# Streaming setup

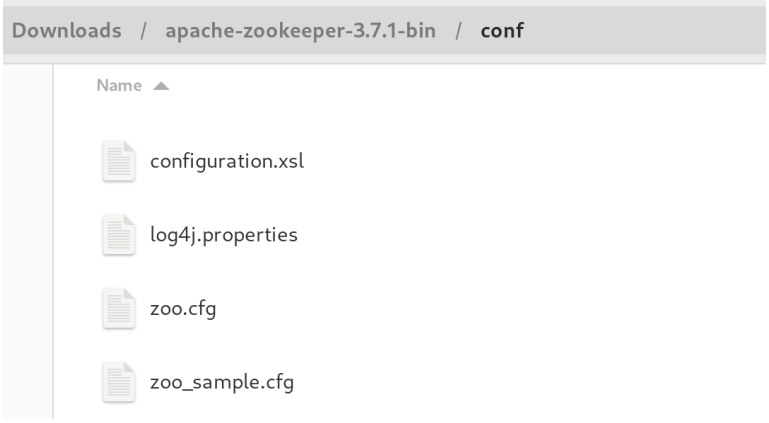
** forget the mandril, now we are stream clustering!!

## Part 0.0: install python libraries,

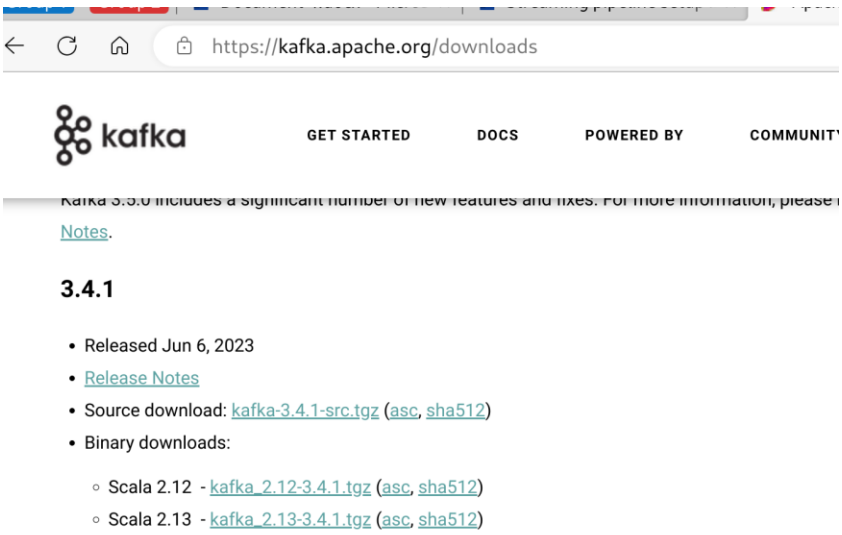
1. Create a new virtual environment named “stream”, activate the stream environment, use the same environment inside VS Code and run the following commands:
    a. pip install opencv-python
    b. pip install kafka-python
    c. pip install loguru
    d. Also install NumPy and pandas

