Birkbeck
UNIVERSITY OF LONDON

Report: AudioPulse Capstone Project

Student: Elena Milanese
Professor: Stelios Sotiriadis
Course: Big Data Analytics

July 2025

Part 0) ASSIGNMENT PREPARATION AND BOILERPLATE

Describe how your pipeline works from start to finish. Compare the performance of both serial and parallel downloading (reference: week 5 lab). Report which approach was faster, and explain any trade-offs you observed, such as added complexity or system load. Provide script snippets as needed.

To create a new virtual Python environment where to install the libraries needed for the "AudioPulse Capstone Project", the following code has been used (Fig.1):

```
python -m venv AudioPulse
```

where "AudioPulse" is the name of the virtual environment. To run the virtual environment, we need to activate it first:

```
source AudioPulse/bin/activate
```

Lastly, it is always a good idea to upgrade pip:

```
-m pip install —-upgrade pip
```

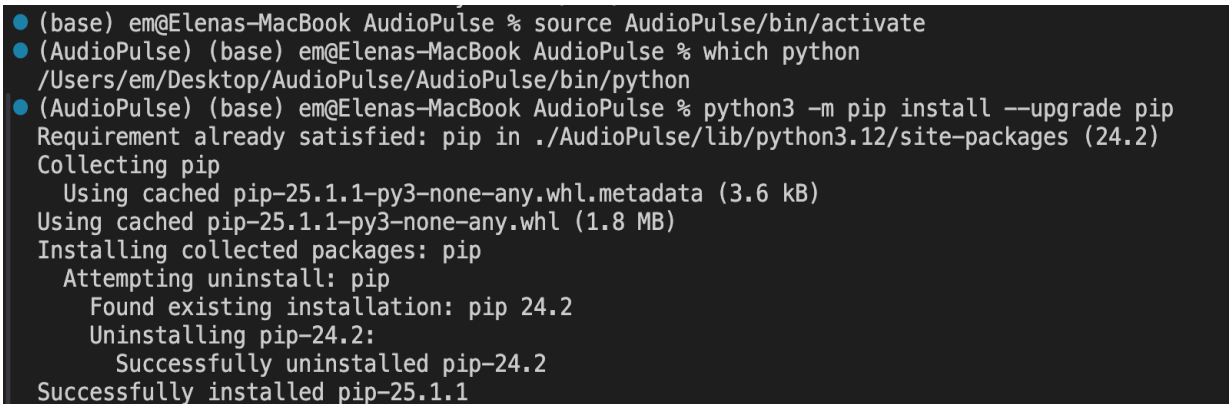


Fig.1

In Fig. 2 and Fig. 3 I have installed the libraries listed in the requirements.txt file: **yt_dlp**, **certify**, **pandas**, and **pyspark** in the virtual environment.

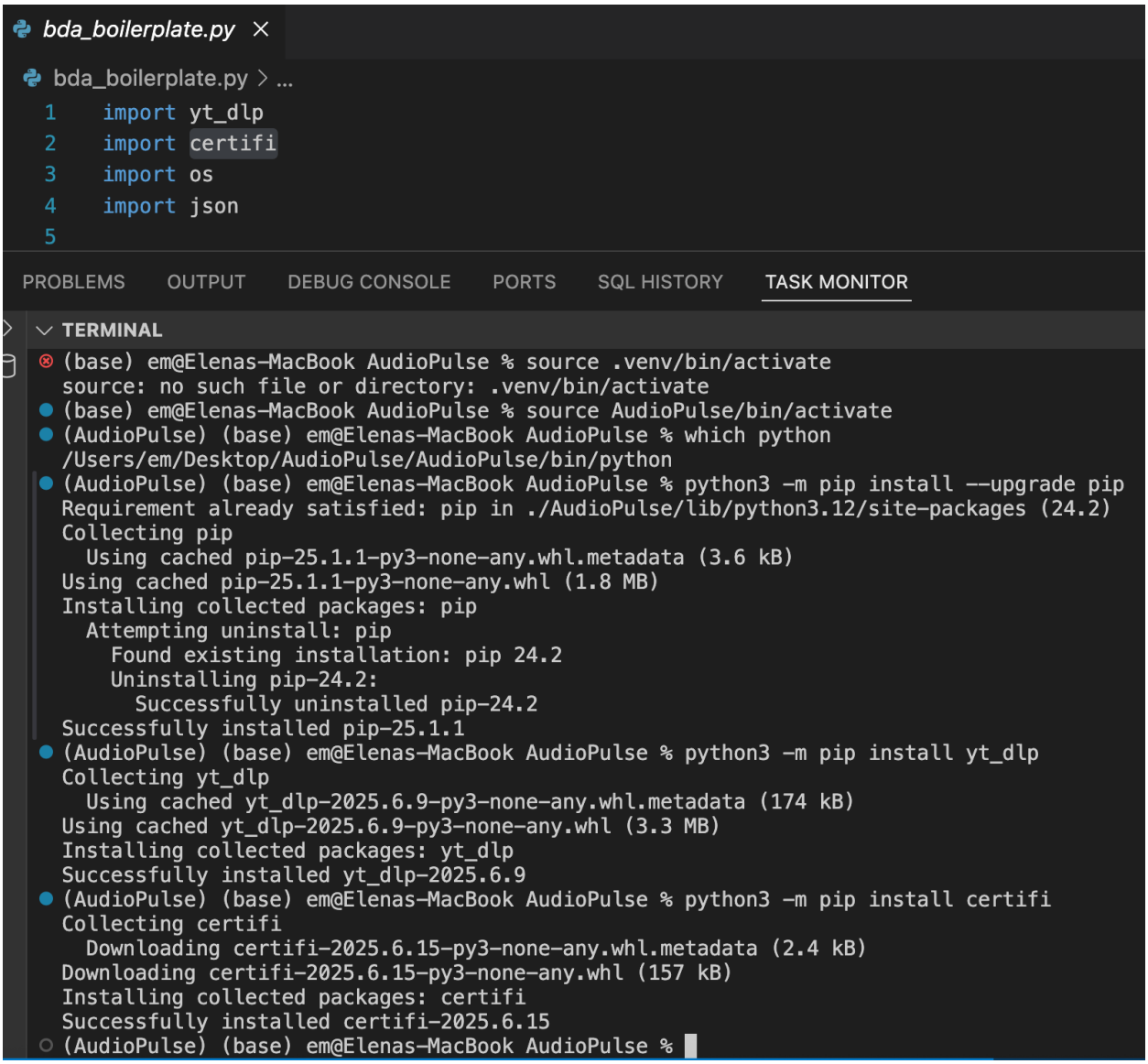

bda_boilerplate.py
bda_boilerplate.py > ...
1 import yt_dlp
2 import certifi
3 import os
4 import json
5
PROBLEMS
OUTPUT
DEBUG CONSOLE
PORTS
SQL HISTORY
TASK MONITOR
TERMINAL
(base) em@Elenas-MacBook AudioPulse % source .venv/bin/activate
source: no such file or directory: .venv/bin/activate
(base) em@Elenas-MacBook AudioPulse % source AudioPulse/bin/activate
(AudioPulse) (base) em@Elenas-MacBook AudioPulse % which python
/Users/em/Desktop/AudioPulse/AudioPulse/bin/python
(AudioPulse) (base) em@Elenas-MacBook AudioPulse % python3 -m pip install --upgrade pip
Requirement already satisfied: pip in ./AudioPulse/lib/python3.12/site-packages (24.2)
Collecting pip
Using cached pip-25.1.1-py3-none-any.whl.metadata (3.6 kB)
Using cached pip-25.1.1-py3-none-any.whl (1.8 MB)
Installing collected packages: pip
Attempting uninstall: pip
Found existing installation: pip 24.2
Uninstalling pip-24.2:
Successfully uninstalled pip-24.2
Successfully installed pip-25.1.1
(AudioPulse) (base) em@Elenas-MacBook AudioPulse % python3 -m pip install yt_dlp
Collecting yt_dlp
Using cached yt_dlp-2025.6.9-py3-none-any.whl.metadata (174 kB)
Using cached yt_dlp-2025.6.9-py3-none-any.whl (3.3 MB)
Installing collected packages: yt_dlp
Successfully installed yt_dlp-2025.6.9
(AudioPulse) (base) em@Elenas-MacBook AudioPulse % python3 -m pip install certifi
Collecting certifi
Downloading certifi-2025.6.15-py3-none-any.whl.metadata (2.4 kB)
Downloading certifi-2025.6.15-py3-none-any.whl (157 kB)
Installing collected packages: certifi
Successfully installed certifi-2025.6.15
(AudioPulse) (base) em@Elenas-MacBook AudioPulse %

