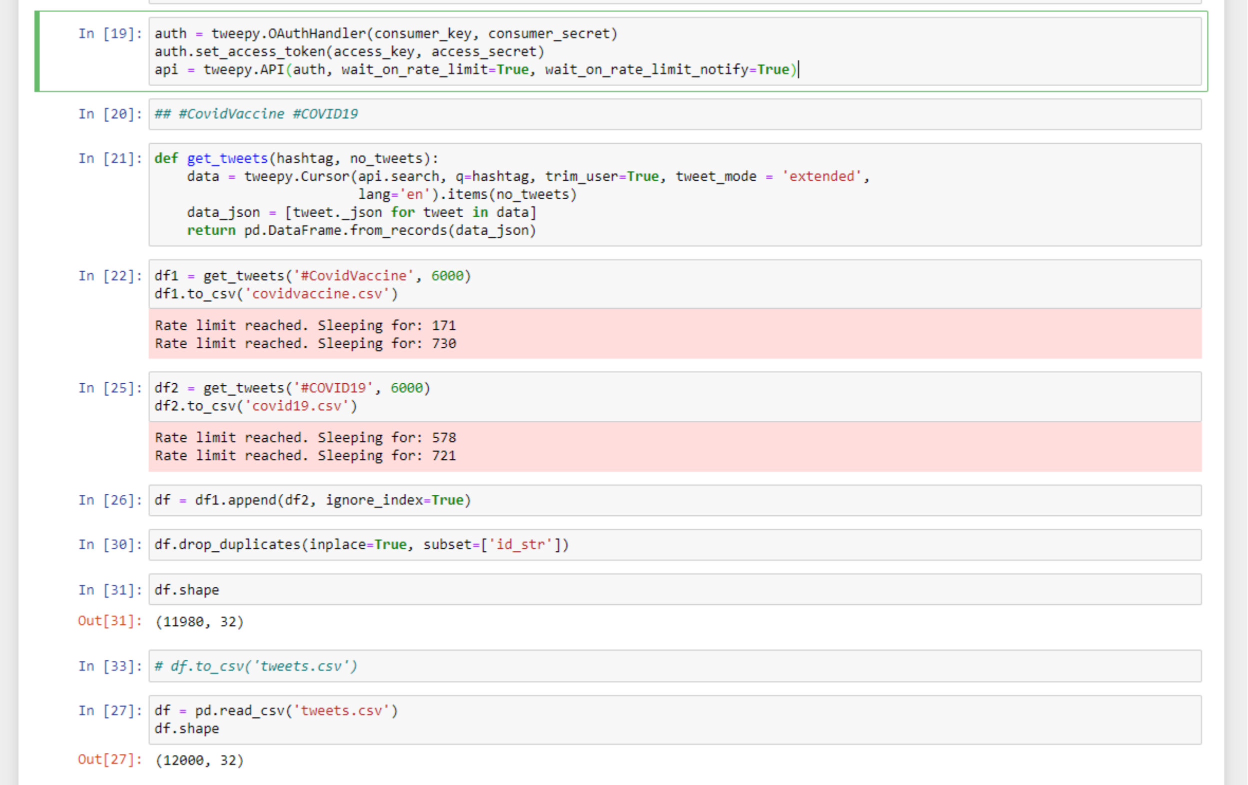
Assignment-4

Garvita Jain (2018034) | Arnav Tandon (2018278)

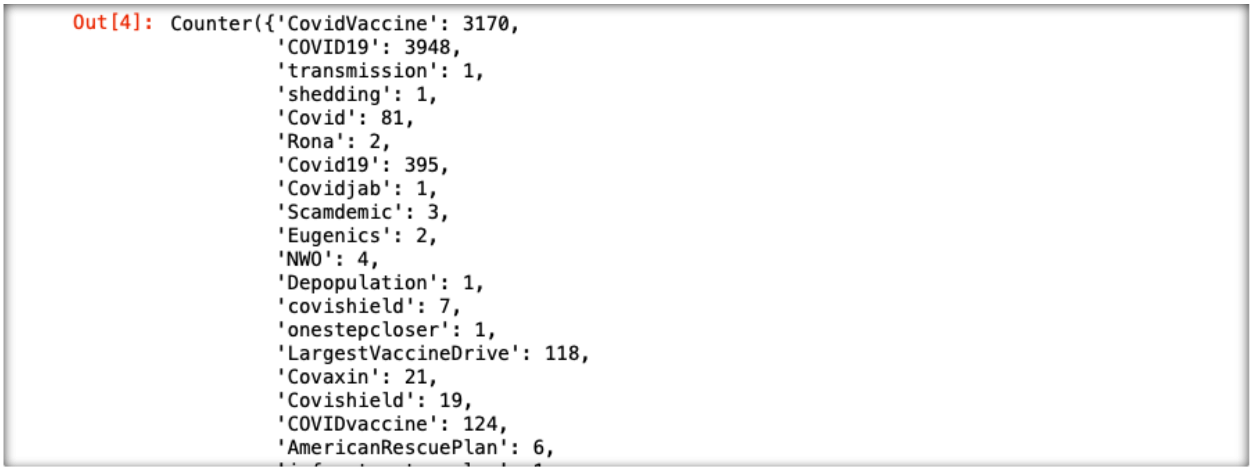
Ans 1)

We used two hashtags to collect data : #CovidVaccine #COVID19
Total unique tweets collected : 11,980



```
In [19]: auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
         auth.set_access_token(access_key, access_secret)
         api = tweepy.API(auth, wait_on_rate_limit=True, wait_on_rate_limit_notify=True)

In [20]: ## #CovidVaccine #COVID19

In [21]: def get_tweets(hashtag, no_tweets):
             data = tweepy.Cursor(api.search, q=hashtag, trim_user=True, tweet_mode = 'extended',
                                  lang='en').items(no_tweets)
             data_json = [tweet._json for tweet in data]
             return pd.DataFrame.from_records(data_json)

In [22]: df1 = get_tweets('#CovidVaccine', 6000)
         df1.to_csv('covidvaccine.csv')

         Rate limit reached. Sleeping for: 171
         Rate limit reached. Sleeping for: 730

In [25]: df2 = get_tweets('#COVID19', 6000)
         df2.to_csv('covid19.csv')

         Rate limit reached. Sleeping for: 578
         Rate limit reached. Sleeping for: 721

In [26]: df = df1.append(df2, ignore_index=True)

In [30]: df.drop_duplicates(inplace=True, subset=['id_str'])

In [31]: df.shape

Out[31]: (11980, 32)

In [33]: # df.to_csv('tweets.csv')

In [27]: df = pd.read_csv('tweets.csv')
         df.shape

Out[27]: (12000, 32)
```

We stored the collected in a CSV file (tweets.csv) and then used them for further analysis.



