**Sanjivani Rural Education Society’s**

**Sanjivani College of Engineering, Kopargaon-423603**

***(An Autonomous Institute Affiliated to Savitribai Phule Pune University, Pune) NAAC ‘A’ Grade Accredited***

### Department of Computer Engineering

***NBA Accredited-UG Program***

A Report on

**Mini Project-Problem Solving Using Python Lab (ESIT136)**

**Title: Social Media Analytics Dashboard**

By **Group ID-**

F.Y.B.Tech. Computer Engineering

Under the Guidance of

Prof . S.S. Gawali

DURING THE ACADEMIC YEAR 2024-25 (Semester-II)

**Team Members**

|  |  |  |
| --- | --- | --- |
| **Roll No.** | **PRN No.** | **Name of Team Members** |
| **124** | **0124UCSM1124** | **Metkar Nikhil** |
| **134** | **0124UCSM1134** | **Dawange vaibhav** |
| **146** | **UCS23M1072** | **Kolhe Rohan** |

Prof . S.S. Gawali Prof. Dr. M.A Jawale

Head, Computer Engineering

Course Coordinator

| **Sr No.** | **Title** | **Page No.** |
| --- | --- | --- |
| 1. | Introduction | 3 |
| 1.1 | Functionality | 3 |
| 1.2 | Objectives | 4 |
| 2. | Modules | 6 |
| 2.1 | YouTube API Integration | 6 |
| 2.2 | Graphical user interface (GUI) | 8 |
| 2.3 | Visualization with Matplotlib | 10 |
| 3. | Basic Python Concept | 12 |
| 3.1 | Modules and Libraries | 12 |
| 3.2 | Functions | 12 |
| 3.3 | Global Variables | 13 |
| 3.4 | Conditional Statements | 13 |
| 3.5 | Loops | 14 |
| 3.6 | String Handling | 14 |
| 3.7 | List Operations | 15 |
| 3.8 | Dictionaries | 15 |
| 4 | Code | 17 |
| 5 | Conclusion | 23 |

**CHAPTER 1: INTRODUCTION**

This project, titled "YouTube Channel Statistics and Engagement Analyzer", is a Python-based application that utilizes the YouTube Data API to retrieve and visualize important statistics about a YouTube channel. The application is developed using the Tkinter library for building a graphical user interface (GUI) and Matplotlib for data visualization.

The tool allows users to search for any YouTube channel by name, fetch channel-level statistics such as subscriber count, total views, and video count, and further analyze the performance of the channel’s latest five videos. It calculates the engagement rate based on likes, comments, and views, providing a measure of audience interaction. Additionally, the project visualizes video views using an interactive bar chart, enhancing the user's understanding of the channel's recent performance.

The main objective of the project is to demonstrate the integration of API usage with Python GUI programming and to offer a simple yet powerful interface for analyzing YouTube content creators' performance metrics. This application can assist creators, marketers, and researchers in better understanding audience engagement trends.

**1.1 Functionality:**

This program is like a small app that helps you check stats of any YouTube channel. You just type the channel name and hit "Search," and it shows you useful numbers like subscribers, total views, and how well recent videos are performing. Here’s what it does step by step:

1. **You Type a Channel Name:**
   * There's a text box where you write the name of the YouTube channel you want to check.
2. **It Looks Up the Channel:**
   * The program connects to YouTube using the API (a special way apps talk to YouTube).
   * It searches for the channel you typed in and finds its ID and details.
3. **It Gets Channel Stats:**
   * After finding the channel, it pulls information like:
     + Number of subscribers
     + Total views
     + Total number of videos
4. **It Fetches the Last 5 Videos:**
   * The program checks out the latest 5 videos uploaded by the channel.
5. **It Collects Video Stats:**
   * For each of those 5 videos, it grabs:
     + Number of views
     + Number of likes
     + Number of comments
6. **It Calculates Engagement Rate:**
   * It uses likes, comments, and views to figure out how much people are interacting (engagement rate).
7. **It Shows You the Numbers:**
   * On the main window, it displays:
     + Channel name
     + Subscriber count
     + Total views
     + Total videos
     + Engagement rate
8. **It Can Show a Graph:**
   * If you click the "Show Views Graph" button:
     + A new window pops up with a bar chart showing views for each of the 5 videos.
     + It also lists the video titles and their exact view counts.