Fig. 2

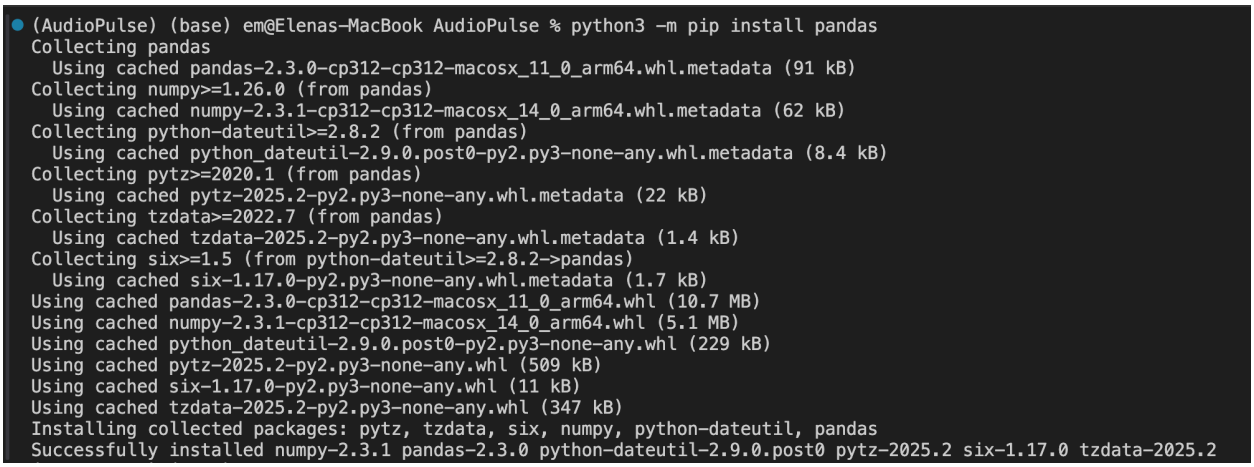


Fig.3

I have run the **bda_boilerplate.py** file and explored the functions and their functionalities.

Part 1) AUDIO DOWNLOAD PIPELINE AND LOGGING

Step 1: Collect video URLs

- **Find and copy 10 to 15 random YouTube video links that include relevant audio content (e.g. music, speech, podcasts).**
- **Save them in a plain text file named video_urls.txt**
- **Ensure each URL is on a separate line.**

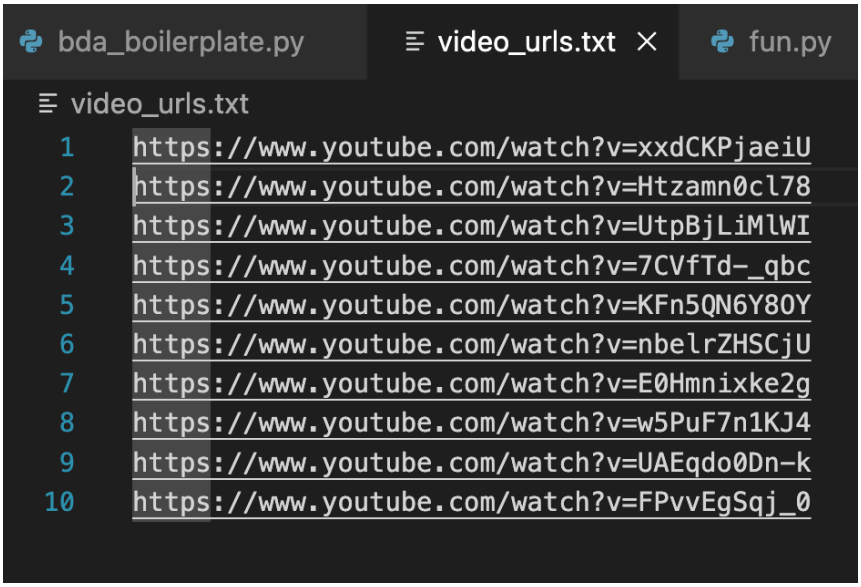


Fig. 4

I created the video_urls.txt file and saved 10 YouTube URLs following the instructions on Step 1 (note: later I added a new video from the same uploader to test 3.1.2, so the total number of YouTube URLs would be 11).

Step 2: Load the URLs into Python

- **Write a Python script that reads video_urls.txt and loads the URLs into a list or another suitable data structure.**
- **You will later pass these URLs to the download script.**

I have created a fun.py file, where I stored all my functions. As recommended in the Step 5, I am wrapping my code in "try/except" blocks, and I have found some interesting errors to raise: a type of error I decided to include is the common *FileNotFoundError*. If the files are not to be found, **load_url_list()** will return an empty list (Fig. 5).

```python
def load_url_list(file_path):
    """Load YouTube URLs from a text file into a list."""
    try:
        with open(file_path) as f:
            return [line.strip() for line in f if line.strip()]
    except FileNotFoundError:
        print(f"File not found: {file_path}")
        return [] #return an empty list if the file is not found
```

Fig. 5

Step 3: Download audio and metadata

- **Write a script that takes each URL, downloads the audio using yt-dlp, and stores metadata as JSON files.**
- **Save all:**
  a) **Audio files to a folder called audio_output**
  b) **Metadata files to the same folder**
  c) **Logs to a folder called logs. You can use the os.makedirs("logs", exist_ok= True) to create the folder.**

I have used some code from the **bda_boilerplate.py** file, in particular the function **download_youtube_audio_with_metadata(url)** that is already doing what requested, and I added a new feature. The function will retry to download the file 5 times before giving up. To do so, I used a while loop and I initialized the retry counter at 1 (Fig.6):

```python
retry = 1
while True:
    try:
        info = bda_boilerplate.get_video_info(url)
        metadata = bda_boilerplate.extract_metadata(info)
        json_path = bda_boilerplate.save_metadata_to_file(metadata, metadata["title"])
        print(f"✅ Done: {metadata['title']}\n📄 Metadata: {json_path}")
        log_entry(url, True)
        return
    except Exception as e:
        log_entry(url, False)
        if retry <= 5:
            print(f"🚧 Failed to download, retry attempt number: {retry}")
            retry += 1
        else:
            print(f"❌ Failed to download: {url}\n   Error: {e}")
            return
```

Fig. 6

d) **Test your script in two modes:**
   - **Serial mode -> download one video at a time (reference: notes from week 5)**
   - **Parallel mode -> download multiple videos at once using threads (reference: notes from week 6)**
   - **Comment: implement a parallel version that can download up to 5 videos at the same time. After both versions run, compare the performance and explain:**
   - **Which one is faster?**
   - **What are the trade-offs (complexity, system load..)?**
   - **How do time and space complexity change?**

