Date: 19-02-2023

***BIG DATA ANALYTICS: Assignment #1 – Report***

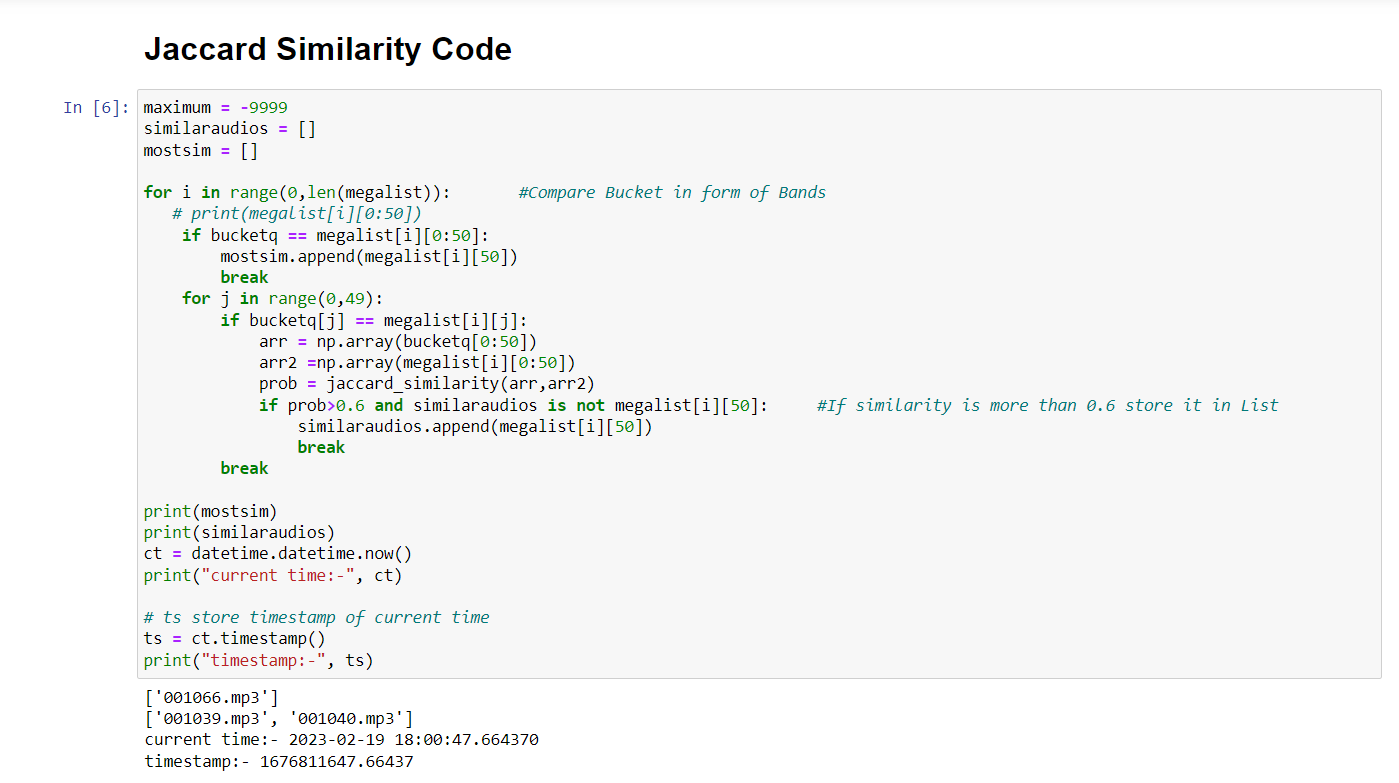
***Group Members:***

* Mubashir Hassan i21-1764
* Mohammad Osman i21-1723
* Abdulmoiz i21-2687

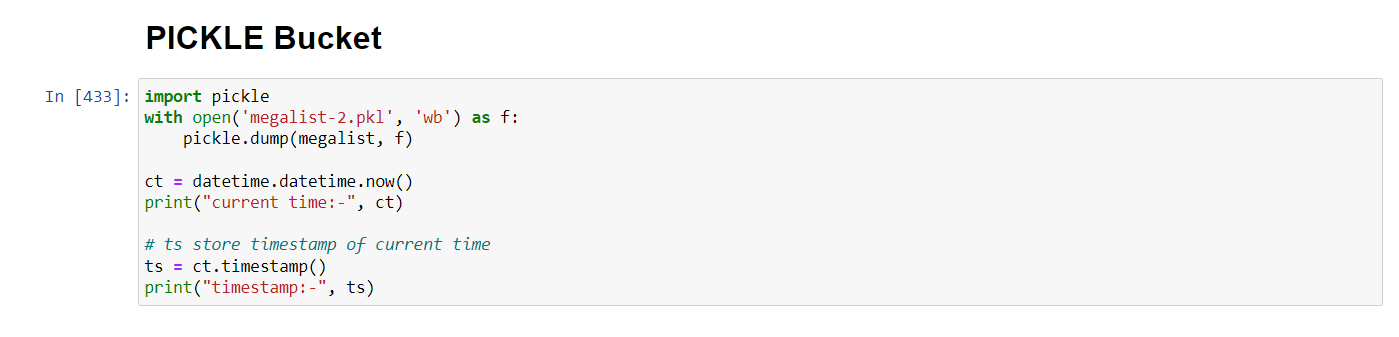
**Question 1 Explanation:**

First of all, we tried to import ‘data sketch’ and convert mfccs into string to do min hash and then implement LSH table to find similarities. But the problem with this was that it was only returning us the exact similar audios. The rest of the audios were not recognized percentage wise. From that, we got to know that we will have to implement it by ourselves through mfccs.

