A. **General Overview**

This system is designed to manage and interact with a collection of tweets stored in a MongoDB database. It consists of two main phases:

1. Building a document store: a program named load-json is provided. This program takes a JSON file containing tweet data as input and constructs a MongoDB collection. It connects to a MongoDB server using a specified port, creates a database named 291db, and builds a collection named tweets. The program inserts data into MongoDB in small batches using the insert_many command.
2. Operating on the document store: a program is designed to support various operations on the MongoDB databases created in the first part. Users can search for tweets and users, list top tweets and users, and compose new tweets. A main menu is provided for users to interact with these functionalities.

User guide:

1. Connect to the databases: provide the JSON file name and MongoDB server port as command-line arguments.
2. Perform operations: select an option(search for tweets or users, list top tweets or users, or compose new tweets) from the main menu and follow the prompts to perform the desired operation. Then follow the instructions to input the keywords, numbers or contents.
3. Perform further operations or exit: after each operation, users have the option for further actions or exit.

B. **Details of Algorithms**

1. Search_tweets: The search for tweet function searches for keywords in the content field of the data, so we can create an index to efficiently search the keywords within the field. The function will ask the user for keywords and will then perform the search. If no tweets are found, the function will display 'no tweets found' and will return to the main menu. If there are tweets found that match the keyword (case insensitive), the function will display the tweets and offer the user the option to go back to the main menu or see more information about a specific tweet by entering its tweet_id.
2. List_top_tweets: This function will ask the user to select a field to rank the tweets by (retweetCount, likeCount, quoteCount). The function will first ask for user-input. The user-input must exactly match the field (case sensitive). The function will then ask the user for the number of tweets to list. The function will sort the tweets using collection.find().sort(field, -1).limit(n). Once the tweets are listed in descending order, the user will be given the same option to either exit to the main menu or see full information about a specific tweet, and the function will call the get_tweet_stats() function.
3. Compose_tweets: This function will ask the user to input the tweet content. And then create a new and unique id for the new tweet. Then it will be inserted into the database.
4. Search_users: This function searches for users based on a keyword the user inputs (case insensitive). After querying the database and retrieving all the users that match the keyword, the function will filter out the duplicate users, and display the rest. After the search, even if no users

are found, the function will still give the user the option to enter the username of a specific user to see all information on him/her.

5. List_top_users: This function ranks the users on followersCount and displays them in descending order. The function will first ask the user to input an integer as the maximum number of users to display, and then it will rank the users based on followersCount and remove duplicates. After that, users will be displayed with their followersCount and the user can choose to exit to the main menu or to see more information about a user.

C. Testing Strategy

1. At the start load-json.py was tested with 10.json to check whether the database was getting initialized correctly or not. After that the same json file was used to check the functions to see whether correct output was obtained or not. This was the stage where most of the functions were created and tested for the first time.
2. The same strategy was used for the second phase of testing where rather than using 10.json, 100.json was used to check whether the function worked as intended or not.
3. The last stage of testing was using the farmers-protest-tweets-2021-03-5.json and the code was checked for any final errors or edge cases. The code was also evaluated and tested according to the marking rubric provided on the eclass and any errors found were fixed immediately.

D. Source Code Quality

Structure and Organization: The code is structured into functions, each handling a specific task, which aids in readability and maintainability. This modular approach facilitates easier updates and debugging.
Comments and Documentation: The code has a lot of comments, making it easier for a new developer to understand the purpose of certain blocks or the logic behind specific implementations.

E. Group Work Strategy

Priyanshu (pusola) : worked on the creation of the load-json.py and was responsible for checking and fixing all the functions that were uploaded by the other group mates. Time spent: 4-5 hours on creation of load-json.py. Further 9-10 hours to debug the code.
Hao (hluo6) : worked on the creation of the function compose_tweet. Time spent: 4-5 hours on creation of the function. Further 9-10 hours to debug the code.
Haoze (haoze7) : worked on the creation of the functions search_tweet, list_top_tweet, search_user and list_top_users and its helper function. Time spent: 6-7 hours on creation of the function. Further 9-10 hours to debug the code.
Komal (kjot) : worked on the creation of the functions search_user and list_top_users. Time spent: 2-3 hours on creation of the functions.

All the group members were equally responsible for testing the code and reporting the errors to the group on a regular basis with the use of discord which was the primary mode of communication between all the members. All the given tasks to group members were completed on time, incase of late submission, prior notice was given by the group-mates to each other in order to accommodate for midterms and other assignments. All the functions work as intended.

F. Assumption and Limitations

Assumptions:-
MongoDB Database Structure: The code assumes a specific structure for the MongoDB collections (e.g., fields like id, date, content, user.username in the tweets collection). Changes in the database schema could break the code.
Local MongoDB Instance: The script assumes the MongoDB instance is running locally (MongoClient('localhost', int(port))). It may not work with a remote or cloud-based database without modifications.
Python Environment: It assumes Python is installed with necessary libraries (pymongo) and that the script is run with a command-line argument specifying the port.
Data Types and Formats: The code assumes specific data types (e.g., id as integer, date in a certain format) and will malfunction if the actual data differs.
Input Validity: The script assumes that user input is mostly correct and does not extensively validate it (except for some basic checks like ensuring tweet ID is an integer).
Case Sensitivity: In functions like list_top_tweets, certain inputs (e.g., retweetCount) are case-sensitive, assuming the user will input them correctly.

Limitations : -
Limited Search Functionality: Search functionality is basic and might not be effective for complex queries.
Dependency on System Time: In compose_tweet, it uses the system's local time, which might not be accurate or consistent in different environments.
Limited Update Functionality: The script focuses on reading and inserting data, with no options for updating or deleting existing records.
User Interface: It uses a simple command-line interface which might not be user-friendly for all users. There's no graphical interface.
Limited on Large Databases: When creating a new id in compose_tweet, it goes through the tweets that exist, which means it will spend too much time on it.