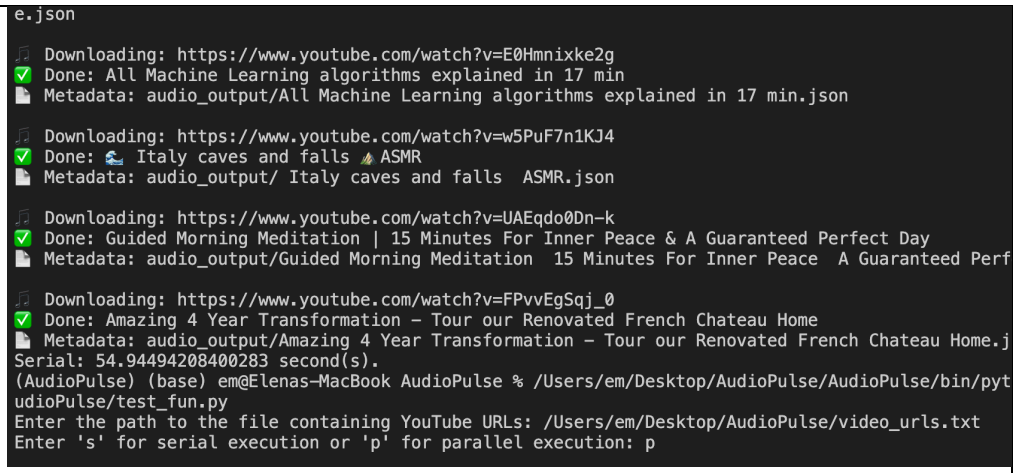
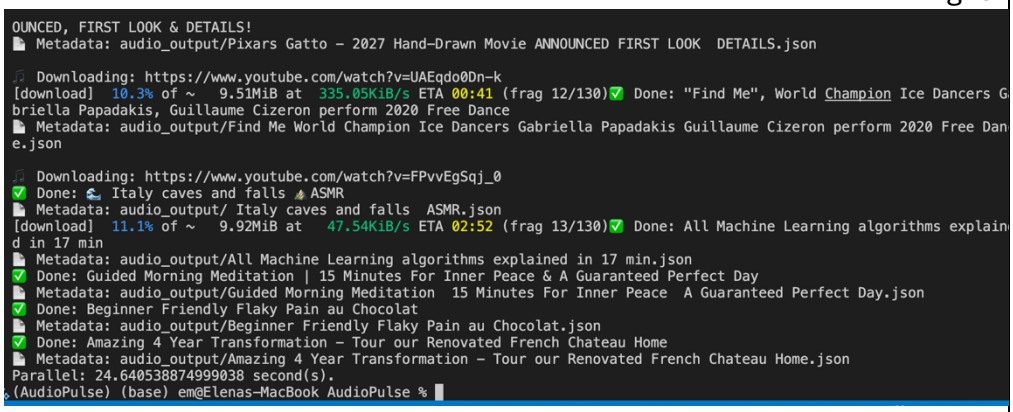
In the fun.py file, I added two functions: one for downloading files using serial execution and another utilizing parallel execution with some limitations (Fig. 7). For the parallel version, I employed **ThreadPoolExecutor** (reference: lesson 6 slides), which creates a pool of up to 5 worker" threads. These threads can execture tasks simultaneously, improving efficiency. Using the **ThreadPoolExecutor** is beneficial because it eliminates the need to create a new thread for each task. Instead, the thread pool is always available, ready to process tasks as soon as one finishes, making the execution faster and more resounce-efficient.

```
39  #serial execution
40  def download_youtube_audio_with_metadata_serial(youtube_urls):
41      for url in youtube_urls:
42          download_youtube_audio_with_metadata(url)
43
44  #parallel execution with ThreadPoolExecutor from lesson 6
45  def download_youtube_audio_with_metadata_parallel(youtube_urls, max_threads=5):
46      with concurrent.futures.ThreadPoolExecutor(max_workers=max_threads) as executor:
47          executor.map(download_youtube_audio_with_metadata, youtube_urls)
48
```

Fig. 7

Performance comparison:

	Serial/Parallel

Seconds	 Fig. 8  Fig. 9 Serial (Fig. 7): 54.94494... Parallel (Fig. 8): 24.64053...
Complexity and system load	The parallel algorithm requires a thread synchronization and a mechanism for distributing the workload among the threads of the pool. The system load is higher in the parallel version because it has to handle several downloads at the same time (network traffic, etc.), whereas in the serial version there is only one download at a time.
How do time and space complexity change?	**Time complexity**: both versions have an asymptotic linear time complexity with respect to the number of videos to download, however in practice the parallel version reduces the required time by a factor equal to the number of workers (in this case, 5). **Space complexity**: both versions have a linear space complexity (n), with respect to the number of the files to download, because we store for each video a fixed and defined amount of information.

Step 4: Create a safe logger

- **Build a logging function that runs safely code in parallel**
- **After each download completes, write a log entry to a file named download_log.txt. Log in any format you prefer.**
- **Save all logs in the logs folder.**

It is important to try to prevent **race conditions**, which happen when multiple downloads try to write to the log file at once (reference: slides and notes week 6). To do so I added some functions to the fun.py file.

First, a global variable was added (log_f), which stores the log file object. This variable holds the actual file object that the function writes to. It is initially set to None, because the log file hasn't been opened yet (Fig. 10).

```
49    log_lock = threading.Lock()
50    log_f = None
51
```

Fig. 10

The **log_init()** function checks if the specified log directory (**log_dir**, default is "logs") exists. If not, it creates the directory

```
os.makedirs()
```

with the

```
exisit_ok=True
```

argument to avoid errors if the directory already exists. Then, the function constructs the log file's full path by joining the directory and file name **log_file.** If the file doesn't exist, it will be created, and if it exists, new entries will be added at the end of it. The global variable **log_f** is set to the opened log file object, allowing other functions to append data to this file. This function sets up everything needed for logging: the directory and the file are created, and the log file is opened and ready for future writes.

Log_entry(url, download) appends a structured log entry to the log file. The log entry structure is as follows:

- Timestamp. I used

  ```
  datetime.now().isformat(timespec='seconds'
  ```

  to get the curreny time in ISO format);
- url: the url being logged is passed as an argument;
- download status (Boolean: True if the download was successful, False if it wasn't).

The function uses the **log_lock** to ensure that only one thread can write to the log at a time. This function provides a structured log format, making sure that every log entry includes the relevant information. The use of thread synchronization ensures that the logs won't be corrupted when multiple threads are used.

Finally, the **log_close()** function closes the log file when done. It checks if the **log_f** is initialized, and if so, it closes the file and sets **log_f** again to **None**, to indicate that the log file is now closed. To handle any possible error, the function will print a message in the case that the log file was never opened. (Fig. 11)

```
log_lock = threading.Lock()
log_f = None

def log_init(log_dir="logs", log_file="download_log.txt"):
    """Initialize the log directory and file."""
    global log_f
    os.makedirs(log_dir, exist_ok=True)
    log_path = os.path.join(log_dir,log_file)
    log_f = open(log_path, "a")

def log_entry(url: str, download: bool):
    """Append a structured log entry to the log txt file with timestamp, in a JSON-like format."""
    if not log_f:
        print("Log file is not initialized. Call log_init() first.")
        return

    with log_lock:
        log_f.write(f"[{datetime.now().isoformat(timespec='seconds')}] - URL: {url} - Downloaded: {download}\n"

def log_close():
    """Close the log file."""
    global log_f
    if log_f:
        log_f.close()
        log_f = None
    else:
        print("Log file is not open.")
```

Fig. 11

A screenshot of the download_log.txt file can be found in Fig. 12.