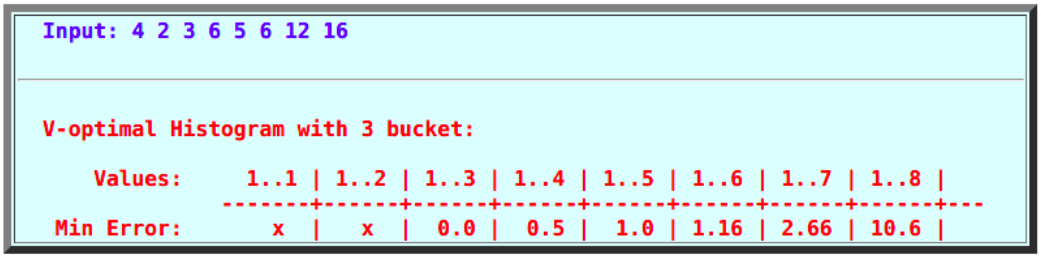
```
Out[4]: Counter({'CovidVaccine': 3170,
                 'COVID19': 3948,
                 'transmission': 1,
                 'shedding': 1,
                 'Covid': 81,
                 'Rona': 2,
                 'Covid19': 395,
                 'Covidjab': 1,
                 'Scamdemic': 3,
                 'Eugenics': 2,
                 'NWO': 4,
                 'Depopulation': 1,
                 'covishield': 7,
                 'onestepcloser': 1,
                 'LargestVaccineDrive': 118,
                 'Covaxin': 21,
                 'Covishield': 19,
                 'COVIDvaccine': 124,
                 'AmericanRescuePlan': 6,
```

Ans 2)
Implementation of Jagadish Algorithm :

Example shown in book :

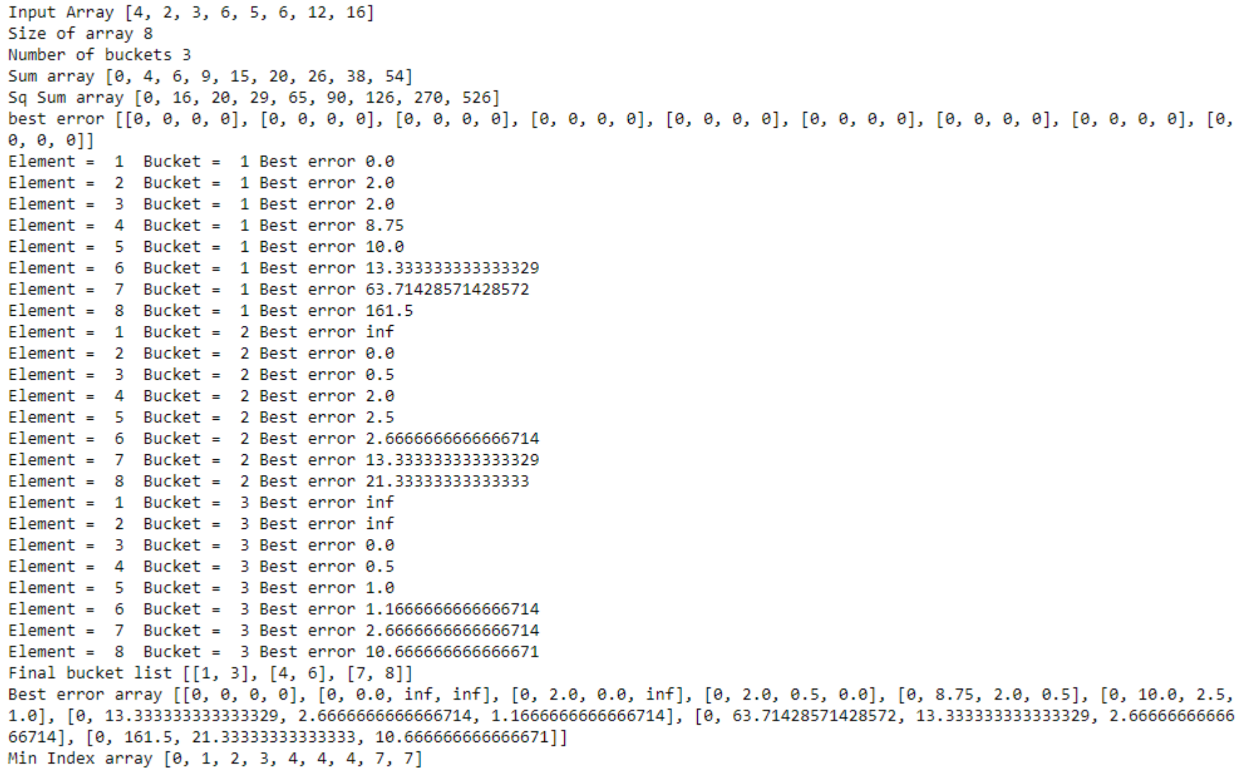
For input array = [4,2,3,6,5,6,12,16]
Buckets = 3

- **And so on... - Final result:**



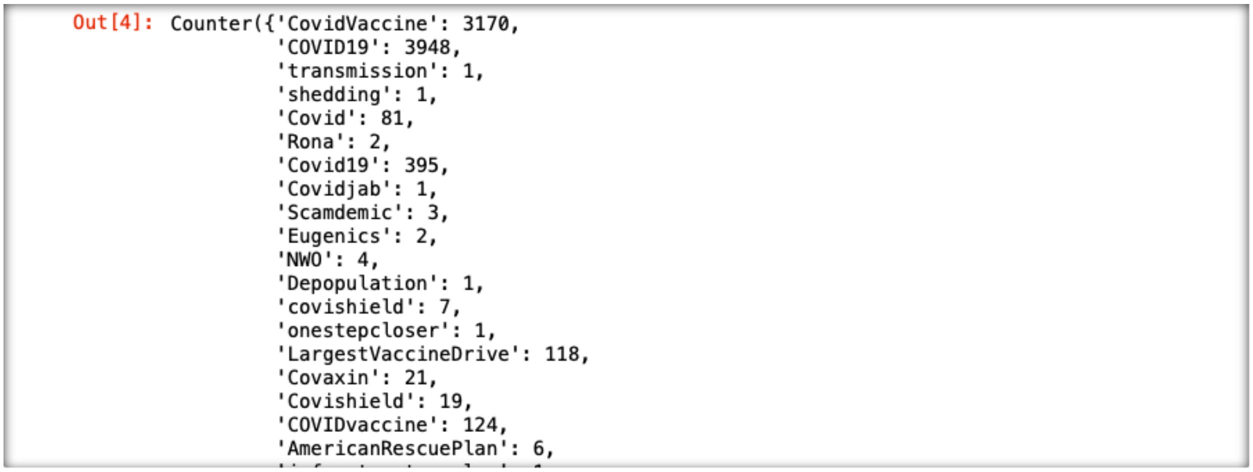
```
Input: 4 2 3 6 5 6 12 16

V-optimal Histogram with 3 bucket:

    Values:     1..1 | 1..2 | 1..3 | 1..4 | 1..5 | 1..6 | 1..7 | 1..8 |
              -------+------+------+------+------+------+------+------+---
 Min Error:       x  |    x  |  0.0 |  0.5 |  1.0 | 1.16 | 2.66 | 10.6 |
```

