INTRODUCTION

1.1 INTRODUCTION

With over 2 billion logged-in monthly users and more than a billion hours of video watched every day, YouTube has emerged as one of the most dominant digital platforms in the world. As a multimedia content-sharing website, YouTube allows millions of creators to upload videos, which are then consumed by users across the globe. This massive user base, combined with an ever-growing volume of content, has created a complex ecosystem that generates a wealth of data daily. This data spans a wide array of metrics, including video views, comments, likes, shares, user demographics, and engagement patterns, which, if properly analyzed, can provide valuable insights into content performance, audience behavior, and emerging trends. However, managing and extracting actionable intelligence from this enormous and continually expanding volume of data presents significant challenges, both from a technological and analytical perspective.

To make sense of this vast repository of information, there is a pressing need for effective **data harvesting** and **warehouse management systems**. Data harvesting refers to the process of systematically collecting, aggregating, and storing content-related data from YouTube's extensive platform, often in real-time. This process involves extracting raw data, which is then transformed into usable formats for analysis and decision-making. On the other hand, a **data warehouse management system** is crucial for organizing, storing, and retrieving large datasets in an efficient and scalable manner. These systems support various querying and reporting functions, allowing users to derive meaningful insights from the data collected.

In this context, the **YouTube Data Harvesting and Warehouse Management System** aims to solve the challenges associated with data collection, storage, and analysis by providing a framework that facilitates seamless data extraction and efficient management of YouTube-related metrics. The goal is to enable businesses, content creators, marketers, and researchers to access structured data and gain insights into patterns of content consumption, audience interaction, and engagement. By analyzing the harvested data, users can make informed decisions about content strategy, audience targeting, advertising, and platform optimization. In particular, data harvesting and warehouse management systems make it possible to track and analyze trending videos, audience preferences, user behavior, content virality, and much more, thus playing a critical role in driving the growth of YouTube as a platform.

THE NEED FOR DATA HARVESTING ON YOUTUBE

YouTube's massive data infrastructure can be daunting to navigate. Although YouTube provides an API that allows developers to access certain data points, this API is limited in scope and often does not meet the needs of businesses or researchers looking to collect comprehensive data for large-scale analysis. For example, the YouTube API provides access to video metadata (such as title, description, and publication date), user interaction data (views, likes, comments), and some audience data, but it does not offer granular details such as user sentiment, geographic breakdowns, or detailed watch history across all users. This limitation necessitates the development of more advanced data harvesting systems that can scrape or pull data from multiple sources within the platform and present it in a coherent format.

Data harvesting systems can gather more detailed and specific metrics about YouTube content, such as video performance over time, detailed audience demographic breakdowns, and engagement metrics across multiple videos. Moreover, these systems can collect data about external interactions with YouTube videos, such as shares, embeds, and social media mentions. This granular level of data can provide key insights into how a video or channel is performing in real-time, enabling content creators to adjust their strategies or marketing teams to optimize ad targeting.

For example, content creators can analyze the time of day when their audience is most active, allowing them to schedule their video uploads accordingly for maximum visibility. Similarly, brands and advertisers can use audience demographic data to tailor their advertising strategies to specific user groups, improving the effectiveness of ad campaigns. In essence, data harvesting ensures that relevant and up-to-date information is continuously available to stakeholders.

DATA WAREHOUSE MANAGEMENT: ORGANIZING AND STORING DATA

While harvesting data from YouTube is a crucial first step, effectively managing that data for analysis is equally important. The raw data collected from YouTube can be vast and unstructured, making it difficult to extract actionable insights without proper organization and storage. This is where the concept of a **data warehouse** becomes important.

A **data warehouse** is a centralized repository that stores data from different sources in a structured and organized way, allowing for easy querying and analysis. For a YouTube data warehouse, this means organizing data into tables or databases that categorize key performance indicators (KPIs), user demographics, content engagement metrics, and more. Data warehouses typically store

historical data, enabling longitudinal analysis over extended periods.

One of the key challenges in designing a YouTube data warehouse is ensuring scalability. Given the constantly increasing volume of content and user interactions, the warehouse must be capable of handling vast amounts of data without compromising performance. This requires the use of modern storage technologies, such as cloud-based data warehouses, distributed databases, and data lakes, which allow for flexible scaling of storage capacity and processing power. Furthermore, ensuring that the data warehouse is optimized for performance means implementing robust indexing, query optimization, and data partitioning strategies, which enable users to retrieve data efficiently without long processing times.

Effective data warehouse management also involves ensuring that data remains clean, consistent, and up-to-date. This requires implementing processes such as data validation, deduplication, and error correction. Additionally, considering privacy and data protection regulations, such as GDPR, is crucial when managing user data. A well-managed data warehouse ensures that the data remains secure, accurate, and reliable for analysis.

THE IMPORTANCE OF ANALYTICS AND INSIGHTS

Once the YouTube data has been harvested and stored in a warehouse, the next step is analysis. With the right tools in place, businesses and content creators can use analytics to gain valuable insights into user engagement, content trends, and audience preferences. These insights can inform decision-making across a variety of domains, from marketing and content creation to platform optimization.

For example, content creators can track the performance of individual videos across different segments, identifying which topics, formats, or production styles resonate most with their audience. Similarly, brands can analyze audience behavior patterns to develop targeted advertising strategies, ensuring that they reach the right viewers at the right time with personalized content.

Additionally, YouTube’s recommendation algorithm plays a crucial role in determining which content gets recommended to viewers, making understanding this system imperative. By analyzing data from the platform’s recommendation system, businesses and content creators can optimize their content to appear more frequently in recommended videos, increasing visibility and engagement.

1.1 OBJECTIVE

The **YouTube Data Harvesting and Warehouse Management System** aims to address the challenges associated with collecting, storing, and analyzing large-scale data from YouTube in an organized and efficient manner. This system serves as a comprehensive framework for gaining valuable insights into video performance, user engagement, and audience behavior on the platform. Below are the key objectives of implementing such a system:

1. EFFICIENT DATA COLLECTION FROM MULTIPLE SOURCES

OBJECTIVE: To build a robust and scalable data harvesting mechanism that can extract a wide range of metrics from YouTube in real-time.

DESCRIPTION: The system should gather relevant data from YouTube's API, as well as from supplementary sources, such as web scraping or social media platforms, to collect both internal and external engagement data. Key data points may include video views, likes, comments, shares, subscriptions, demographics, watch time, and content metadata.

GOAL: To ensure comprehensive data collection across a variety of content types and user interactions that reflects the dynamic nature of the platform.

2. DATA AGGREGATION AND STRUCTURING FOR ANALYSIS

OBJECTIVE: To aggregate raw data into structured formats that are suitable for querying and analysis.

DESCRIPTION: The raw data harvested from YouTube must be processed, cleaned, and structured into a usable format for deeper analysis. This involves filtering out irrelevant data, handling missing or incomplete information, and ensuring consistency across different datasets.

GOAL: To create a data structure that supports efficient querying and insights generation, making it easier for users to analyze trends, audience behavior, and video performance.

3. DEVELOPMENT OF A SCALABLE DATA WAREHOUSE

OBJECTIVE: To establish a scalable and efficient data warehouse capable of storing large volumes of YouTube data while ensuring fast retrieval and minimal latency.

DESCRIPTION: The data warehouse must support the growing volume of data generated by

YouTube over time. This includes adopting cloud-based or distributed storage solutions, implementing efficient indexing techniques, and ensuring the system scales dynamically as new data is added.

GOAL: To ensure the system can handle large datasets and high-frequency data updates without performance degradation, while providing a seamless user experience for analysts and content creators.

4. OPTIMIZATION FOR HIGH-PERFORMANCE DATA RETRIEVAL

OBJECTIVE: To optimize the querying process for fast data retrieval and analysis.

DESCRIPTION: Given the large scale of data involved, query optimization becomes a key objective. By implementing advanced indexing methods, data partitioning, and caching mechanisms, the system should enable users to run complex queries (e.g., trend analysis, sentiment analysis) without significant delays.

GOAL: To reduce the time required to access and analyze data, thus providing timely insights into YouTube content performance and user engagement.

5. FACILITATE REAL-TIME DATA PROCESSING AND ANALYTICS

OBJECTIVE: To enable real-time data processing for continuous tracking of YouTube metrics.