Fig. 12

To run the functions I created a new file: **Part_1.py**, that acts as a main. In the file I first initialized the log file calling the **log_init()** function. Then, I created a little console to test the serial and parallel functions. The program asks the user to input the path containing the YouTube URLs; then, it asks to write an “s” if we want the serial execution, or a “p” if we want a parallel execution of the functions. It will also print the seconds it takes to execute the tasks. At the end, the **log_close()** is called to ensure the log file is closed (Fig. 13-14).

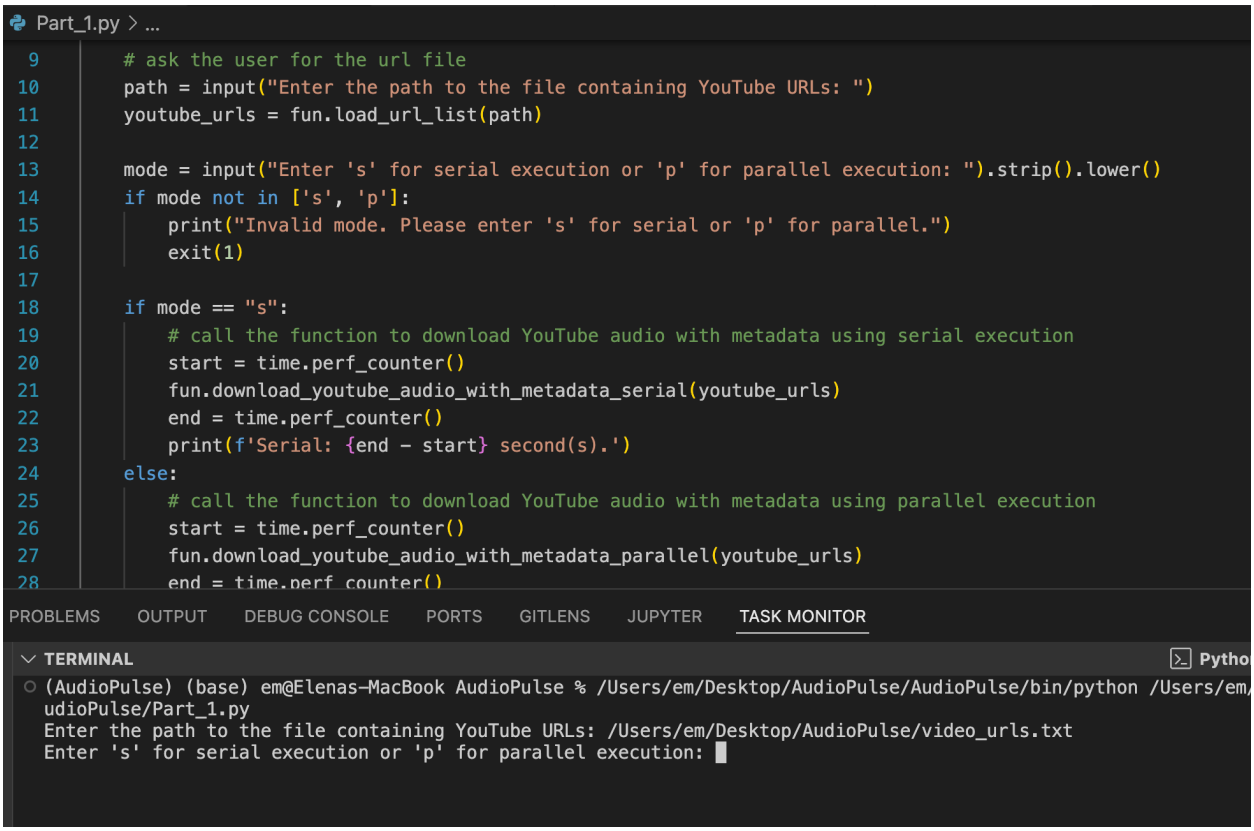

Part_1.py > ...
# ask the user for the url file
path = input("Enter the path to the file containing YouTube URLs: ")
youtube_urls = fun.load_url_list(path)
mode = input("Enter 's' for serial execution or 'p' for parallel execution: ").strip().lower()
if mode not in ['s', 'p']:
print("Invalid mode. Please enter 's' for serial or 'p' for parallel.")
exit(1)
if mode == "s":
# call the function to download YouTube audio with metadata using serial execution
start = time.perf_counter()
fun.download_youtube_audio_with_metadata_serial(youtube_urls)
end = time.perf_counter()
print(f'Serial: {end - start} second(s).')
else:
# call the function to download YouTube audio with metadata using parallel execution
start = time.perf_counter()
fun.download_youtube_audio_with_metadata_parallel(youtube_urls)
PROBLEMS
OUTPUT
DEBUG CONSOLE
PORTS
GITLENS
JUPYTER
TASK MONITOR
TERMINAL
(AudioPulse) (base) em@Elenas-MacBook AudioPulse % /Users/em/Desktop/AudioPulse/AudioPulse/bin/python /Users/em/AudioPulse/Part_1.py
Enter the path to the file containing YouTube URLs: /Users/em/Desktop/AudioPulse/video_urls.txt
Enter 's' for serial execution or 'p' for parallel execution:

Fig. 13

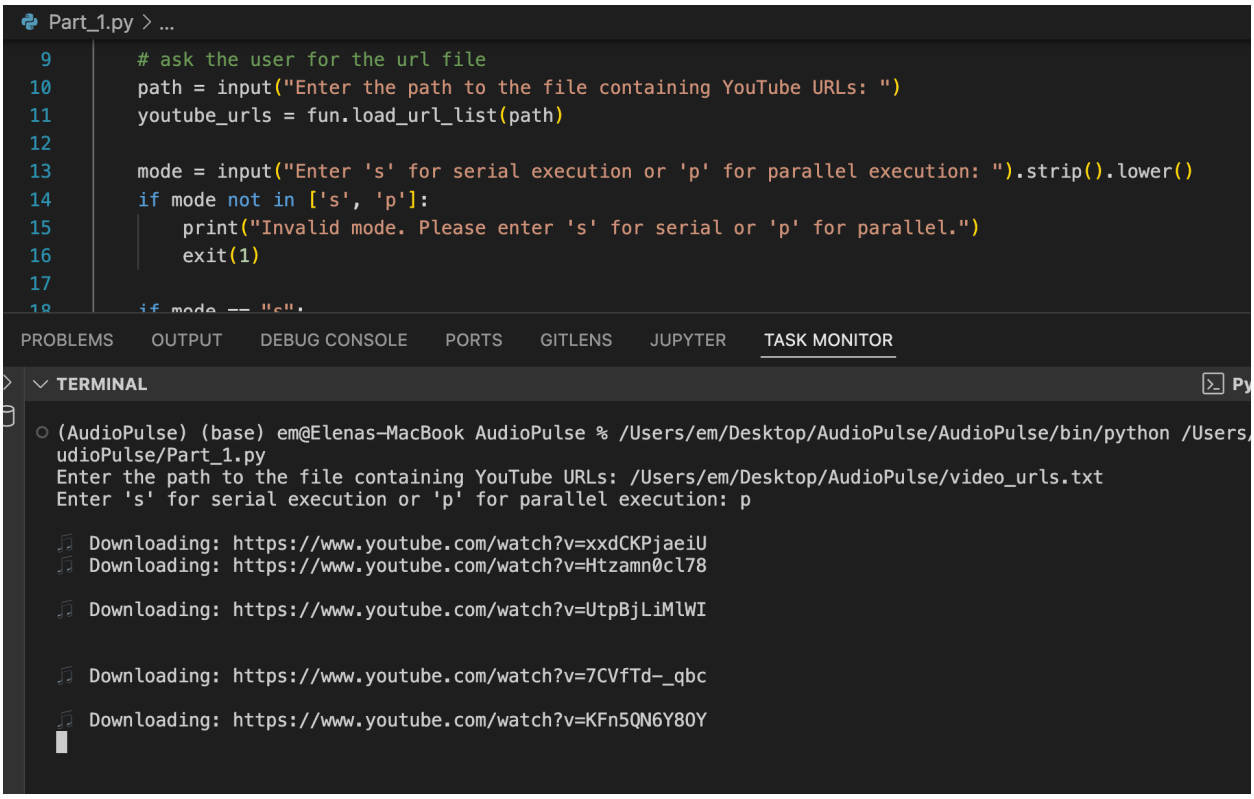


Fig. 14

Step 5: Handle errors gracefully

- **Wrap the download logic in a tr/except block**
- **If a download fails, print a helpful message, or optionally, retry the download a second time.**