From our implementation :



```
Input Array [4, 2, 3, 6, 5, 6, 12, 16]
Size of array 8
Number of buckets 3
Sum array [0, 4, 6, 9, 15, 20, 26, 38, 54]
Sq Sum array [0, 16, 20, 29, 65, 90, 126, 270, 526]
best error [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0,
0, 0, 0]]
Element =  1  Bucket =  1 Best error 0.0
Element =  2  Bucket =  1 Best error 2.0
Element =  3  Bucket =  1 Best error 2.0
Element =  4  Bucket =  1 Best error 8.75
Element =  5  Bucket =  1 Best error 10.0
Element =  6  Bucket =  1 Best error 13.333333333333329
Element =  7  Bucket =  1 Best error 63.71428571428572
Element =  8  Bucket =  1 Best error 161.5
Element =  1  Bucket =  2 Best error inf
Element =  2  Bucket =  2 Best error 0.0
Element =  3  Bucket =  2 Best error 0.5
Element =  4  Bucket =  2 Best error 2.0
Element =  5  Bucket =  2 Best error 2.5
Element =  6  Bucket =  2 Best error 2.6666666666666714
Element =  7  Bucket =  2 Best error 13.333333333333329
Element =  8  Bucket =  2 Best error 21.33333333333333
Element =  1  Bucket =  3 Best error inf
Element =  2  Bucket =  3 Best error inf
Element =  3  Bucket =  3 Best error 0.0
Element =  4  Bucket =  3 Best error 0.5
Element =  5  Bucket =  3 Best error 1.0
Element =  6  Bucket =  3 Best error 1.1666666666666714
Element =  7  Bucket =  3 Best error 2.6666666666666714
Element =  8  Bucket =  3 Best error 10.666666666666671
Final bucket list [[1, 3], [4, 6], [7, 8]]
Best error array [[0, 0, 0, 0], [0, 0.0, inf, inf], [0, 2.0, 0.0, inf], [0, 2.0, 0.5, 0.0], [0, 8.75, 2.0, 0.5], [0, 10.0, 2.5,
1.0], [0, 13.333333333333329, 2.6666666666666714, 1.1666666666666714], [0, 63.71428571428572, 13.333333333333329, 2.66666666666
66714], [0, 161.5, 21.33333333333333, 10.666666666666671]]
Min Index array [0, 1, 2, 3, 4, 4, 4, 7, 7]
```

Here we can verify our implementation by seeing that both the results are the same.



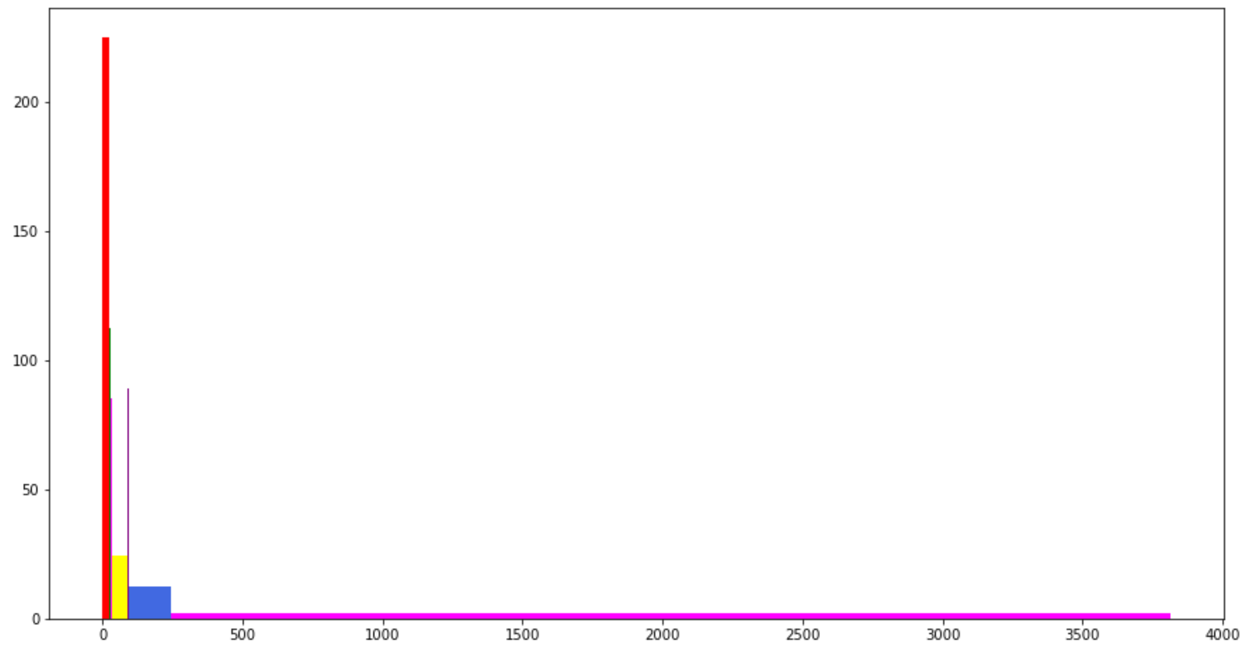
```
Out[4]: Counter({'CovidVaccine': 3170,
                 'COVID19': 3948,
                 'transmission': 1,
                 'shedding': 1,
                 'Covid': 81,
                 'Rona': 2,
                 'Covid19': 395,
                 'Covidjab': 1,
                 'Scamdemic': 3,
                 'Eugenics': 2,
                 'NWO': 4,
                 'Depopulation': 1,
                 'covishield': 7,
                 'onestepcloser': 1,
                 'LargestVaccineDrive': 118,
                 'Covaxin': 21,
                 'Covishield': 19,
                 'COVIDvaccine': 124,
                 'AmericanRescuePlan': 6,
100%|██████████| 3814/3814 [00:00<00:00, 264515.03it/s]
  9%|█         | 358/3814 [00:00<00:00, 3572.60it/s]
```

```
Size of array 3814
Number of buckets 10
```

```
100%|██████████| 3814/3814 [00:10<00:00, 368.89it/s]
100%|██████████| 3814/3814 [00:09<00:00, 386.62it/s]
100%|██████████| 3814/3814 [00:10<00:00, 370.37it/s]
100%|██████████| 3814/3814 [00:09<00:00, 412.27it/s]
100%|██████████| 3814/3814 [00:09<00:00, 389.98it/s]
100%|██████████| 3814/3814 [00:09<00:00, 410.80it/s]
100%|██████████| 3814/3814 [00:10<00:00, 381.09it/s]
100%|██████████| 3814/3814 [00:09<00:00, 413.65it/s]
100%|██████████| 3814/3814 [00:09<00:00, 399.17it/s]
```

```
Final bucket list [[1, 22], [23, 24], [25, 25], [26, 30], [31, 31], [32, 89], [90, 91], [92
, 245], [246, 246], [247, 3814]]
```

```
Average Values for The Buckets
Squared Error of Bucket:  14703638.954545464
[1, 22]  =  225.04545454545453
Squared Error of Bucket:  8064.5
[23, 24]  =  154.5
Squared Error of Bucket:  0.0
[25, 25]  =  10.0
Squared Error of Bucket:  202165.20000000004
[26, 30]  =  112.4
Squared Error of Bucket:  0.0
[31, 31]  =  85.0
Squared Error of Bucket:  101750.4827586207
[32, 89]  =  24.482758620689655
Squared Error of Bucket:  14792.0
[90, 91]  =  89.0
Squared Error of Bucket:  100319.22727272732
[92, 245]  =  12.590909090909092
Squared Error of Bucket:  0.0
[246, 246]  =  6.0
Squared Error of Bucket:  130459.36274977843
[247, 3814]  =  2.220011210762332
Squared Error of V-Optimal Histogram:  15261189.727326589
```