**1.2 Objectives:**

This project aims to build a simple and user-friendly app that helps people check YouTube channel statistics and video performance easily. Below are the main objectives of this project:

1. **To Create a Simple Tool for YouTube Stats:**

* Build an app where users can type a YouTube channel name and quickly get its stats without any technical skills.

1. **To Fetch and Display Channel Information:**

* Show important channel details like:
* Number of subscribers
* Total views
* Total number of videos uploaded

1. **To Analyze the Latest 5 Videos:**

* Automatically find and display data about the 5 most recent videos from the selected channel.

1. **To Collect Detailed Video Statistics**:

* For each of those 5 videos, the app fetches:
* Views count
* Likes count
* Comments count

1. **To Calculate the Engagement Rate:**

* Use the collected likes, comments, and views to compute an engagement rate, showing how actively people are interacting with the content.

1. **To Visualize the Data in Graph Form:**

* Display a bar chart that shows views for each of the last 5 videos to make the data visually clear and easy to understand.

1. **To Provide an Easy User Interface Using Tkinter:**

* Design the app with a simple and clean interface where users can search and view results smoothly with buttons and text boxes.

CHAPTER 2: MODULES

The project is divided into several logical modules, each handling specific functionality to ensure clarity, maintainability, and scalability:

**2.1 YouTube API Integration**

**Purpose:**

The YouTube API Integration module forms the backbone of the project by connecting the application to the **YouTube Data API v3**. It enables the system to dynamically search for a channel, retrieve channel-level statistics, and collect detailed information about individual videos.  
Without this module, real-time data fetching from YouTube would not be possible. It ensures that the user interacts with live, updated information rather than static or outdated datasets.

This module is crucial because it automates the process of extracting public data from YouTube, reducing manual efforts and errors, and provides structured data needed for further processing, analysis, and visualization.

**Key Functions:**

* **get\_channel\_stats(channel\_name)**
  + **Functionality:**  
    Searches for a YouTube channel based on the user's input (channel name) and retrieves important metadata.
  + **Process:**
    - Initiates a search request using the search().list() method of the YouTube API.
    - Extracts the channel ID from the search result.
    - Fetches channel statistics such as **subscriber count**, **total view count**, and **total video count** by calling the channels().list() method with the retrieved channel ID.
    - If the channel is not found, the function gracefully handles the error and returns a None response, ensuring that the application remains stable.
* **get\_video\_ids(channel\_id, max\_results=5)**
  + **Functionality:**  
    Retrieves a list of recent videos uploaded by the channel.
  + **Process:**
    - Sends a request to the YouTube API to fetch videos associated with the given channel ID.
    - Limits the results to the latest **five videos** to keep the application lightweight and focused.
    - Extracts both the **video IDs** (for internal processing) and **video titles** (for user-friendly display).
* **get\_video\_stats(videos)**
  + **Functionality:**  
    Gathers detailed statistics for each video, such as the number of views, likes, and comments.
  + **Process:**
    - Compiles a list of video IDs and sends a batch request to the YouTube API.
    - Parses the response to extract relevant statistics for each video.
    - Stores **views** and **shortened titles** in global variables, which are later used for graph plotting and detailed reporting.

**Technologies and Libraries Used:**

* **Google API Client Library (googleapiclient.discovery)**
  + Provides a Python interface for interacting with Google's APIs, including YouTube.
  + Simplifies building API requests and handling responses in a structured manner.
* **YouTube Data API v3**
  + An official API service provided by YouTube, enabling developers to programmatically access YouTube content such as videos, playlists, and channels.
  + Supports methods like search().list(), channels().list(), and videos().list(), all of which are used in this project.