DESCRIPTION: In addition to batch processing, the system should support real-time data feeds that allow businesses and content creators to monitor video performance and audience interactions instantaneously. This will allow for quicker decision-making, such as modifying content strategies or targeting specific audience segments based on real-time data.

GOAL: To provide up-to-date insights that reflect the current state of the platform, helping users stay competitive and responsive to emerging trends or shifts in audience behavior.

6. ADVANCED DATA ANALYSIS AND REPORTING

OBJECTIVE: To provide powerful analytics tools that help users extract actionable insights from the data collected.

DESCRIPTION: The system should incorporate advanced analytics features, including data visualization (e.g., graphs, charts, heatmaps), trend analysis, and predictive modeling. Users should

be able to generate custom reports on key performance indicators (KPIs), video engagement, audience demographics, and more.

GOAL: To empower content creators, marketers, and analysts to make informed decisions based on data-driven insights, improving content creation, marketing strategies, and overall platform engagement.

7. DATA SECURITY AND PRIVACY COMPLIANCE

OBJECTIVE: To ensure that the data collected, stored, and analyzed complies with privacy regulations and is securely protected.

DESCRIPTION: As YouTube data can include sensitive user information, it is critical to ensure that the system adheres to relevant privacy laws and regulations (e.g., GDPR, CCPA). This includes encrypting personal data, implementing access controls, and ensuring data anonymization where necessary to protect user privacy.

GOAL: To provide a secure environment for data storage and processing that adheres to legal standards and safeguards the integrity and confidentiality of user data.

8. PROVIDE PERSONALIZED CONTENT AND ADVERTISING INSIGHTS

OBJECTIVE: To generate insights into content preferences and advertising effectiveness for targeted campaigns.

DESCRIPTION: With the rich data collected, the system should help marketers and content creators understand user interests, engagement patterns, and demographic preferences. It should support the identification of high-performing content and audience segments, which can inform personalized content recommendations and targeted advertising.

GOAL: To help brands optimize their advertising campaigns and assist content creators in refining their content strategies based on user interests and engagement trends.

9. TREND AND SENTIMENT ANALYSIS

OBJECTIVE: To identify emerging trends and perform sentiment analysis on content and user feedback.

DESCRIPTION: By analyzing comments, social media interactions, and viewership data, the

system should be capable of detecting trending topics, popular videos, and shifts in audience sentiment. This can be particularly useful for brands looking to ride the wave of viral content or identify key themes that resonate with viewers.

GOAL: To provide stakeholders with predictive insights that can help guide content development, product launches, and marketing strategies.

10. SUPPORT FOR CONTINUOUS SYSTEM IMPROVEMENT

OBJECTIVE: To create a flexible and adaptable system that evolves with the platform's changing data landscape.

DESCRIPTION: YouTube's algorithms, features, and data sources continuously evolve. As such, the data harvesting and warehouse management system must be designed with scalability and adaptability in mind. Regular updates to the system, as well as the ability to incorporate new data points, ensure that the system remains effective over time.

GOAL: To maintain the system's relevance and accuracy by incorporating new features, data sources, and technologies as YouTube's platform and data ecosystem evolve.

1.2 MODULES

Creating a system for **YouTube Data Harvesting and Warehouse Management** involves designing modules that can efficiently gather data from YouTube, store it in a structured format, and manage the data for further analysis or reporting. Here's a breakdown of possible modules that could be part of such a system:

1. YOUTUBE DATA HARVESTING MODULES

These modules will focus on scraping, accessing, and collecting data from YouTube, either via YouTube API or other scraping methods.

1.1 YOUTUBE API INTEGRATION

YOUTUBE DATA API V3: This API allows you to access public data on YouTube, including:

- Video details (title, description, views, likes, comments, etc.)
- Channel information (subscriber count, uploaded videos, etc.)
- Playlist details
- Comments on videos
- Video search results

API AUTHENTICATION & RATE LIMITING: Handle API key management, user authentication, and prevent hitting API rate limits.

1.2 VIDEO AND CHANNEL DATA COLLECTION

VIDEO METADATA HARVESTING: Collect information like video title, description, tags, view counts, likes, dislikes, comments, upload date, etc.

CHANNEL DATA HARVESTING: Collect information about the channel, including subscriber count, total views, upload history, etc.

1.3 COMMENT DATA EXTRACTION

Collect and store comments from videos, which could be used for sentiment analysis, trend tracking, etc.

Support for pagination and managing large volumes of comments.

1.4 SEARCH AND TRENDING DATA

SEARCH API: To gather data based on specific search queries (e.g., videos about a certain topic).

TRENDING API: Get a list of trending videos in different regions.

1.5 REAL-TIME DATA COLLECTION

If needed, real-time tracking of videos (e.g., views, likes, comments) or channel statistics.

2. DATA STORAGE & WAREHOUSE MANAGEMENT MODULES

Once the data is harvested, you need to structure and store it efficiently for analysis, reporting, or future use.

2.1 DATA STRUCTURING & NORMALIZATION

Normalize data to fit a consistent schema.

Convert different data formats into a relational or non-relational format (JSON, CSV, SQL, etc.).

2.2 DATABASE MANAGEMENT

SQL DATABASE: If you prefer structured data (e.g., PostgreSQL, MySQL).

NOSQL DATABASE: For large-scale unstructured or semi-structured data (e.g., MongoDB, Cassandra).

CLOUD STORAGE: For scalable data storage (e.g., Amazon S3, Google Cloud Storage).

DATA INDEXING: Efficient indexing of data for fast retrieval (e.g., Elasticsearch for search purposes).

2.3 DATA WAREHOUSING

ETL (EXTRACT, TRANSFORM, LOAD): A process for extracting data from YouTube, transforming it into the desired format, and loading it into a data warehouse (e.g., Redshift, Snowflake, Google BigQuery).

DATA AGGREGATION: Aggregate video statistics, views, likes, etc., into a summarized form for easy querying and analysis.

DATA PARTITIONING: Partition data based on criteria like time, video type, channel, etc.

2.4 DATA CLEANING & VALIDATION

Check for duplicates, handle missing or incorrect data, and ensure the data conforms to the desired format.

2.5 DATA ARCHIVING

Long-term storage of old or unused data in cheaper storage solutions (e.g., AWS Glacier or Google Coldline).

Implement data retention policies.

3. DATA ANALYTICS & REPORTING MODULES

This layer is for analyzing and presenting the harvested YouTube data.

3.1 ANALYTICS ENGINE

VIDEO PERFORMANCE ANALYTICS: Analyze video metrics such as views, likes/dislikes, comments, and shares over time.

CHANNEL GROWTH ANALYTICS: Track channel growth, subscriber increases, and video performance over time.

AUDIENCE DEMOGRAPHICS: Analyze user demographics based on video views and engagement (age, gender, location, etc.).

TRENDING TOPICS: Identify trending keywords, hashtags, and topics across videos and channels.

3.2 SENTIMENT ANALYSIS

Use natural language processing (NLP) to analyze comments, video descriptions, and titles for sentiment (positive, negative, neutral).

3.3 VISUALIZATION TOOLS

Dashboards to visualize data trends, engagement, video performance, etc. (e.g., using Power BI, Tableau, or custom-built solutions).

Time-series graphs, bar charts, and pie charts for showing view count growth, like/dislike ratios, etc.

4. DATA EXPORT & INTEGRATION MODULES

These modules handle the exporting and integration of YouTube data with other systems.

4.1 API FOR DATA ACCESS

Expose collected data via custom APIs for external systems to query.

Provide options to filter and retrieve data based on criteria such as date range, video type, or channel.

4.2 DATA EXPORT OPTIONS

CSV/EXCEL EXPORT: Allow users to export the data to CSV or Excel for offline use.

DATA PIPELINES: Send data to external systems (e.g., CRM, marketing platforms, or social media tools) via data pipelines.

5. DATA SECURITY & PRIVACY MANAGEMENT

Ensure that the collected data is secure and complies with privacy regulations.

5.1 USER AUTHENTICATION & AUTHORIZATION

Manage user roles and permissions, e.g., who can access specific data or perform data operations.

5.2 DATA ENCRYPTION

Encrypt sensitive data in transit (using TLS/SSL) and at rest (using AES encryption).

5.3 COMPLIANCE WITH DATA PRIVACY LAWS

Make sure the system complies with GDPR, CCPA, and other privacy laws related to user data.

6. MONITORING & MAINTENANCE MODULES

Ongoing system health monitoring and maintenance.