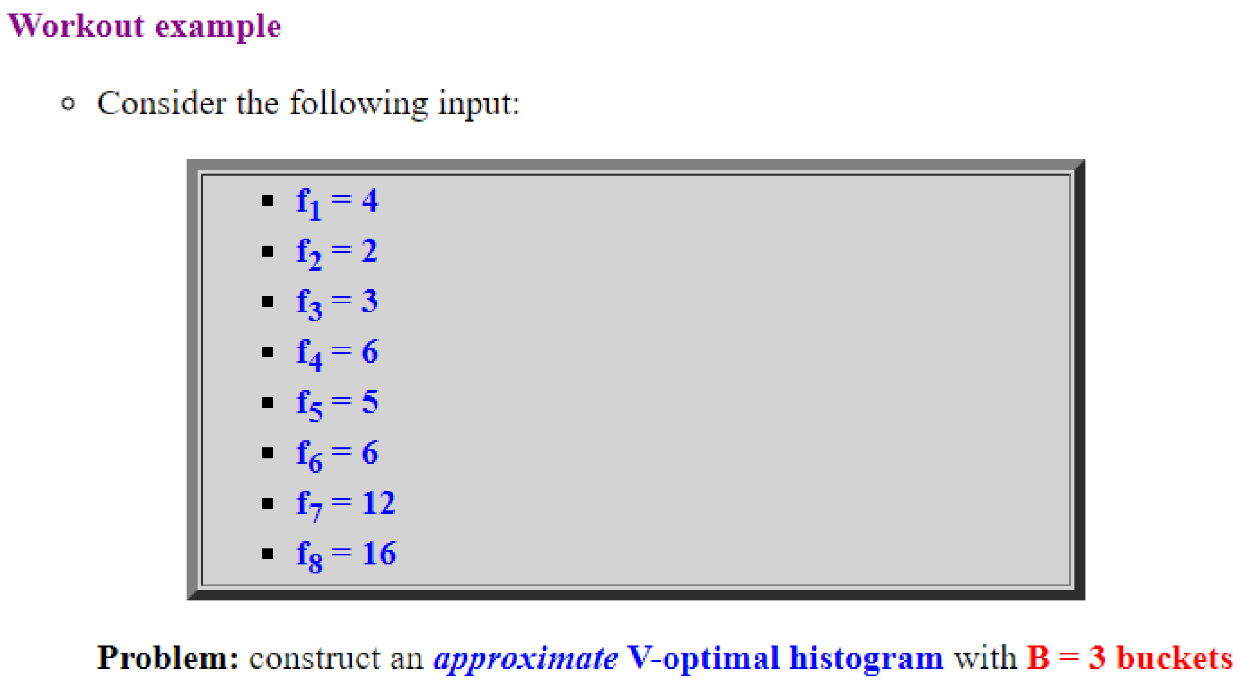


Y-axis : Frequency of hashtag
X-axis : Hashtag by index

Ans 3)

We did the implementation of Guha’s algorithm and the output is shown as below :

Example Shown in Book for implementation of Guha :

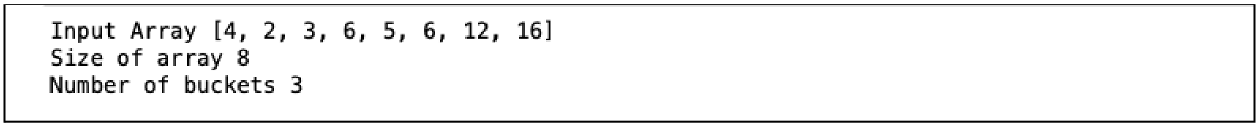


Workout example

- Consider the following input:

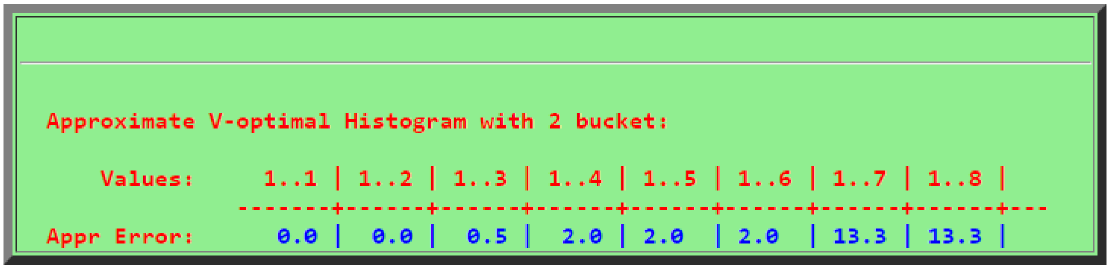
  - $f_1 = 4$
  - $f_2 = 2$
  - $f_3 = 3$
  - $f_4 = 6$
  - $f_5 = 5$
  - $f_6 = 6$
  - $f_7 = 12$
  - $f_8 = 16$

Problem: construct an ***approximate*** **V-optimal histogram** with **B = 3 buckets**



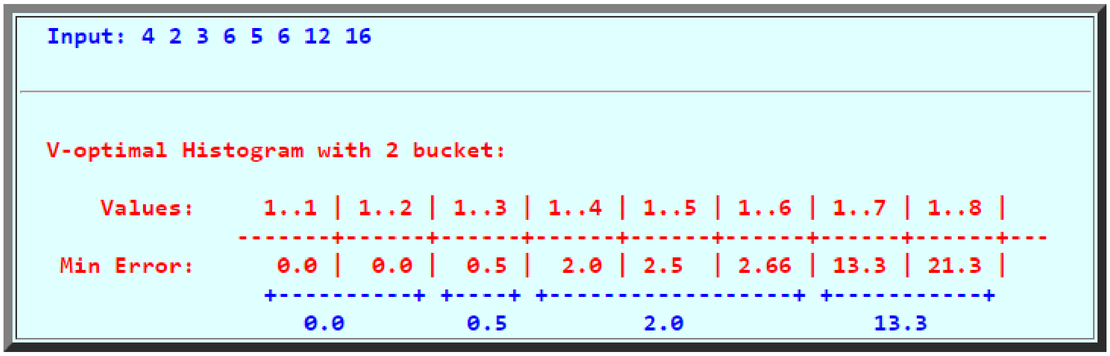
```
Input Array [4, 2, 3, 6, 5, 6, 12, 16]
Size of array 8
Number of buckets 3
```

- **And so on... - Final result:**

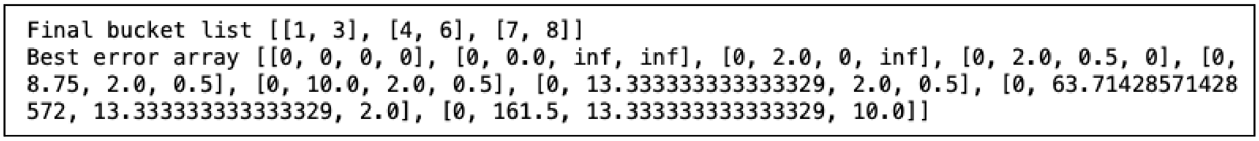


```
Approximate V-optimal Histogram with 2 bucket:

    Values:      1..1 | 1..2 | 1..3 | 1..4 | 1..5 | 1..6 | 1..7 | 1..8 |
                -------+------+------+------+------+------+------+------+---
Appr Error:       0.0 |  0.0 |  0.5 |  2.0 | 2.0  | 2.0  | 13.3 | 13.3 |
```