**2.2. GRAPHICAL USER INTERFACE (GUI)**

**Purpose:**

The Graphical User Interface (GUI) module is designed to provide an intuitive, user-friendly platform through which users can interact with the application.  
Instead of working with command-line inputs and outputs, users are presented with a well-structured window where they can easily enter the channel name, trigger data fetching, view the results, and interact with visualizations.

This module enhances the overall user experience by making the application **accessible to non-programmers** and by presenting complex data in an organized, readable manner.

**Key Components and Functionalities:**

* **Main Window Setup**
  + Created using the Tkinter library.
  + The main window is configured with an appropriate **title** ("YouTube Channel Stats") and **size** (500x400 pixels) to fit all UI components comfortably.
  + Ensures that all interactive elements are neatly packed with proper padding for a clean and uncluttered look.
* **Text Input Box (Text)**
  + A text box is placed at the top of the window where users can enter the name of the YouTube channel they want to search.
  + Height and width are optimized to allow easy typing without overwhelming the UI.
* **Search Button (Button)**
  + Labeled "Search," this button is linked to the on\_search() function.
  + When clicked, it triggers the complete process of searching for the channel, fetching its data, and displaying the results.
  + Provides immediate, responsive feedback to user actions.
* **Dynamic Display of Results (Message)**
  + After fetching the channel statistics, a **Message widget** is used to display the detailed results.
  + Statistics such as subscriber count, total views, video count, and engagement rate are shown in a **formatted, easy-to-read text block**.
  + If errors occur (like channel not found or no videos retrieved), appropriate error messages are dynamically generated and shown in the interface.
* **Graph Button (Button)**
  + After a successful search, an additional button labeled "Show Views Graph" appears.
  + Clicking this button calls the create\_views\_graph() function, opening a **new window** with a detailed graph of the latest video views.
  + This layered interaction keeps the main window uncluttered while allowing more advanced users to explore deeper insights.
* **New Window for Graph (Toplevel)**
  + The graph visualization appears in a separate pop-up window, maintaining focus on specific tasks.
  + This multi-window approach improves usability and organizes different types of content into dedicated spaces.

**Technologies and Libraries Used:**

* **Tkinter**
  + Python’s standard GUI toolkit, used for creating windows, frames, labels, buttons, text boxes, and messages.
  + Provides simplicity, lightness, and full control over event-driven programming (such as button click actions).
* **Matplotlib (FigureCanvasTkAgg)**
  + Although primarily a plotting library, Matplotlib is integrated into Tkinter using FigureCanvasTkAgg.
  + Enables embedding complex graphs directly into Tkinter windows.

**2.3. DATA VISUALIZATION WITH MATPLOTLIB**

**Purpose:**

The Data Visualization module is responsible for converting raw numerical data (such as video view counts) into clear, engaging, and informative visual formats.  
Rather than displaying statistics in plain text, the application uses **bar graphs** to make data interpretation faster, more intuitive, and visually appealing.  
Graphs help users quickly identify trends, compare video performance, and understand channel engagement at a glance.

This module significantly enhances the application's analytical capabilities by providing **graphical insights**, making the experience more interactive and data-driven.