Part 2) AUDIO DATA EXTRACTION

Provide comments on the code regarding your implementation. If needed, provide details in the report.

Step 1: write a Python script to

- **Read the .json metadata files from the audio_output directory**
- **USE THE PROVIDED SCRIPT to construct the Python dataframe and save it to a file called example.csv**
- **Extract the following fields from each JSON file:**
  a) **Id**
  b) **Title**
  c) **Uploader**
  d) **... see instructions**
- **Save the final DataFrame as a combined_metadata.csv in your preferred directory.**

In **Part_2.py**, the script provided reads the JSON files from the audio_output directory, and it saves it in a Pandas dataframe called **combined_metadata.csv**.

Part 3) DATA ANALYSIS

Provide comments on the code regarding your implementation. If needed provide details in the report.

Use the combined_metadata.csv file to answer the following questions using Pandas and/or the Spark libraries.

To solve the queries listed under Part 3, I used the slides/notes/labs from week 7 and 8 of the class as reference.

To run the queries in both Pandas and Spark, I first tried to install Spark locally.

Since there is a known incompatibility between Java 21+ and Spark/Hadoop on macOS ARM (M1/M2), which is the device I use, I had to first install Java 17 using Homebrew, and then I verified the installation went well by typing the command:

```
/usr/libexec/java_home -V
```

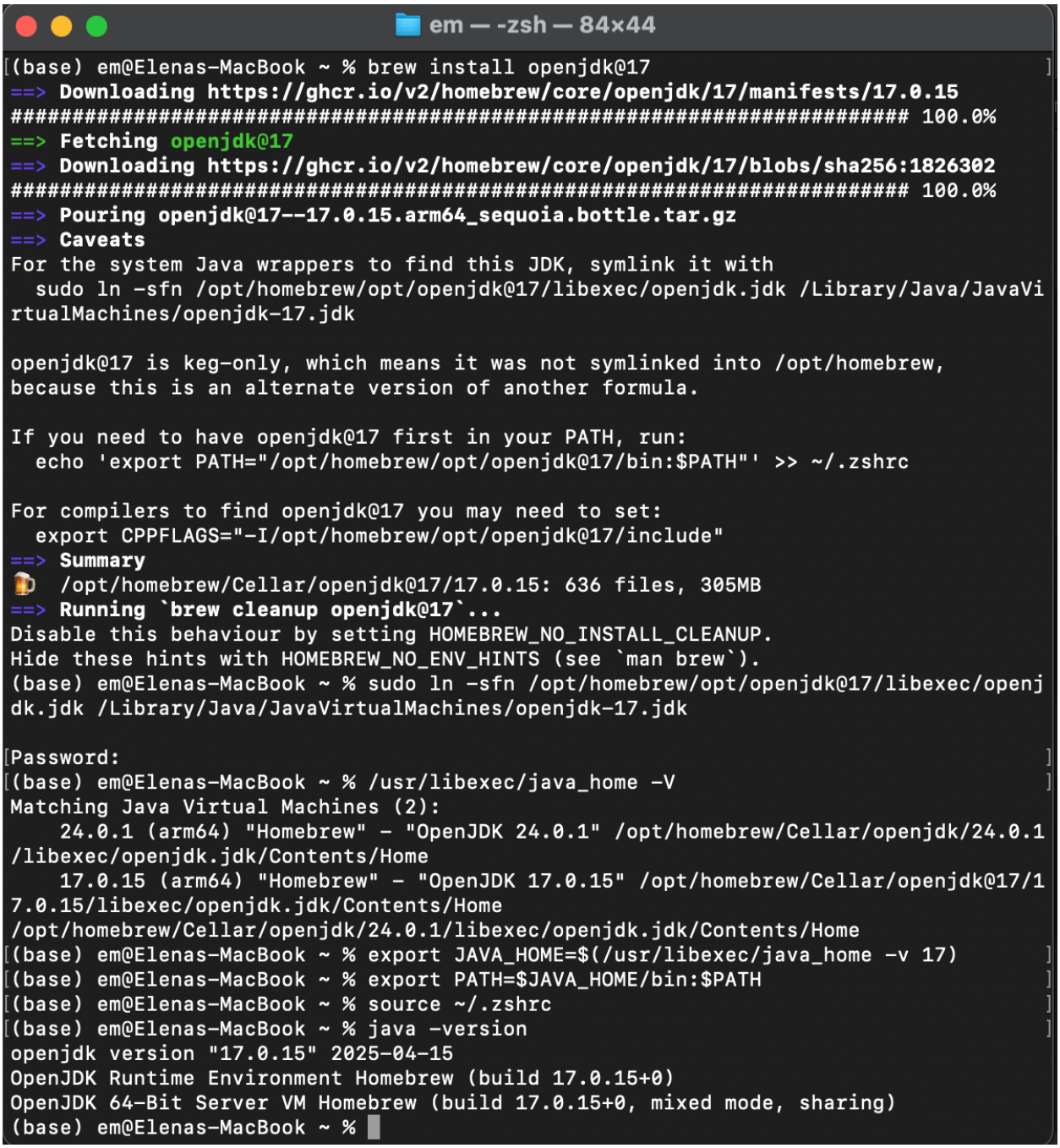


Fig. 15

And finally the dataset was visualized also in Spark (Fig. 16):

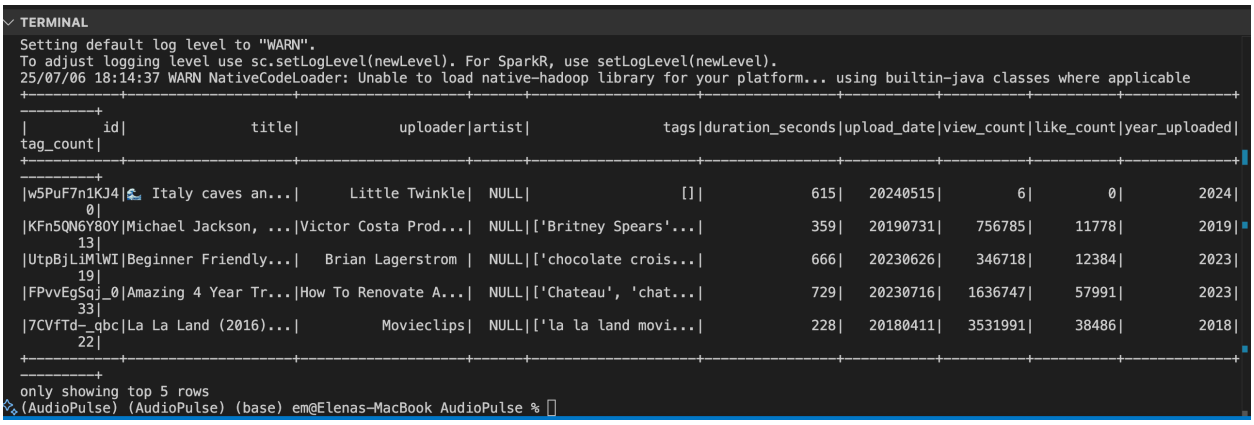


Fig.16

Despite being able to visualize the dataset using Spark, it was not possible to run the queries locally because of the following error:

```
java.lang.UnsupportedOperationException: getSubject is not supported
```

the error raises from “javax.security.auth.Subject.getSubject(Subject.java:277)”, and makes

```
spark = SparkSession.builder.appName("combined_metadata").getOrCreate()
```

fail.