6.1 DATA HARVESTING MONITORING

Monitor API usage, success rates, and errors during data harvesting.

Alerting on failed data collection attempts or API limit breaches.

6.2 SYSTEM HEALTH MONITORING

Track the status of the entire system, including data storage, ETL jobs, and analytics pipelines.

Logging and exception handling.

6.3 UPDATE MECHANISM

Regularly update the system to handle changes in the YouTube API, platform updates, or new features.

7. USER INTERFACE (UI) MODULES

For users to interact with the data, these modules could provide a web-based or desktop application interface.

7.1 DASHBOARD

A central dashboard to track the status of YouTube data harvesting and warehouse health.

7.2 REPORTING INTERFACE

User-friendly tools to generate and download reports based on specific metrics, such as video performance, channel growth, sentiment, and more.

7.3 SEARCH & QUERY INTERFACE

Allow users to search and query the data warehouse for specific data, e.g., finding top-performing videos in a specific time range.

SURVEY OF TECHNOLOGY

2.1 SOFTWARE DESCRIPTION

VISUAL STUDIO CODE

Visual Studio Code, commonly referred to as VS Code, is a powerful, open-source code editor developed by Microsoft. Designed with flexibility and productivity in mind, VS Code combines the simplicity of a text editor with advanced developer tooling, making it suitable for a wide range of programming tasks across different languages and frameworks. It has gained immense popularity in the developer community due to its user-friendly interface, extensibility, and efficient features that streamline the development process.

KEY FEATURES AND BENEFITS

INTELLISENSE CODE COMPLETION:

VS Code's IntelliSense offers intelligent auto completion for variables, methods, and functions, speeding up the coding process and reducing errors by suggesting the correct syntax and function names.

BUILT-IN DEBUGGING:

VS Code includes integrated debugging tools, allowing developers to set breakpoints and inspect code execution, making bug fixing faster and more efficient without switching between tools.

CUSTOMIZATION AND EXTENSIONS:

With a vast marketplace of extensions, VS Code can be tailored to specific needs, enhancing functionality with support for new languages, Git integration, and tools for testing and deployment.

VERSION CONTROL WITH GIT INTEGRATION:

VS Code’s built-in Git support enables easy version control, allowing developers to manage code, commit changes, and push updates directly within the editor.

LANGUAGES

2.2.1 STREAMLIT FRONT END

INTERACTIVE DASHBOARDS FOR DATA VISUALIZATION

Streamlit can be used to build real-time, interactive dashboards to visualize YouTube data collected via APIs. These dashboards can display key metrics and analytics, such as video views, likes, comments, engagement rates, and subscriber counts. The following elements can be added:

FEATURES:

REAL-TIME UPDATES: Streamlit can be connected to your backend database (SQL/NoSQL) or data warehouse (e.g., BigQuery, Snowflake) to provide live updates on YouTube video or channel metrics.

CHARTS AND GRAPHS: You can integrate matplotlib, plotly, or altair libraries to generate graphs.

DATA QUERYING AND FILTERING INTERFACE

One of the main functionalities that Streamlit can provide as a front-end is an intuitive user interface for querying and filtering the YouTube data stored in the warehouse.

FEATURES:

CUSTOM FILTERS: Users can use Streamlit widgets like text boxes, dropdowns, and date pickers to filter data based on:

- Time range (e.g., daily, weekly, monthly metrics).
- Video parameters (e.g., title, views, likes).
- Channel attributes (e.g., channel name, subscriber count).
- Geographical or demographic filters (if the data is enriched with user location or age group).

SEARCHABLE DATA: Implement search functionality for users to quickly find specific videos or channels based on keywords or tags.

SENTIMENT ANALYSIS AND COMMENTS SECTION

You can integrate **sentiment analysis** or **comment analysis** into the dashboard, showing how the audience feels about certain videos based on their comments.

FEATURES:

SENTIMENT VISUALIZATIONS: Display the distribution of sentiment scores (positive, negative, neutral) for comments on a particular video.

COMMENT WORD CLOUD: Show a word cloud of the most frequent words or hashtags in the comments section.

INTERACTIVE FILTERS: Allow users to filter comments based on sentiment, comment length, or keywords, to gain deeper insights into user engagement.

DATA EXPORT AND REPORTING

Streamlit provides an easy way for users to export YouTube data for offline use or further analysis. For example, users could download CSVs of videos, comments, or channel statistics.

FEATURES:

DOWNLOAD BUTTONS: Use st.download_button to allow users to download data in formats like CSV, Excel, or JSON.

CUSTOM REPORTS: Generate custom reports based on user queries or selections and let users download them.

SCHEDULED EXPORT: Schedule automated exports based on specific time intervals or triggers.

2.2.2 MONGODB (NOSQL DATABASE) - DATABASE

MongoDB is a powerful NoSQL database known for its flexibility and high performance. Unlike traditional SQL databases, MongoDB stores data in a JSON-like, document-oriented format, allowing for the storage of semi-structured data without a fixed schema. This structure enables MongoDB to handle large datasets with ease and adapt as the data needs of applications evolve. MongoDB's document-based model supports complex data types, which makes it ideal for applications that require a highly flexible data structure. Additionally,

its scalability ensures that MongoDB can handle growing amounts of data and users without compromising performance, making it suitable for both small and large-scale applications.

PURPOSE OF MONGODB IN YOUTUBE DATA HARVESTING :

In a YouTube Data Harvesting and Warehouse Management System, MongoDB can serve as an important component of the system's data storage layer. MongoDB is a popular NoSQL database known for its flexibility, scalability, and ability to handle large volumes of unstructured or semi-structured data, which makes it an ideal choice for managing the dynamic and diverse data collected from YouTube.

STORING UNSTRUCTURED AND SEMI-STRUCTURED DATA

YouTube data, such as video metadata (e.g., title, description, views, likes, comments), is typically semi-structured and may vary from video to video or channel to channel. MongoDB's **document-based** storage model makes it highly suitable for handling such data without the rigid schema requirements of relational databases.

HOW MONGODB HELPS:

FLEXIBLE SCHEMA: MongoDB allows you to store YouTube data in JSON-like documents (BSON format), meaning that you don't need to define a strict schema upfront. You can store different attributes for different videos and channels without the need for complex joins.

HANDLING NESTED DATA: YouTube data can include nested or hierarchical information, such as lists of comments, playlists, or channel statistics. MongoDB is well-suited to store this kind of nested, hierarchical data (e.g., as arrays or embedded documents), which makes querying and retrieval more natural and efficient.

SCALABILITY AND HIGH THROUGHPUT

YouTube is a massive platform with millions of videos and users, generating vast amounts of data. MongoDB is known for its **horizontal scalability** through **sharding**, allowing it to scale efficiently as the amount of data grows. This is crucial in handling the large volume of data generated from YouTube's ever-expanding catalog of videos, channels, and interactions.

HOW MONGODB HELPS:

SHARDING: MongoDB's sharding capability allows you to distribute data across multiple servers, improving performance and ensuring the system can handle large datasets (such as millions of videos, comments, and user interactions).

REPLICATION: MongoDB also supports replication for high availability, ensuring that the data is protected against failure by maintaining multiple copies of the data on different servers.

HANDLING TIME-SERIES DATA

YouTube data, such as video views, likes, and comments, are typically recorded over time. MongoDB can be effective in managing time-series data, allowing you to store metrics such as views per day, changes in subscriber count, or engagement trends over time.

HOW MONGODB HELPS:

TIME-BASED DATA: MongoDB supports efficient querying of time-based data, which is crucial for tracking **video performance over time**. For instance, you can store daily metrics for views, likes, and comments for each video in separate collections or documents, and easily track trends over periods (e.g., the last 30 days).

TTL INDEXES: MongoDB also offers **TTL (Time-to-Live)** indexes, which allow automatic deletion of documents after a certain time. This can be useful for cleaning up outdated or irrelevant data, such as metrics for videos that are no longer being updated or engaging a large audience.

2.2.3 PYTHON (PROGRAMMING LANGUAGE) - BACKEND

Python is one of the most popular and versatile programming languages, and it's widely used for backend development due to its simplicity, readability, and powerful libraries. When it comes to backend development, Python offers a variety of tools and frameworks that make building web applications and services fast and efficient.