**Key Components and Functionalities:**

* **Graph Window Creation (Toplevel)**
  + A new window is created specifically for displaying the graph, ensuring that the main application window remains uncluttered.
  + This separate window is titled appropriately ("Video Views Graph") and sized (700x500 pixels) to ensure the graph and other elements fit comfortably.
* **Graph Construction (Figure and plot.bar())**
  + A Figure object is created using Matplotlib, which serves as a blank canvas for plotting.
  + A subplot is added to the figure where the actual bar graph is drawn.
  + The x-axis represents the **latest video titles** (shortened for better readability), while the y-axis represents the corresponding **view counts**.
* **Data Labeling**
  + Each bar in the graph is labeled with the exact view count value.
  + View numbers are formatted using a custom **format\_number()** function, converting large numbers into human-readable formats like K (thousand) or M (million).
  + Value labels are placed just above each bar for maximum clarity.
* **Axis Formatting**
  + The y-axis uses a customized formatter (FuncFormatter) to consistently display numbers in a readable format, avoiding overwhelming the user with long digit strings.
  + X-axis labels are rotated slightly (rotation=20) to avoid overlapping text, improving the visual neatness of the graph.
* **Graph Layout**
  + Grid lines are added along the y-axis to make it easier to visually trace data values.
  + The graph layout is adjusted (tight\_layout()) to ensure all elements fit properly without cutting off labels or titles.
* **Details Section**
  + Below the graph, a **text area** is added displaying a **Detailed Video Views** report.
  + This report lists each video title (full name) and its exact view count in a neatly formatted style.
  + The text area is set to **read-only** (state=DISABLED) to prevent accidental editing.

**Technologies and Libraries Used:**

* **Matplotlib**
  + A widely-used plotting library for Python that enables high-quality static, animated, and interactive visualizations.
  + Used here to create bar graphs that represent video view data clearly and aesthetically.
* **Tkinter Integration (FigureCanvasTkAgg)**
  + Bridges Matplotlib figures into Tkinter windows seamlessly.
  + Allows embedding interactive and responsive graphs within a Tkinter-based graphical user interface.

**CHAPTER 3: BASIC PYTHON CONCEPT**

There are many python concepts used in this project. Some concepts are explained below:

**3.1. Modules and Libraries:**

Modules allow you to organize your Python programs by separating functionalities. Libraries are collections of pre-written code that provide specific capabilities without reinventing the wheel.

**from** **tkinter** **import** \* # For creating the GUI

**import** **tkinter** **as** **tk** # Alternative way to access Tkinter components

**from** **googleapiclient.discovery** **import** build # For accessing YouTube Data API

**import** **matplotlib** # For plotting graphs

**from** **matplotlib.figure** **import** Figure

**from** **matplotlib.backends.backend\_tkagg** **import** FigureCanvasTkAgg

 tkinter is used to build windows, buttons, text boxes, etc.

 googleapiclient.discovery.build is used to interact with the YouTube API.

 matplotlib is used to create and customize the graph.

**3.2. FUNCTIONS:**

Functions in Python are blocks of organized, reusable code used to perform a single, related action. They help break the program into smaller modules for better readability and maintainability.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | **def** **get\_channel\_stats**(channel\_name):  # Fetches basic statistics of a YouTube channel  ...  **def** **get\_video\_ids**(channel\_id, max\_results=**5**):  # Retrieves video IDs from the channel  ...  **def** **get\_video\_stats**(videos):  # Gets view, like, and comment statistics for each video  ...  **def** **calculate\_engagement**(video\_stats):  # Calculates engagement based on views, likes, comments  ...  **def** **create\_views\_graph**():  # Creates and shows a bar graph of video views  ...  **def** **on\_search**():  # Main function that ties the search button to the logic  ... |

Each function has a specific responsibility, making your code modular and easy to debug.

**3.3. GLOBAL VARIABLES**

Global variables are variables declared outside of any function and accessible across the entire file or module.

|  |  |
| --- | --- |
| 1  2 | video\_views = []  video\_titles = [] |

And inside functions, you used the “global” keyword:

|  |  |
| --- | --- |
| 1  2  3 | **global** video\_views, video\_titles  video\_views = []  video\_titles = [] |

This allows multiple functions (create\_views\_graph, get\_video\_stats, etc.) to access and modify the same lists.

**3.4.** **CONDITIONAL STATEMENTS**

Conditional statements (if, else, elif) allow decision-making in programs based on certain conditions.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1  2  3  4  5  6  7  8 | **if** **not** search\_response["items"]:  **print**("Channel not found!")  **return** None, None, None  **if** **not** channel\_response["items"]:  **print**("Could not fetch channel details.")  **return** None, None, None  **if** **not** channel\_name:  **return** |  |  |  |  |

Conditional statements make sure your program responds correctly if the user input is wrong or if the API does not return expected data — **preventing crashes**.