This happens during Spark initialization because Spark (via Hadoop) is trying to determine the current user, and that call relies on a Java API that is no longer supported or is restricted in newer versions of the JDK (Java Development Kit). The cause of this is that the method `Subject.getSubject()` from `javax.security.auth` module is restricted or unsupported in Java 17 and above. This means, I had to downgrade from Java 17 to an even previous version (Java 11). For some reason, after installing Java 11 I still couldn’t use it, so I had to force my terminal to use it (Fig.17).

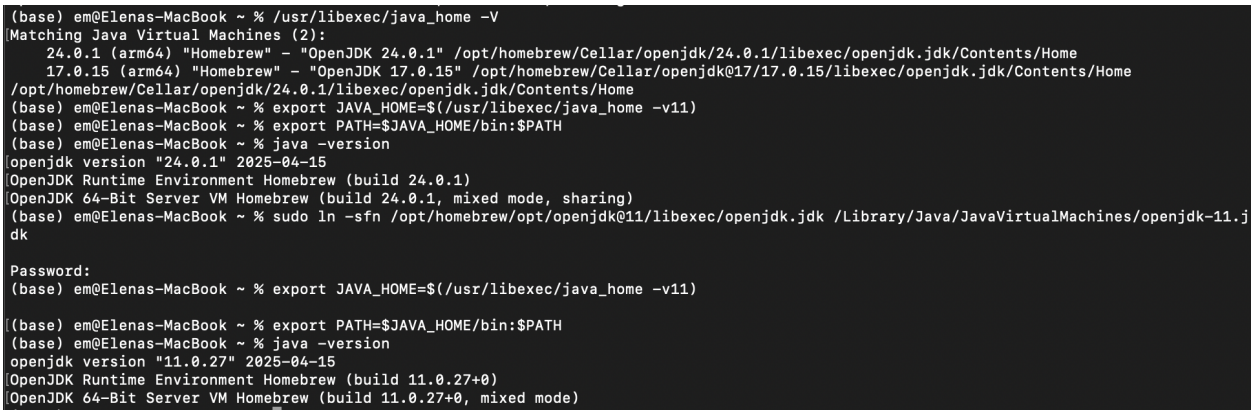


Fig. 17

Spark still didn't work because of a mismatch: I was trying to run Spark using Java 11, but I installed a version of PySpark that was compiled with Java 17. So I had to unistall PySpark, and install a version of PySpark that supports Java 11:

```
pip install pyspark==3.3.1
```

And then finally I was able to run the code locally. Below there is a screenshot of my scrip running locally on VSCode (Fig. 18):

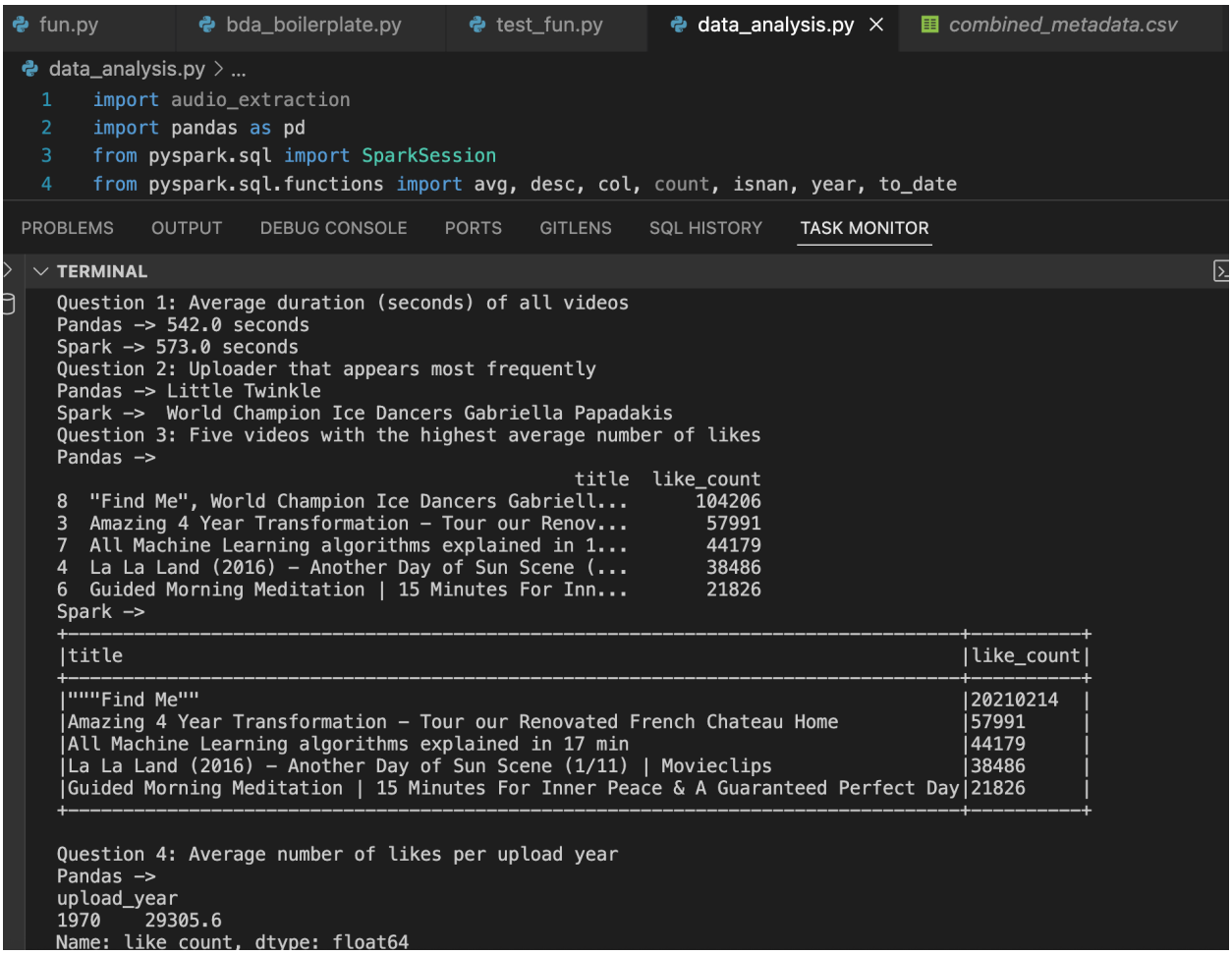


Fig. 18

To ensure smooth execution of the queries, I ran them on Google Colab to avoid any local environment issues. The summarized results are presented in the table below. For queries returning numerical or text data, I provided the direct answers. For more detailed or complex results, I included screenshots to illustrate the output clearly.