- **Compare** the **result** of **Guha's** algorithm and the **actual minimum error result**:

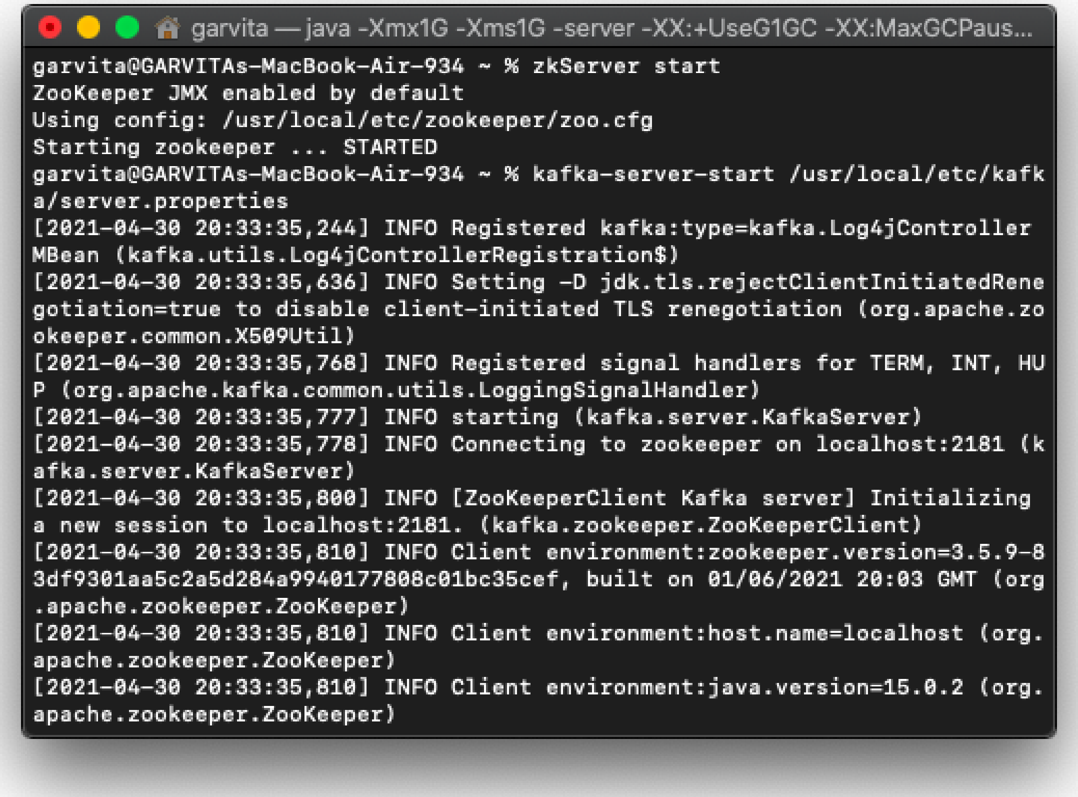


```
Input: 4 2 3 6 5 6 12 16

V-optimal Histogram with 2 bucket:

    Values:      1..1 | 1..2 | 1..3 | 1..4 | 1..5 | 1..6 | 1..7 | 1..8 |
                -------+------+------+------+------+------+------+------+---
 Min Error:       0.0 |  0.0 |  0.5 |  2.0 | 2.5  | 2.66 | 13.3 | 21.3 |
                  +-----------+ +-----+ +-------------------+ +------------+
                    0.0           0.5          2.0                13.3
```



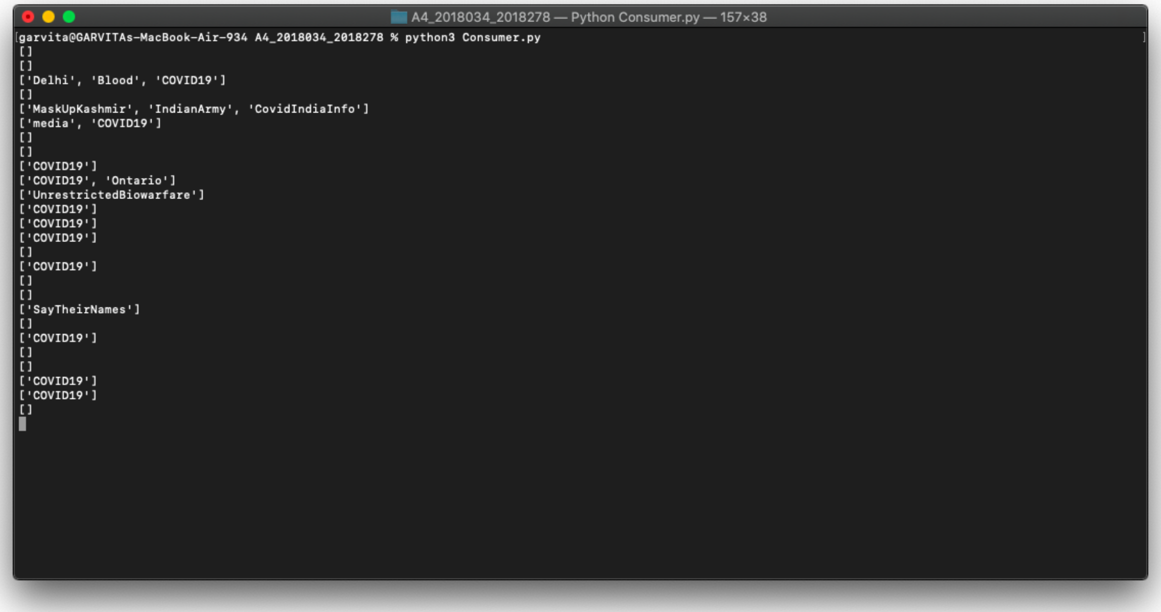
```
Final bucket list [[1, 3], [4, 6], [7, 8]]
Best error array [[0, 0, 0, 0], [0, 0.0, inf, inf], [0, 2.0, 0, inf], [0, 2.0, 0.5, 0], [0,
8.75, 2.0, 0.5], [0, 10.0, 2.0, 0.5], [0, 13.333333333333329, 2.0, 0.5], [0, 63.71428571428
572, 13.333333333333329, 2.0], [0, 161.5, 13.333333333333329, 10.0]]
```