3.5. LOOPS

Loops (for, while) are used to repeat actions multiple times efficiently.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | **for** video **in** video\_response["items"]:  videos.append({  "id": video["id"]["videoId"],  "title": video["snippet"]["title"]  })  **for** i, video **in** enumerate(video\_response["items"]):  ...  **for** bar, value **in** zip(bars, y):  height = bar.get\_height()  plot.text(bar.get\_x() + bar.get\_width()/**2.**, height,  format\_number(value),  ha='center', va='bottom', rotation=**0**) |

Loops allow your program to handle multiple videos, titles, and graphs dynamically, without hardcoding for each video

3.6. **STRING HANDLING**

String operations like slicing, concatenation, and formatting allow you to manipulate text data efficiently.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | short\_title = title[:**15**] + "..." **if** len(title) > **15** **else** title  f"Subscribers: {int(stats.get('subscriberCount', 0)):,}"  stats\_message = (  f"📊 Channel Statistics 📊**\n**"  f"Channel Name: {channel\_name}**\n**"  ...  ) |

String handling makes your displayed messages more clean, readable, and professional.

3.7. **LIST OPERATIONS**

Lists are used to store multiple items in a single variable and allow operations like adding, updating, and looping through them.

* To store video data:

|  |  |
| --- | --- |
| 1  2 | video\_views = []  video\_titles = [] |

* Appending views and titles dynamically:

|  |  |
| --- | --- |
| 1  2 | video\_views.append(views)  video\_titles.append(short\_title) |

* Managing structured data:

1 videos = []

**2 for** video **in** video\_response["items"]:

3 videos.append({...})

Lists make it easy to collect all the video data and process it together (e.g., displaying in a graph, summarizing views).

**3.8. DICTIONARIES**

Dictionaries store data in key-value pairs and are perfect for structured, related data.

* To collect and store detailed video statistics:

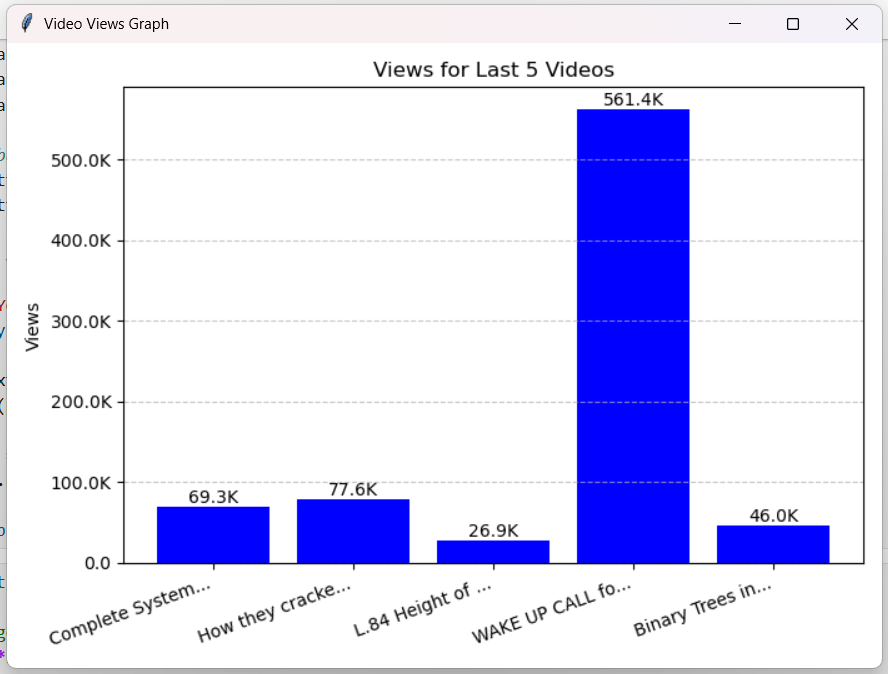
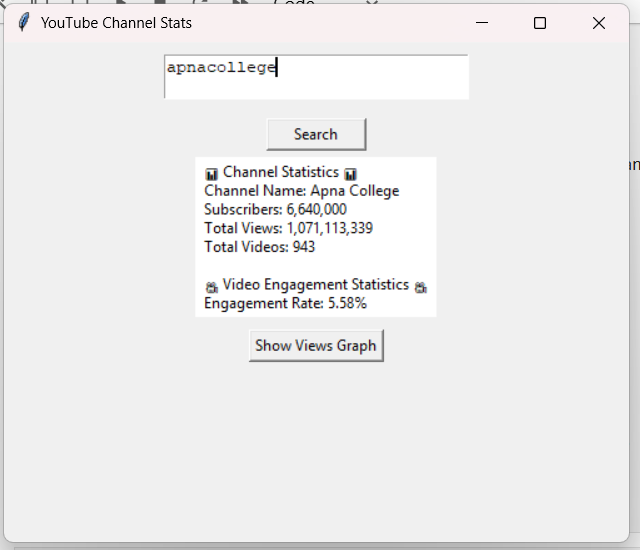
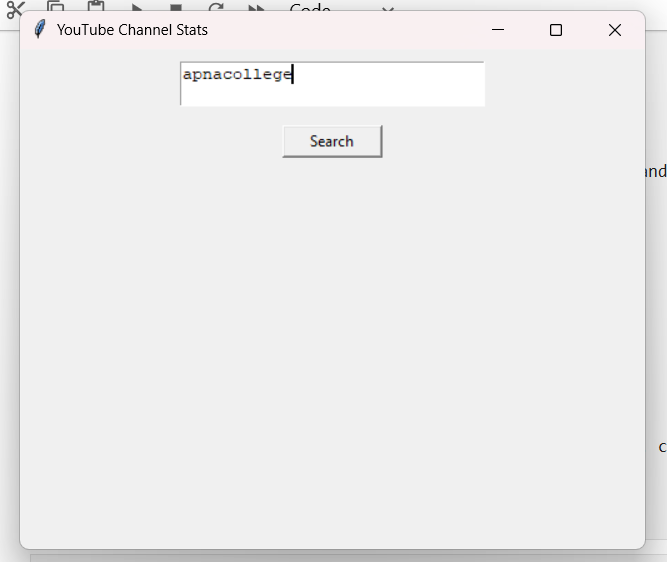
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | video\_stats.append({  "video\_id": video\_id,  "title": title,  "views": views,  "likes": likes,  "comments": comments  }) |

* To read structured API responses:

|  |  |
| --- | --- |
| 1 | search\_response["items"][**0**]["snippet"]["title"] |