PURPOSE IN YOUTUBE DATA HARVESTING :

In a **YouTube Data Harvesting and Warehouse Management System**, **Python** is an ideal backend programming language due to its flexibility, vast ecosystem of libraries, and ease of integration with data processing, storage, and visualization tools. Below is an explanation of how Python can be used as the backend in such a system:

Data Collection (YouTube API Integration)

The first step in harvesting YouTube data involves using the **YouTube Data API** or other methods to extract data from YouTube (e.g., video details, comments, likes, views, channel data). Python's extensive library support makes it easy to interact with APIs and gather data.

Key Libraries/Tools:

google-api-python-client: A client library that facilitates interacting with the YouTube Data API, allowing Python to pull data like videos, playlists, channels, and comments.

requests: A powerful HTTP library for making API calls to YouTube or other RESTful services.

oauth2client: Used for handling OAuth authentication when accessing YouTube API data with user-specific permissions.

Data Processing and Transformation

Once data is collected, it needs to be processed, cleaned, and transformed into a structure suitable for storage and analysis. Python is commonly used for data processing tasks because of its rich ecosystem of libraries designed for data manipulation.

Key Libraries/Tools:

pandas: A powerful data manipulation library for cleaning, transforming, and analyzing data. It can handle large datasets and export data in various formats like CSV, Excel, or JSON.

numpy: A library for numerical computing, particularly useful for working with large arrays and matrices.

json: To work with JSON data, commonly used for API responses.

regex: For text processing and cleaning, such as cleaning comments or descriptions.

Data Storage and Management

Python interacts seamlessly with a variety of databases for storing large datasets, including SQL and NoSQL databases, as well as cloud storage systems.

Key Libraries/Tools for Storage:

SQL (PostgreSQL, MySQL):

psycopg2 (for PostgreSQL) or **mysql-connector-python** (for MySQL) can be used for interfacing with relational databases.

SQLAlchemy: An ORM (Object-Relational Mapping) library that simplifies interaction with relational databases.

NoSQL (MongoDB):

pymongo: A MongoDB driver for Python, which makes it easy to interact with a MongoDB database.

REQUIREMENTS AND ANALYSIS

3.1 REQUIREMENT SPECIFICATION

A YouTube Data Harvesting System is designed to collect, store, and analyze data from YouTube, such as video statistics, user engagement, comments, and more. This system typically interacts with the YouTube Data API to fetch information and can be used for various purposes such as analytics, reporting, sentiment analysis, or even monitoring trends in video content.

3.1.1 FUNCTIONAL REQUIREMENTS:

a. Data Collection

YouTube Data API Integration: The system must connect to the YouTube Data API to harvest information about YouTube videos, channels, comments, and more.

Video Details: The system should be able to collect data for individual videos, including:

- Video title
- Description
- Published date
- View count
- Like/dislike count
- Comment count
- Video tags and categories
- Channel ID and owner info

Channel Data: Information about the channel that owns the video, including:

- Subscriber count
- Channel description
- Video count
- Channel statistics (e.g., growth over time)

Comments: Harvest and store comments for each video, along with metadata such as:

- Comment text
- Author

- Sentiment (optional analysis)
- Timestamp

Trending Data: Ability to track trending videos (by views, likes, comments).

Live Video Data: For live streams, real-time data such as viewer count and engagement metrics should be collected.

b. Data Storage

The system should store harvested data in a structured or semi-structured format, depending on the type of database used (e.g., SQL or NoSQL).

Database Requirements:

Relational Databases: If using SQL, the system needs tables for videos, channels, comments, and other data points, with relationships defined (e.g., foreign keys between video data and channel data).

NoSQL Databases: If using MongoDB, data can be stored in collections with flexible schemas, supporting varied data structures (nested comments, video metadata).

c. Data Processing

Real-Time Processing: The system may need to process data in real-time, especially for live videos or near-real-time tracking of video metrics.

Data Aggregation: Aggregating data over time for reporting (e.g., average views, engagement rate).

Data Transformation: Cleaning and transforming the raw YouTube data into usable formats for analysis (e.g., converting view counts into percentages, grouping videos by categories).

d. Analytics and Reporting

Trending Video Analytics: Identifying trends in video popularity or engagement over a set period (e.g., top 10 videos by views in the last week).

Sentiment Analysis: Analyzing the sentiment of video comments using Natural Language Processing (NLP) to determine whether the public's opinion is positive, negative, or neutral.

Engagement Metrics: Metrics like the like-to-view ratio, comment-to-view ratio, and subscriber growth.

Time-Series Analysis: Tracking how metrics evolve over time (e.g., views per day for a video, or subscribers per month for a channel).

Visual Dashboards: Creating user-friendly dashboards to display the harvested and processed data, such as:

- Graphs (e.g., view counts over time)
- Tables (e.g., comment sentiment breakdown)
- Interactive filters (e.g., sorting by video category, views, or engagement rate)

e. Automation

Scheduled Harvesting: The system must automate data collection at regular intervals (e.g., daily, weekly) to track video/channel performance over time.

Alerting System: Alerts for milestones, such as when a video reaches a certain number of views or when a channel grows its subscriber base rapidly.

f. User Authentication and Access Control

User Authentication: The system may need to provide role-based access to different levels of data or features (e.g., admin users vs. regular users).

API Access Control: If other users or systems are accessing the API to retrieve YouTube data, access control and rate-limiting should be implemented to ensure that the system adheres to YouTube's API usage policies.

3.1.2 NON-FUNCTIONAL REQUIREMENTS:

a. Performance

Scalability: The system should be able to scale to handle the large volume of data YouTube generates. For instance, if a user tracks thousands of videos or channels, the system should handle this without performance degradation.

Horizontal Scaling: To handle large datasets, databases (like MongoDB) can be shared across multiple servers to distribute the load.

Caching: Use of caching mechanisms (e.g., Redis) for frequently requested data to reduce the load on the YouTube API and database.

b. Availability and Reliability

High Availability: The system must be available to users at all times, with failover mechanisms in place (e.g., database replication).

Backup and Recovery: Data should be regularly backed up to prevent loss of important data.

c. Security

Data Encryption: Sensitive data, such as user information and API keys, should be encrypted in transit (using HTTPS) and at rest (using strong encryption algorithms).

Authentication: OAuth or API keys for authenticating access to the YouTube Data API.

Access Control: Role-based access control (RBAC) to ensure only authorized users have access to sensitive data or features.

d. Compliance

Adherence to YouTube API Policies: The system should comply with YouTube's API terms of service, including rate-limiting requests and ensuring that data is not misused.

Data Privacy: The system should respect user privacy, especially when dealing with user data (e.g., comments or personal details associated with a video).

3.2 HARDWARE AND SOFTWARE REQUIREMENTS

3.2.1 HARDWARE REQUIREMENTS:

1. **COMPUTER WITH WINDOWS 10/LINUX OPERATING SYSTEM**:
   Required For Both Development And Server Deployment.
2. **WEB SERVER OR CLOUD-BASED SERVER**:
   For Hosting The Application And Providing Real-Time Access To All Users.
3. **MINIMUM 4 GB RAM AND 500 GB STORAGE**:
   Ensures That The Application Operates Efficiently And Can Handle Data Storage And High-Performance Requirements

3.2.2 SOFTWARE REQUIREMENTS:

1. **STREAMLIT FRONT END:**

Streamlit can be used to build real-time, interactive dashboards to visualize YouTube data collected via APIs.