We used Kafka for Twitter Streaming Output

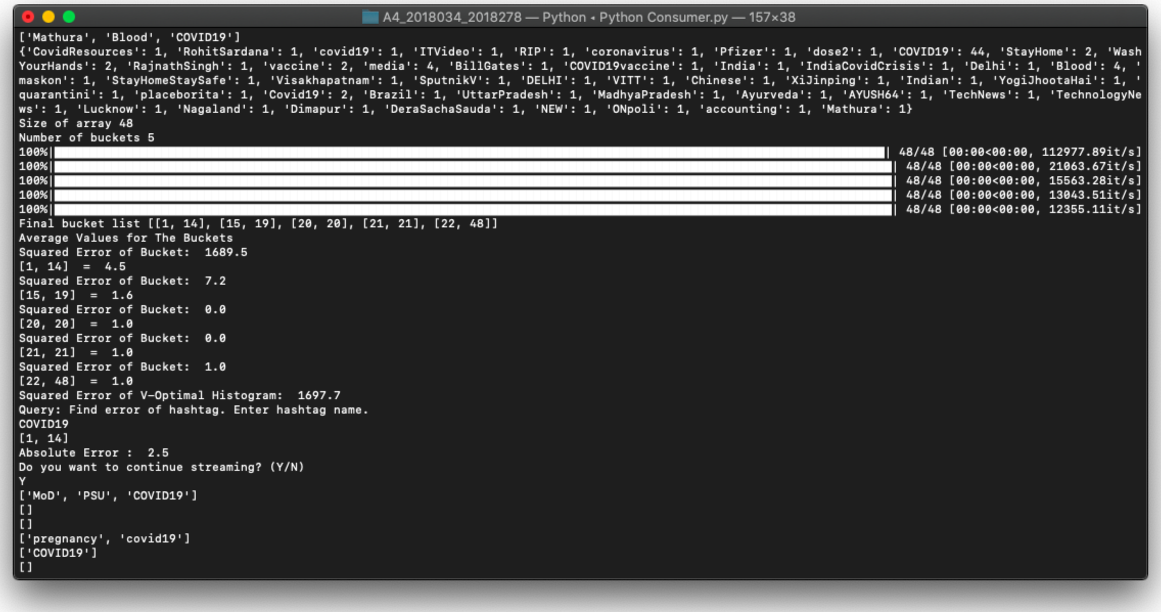


garvita — java -Xmx1G -Xms1G -server -XX:+UseG1GC -XX:MaxGCPaus...

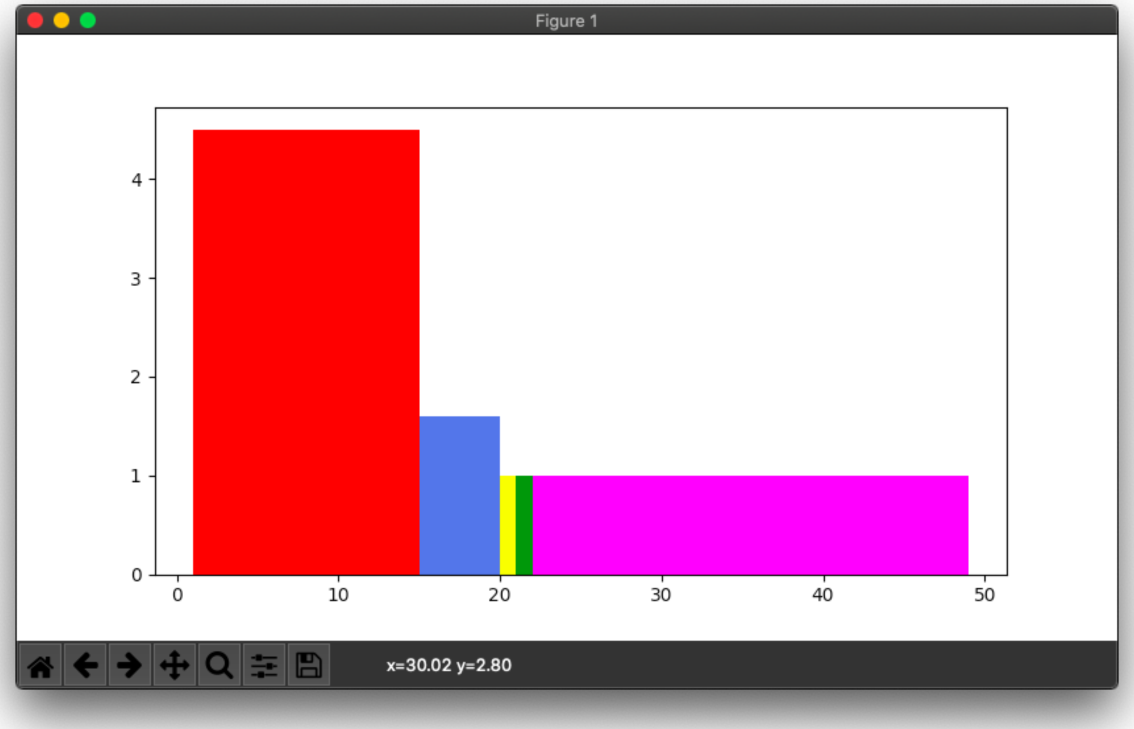
```
garvita@GARVITAs-MacBook-Air-934 ~ % zkServer start
ZooKeeper JMX enabled by default
Using config: /usr/local/etc/zookeeper/zoo.cfg
Starting zookeeper ... STARTED
garvita@GARVITAs-MacBook-Air-934 ~ % kafka-server-start /usr/local/etc/kafk
a/server.properties
[2021-04-30 20:33:35,244] INFO Registered kafka:type=kafka.Log4jController
MBean (kafka.utils.Log4jControllerRegistration$)
[2021-04-30 20:33:35,636] INFO Setting -D jdk.tls.rejectClientInitiatedRene
gotiation=true to disable client-initiated TLS renegotiation (org.apache.zo
okeeper.common.X509Util)
[2021-04-30 20:33:35,768] INFO Registered signal handlers for TERM, INT, HU
P (org.apache.kafka.common.utils.LoggingSignalHandler)
[2021-04-30 20:33:35,777] INFO starting (kafka.server.KafkaServer)
[2021-04-30 20:33:35,778] INFO Connecting to zookeeper on localhost:2181 (k
afka.server.KafkaServer)
[2021-04-30 20:33:35,800] INFO [ZooKeeperClient Kafka server] Initializing
a new session to localhost:2181. (kafka.zookeeper.ZooKeeperClient)
[2021-04-30 20:33:35,810] INFO Client environment:zookeeper.version=3.5.9-8
3df9301aa5c2a5d284a9940177808c01bc35cef, built on 01/06/2021 20:03 GMT (org
.apache.zookeeper.ZooKeeper)
[2021-04-30 20:33:35,810] INFO Client environment:host.name=localhost (org.
apache.zookeeper.ZooKeeper)
[2021-04-30 20:33:35,810] INFO Client environment:java.version=15.0.2 (org.
apache.zookeeper.ZooKeeper)
```



```
garvita@GARVITAs-MacBook-Air-934 A4_2018034_2018278 % python3 Consumer.py
[]
[]
['Delhi', 'Blood', 'COVID19']
[]
['MaskUpKashmir', 'IndianArmy', 'CovidIndiaInfo']
['media', 'COVID19']
[]
[]
['COVID19']
['COVID19', 'Ontario']
['UnrestrictedBiowarfare']
['COVID19']
['COVID19']
['COVID19']
[]
['COVID19']
[]
[]
['SayTheirNames']
[]
['COVID19']
[]
[]
['COVID19']
['COVID19']
[]
```



```
['Mathura', 'Blood', 'COVID19']
{'CovidResources': 1, 'RohitSardana': 1, 'covid19': 1, 'ITVideo': 1, 'RIP': 1, 'coronavirus': 1, 'Pfizer': 1, 'dose2': 1, 'COVID19': 44, 'StayHome': 2, 'Wash
YourHands': 2, 'RajnathSingh': 1, 'vaccine': 2, 'media': 4, 'BillGates': 1, 'COVID19vaccine': 1, 'India': 1, 'IndiaCovidCrisis': 1, 'Delhi': 1, 'Blood': 4, '
maskon': 1, 'StayHomeStaySafe': 1, 'Visakhapatnam': 1, 'SputnikV': 1, 'DELHI': 1, 'VITT': 1, 'Chinese': 1, 'XiJinping': 1, 'Indian': 1, 'YogiJhootaHai': 1, '
quarantini': 1, 'placeborita': 1, 'Covid19': 2, 'Brazil': 1, 'UttarPradesh': 1, 'MadhyaPradesh': 1, 'Ayurveda': 1, 'AYUSH64': 1, 'TechNews': 1, 'TechnologyNe
ws': 1, 'Lucknow': 1, 'Nagaland': 1, 'Dimapur': 1, 'DeraSachaSauda': 1, 'NEW': 1, 'ONpoli': 1, 'accounting': 1, 'Mathura': 1}
Size of array 48
Number of buckets 5
100%|████████████████████████████████████████████████████████████████████████████████████████████████████████████████████| 48/48 [00:00<00:00, 112977.89it/s]
100%|█████████████████████████████████████████████████████████████████████████████████████████████████████████████████████| 48/48 [00:00<00:00, 21063.67it/s]
100%|█████████████████████████████████████████████████████████████████████████████████████████████████████████████████████| 48/48 [00:00<00:00, 15563.28it/s]
100%|█████████████████████████████████████████████████████████████████████████████████████████████████████████████████████| 48/48 [00:00<00:00, 13043.51it/s]
100%|█████████████████████████████████████████████████████████████████████████████████████████████████████████████████████| 48/48 [00:00<00:00, 12355.11it/s]
Final bucket list [[1, 14], [15, 19], [20, 20], [21, 21], [22, 48]]
Average Values for The Buckets
Squared Error of Bucket:  1689.5
[1, 14]  =  4.5
Squared Error of Bucket:  7.2
[15, 19]  =  1.6
Squared Error of Bucket:  0.0
[20, 20]  =  1.0
Squared Error of Bucket:  0.0
[21, 21]  =  1.0
Squared Error of Bucket:  1.0
[22, 48]  =  1.0
Squared Error of V-Optimal Histogram:  1697.7
Query: Find error of hashtag. Enter hashtag name.
COVID19
[1, 14]
Absolute Error :  2.5
Do you want to continue streaming? (Y/N)
Y
['MoD', 'PSU', 'COVID19']
[]
[]
['pregnancy', 'covid19']
['COVID19']
[]
```