	Descriptive statistics	**Pandas**	**Spark**
1	What is the average duration (seconds) of all videos in the dataset?	622.09	622.09
Spark was not reading the CSV file correctly: in particular, the second to last row was read as “NULL”, causing a problem when calculating the average video duration (Fig. 19).			

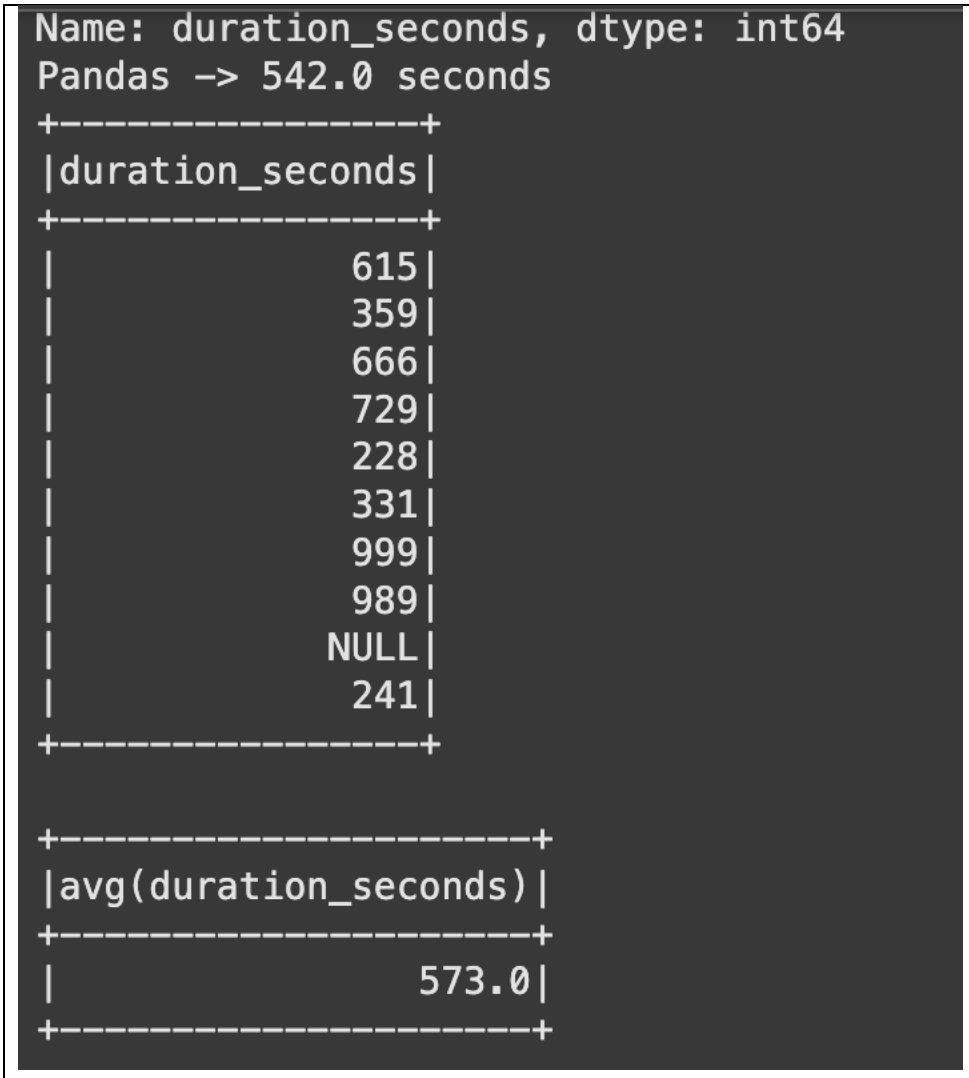


Fig. 19

Note: Fig. 19 is outdated because later on I added a video to the database. I still decided to include it because it shows the error I was getting even if the avg(duration_second) is wrong.

Since in my CSV file there are double-quotes used to enclose fields, then a double-quote appearing inside a field must be escaped by preceding it with another double quote. To solve the issue, I have loaded the Spark dataset using the following code:

```
sdf = spark.read.option("quote", "\"").option("escape", "\"").csv(csv_path, header=True, inferSchema=True)
```

(reference: https://stackoverflow.com/questions/40413526/reading-csv-files-with-quoted-fields-containing-embedded-commas)

2	Which uploader appears most frequently in the dataset?	Boho Beautiful Yoga	Boho Beautiful Yoga
3	Which five videos have the highest average number of likes?	Fig. 20	

```
Question 3: Five videos with the highest average number of likes
Pandas ->
                                          title  like_count
9  "Find Me", World Champion Ice Dancers Gabriell...      104235
3  Amazing 4 Year Transformation - Tour our Renov...       57997
8  All Machine Learning algorithms explained in 1...       44976
4  La La Land (2016) - Another Day of Sun Scene (...       38628
7  Guided Morning Meditation | 15 Minutes For Inn...       21877
Spark ->
+-----------------------------------------------------------------------------------------------------+----------+
|title                                                                                                |like_count|
+-----------------------------------------------------------------------------------------------------+----------+
|"Find Me", World Champion Ice Dancers Gabriella Papadakis, Guillaume Cizeron perform 2020 Free Dance|104235    |
|Amazing 4 Year Transformation - Tour our Renovated French Chateau Home                               |57997     |
|All Machine Learning algorithms explained in 17 min                                                  |44976     |
|La La Land (2016) - Another Day of Sun Scene (1/11) | Movieclips                                     |38628     |
|Guided Morning Meditation | 15 Minutes For Inner Peace & A Guaranteed Perfect Day                    |21877     |
+-----------------------------------------------------------------------------------------------------+----------+
```

Fig. 20

4	For each upload year, what is the average number of likes?	Fig. 21

```
Question 4: Average number of likes per upload year
Pandas ->
upload_year
2013      1625.000000
2018     38628.000000
2019     11993.000000
2021    104235.000000
2023     30759.666667
2024     22488.000000
2025      1550.000000
Name: like_count, dtype: float64
Spark ->
+-----------+------------------+
|upload_year|avg_likes         |
+-----------+------------------+
|2013       |1625.0            |
|2018       |38628.0           |
|2019       |11993.0           |
|2021       |104235.0          |
|2023       |30759.666666666668|
|2024       |22488.0           |
|2025       |1550.0            |
+-----------+------------------+
```

			Fig. 21
5	How many videos are missing artist information?	11 videos	11 videos

	Tag and content characteristics	
1	How many tags does each video have? Visualize the distribution using an histogram.	Fig. 22

```
Question 1: How many tags does each video have? Visualize the distribution using a histogram.
<Figure size 800x500 with 0 Axes>
```