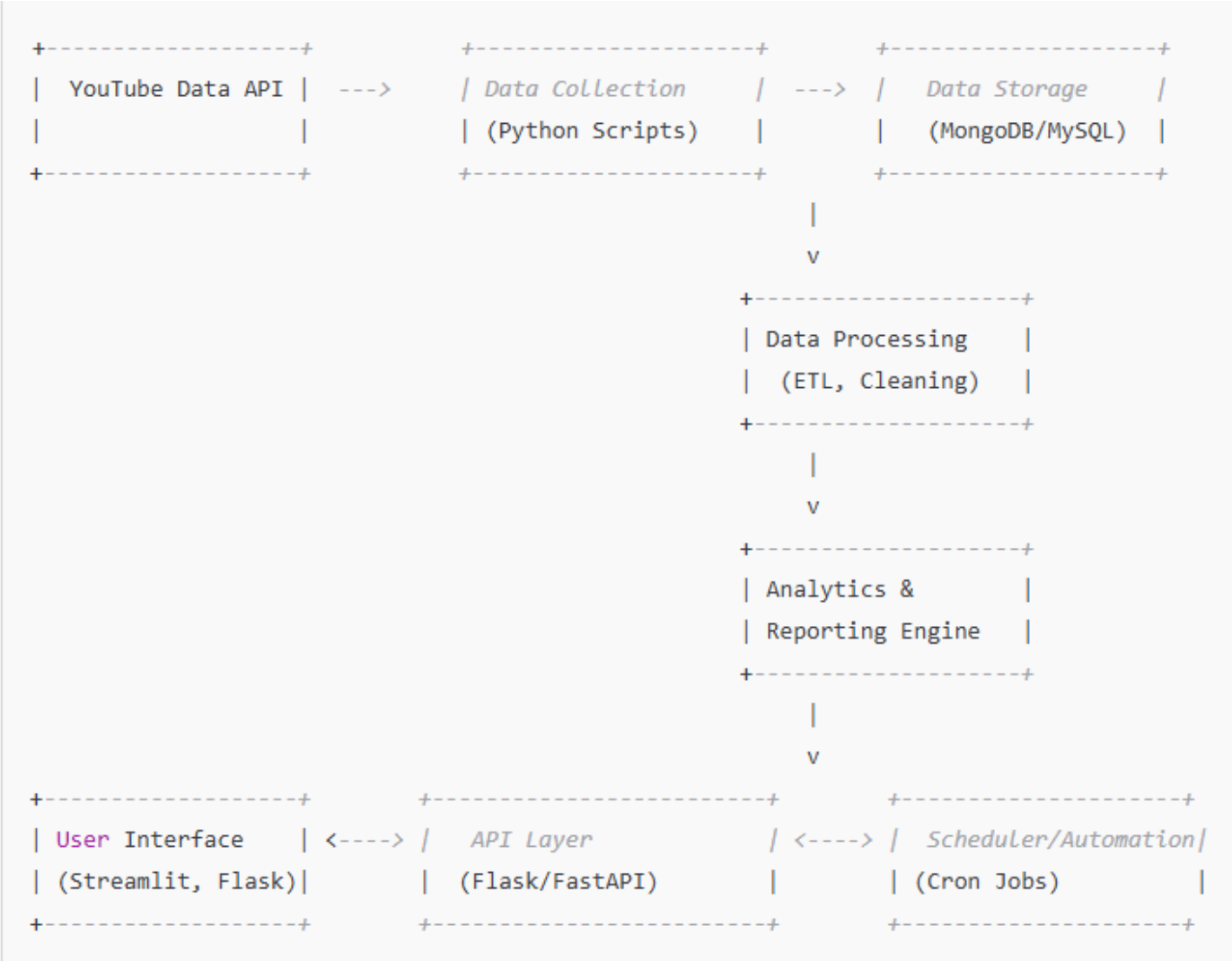
2. **MONGODB (BACK-END)**:

A Nosql Database Chosen For Its Flexibility And Scalability, Suitable For Storing Student Requests, Faculty Decisions, And All Record Data.

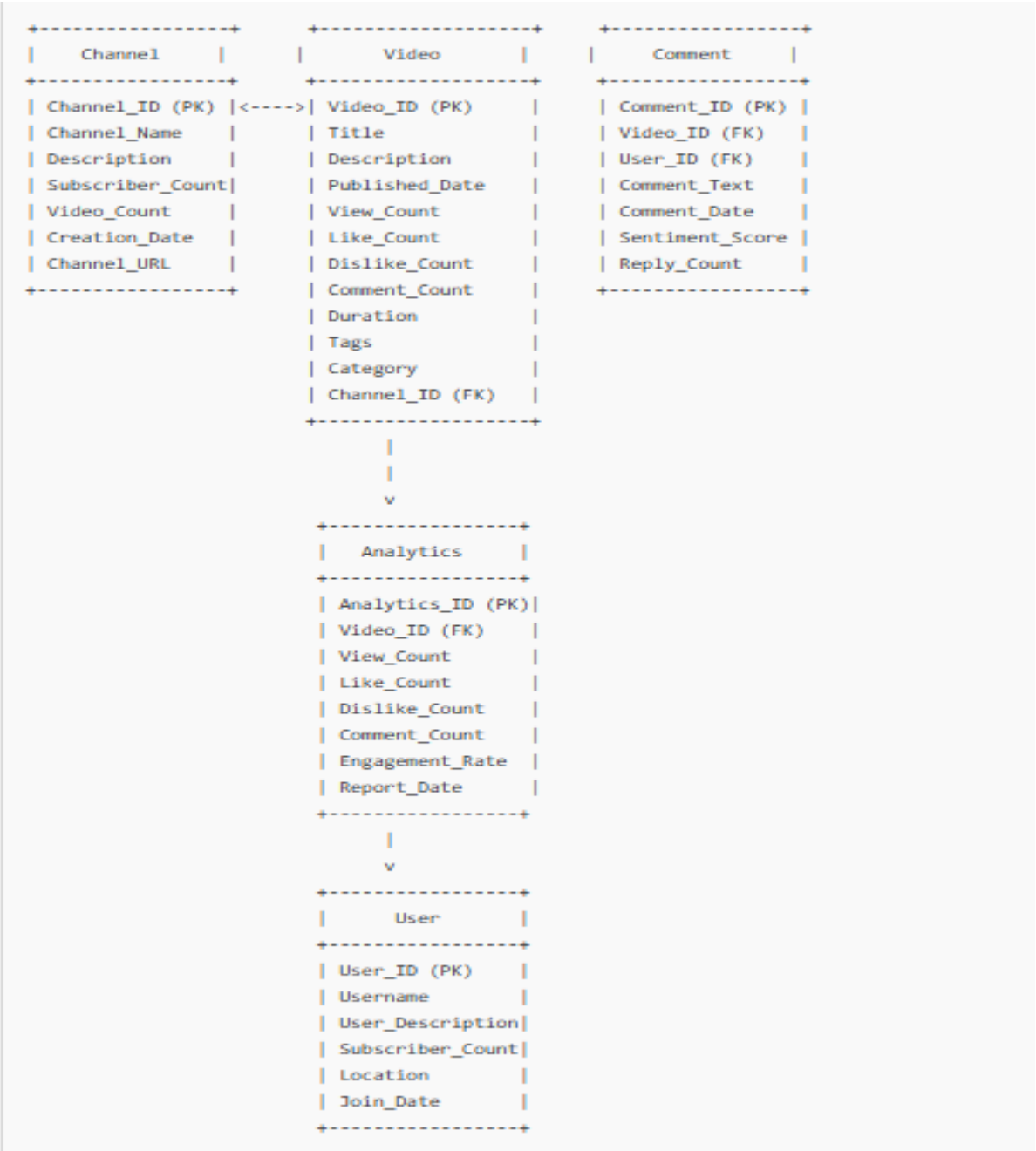
3. **PYTHON (BACK-END)**:

Python is one of the most popular and versatile programming languages, and it's widely used for backend development due to its simplicity, readability, and powerful libraries.

ARCHITECTURE DIAGRAM OF YOUTUBE DATA HARVESTING



ER DIAGRAM


Channel
Channel_ID (PK)
Channel_Name
Description
Subscriber_Count
Video_Count
Creation_Date
Channel_URL
Video
Video_ID (PK)
Title
Description
Published_Date
View_Count
Like_Count
Dislike_Count
Comment_Count
Duration
Tags
Category
Channel_ID (FK)
Comment
Comment_ID (PK)
Video_ID (FK)
User_ID (FK)
Comment_Text
Comment_Date
Sentiment_Score
Reply_Count
Analytics
Analytics_ID (PK)
Video_ID (FK)
View_Count
Like_Count
Dislike_Count
Comment_Count
Engagement_Rate
Report_Date
User
User_ID (PK)
Username
User_Description
Subscriber_Count
Location
Join_Date