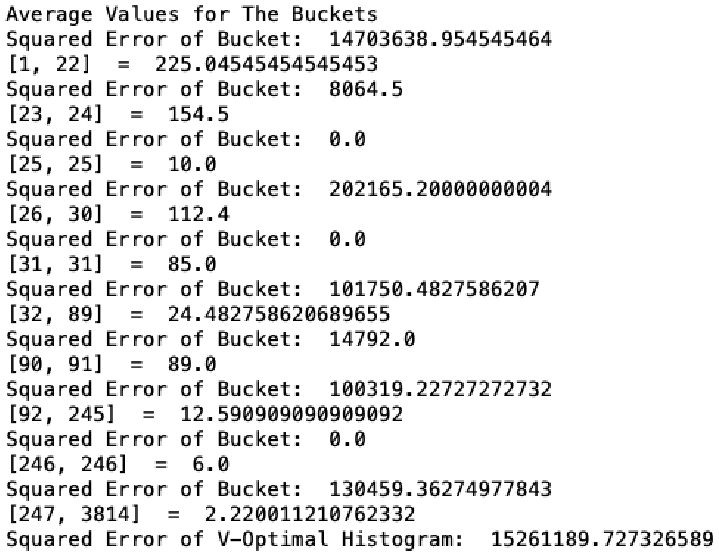


Y-axis : Frequency of hashtag
X-axis : Hashtag by index

Ans 4)

Analyzing the created histogram using some some queries and the error.

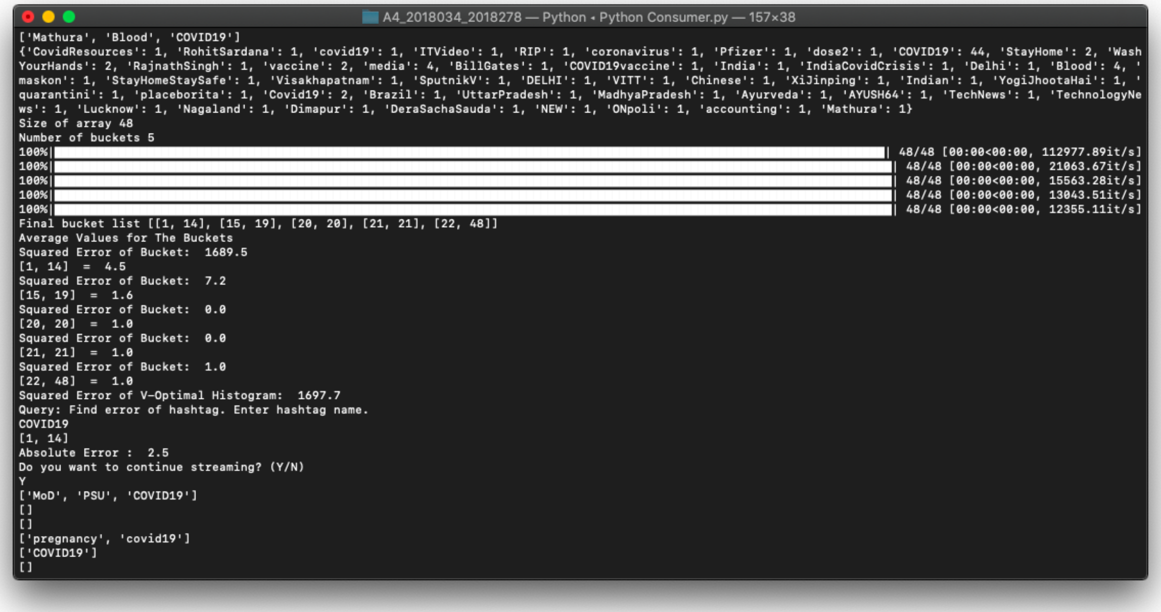
For Jagadish Algorithm for the tweets we collected