## Setting up Apache zookeeper and Kafka

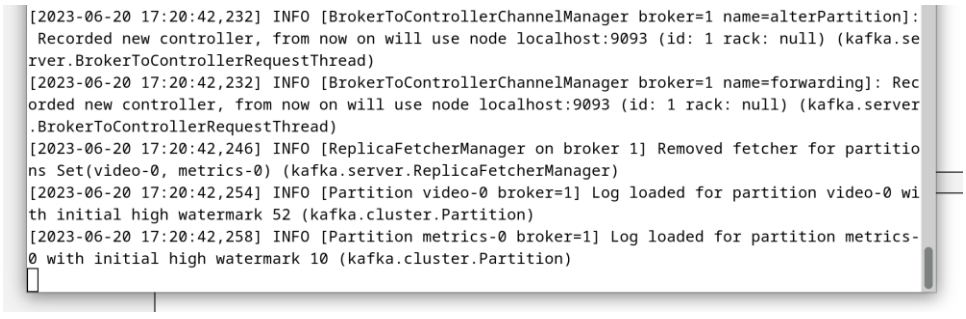
## Part 1: Setup Apache zookeeper

1. Since you have downloaded zookeeper. Copy zoo.cfg file that I have provided to here 


2. Then Go to this page and download Kafka version 2.13_3.4.1.tgz in the last link in the image, click here




3. Extract the downloaded file
4. Then go inside the extracted folder, you will find a config folder,
5. Paste server.properties and zookeeper.properties files inside the config folder
6. Now open a terminal window and run following commands in order,
7. I AM ASSUMING YOU ARE IN YOUR HOME DIRECTORY IN YOUR TERMINAL,
8. TO CHECK, RUN: pwd
9. And you have put your downloaded files in your Downloads folder.
    a. Run: cd Downloads/
    b. Run: cd apache-zookeeper-3.7.1-bin/
    c. Run: bin/zkServer.sh start conf/zoo.cfg
    d. Don't close this window and open another terminal tab
    e. Run: cd ~
    f. Run: cd Downloads/
    g. Run: cd Downloads/kafka_2.13-3.4.1
    h. Run: bin/kafka-server-start.sh config/server.properties
    i. This will start the kafka broker, it will look like this




**A BLINKING CURSOR INDICATES THE BROKER IS RUNNING.**

    j. Don't interrupt this window or close this window, we need this running for streaming,

**k. THATS IT, now we are ready to do real time stream clustering**

# Running the notebooks

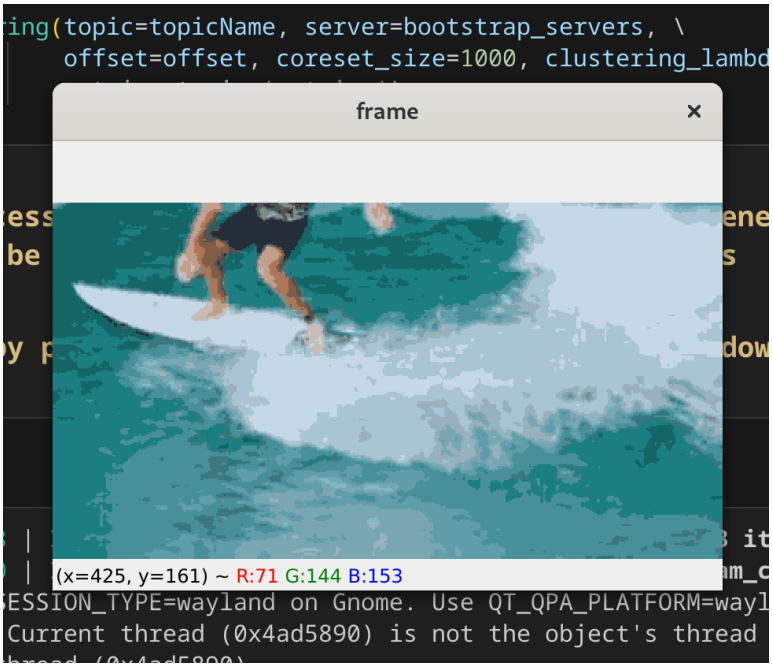
1. Run all the cells of streaming_metrics notebook **except the last cell.**
2. If you want to c**hange the value of coreset_size and lambda**, go to the last cell of video consumer notebook, and change the values, and save the changes
3. **lambda value** represents the **size of the clusters** we want, higher value of lambda will give less and less accurate predictions, for lower values of lambda, the result will resemble the original video.
4. Each frame of the video has around 96000 pixels, for demonstration, I have selected the coreset size of 1000, for all practical purposes this is too low, the ideal value is

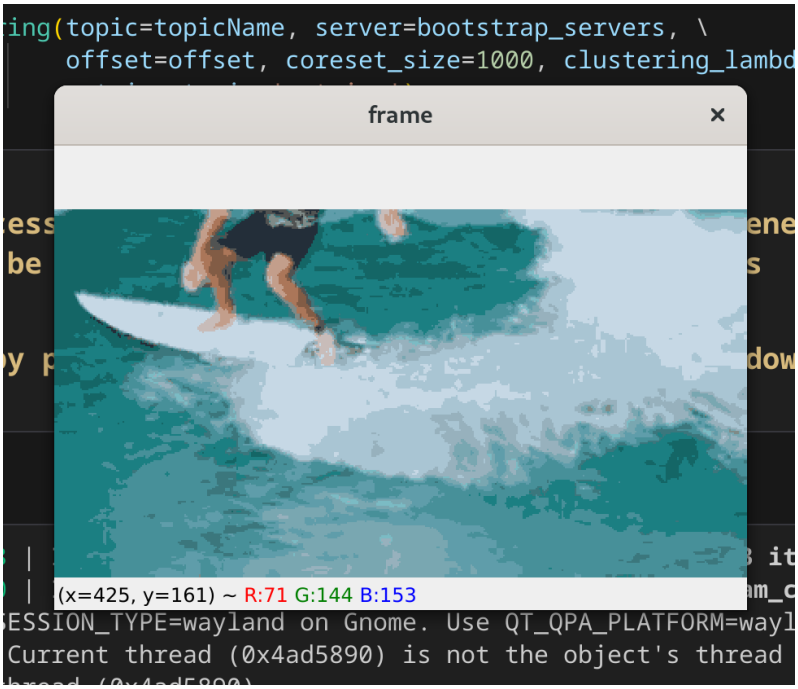
$$\frac{k \log n}{\delta^{d/2} \varepsilon^{d}} \log^{d/2}\left(\frac{k \log n}{\delta^{d/2} \varepsilon^{d}}\right)$$