**CHAPTER 4: CODE**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87    88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255 | **from** **tkinter** **import** \*  **import** **tkinter** **as** **tk**  **from** **googleapiclient.discovery** **import** build  **import** **matplotlib**  matplotlib.use("TkAgg")  **from** **matplotlib.figure** **import** Figure  **from** **matplotlib.backends.backend\_tkagg** **import** FigureCanvasTkAgg  # Replace with your actual API key  API\_KEY = "AIzaSyD0X5NyqaHpblXjHUa7A100OdE9\_OKllr8"  # Global variables to store video data  video\_views = []  video\_titles = []  **def** **get\_channel\_stats**(channel\_name):  youtube = build("youtube", "v3", developerKey=API\_KEY)  # Search for the channel using its name  search\_response = youtube.search().list(  q=channel\_name,  type="channel",  part="snippet",  maxResults=**1**  ).execute()  **if** **not** search\_response["items"]:  **print**("Channel not found!")  **return** None, None, None  channel\_id = search\_response["items"][**0**]["id"]["channelId"]  channel\_name = search\_response["items"][**0**]["snippet"]["title"]  # Fetch channel statistics  channel\_response = youtube.channels().list(  id=channel\_id,  part="statistics"  ).execute()  **if** **not** channel\_response["items"]:  **print**("Could not fetch channel details.")  **return** None, None, None  stats = channel\_response["items"][**0**]["statistics"]    **return** channel\_id, channel\_name, stats  **def** **get\_video\_ids**(channel\_id, max\_results=**5**):  youtube = build("youtube", "v3", developerKey=API\_KEY)  video\_response = youtube.search().list(  channelId=channel\_id,  part="id,snippet",  order="date",  maxResults=max\_results,  type="video"  ).execute()    videos = []  **for** video **in** video\_response["items"]:  videos.append({  "id": video["id"]["videoId"],  "title": video["snippet"]["title"]  })    **return** videos  **def** **get\_video\_stats**(videos):  youtube = build("youtube", "v3", developerKey=API\_KEY)  video\_ids = [video["id"] **for** video **in** videos]  video\_response = youtube.videos().list(  id=",".join(video\_ids),  part="statistics"  ).execute()  **global** video\_views, video\_titles  video\_views = []  video\_titles = []    video\_stats = []  **for** i, video **in** enumerate(video\_response["items"]):  stats = video["statistics"]  video\_id = video["id"]    # Find the corresponding title  title = next((v["title"] **for** v **in** videos **if** v["id"] == video\_id), f"Video {i+1}")    views = int(stats.get("viewCount", **0**))  likes = int(stats.get("likeCount", **0**))  comments = int(stats.get("commentCount", **0**))    # Store views and title  video\_views.append(views)  # Create a shortened version of the title  short\_title = title[:**15**] + "..." **if** len(title) > **15** **else** title  video\_titles.append(short\_title)    video\_stats.append({  "video\_id": video\_id,  "title": title,  "views": views,  "likes": likes,  "comments": comments  })    **return** video\_stats  **def** **calculate\_engagement**(video\_stats):  total\_views = sum(video["views"] **for** video **in** video\_stats)  total\_likes = sum(video["likes"] **for** video **in** video\_stats)  total\_comments = sum(video["comments"] **for** video **in** video\_stats)  **if** total\_views == **0**:  **return** **0**  engagement\_rate = ((total\_likes + total\_comments) / total\_views) \* **100**  **return** round(engagement\_rate, **2**)  **def** **format\_number**(num):  """Format large numbers to K (thousands) or M (millions)"""  **if** num >= **1000000**:  **return** f"{num/1000000:.1f}M"  **elif** num >= **1000**:  **return** f"{num/1000:.1f}K"  **else**:  **return** str(num)  **def** **create\_views\_graph**():  # Create a new window for the graph  graph\_window = Toplevel(root)  graph\_window.title("Video Views Graph")  graph\_window.geometry("700x500")    figure = Figure(figsize=(**7**, **5**), dpi=**100**)  plot = figure.add\_subplot(**1**, **1**, **1**)    # Use the stored video views  y = video\_views  x = range(**1**, len(y) + **1**)    # Create the bar chart  bars = plot.bar(x, y, color="blue")    # Add value labels above each bar with proper formatting  **for** bar, value **in** zip(bars, y):  height = bar.get\_height()  plot.text(bar.get\_x() + bar.get\_width()/**2.