PROGRAM CODE

ARENA FOR CODING : VISUAL STUDIO CODE

CODE FOR YOUTUBE DATA HARVESTING

```
from googleapiclient.discovery import build
import pymongo
import psycopg2
import pandas as pd
import streamlit as st



#API key connection

def Api_connect():
    Api_Id="AIzaSyAAUHDdALApUGmI3OE1cVs4oBtKvcnlBb8"

    api_service_name="youtube"
    api_version="v3"

    youtube=build(api_service_name,api_version,developerKey=Api_Id)

    return youtube

youtube=Api_connect()
```

```
#get channels information
def get_channel_info(channel_id):
    request=youtube.channels().list(
                part="snippet,ContentDetails,statistics",
                id=channel_id
    )
    response=request.execute()

    for i in response['items']:
        data=dict(Channel_Name=i["snippet"]["title"],
                Channel_Id=i["id"],
                Subscribers=i['statistics']['subscriberCount'],
                Views=i["statistics"]["viewCount"],
                Total_Videos=i["statistics"]["videoCount"],
                Channel_Description=i["snippet"]["description"],
                Playlist_Id=i["contentDetails"]["relatedPlaylists"]["uploads"])
    return data

#get video ids
def get_videos_ids(channel_id):
    video_ids=[]
    response=youtube.channels().list(id=channel_id,
                                     part='contentDetails').execute()
    Playlist_Id=response['items'][0]['contentDetails']['relatedPlaylists']['uploads']

    next_page_token=None
```

```
    while True:
        response1=youtube.playlistItems().list(
                                    part='snippet',
                                    playlistId=Playlist_Id,
                                    maxResults=50,
                                    pageToken=next_page_token).execute()
        for i in range(len(response1['items'])):
            video_ids.append(response1['items'][i]['snippet']['resourceId']['videoId'])
        next_page_token=response1.get('nextPageToken')


        if next_page_token is None:
            break
    return video_ids


#get video information
def get_video_info(video_ids):
    video_data=[]
    for video_id in video_ids:
        request=youtube.videos().list(
            part="snippet,ContentDetails,statistics",
            id=video_id
        )
        response=request.execute()


        for item in response["items"]:
            data=dict(Channel_Name=item['snippet']['channelTitle'],
                    Channel_Id=item['snippet']['channelId'],
```

```
                Video_Id=item['id'],
                Title=item['snippet']['title'],
                Tags=item['snippet'].get('tags'),
                Thumbnail=item['snippet']['thumbnails']['default']['url'],
                Description=item['snippet'].get('description'),
                Published_Date=item['snippet']['publishedAt'],
                Duration=item['contentDetails']['duration'],
                Views=item['statistics'].get('viewCount'),
                Likes=item['statistics'].get('likeCount'),
                Comments=item['statistics'].get('commentCount'),
                Favorite_Count=item['statistics']['favoriteCount'],
                Definition=item['contentDetails']['definition'],
                Caption_Status=item['contentDetails']['caption']
                )
        video_data.append(data)
    return video_data



#get comment information
def get_comment_info(video_ids):
    Comment_data=[]
    try:
        for video_id in video_ids:
            request=youtube.commentThreads().list(
                part="snippet",
                videoId=video_id,
                maxResults=50
```

```
            )
            response=request.execute()

            for item in response['items']:
                data=dict(Comment_Id=item['snippet']['topLevelComment']['id'],
                        Video_Id=item['snippet']['topLevelComment']['snippet']['videoId'],
                        Comment_Text=item['snippet']['topLevelComment']['snippet']['textDisplay'],
Comment_Author=item['snippet']['topLevelComment']['snippet']['authorDisplayName'],
Comment_Published=item['snippet']['topLevelComment']['snippet']['publishedAt'])

                Comment_data.append(data)

    except:
        pass
    return Comment_data

#get_playlist_details

def get_playlist_details(channel_id):
        next_page_token=None
        All_data=[]
        while True:
            request=youtube.playlists().list(
                    part='snippet,contentDetails',
                    channelId=channel_id,
```

```
                maxResults=50,
                pageToken=next_page_token
            )
            response=request.execute()

            for item in response['items']:
                data=dict(Playlist_Id=item['id'],
                    Title=item['snippet']['title'],
                    Channel_Id=item['snippet']['channelId'],
                    Channel_Name=item['snippet']['channelTitle'],
                    PublishedAt=item['snippet']['publishedAt'],
                    Video_Count=item['contentDetails']['itemCount'])
                All_data.append(data)

            next_page_token=response.get('nextPageToken')
            if next_page_token is None:
                break
        return All_data



#upload to mongoDB

client=pymongo.MongoClient("mongodb+srv://harshasri1511:Harshasrigee@harsha.6gp0f.mongodb.net/?retryWrites=true&w=majority&appName=Harsha")
db=client["Youtube_data"]

def channel_details(channel_id):
```

```
    ch_details=get_channel_info(channel_id)
    pl_details=get_playlist_details(channel_id)
    vi_ids=get_videos_ids(channel_id)
    vi_details=get_video_info(vi_ids)
    com_details=get_comment_info(vi_ids)

    coll1=db["channel_details"]
    coll1.insert_one({"channel_information":ch_details,"playlist_information":pl_details,
                      "video_information":vi_details,"comment_information":com_details})

    return "upload completed successfully"



#Table creation for channels,playlists,videos,comments



def channels_table(channel_name_s):
    mydb=psycopg2.connect(host="localhost",
                          user="postgres",
                          password="harsha",
                          database="youtube_data",
                          port="5432")
    cursor=mydb.cursor()

    try:
        chann_list.append(df_all_channels[0])
        create_query='''create table if not exists channels(Channel_Name varchar(100),
```

```
                                    Channel_Id varchar(80) primary key,
                                    Subscribers bigint,
                                    Views bigint,
                                    Total_Videos int,
                                    Channel_Description text,
                                    Playlist_Id varchar(80))'''
        cursor.execute(create_query)
        mydb.commit()
    # Attempt to access the first row by column name
    except KeyError:
        # If the column doesn't exist, use integer-based indexing
        chann_list.append(df_all_channels.iloc[0])

    # ... (rest of the code)
    except:
        print("Channels table already created")

    #fetching all datas
    query_1= "SELECT * FROM channels"
    cursor.execute(query_1)
    table= cursor.fetchall()
    mydb.commit()

    chann_list= []
    chann_list2= []
    df_all_channels= pd.DataFrame(table)
```

```
    chann_list.append(df_all_channels[0])
    for i in chann_list[0]:
        chann_list2.append(i)



    if channel_name_s in chann_list2:
        news= f"Your Provided Channel {channel_name_s} is Already exists"
        return news

    else:

        single_channel_details= []
        coll1=db["channel_details"]
        for ch_data in
coll1.find({"channel_information.Channel_Name":channel_name_s},{"_id":0}):
            single_channel_details.append(ch_data["channel_information"])

        df_single_channel= pd.DataFrame(single_channel_details)




        for index,row in df_single_channel.iterrows():
            insert_query='''insert into channels(Channel_Name ,
                                    Channel_Id,
                                    Subscribers,
                                    Views,
                                    Total_Videos,
```

```
                                Channel_Description,
                                Playlist_Id)

                                values(%s,%s,%s,%s,%s,%s,%s)'''
            values=(row['Channel_Name'],
                    row['Channel_Id'],
                    row['Subscribers'],
                    row['Views'],
                    row['Total_Videos'],
                    row['Channel_Description'],
                    row['Playlist_Id'])

            try:
                cursor.execute(insert_query,values)
                mydb.commit()

            except:
                print("Channel values are already inserted")



def playlist_table(channel_name_s):
    mydb=psycopg2.connect(host="localhost",
                          user="postgres",
                          password="harsha",
                          database="youtube_data",
                          port="5432")
    cursor=mydb.cursor()
```

```
    create_query='''create table if not exists playlists(Playlist_Id varchar(100) primary key,
                                                        Title varchar(100),
                                                        Channel_Id varchar(100),
                                                        Channel_Name varchar(100),
                                                        PublishedAt timestamp,
                                                        Video_Count int
                                                        )'''

    cursor.execute(create_query)
    mydb.commit()

    single_channel_details= []
    coll1=db["channel_details"]
    for ch_data in
coll1.find({"channel_information.Channel_Name":channel_name_s},{"_id":0}):
        single_channel_details.append(ch_data["playlist_information"])

    df_single_channel= pd.DataFrame(single_channel_details[0])

    for index,row in df_single_channel.iterrows():
        insert_query='''insert into playlists(Playlist_Id,
                                    Title,
                                    Channel_Id,
                                    Channel_Name,
                                    PublishedAt,
                                    Video_Count
```

```
                                )

                                values(%s,%s,%s,%s,%s,%s)'''

        values=(row['Playlist_Id'],
                row['Title'],
                row['Channel_Id'],
                row['Channel_Name'],
                row['PublishedAt'],
                row['Video_Count']
                )


        cursor.execute(insert_query,values)
        mydb.commit()

def videos_table(channel_name_s):
    mydb=psycopg2.connect(host="localhost",
                          user="postgres",
                          password="harsha",
                          database="youtube_data",
                          port="5432")
    cursor=mydb.cursor()

    create_query='''create table if not exists videos(Channel_Name varchar(100),
                                        Channel_Id varchar(100),
                                        Video_Id varchar(30) primary key,
```

```
                                Title varchar(150),
                                Tags text,
                                Thumbnail varchar(200),
                                Description text,
                                Published_Date timestamp,
                                Duration interval,
                                Views bigint,
                                Likes bigint,
                                Comments int,
                                Favorite_Count int,
                                Definition varchar(10),
                                Caption_Status varchar(50)
                                    )'''

    cursor.execute(create_query)
    mydb.commit()



    single_channel_details= []
    coll1=db["channel_details"]
    for ch_data in
coll1.find({"channel_information.Channel_Name":channel_name_s},{"_id":0}):
        single_channel_details.append(ch_data["video_information"])

    df_single_channel= pd.DataFrame(single_channel_details[0])
```

```
for index,row in df_single_channel.iterrows():
        insert_query='''insert into videos(Channel_Name,
                                           Channel_Id,
                                           Video_Id,
                                           Title,
                                           Tags,
                                           Thumbnail,
                                           Description,
                                           Published_Date,
                                           Duration,
                                           Views,
                                           Likes,
                                           Comments,
                                           Favorite_Count,
                                           Definition,
                                           Caption_Status
                                          )

                                          values(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)'''


        values=(row['Channel_Name'],
                row['Channel_Id'],
                row['Video_Id'],
                row['Title'],
                row['Tags'],
```

```
                row['Thumbnail'],
                row['Description'],
                row['Published_Date'],
                row['Duration'],
                row['Views'],
                row['Likes'],
                row['Comments'],
                row['Favorite_Count'],
                row['Definition'],
                row['Caption_Status']
                )


        cursor.execute(insert_query,values)
        mydb.commit()


def comments_table(channel_name_s):
    mydb=psycopg2.connect(host="localhost",
                user="postgres",
                password="harsha",
                database="youtube_data",
                port="5432")
    cursor=mydb.cursor()

    create_query='''create table if not exists comments(Comment_Id varchar(100) primary key,
                                Video_Id varchar(50),
```

```
                                    Comment_Text text,
                                    Comment_Author varchar(150),
                                    Comment_Published timestamp
                                    )'''

    cursor.execute(create_query)
    mydb.commit()

    single_channel_details= []
    coll1=db["channel_details"]
    for ch_data in
coll1.find({"channel_information.Channel_Name":channel_name_s},{"_id":0}):
        single_channel_details.append(ch_data["comment_information"])

    df_single_channel= pd.DataFrame(single_channel_details[0])

    for index,row in df_single_channel.iterrows():
            insert_query='''insert into comments(Comment_Id,
                                    Video_Id,
                                    Comment_Text,
                                    Comment_Author,
                                    Comment_Published
                                )

                                values(%s,%s,%s,%s,%s)'''
```

```
        values=(row['Comment_Id'],
                row['Video_Id'],
                row['Comment_Text'],
                row['Comment_Author'],
                row['Comment_Published']
                )



        cursor.execute(insert_query,values)
        mydb.commit()



def tables(channel_name):

    news= channels_table(channel_name)
    if news:
        st.write(news)
    else:
        playlist_table(channel_name)
        videos_table(channel_name)
        comments_table(channel_name)

    return "Tables Created Successfully"

def show_channels_table():
    ch_list=[]
    db=client["Youtube_data"]
```

```
    coll1=db["channel_details"]
    for ch_data in coll1.find({},{"_id":0,"channel_information":1}):
        ch_list.append(ch_data["channel_information"])
    df=st.dataframe(ch_list)


    return df


def show_playlists_table():
    pl_list=[]
    db=client["Youtube_data"]
    coll1=db["channel_details"]
    for pl_data in coll1.find({},{"_id":0,"playlist_information":1}):
        for i in range(len(pl_data["playlist_information"])):
            pl_list.append(pl_data["playlist_information"][i])
    df1=st.dataframe(pl_list)


    return df1


def show_videos_table():
    vi_list=[]
    db=client["Youtube_data"]
    coll1=db["channel_details"]
    for vi_data in coll1.find({},{"_id":0,"video_information":1}):
        for i in range(len(vi_data["video_information"])):
            vi_list.append(vi_data["video_information"][i])
    df2=st.dataframe(vi_list)
```

```
    return df2

def show_comments_table():
    com_list=[]
    db=client["Youtube_data"]
    coll1=db["channel_details"]
    for com_data in coll1.find({},{"_id":0,"comment_information":1}):
        for i in range(len(com_data["comment_information"])):
            com_list.append(com_data["comment_information"][i])
    df3=st.dataframe(com_list)

    return df3

#streamlit part

with st.sidebar:
    st.title(":red[YOUTUBE DATA HAVERSTING AND WAREHOUSING]")
    st.header("Skill Take Away")
    st.caption("Python Scripting")
    st.caption("Data Collection")
    st.caption("MongoDB")
    st.caption("API Integration")
    st.caption("Data Management using MongoDB and SQL")

channel_id=st.text_input("Enter the channel ID")

if st.button("collect and store data"):
```

```
    ch_ids=[]
    db=client["Youtube_data"]
    coll1=db["channel_details"]
    for ch_data in coll1.find({},{"_id":0,"channel_information":1}):
        ch_ids.append(ch_data["channel_information"]["Channel_Id"])

    if channel_id in ch_ids:
        st.success("Channel Details of the given channel id already exists")

    else:
        insert=channel_details(channel_id)
        st.success(insert)

#New code

all_channels= []
coll1=db["channel_details"]
for ch_data in coll1.find({},{"_id":0,"channel_information":1}):
    all_channels.append(ch_data["channel_information"]["Channel_Name"])

unique_channel= st.selectbox("Select the Channel",all_channels)

if st.button("Migrate to Sql"):
    Table=tables(unique_channel)
    st.success(Table)

show_table=st.radio("SELECT THE TABLE FOR
```

```
VIEW",("CHANNELS","PLAYLISTS","VIDEOS","COMMENTS"))

if show_table=="CHANNELS":
    show_channels_table()

elif show_table=="PLAYLISTS":
    show_playlists_table()

elif show_table=="VIDEOS":
    show_videos_table()

elif show_table=="COMMENTS":
    show_comments_table()

#SQL Connection

mydb=psycopg2.connect(host="localhost",
                      user="postgres",
                      password="harsha",
                      database="youtube_data",
                      port="5432")
cursor=mydb.cursor()

question=st.selectbox("Select your question",("1. All the videos and the channel name",
                                              "2. channels with most number of videos",
                                              "3. 10 most viewed videos",
                                              "4. comments in each videos",
```