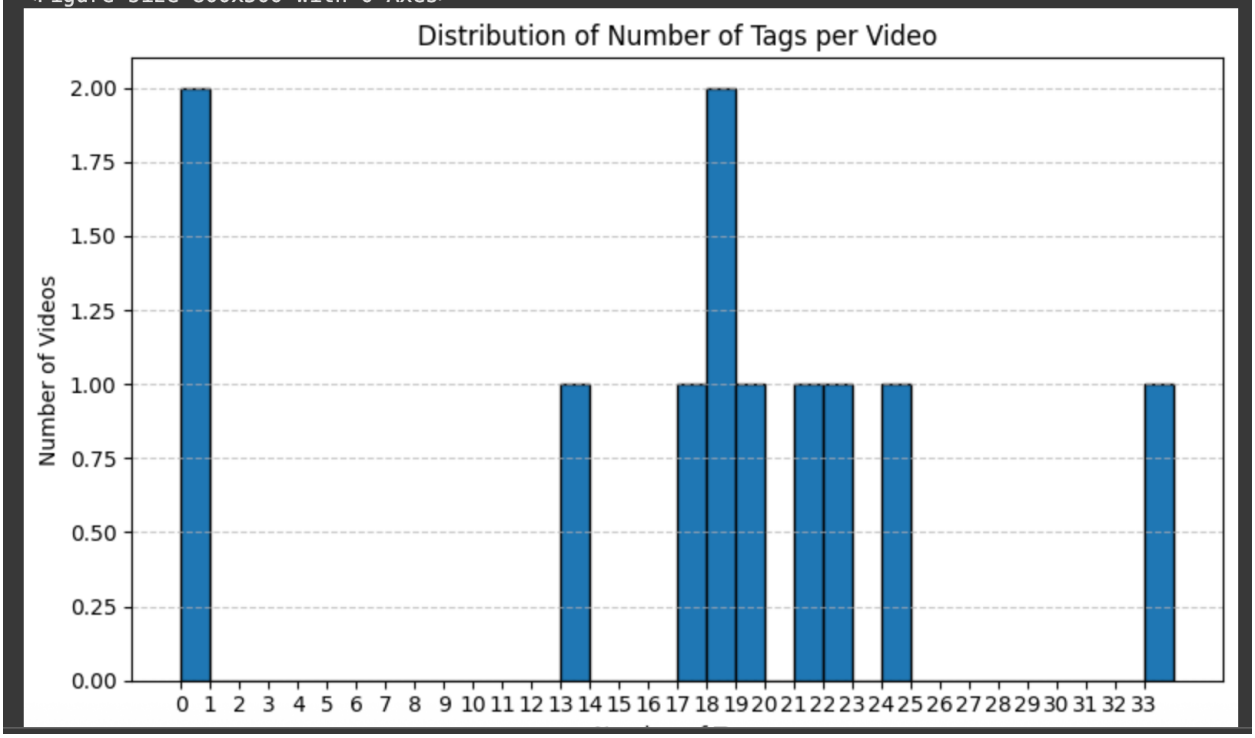


Fig. 22

2	What is the total number of views per uploader? Rank the results in descending order.	Fig. 23

```
Question 2: What is the total number of views per uploader? Rank the results in descending order.
+-------------------------+-----------+
|uploader                 |total_views|
+-------------------------+-----------+
|On Ice Perspectives      |4894802    |
|Movieclips               |3550100    |
|How To Renovate A Chateau|1637189    |
|Boho Beautiful Yoga      |1455935    |
|Infinite Codes           |1110907    |
|Victor Costa Productions |772517     |
|Brian Lagerstrom         |347488     |
|trailersencasa           |220851     |
|Dave Lee Down Under      |19913      |
|Little Twinkle           |6          |
+-------------------------+-----------+
```

Fig. 23

3	Which video has the longest duration? List the titles and each duration.	Fig. 24

```
Question 3: Which video has the longest duration?
+-----------------------------------------------------------------------------------------------------+----------------+
|title                                                                                                |duration_seconds|
+-----------------------------------------------------------------------------------------------------+----------------+
|20 Min Yin Yoga For Sore Hips | Ease Discomfort, Increase Flexibility & Clear Your Mind Perfectly|1423            |
|Guided Morning Meditation | 15 Minutes For Inner Peace & A Guaranteed Perfect Day                |999             |
|All Machine Learning algorithms explained in 17 min                                                  |989             |
|Amazing 4 Year Transformation — Tour our Renovated French Chateau Home                               |729             |
|Beginner Friendly Flaky Pain au Chocolat                                                             |666             |
+-----------------------------------------------------------------------------------------------------+----------------+
only showing top 5 rows
```

Fig. 24

4	How many videos were uploaded in each year? Present the results sorted by year.	Fig. 25

```
Question 4: How many videos were uploaded in each year? Present the results sorted by year.
Pandas ->
upload_year
2013    1
2018    1
2019    1
2021    1
2023    3
2024    2
2025    2
dtype: int64
```

Fig. 25

5	Is there a correlation between the number of views and the number of likes? Feel free to drop or filter rows missing or zero values before computing correlation.	Question 5: Correlation between views and likes Correlation between views and likes: 0.8802

	Derive metrics and custom analysis	
1	Which video has the highest number of likes per second of duration?	Fig. 26

```
Question 1: Which video has the highest number of likes per second of duration?
+--------------------------------------------------------------------------------------------------------------+------------------+----------+----------------+
|title                                                                                                         |likes_per_second  |like_count|duration_seconds|
+--------------------------------------------------------------------------------------------------------------+------------------+----------+----------------+
|"Find Me", World Champion Ice Dancers Gabriella Papadakis, Guillaume Cizeron perform 2020 Free Dance|396.33079847908743|104235    |263             |
+--------------------------------------------------------------------------------------------------------------+------------------+----------+----------------+
only showing top 1 row
```

Fig. 26

2	Which uploader has the longest duration of all their uploaded videos combined?	Fig. 27

```
Question 2: Which uploader has the longest duration of all their uploaded videos combined?
+--------------------+--------------+
|uploader            |total_duration|
+--------------------+--------------+
|Boho Beautiful Yoga|2422          |
+--------------------+--------------+
only showing top 1 row
```

Fig. 27

3	What is the ration of views to likes for each video?	Fig. 28

```
Question 3: What is the ration of views to likes for each video?
+-----------------------------------------------------------------------------------------------------------+-----------------+----------+----------+
|title                                                                                                      |views_to_likes   |view_count|like_count|
+-----------------------------------------------------------------------------------------------------------+-----------------+----------+----------+
|El tiempo entre costuras – Trailer HD                                                                      |135.9083076923077|220851    |1625      |
|La La Land (2016) – Another Day of Sun Scene (1/11) | Movieclips                                           |91.90483587035311|3550100   |38628     |
|Michael Jackson, Britney Spears – The Way You Make Me Feel 4K                                              |64.41399149503877|772517    |11993     |
|Guided Morning Meditation | 15 Minutes For Inner Peace & A Guaranteed Perfect Day                          |60.54961832061068|1324644   |21877     |
|20 Min Yin Yoga For Sore Hips | Ease Discomfort, Increase Flexibility & Clear Your Mind Perfectly|52.4954018392643 |131291    |2501      |
+-----------------------------------------------------------------------------------------------------------+-----------------+----------+----------+
only showing top 5 rows
```

Fig. 28

References

1. Class labs/notes/materials
2. “JSONDecodeError”: https://docs.python.org/3/library/json.html
3. “UnicodeDecodeError” https://www.omi.me/blogs/tensorflow-errors/unicodedecodeerror-in-tensorflow-causes-and-how-to-fix#:~:text=This%20error%20can%20emerge%20when,trying%20to%20process%20these%20bytes
   https://gist.github.com/dbeley/9363640dbe7cf410c995588585d33038
4. “Hadoop Installation, Error: getSubject is supported only if a security manager is allowed:” https://stackoverflow.com/questions/79016199/hadoop-installation-error-getsubject-is-supported-only-if-a-security-manager-i
5. “install pyspark locally. Pain.”: https://www.reddit.com/r/dataengineering/comments/1dupogi/wasted_45_hours_to_install_pyspark_locally_pain/
6. “Any plans for Spark to support Java 17”: https://www.reddit.com/r/apachespark/comments/tkifvl/any_plans_for_spark_to_support_java_17/
7. “Java 17 vs Java 11: Exploring the Latest Features and Improvements”: https://medium.com/javarevisited/java-17-vs-java-11-exploring-the-latest-features-and-improvements-6d13290e4e1a
8. “Reading csv files with quoted fields containing embedded commas”: https://stackoverflow.com/questions/40413526/reading-csv-files-with-quoted-fields-containing-embedded-commas