calculated as

Where n = no of pixels, d = no of dimensions, k = no of clusters, delta and epsilon are both selected by user and they represent error tolerance and confidence interval.

5. After running all the cells of streaming metrics notebook **except the last cell**, run all the cells of video consumer notebook and then run all the cell of video producer notebook
6. Check all the three notebook and see if any error hasn't occurred,
7. If no error occurred, a new frame window will get created and processed frames will start to appear, the results will look like this:




NOTICE THE DIFFERENCES BETWEEN TWO IMAGES,

When the real time stream clustering starts it will look like a video.

Inside the streaming metrics notebook, a log will be generated like below, it shows how many colors were detected in each frame,

```
2023-06-20 17:22:24.119 | INFO     | __main__:<module>:19 - Number of colors detected: 13
2023-06-20 17:22:29.377 | INFO     | __main__:<module>:19 - Number of colors detected: 17
2023-06-20 17:22:34.638 | INFO     | __main__:<module>:19 - Number of colors detected: 14
2023-06-20 17:22:39.577 | INFO     | __main__:<module>:19 - Number of colors detected: 13
2023-06-20 17:22:47.703 | INFO     | __main__:<module>:22 - Streaming interrupted


Calculating silhouette score is very very computationally expensive
Try out a single frame by running the cell below
```

**HOW TO STOP THE STREAM?**

1. **I have configured such that when you press Esc, the streaming stops,**
2. **So press Esc when you are on the video streaming window**

Last cell of video consumer notebook will look like this:

```
2023-06-20 18:26:46.616 | INFO     | Algorithms.dcdp:fit:47 - DCDP took 36 iterations to converge
2023-06-20 18:26:46.790 | INFO     | __main__:wrapper:7 - Runtime of stream_clustering is 4.085196018218994
2023-06-20 18:26:51.736 | INFO     | Algorithms.dcdp:fit:47 - DCDP took 50 iterations to converge
2023-06-20 18:26:51.829 | INFO     | __main__:wrapper:7 - Runtime of stream_clustering is 4.0234479904174805
```

As you can see, we are logging the runtime cost of clustering each frame as it arrives and no of iterations dcdp took to converge

3. **This will stop video consumer notebook's last cell,**
4. **Now stop the last cell of video producer notebook**
5. **Then stop the 2nd last cell of streaming metrics notebook.**
6. **Now run the last cell of streaming metrics notebook, the result will look like this,**

**This is the silhouette diagram,**

1. On the y axis we have the labels from 0 to 12, where 12 in the no of colors detected in the last frame I received from kafka.
2. The size of each knife shows how many pixels each cluster contains.
3. Since we detected 12 colors, there are 12 clusters
4. You will also notice that on the left the knives are dropping out, this shows the how poorly each cluster was formed, higher dropoff means bad clustering quality.
5. Try **different values of lambda, rerun the stream clustering,** and **observe** the change in the **shape** of the knives and the **average silhouette score** (which is represented by the red dotted line)
6. NOTE: the video is 10 seconds long, and each second contains 30 frames, for a total of 300 frames,
7. **NOTE: CALCULATING SILHOUETTE SCORE IS COMPUTATIONALLY EXPENSIVE,**

8. **ON A QUAD CORE SYSTEM, IT TAKES 6 MINUTES TO CALCULATE SILHOUETTE SCORE ON ONE FRAME,**