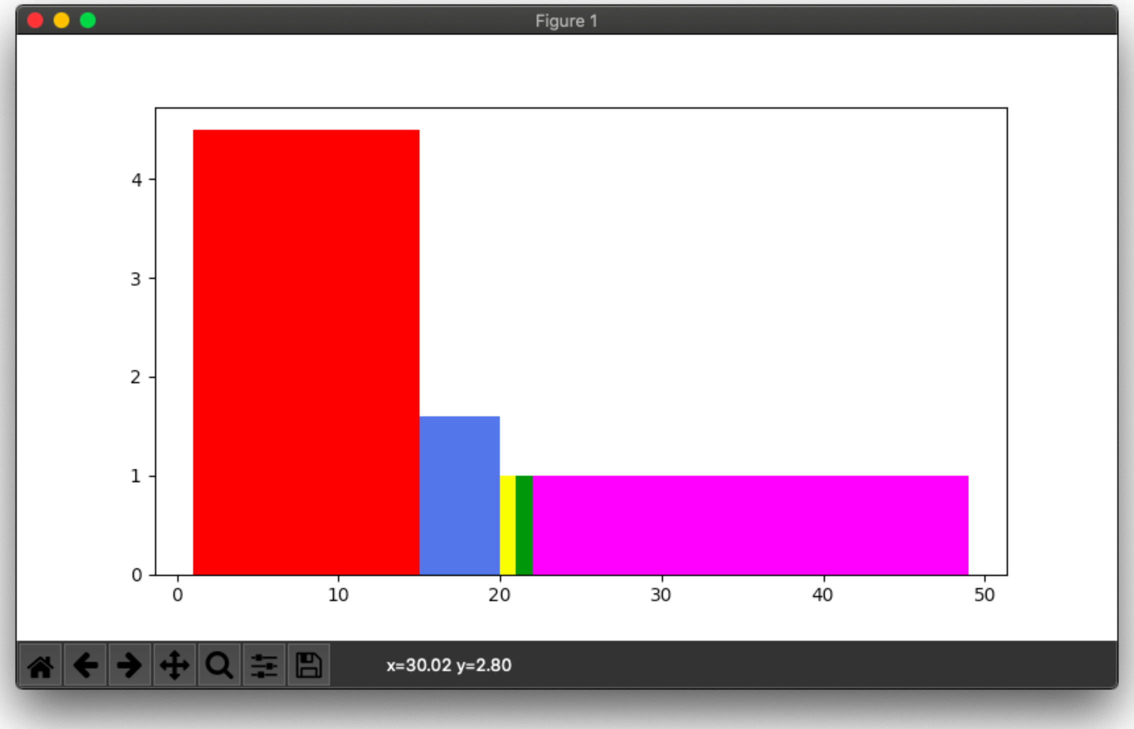
```
Average Values for The Buckets
Squared Error of Bucket:  14703638.954545464
[1, 22]  =  225.04545454545453
Squared Error of Bucket:  8064.5
[23, 24]  =  154.5
Squared Error of Bucket:  0.0
[25, 25]  =  10.0
Squared Error of Bucket:  202165.20000000004
[26, 30]  =  112.4
Squared Error of Bucket:  0.0
[31, 31]  =  85.0
Squared Error of Bucket:  101750.4827586207
[32, 89]  =  24.482758620689655
Squared Error of Bucket:  14792.0
[90, 91]  =  89.0
Squared Error of Bucket:  100319.22727272732
[92, 245]  =  12.590909090909092
Squared Error of Bucket:  0.0
[246, 246]  =  6.0
Squared Error of Bucket:  130459.36274977843
[247, 3814]  =  2.220011210762332
Squared Error of V-Optimal Histogram:  15261189.727326589
```

For Guha Algorithm when we are streaming the data

Here we are reporting the error for each bucket along with the total error, here in this we can restart the query, for demo purposes we are streaming 100 tweets at a time.
We can see this from the below screenshot :



```
['Mathura', 'Blood', 'COVID19']
{'CovidResources': 1, 'RohitSardana': 1, 'covid19': 1, 'ITVideo': 1, 'RIP': 1, 'coronavirus': 1, 'Pfizer': 1, 'dose2': 1, 'COVID19': 44, 'StayHome': 2, 'Wash
YourHands': 2, 'RajnathSingh': 1, 'vaccine': 2, 'media': 4, 'BillGates': 1, 'COVID19vaccine': 1, 'India': 1, 'IndiaCovidCrisis': 1, 'Delhi': 1, 'Blood': 4, '
maskon': 1, 'StayHomeStaySafe': 1, 'Visakhapatnam': 1, 'SputnikV': 1, 'DELHI': 1, 'VITT': 1, 'Chinese': 1, 'XiJinping': 1, 'Indian': 1, 'YogiJhootaHai': 1, '
quarantini': 1, 'placeborita': 1, 'Covid19': 2, 'Brazil': 1, 'UttarPradesh': 1, 'MadhyaPradesh': 1, 'Ayurveda': 1, 'AYUSH64': 1, 'TechNews': 1, 'TechnologyNe
ws': 1, 'Lucknow': 1, 'Nagaland': 1, 'Dimapur': 1, 'DeraSachaSauda': 1, 'NEW': 1, 'ONpoli': 1, 'accounting': 1, 'Mathura': 1}
Size of array 48
Number of buckets 5
100%|████████████████████████████████████████████████████████████| 48/48 [00:00<00:00, 112977.89it/s]
100%|█████████████████████████████████████████████████████████████| 48/48 [00:00<00:00, 21063.67it/s]
100%|█████████████████████████████████████████████████████████████| 48/48 [00:00<00:00, 15563.28it/s]
100%|█████████████████████████████████████████████████████████████| 48/48 [00:00<00:00, 13043.51it/s]
100%|█████████████████████████████████████████████████████████████| 48/48 [00:00<00:00, 12355.11it/s]
Final bucket list [[1, 14], [15, 19], [20, 20], [21, 21], [22, 48]]
Average Values for The Buckets
Squared Error of Bucket:  1689.5
[1, 14]  =  4.5
Squared Error of Bucket:  7.2
[15, 19]  =  1.6
Squared Error of Bucket:  0.0
[20, 20]  =  1.0
Squared Error of Bucket:  0.0
[21, 21]  =  1.0
Squared Error of Bucket:  1.0
[22, 48]  =  1.0
Squared Error of V-Optimal Histogram:  1697.7
Query: Find error of hashtag. Enter hashtag name.
COVID19
[1, 14]
Absolute Error :  2.5
Do you want to continue streaming? (Y/N)
Y
['MoD', 'PSU', 'COVID19']
[]
[]
['pregnancy', 'covid19']
['COVID19']
[]
```



Learnings :

- Understanding different Real-Time Data Streaming softwares.
- Difference between Apache Storm and Apache Kafka (and their installation processes).
- Twitter streaming using Apache Kafka
- V-Optimal Histograms
- Dynamic Programming based algorithms - Jagdish and Guha's algorithm
- Collection of tweets using Tweepy's streaming and search api's
- Implementation of learnings from BDA lecture class into real life scenarios by using actual data from twitter
- Using Twitter developer account

Resources used for this assignment :

For Jagadish algorithm implementation

- http://www.mathcs.emory.edu/~cheung/Courses/584/Syllabus/06-Histograms/Progs/
- http://www.mathcs.emory.edu/~cheung/Courses/584/Syllabus/06-Histograms/OLD-v-opt3.html

For Kafka algorithm implementation

- http://www.mathcs.emory.edu/~cheung/Courses/584/Syllabus/06-Histograms/guha.html
- http://www.mathcs.emory.edu/~cheung/Courses/584/Syllabus/06-Histograms/

For installation of Kafka

- https://towardsdatascience.com/getting-started-with-apache-kafka-in-python-604b3250aa05
- https://www.techwasti.com/installation-of-apache-kafka-on-macosx/
- https://www.bmc.com/blogs/working-streaming-twitter-data-using-kafka/
- https://towardsdatascience.com/running-zookeeper-kafka-on-windows-10-14fc70dcc771
- http://www.mathcs.emory.edu/~cheung/Courses/584/Syllabus/06-Histograms/Progs/
- https://www.tutorialspoint.com/apache_kafka/apache_kafka_installation_steps.htm

---