**, height,  format\_number(value),  ha='center', va='bottom', rotation=**0**)    # Format the y-axis to display numbers in K and M format  **def** **y\_fmt**(x, pos):  **return** format\_number(x)    **from** **matplotlib.ticker** **import** FuncFormatter  plot.yaxis.set\_major\_formatter(FuncFormatter(y\_fmt))    # Set up the chart labels  plot.set\_ylabel("Views")  plot.set\_title("Views for Last 5 Videos")  plot.set\_xticks(x)  plot.set\_xticklabels(video\_titles, rotation=**20**, ha='right')    # Add grid lines for better readability  plot.grid(axis='y', linestyle='--', alpha=**0.7**)    # Adjust layout to ensure everything fits  figure.tight\_layout()    # Display the figure  canvas = FigureCanvasTkAgg(figure, graph\_window)  canvas.get\_tk\_widget().pack(fill=tk.BOTH, expand=True)  canvas.draw()    # Add a details section below the graph  details\_frame = Frame(graph\_window)  details\_frame.pack(fill=tk.X, pady=**10**)    details\_text = Text(details\_frame, height=**6**, width=**80**)  details\_text.pack(padx=**10**)    # Add detailed view information  details\_text.insert(END, "Detailed Video Views:**\n\n**")  **for** i, (title, views) **in** enumerate(zip(video\_titles, video\_views)):  full\_title = next((v["title"] **for** v **in** video\_stats **if** v["views"] == views), title)  details\_text.insert(END, f"Video {i+1}: {full\_title}**\n**")  details\_text.insert(END, f"Views: {views:,}**\n\n**")    details\_text.config(state=DISABLED) # Make it read-only  **def** **on\_search**():  **global** video\_stats  channel\_name = textBox.get("1.0", "end-1c").strip()  **if** **not** channel\_name:  **return**  # Clear previous messages  **for** widget **in** root.winfo\_children():  **if** isinstance(widget, Message) **or** isinstance(widget, Button) **and** widget != buttonCommit:  widget.destroy()  channel\_id, channel\_name, stats = get\_channel\_stats(channel\_name)  **if** **not** channel\_id:  messageVar = Message(root, text="Channel not found!", width=**400**)  messageVar.config(bg='white')  messageVar.pack()  **return**  videos = get\_video\_ids(channel\_id)  **if** **not** videos:  messageVar = Message(root, text="No videos found!", width=**400**)  messageVar.config(bg='white')  messageVar.pack()  **return**  video\_stats = get\_video\_stats(videos)  **if** **not** video\_stats:  messageVar = Message(root, text="Could not fetch video statistics.", width=**400**)  messageVar.config(bg='white')  messageVar.pack()  **return**  engagement\_rate = calculate\_engagement(video\_stats)    stats\_message = (  f"📊 Channel Statistics 📊**\n**"  f"Channel Name: {channel\_name}**\n**"  f"Subscribers: {int(stats.get('subscriberCount', 0)):,}**\n**"  f"Total Views: {int(stats.get('viewCount', 0)):,}**\n**"  f"Total Videos: {int(stats.get('videoCount', 0)):,}**\n\n**"  f"🎥 Video Engagement Statistics 🎥**\n**"  f"Engagement Rate: {engagement\_rate}%"  )  messageVar = Message(root, text=stats\_message, width=**400**)  messageVar.config(bg='white')  messageVar.pack()    # Add a button to show the graph  graph\_button = Button(root, text="Show Views Graph", command=create\_views\_graph)  graph\_button.pack(pady=**10**)  # Tkinter UI Setup  root = Tk()  root.title("YouTube Channel Stats")  root.geometry("500x400")  textBox = Text(root, height=**2**, width=**30**)  textBox.pack(pady=**10**)  buttonCommit = Button(root, height=**1**, width=**10**, text="Search", command=on\_search)  buttonCommit.pack(pady=**5**)  root.mainloop() |



**CHAPTER 5: CONCLUSION**

The development of the YouTube Channel Statistics and Engagement Analyzer has provided an effective and user-friendly tool for creators, marketers, and analysts to gain insightful metrics about YouTube channels. By utilizing the YouTube Data API in combination with Python-based technologies such as Tkinter for GUI and Matplotlib for data visualization, the system successfully extracts and presents key performance indicators like subscriber count, video views, likes/dislikes, comment analysis, and engagement rates. This application not only simplifies the process of gathering analytics but also enables users to make informed decisions for content strategy and audience growth.

Moreover, the modular design and integration of open-source libraries make the project both flexible and extendable. Despite some limitations related to API usage quotas and real-time data constraints, the project lays a solid foundation for future enhancements such as sentiment analysis, competitor comparisons, or dashboard integration. In conclusion, this project demonstrates the practical application of programming and data analysis in the digital content ecosystem and serves as a valuable asset for anyone looking to monitor or optimize YouTube performance.