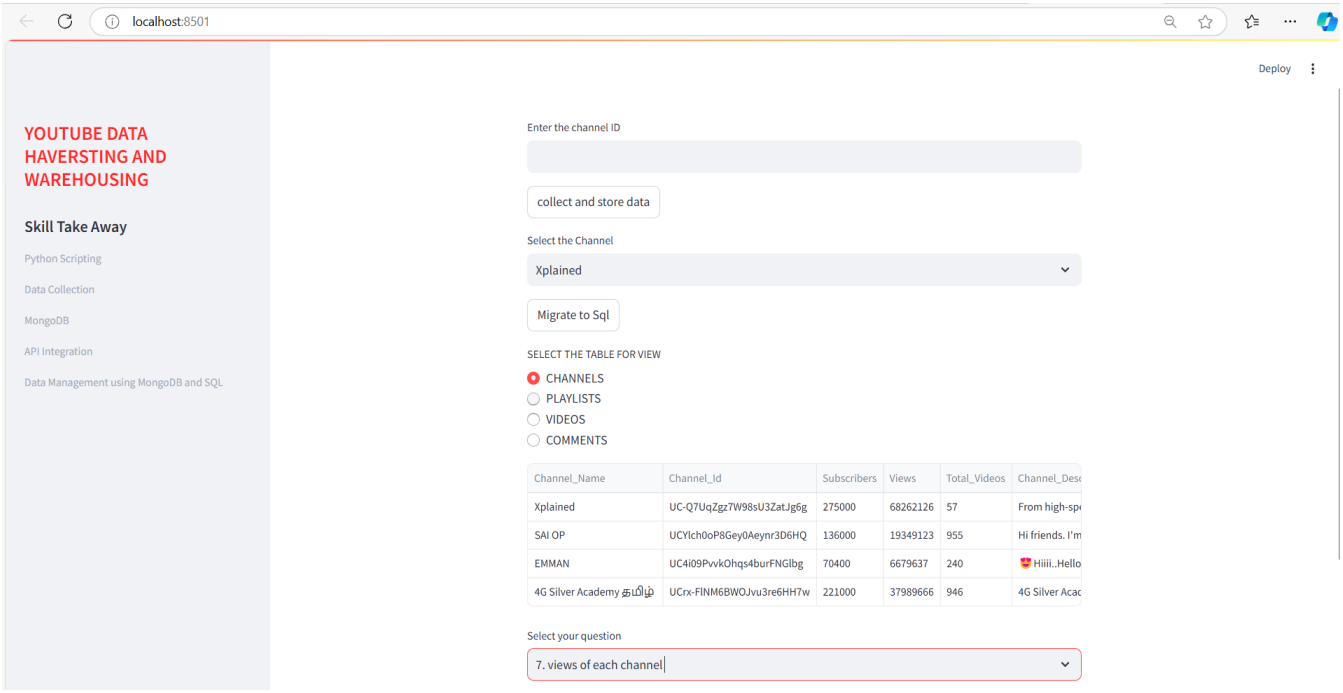
```
                        "5. Videos with higest likes",
                        "6. likes of all videos",
                        "7. views of each channel",
                        "8. videos published in the year of 2022",
                        "9. average duration of all videos in each channel",
                        "10. videos with highest number of comments"))

if question=="1. All the videos and the channel name":
    query1="'select title as videos,channel_name as channelname from videos'"
    cursor.execute(query1)
    mydb.commit()
    t1=cursor.fetchall()
    df=pd.DataFrame(t1,columns=["video title","channel name"])
    st.write(df)

elif question=="2. channels with most number of videos":
    query2="'select channel_name as channelname,total_videos as no_videos from channels
            order by total_videos desc'"
    cursor.execute(query2)
    mydb.commit()
    t2=cursor.fetchall()
    df2=pd.DataFrame(t2,columns=["channel name","No of videos"])
    st.write(df2)

elif question=="3. 10 most viewed videos":
    query3="'select views as views,channel_name as channelname,title as videotitle from videos
            where views is not null order by views desc limit 10'"
```

```
    cursor.execute(query3)
    mydb.commit()
    t3=cursor.fetchall()
    df3=pd.DataFrame(t3,columns=["views","channel name","videotitle"])
    st.write(df3)

elif question=="4. comments in each videos":
    query4="'select comments as no_comments,title as videotitle from videos where comments is
not null"'
    cursor.execute(query4)
    mydb.commit()
    t4=cursor.fetchall()
    df4=pd.DataFrame(t4,columns=["no of comments","videotitle"])
    st.write(df4)

elif question=="5. Videos with higest likes":
    query5="'select title as videotitle,channel_name as channelname,likes as likecount
                from videos where likes is not null order by likes desc"'
    cursor.execute(query5)
    mydb.commit()
    t5=cursor.fetchall()
    df5=pd.DataFrame(t5,columns=["videotitle","channelname","likecount"])
    st.write(df5)

elif question=="6. likes of all videos":
    query6="'select likes as likecount,title as videotitle from videos"'
    cursor.execute(query6)
```

```
    mydb.commit()
    t6=cursor.fetchall()
    df6=pd.DataFrame(t6,columns=["likecount","videotitle"])
    st.write(df6)

elif question=="7. views of each channel":
    query7='''select channel_name as channelname ,views as totalviews from channels'''
    cursor.execute(query7)
    mydb.commit()
    t7=cursor.fetchall()
    df7=pd.DataFrame(t7,columns=["channel name","totalviews"])
    st.write(df7)

elif question=="8. videos published in the year of 2022":
    query8='''select title as video_title,published_date as videorelease,channel_name as channelname from videos
                where extract(year from published_date)=2022'''
    cursor.execute(query8)
    mydb.commit()
    t8=cursor.fetchall()
    df8=pd.DataFrame(t8,columns=["videotitle","published_date","channelname"])
    st.write(df8)

elif question=="9. average duration of all videos in each channel":
    query9='''select channel_name as channelname,AVG(duration) as averageduration from videos group by channel_name'''
    cursor.execute(query9)
```

```
    mydb.commit()
    t9=cursor.fetchall()
    df9=pd.DataFrame(t9,columns=["channelname","averageduration"])


    T9=[]
    for index,row in df9.iterrows():
        channel_title=row["channelname"]
        average_duration=row["averageduration"]
        average_duration_str=str(average_duration)
        T9.append(dict(channeltitle=channel_title,avgduration=average_duration_str))
    df1=pd.DataFrame(T9)
    st.write(df1)


elif question=="10. videos with highest number of comments":
    query10='''select title as videotitle, channel_name as channelname,comments as comments from videos where comments is
                not null order by comments desc'''
    cursor.execute(query10)
    mydb.commit()
    t10=cursor.fetchall()
    df10=pd.DataFrame(t10,columns=["video title","channel name","comments"])
    st.write(df10)
```