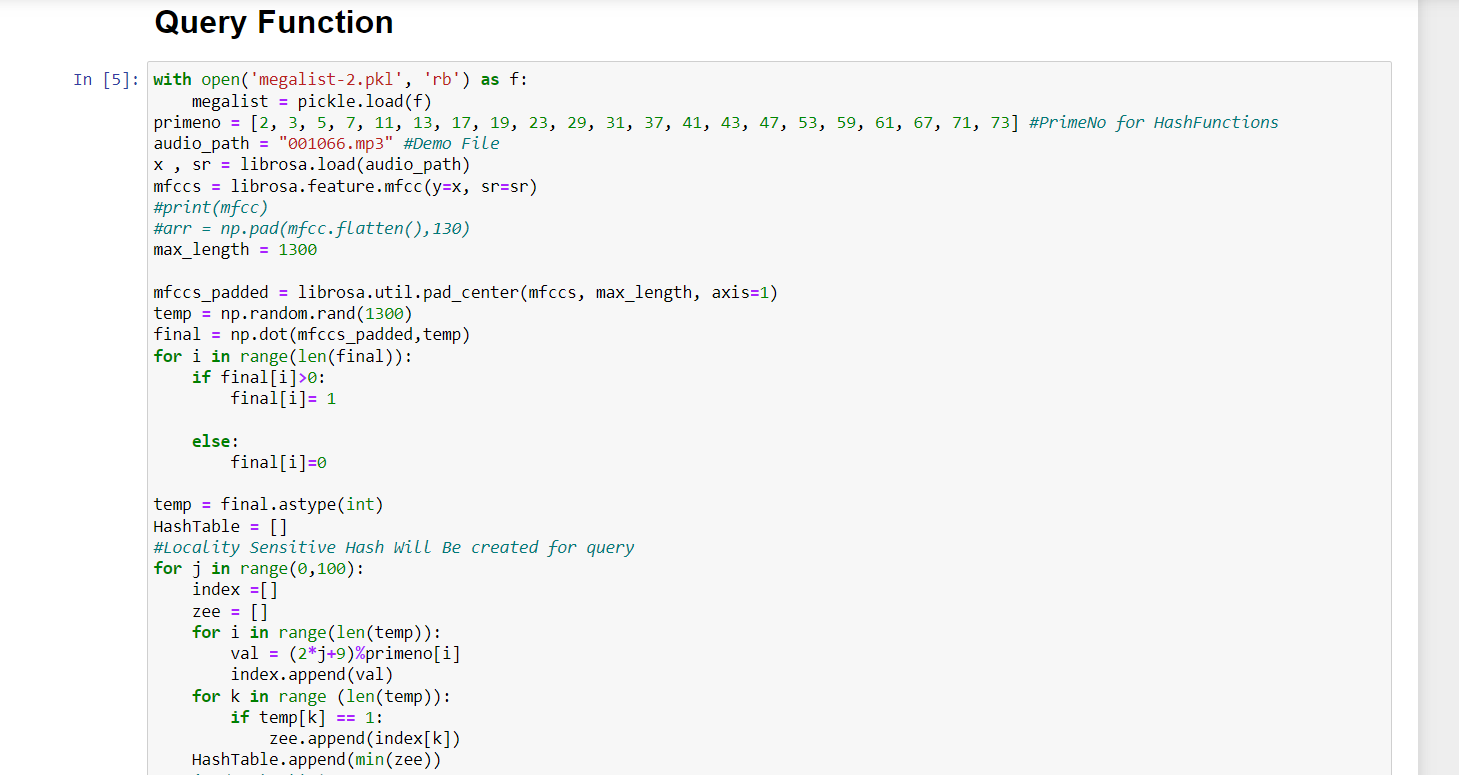
Now, as our mfccs were in 2D arrays, we made a temporary NumPy random array and multiplied the two arrays so that the array comes to twenty. If the number is negative in the mfcc, then we set it to zero and if the number is positive then set it to one. Now, it is mapped in the form of zeros’ s and one’s. The min hash is created by randomly generated quadratic functions that goes in min hashes and creates another table on another index. After all of that, our LSH table is ready.



Through LSH tables, we made buckets. Buckets are made that the common values in the audio files are combined and put in a different dictionary. But it was difficult to append in the dictionary. So, we put those audios in an arranged form of list. Now the audios are in the bucket. LSH is a technique used to find similar items in a large dataset efficiently. It works by hashing similar items into the same "bucket" with a high probability, so that when a query is made, we only need to compare the items in the same bucket as the query, rather than all the items in the dataset. This greatly reduces the time and computational resources required to find similar items.

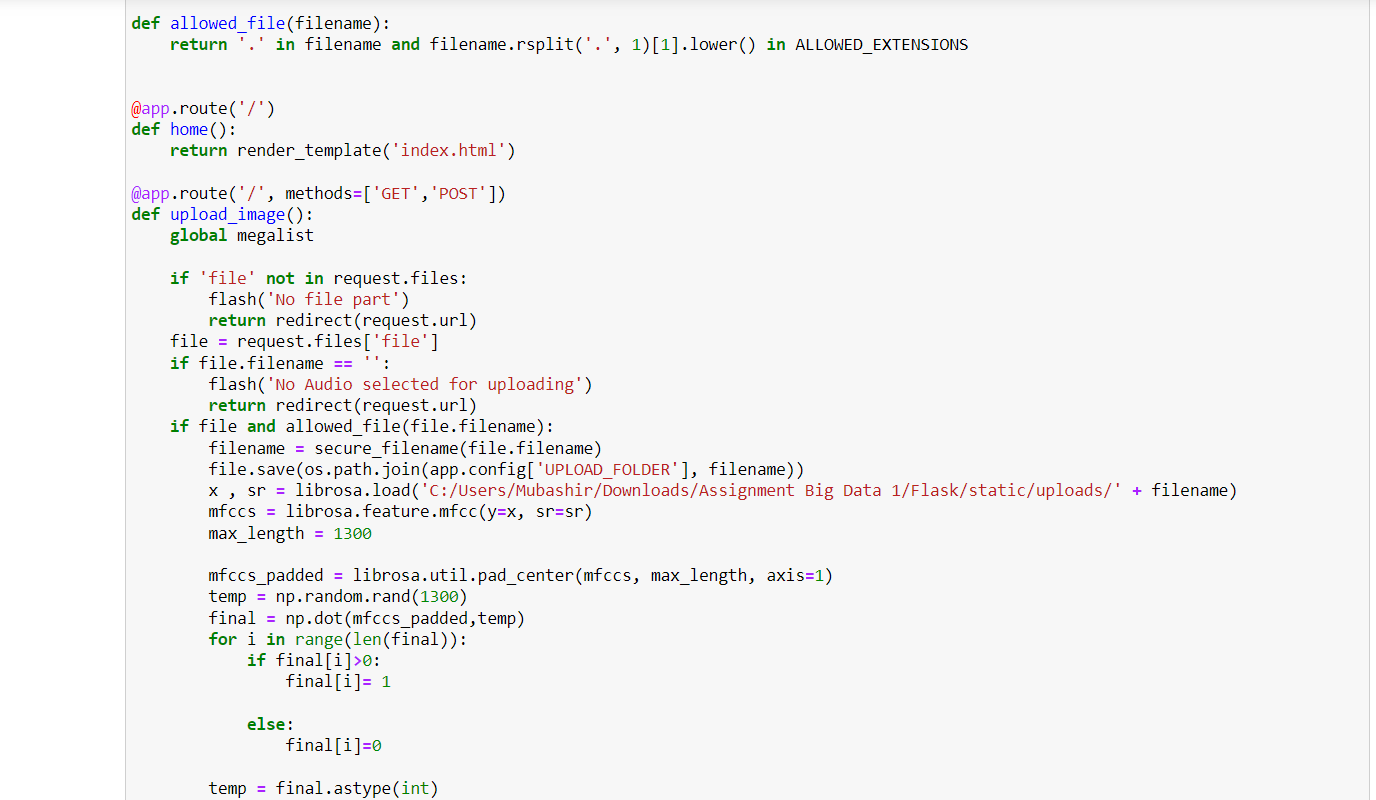


After that, the query function is applied and after that, Jaccard similarity to which query function’s band was matching and calculating the Jaccard similarity is implemented in the flask.



**Flask Implementation:**

There is one web page and one html page. The audio is processed through static upload file. In the static upload file, the URL comes back again on the html file and gets the file name. Through the file name, the whole procedure of the query function runs. After the query function is called, it tells us the similarity through the Jaccard function i.e., how much the file is similar. After that, we pickled the buckets of the mfccs and upload it on the flask front end display.



**Question 2 Explanation:**

In Question 1, the goal was to find the most similar audio file based on a query audio file. However, in Question 2, we want to display not only the most similar audio file, but also other audio files that have some degree of similarity with the query audio file, even if they are less similar or less accurate. This is where Locality Sensitive Hashing (LSH) comes in handy.

In the context of audio files, LSH can be used to group together audio files that have some degree of similarity with the query audio file, even if they are not an exact match. This allows us to display a range of similar audio files to the user, rather than just the most similar one