PROJECT SCREENSHOTS



RESULTS AND DISCUSSION

The result of the YouTube Data Harvesting and Warehouse Management System refers to the successful implementation and output of a functional system that collects, processes, stores, and analyzes YouTube data. This system allows users to efficiently manage large volumes of YouTube data (videos, comments, user engagement, etc.) while providing meaningful insights and visualizations.

The **result of the YouTube Data Harvesting and Warehouse Management System** would be:

Efficient data storage for large-scale YouTube data.

Actionable insights through data analysis and visualizations.

Scalability to handle growing data over time.

A **user-friendly interface** for exploring YouTube data and tracking performance.

Real-time or batch **report generation** for stakeholders (e.g., content creators, marketers).

This system will make it easier for users to analyze YouTube data, track performance, and make informed decisions based on data-driven insights.

CONCLUSION

The **YouTube Data Harvesting and Warehouse Management System** represents a powerful solution for collecting, processing, storing, and analyzing data from YouTube. By leveraging YouTube's Data API, a combination of backend technologies (like Python), a flexible database (such as MongoDB or SQL), and a frontend interface (like Streamlit or Flask), this system can effectively provide valuable insights to content creators, marketers, analysts, and decision-makers.

Key Highlights:

Comprehensive Data Collection: The system can collect a wide variety of data, including video metadata, user comments, channel information, and engagement metrics (likes, views, comments). This is achieved by interacting with the YouTube API, which makes the process efficient and scalable.

Efficient Data Storage and Management: Data is organized into a database, allowing for efficient querying and easy access. Whether using a relational database like MySQL/PostgreSQL or a NoSQL solution like MongoDB, the system ensures that large volumes of YouTube data can be stored, processed, and retrieved without performance degradation.

Data Processing and Analysis: The backend Python scripts enable advanced data processing capabilities, such as sentiment analysis of comments, engagement metrics, and performance tracking over time. This makes it easier to understand how content is performing and where improvements can be made.

Actionable Insights and Reporting: Visualizations and dashboards provide meaningful insights, such as identifying trending videos, analyzing comment sentiment, and discovering patterns in video engagement. Reports generated from this data can guide content strategies, marketing campaigns, and even inform future video production efforts.

Scalability and Flexibility: The system is designed to scale as data grows, allowing for the seamless addition of new videos, comments, and analytics. Whether used for small YouTube channels or large media organizations, the architecture is flexible and can adapt to changing data needs over time.

User-Friendly Interface:

The frontend interface, built with tools like Streamlit or Flask, ensures that even non-technical users can interact with and explore the data easily. This empowers content creators and marketers to understand their audience, optimize content, and make data-driven decisions.

REFERENCES

- Database System Concepts(6th Edition) By Abraham Silberschatz, Henry F. Korth,S. Sudarshan.

WEBSITES:

- Www.Geeksforgeeks.Com
- Www.W3schools.Com

REFERENCE VIDEOS :

YouTube Data Harvesting and Warehousing using MongoDB, SQL and Streamlit|Tamil|Creating virtual envi