🎧 AudioPulse Capstone Project

You've just been hired by 🎧 AudioPulse, a fast-growing marketing startup that analyzes online audio content to uncover audience trends and engagement patterns.

Your task?

Design a data analysis pipeline that can collect, process, and analyze YouTube audio data — like building the brains behind a mini Spotify, but focused on research and insights. From downloading content to generating rich metadata and extracting trends, you'll take full ownership of the data engineering flow.

You'll develop a pipeline that:

- Collects YouTube videos with music, or ambient sound (e.g. podcasts, reviews, music sessions).
- Downloads only the audio tracks (no need for video!) and grabs all available metadata about each one.
- Processes and organizes that data into clean `csv` s — ideal for analysis.
- Speeds up the downloads using appropriate parallel processing methods to avoid long waits.
- Logs every download (and any errors) to keep track of what's working and what's not.
- Analyzes patterns: Who's getting the most views? What's the average audio length? Are likes and views related?

Good luck 🚀

> ⓘ **Note**
>
> No strict rules: You're encouraged to be creative, experiment with your own improvements, and explore what's possible.

Part 0 — Assignment Preparation and Boilerplate

> Before starting the assignment tasks, take time to review the provided boilerplate code. This tutorial will guide you through setting up your environment. The code provided as a whole in the `bda_boilerplate.py` enables you to download audio from YouTube videos, extract relevant metadata, and save both to the appropriate folder.

1. Make sure you start by creating a virtual environment using `venv` for your project. This helps keep your project dependencies clean and isolated.
2. You will need to install the following python libraries.

```
pip install yt-dlp
pip install certifi
```

> 💡 **Tip**

> You can install the required libraries by running `pip install -r requirements.txt` in your terminal. You might need to use `pip3` as needed.

3. Use this starter script to download audio and metadata from a YouTube video. This script will:

- Download the audio track (best available quality),
- Save it in an `audio_output` folder,
- Extract and store relevant metadata in a `.json` file.

4. Explore the boilerplate code below to understand what it does. You are expected to use this code to implement your software.

- You will need to import the following libraries.

```
import yt_dlp       # For downloading and extracting metadata/audio from YouTube videos
import certifi      # Provides a trusted CA certificate bundle for secure HTTPS requests
import os           # For interacting with the operating system (e.g., file paths,
directories)
import json         # For reading and writing JSON data (used to save metadata)
```

> **Tip**
>
> `JSON` stands for JavaScript Object Notation. It's a lightweight, human-readable format used to store and exchange data between systems (like between a server and a web app).

- This next code ensures that there's a folder ready to store audio files. The app can securely access HTTPS resources using trusted certificates.

```
OUTPUT_DIR = "audio_output"
os.makedirs(OUTPUT_DIR, exist_ok=True)
os.environ["SSL_CERT_FILE"] = certifi.where()
```

- The `get_video_info` function uses `yt_dlp` to extract metadata (and optionally download audio) from a YouTube video at the given URL, saving the audio in `.m4a` format with minimal output or warnings.

```
def get_video_info(url: str, download: bool = True) -> dict:
    """Extract video info and optionally download the audio without warnings."""
    ydl_opts = {
        'format': 'bestaudio[ext=m4a]/bestaudio',
        'outtmpl': os.path.join(OUTPUT_DIR, '%(title)s.%(ext)s'),
        'noplaylist': True,
        'quiet': True,          # suppress general output
        'no_warnings': True,    # suppress warnings
        'skip_download': not download,
        'postprocessors': [],
    }
    with yt_dlp.YoutubeDL(ydl_opts) as ydl:
        return ydl.extract_info(url, download=download)
```

- The `extract_metadata` function extracts and returns a simplified dictionary with key metadata from a video info object—such as title, uploader, duration, tags, and stats—filtering out only the most relevant fields.

```
def extract_metadata(info: dict) -> dict:
    """Filter and return relevant metadata fields, including derived values."""
    upload_date = info.get("upload_date")
    tags = info.get("tags") or []

    return {
        "id": info.get("id"),
        "title": info.get("title"),
        "uploader": info.get("uploader"),
        "uploader_id": info.get("uploader_id"),
        "channel": info.get("channel"),
        "track": info.get("track"),
        "artist": info.get("artist"),
        "album": info.get("album"),
        "description": info.get("description"),
        "tags": tags,  # Ensure it's a list
        "duration_seconds": info.get("duration"),
        "upload_date": upload_date,
        "view_count": info.get("view_count"),
        "like_count": info.get("like_count"),
        "webpage_url": info.get("webpage_url"),
        # Derived fields
        # Extract the year from upload_date (e.g., "20230521" → 2023)
        "year_uploaded": int(upload_date[:4]) if upload_date else None,
        "tag_count": len(tags),
    }
```

- The `save_metadata_to_file` function saves the given metadata dictionary to a JSON file using a sanitized version of the video title as the filename, and returns the full path to the saved file.

```
def save_metadata_to_file(metadata: dict, title: str) -> str:
    """Save metadata as a JSON file and return the path."""
    safe_title = "".join(c for c in title if c.isalnum() or c in (" ", "_", "-
")).rstrip()
    file_path = os.path.join(OUTPUT_DIR, f"{safe_title}.json")
    with open(file_path, "w", encoding="utf-8") as f:
        json.dump(metadata, f, indent=2, ensure_ascii=False)
    return file_path
```

- This `download_youtube_audio_with_metadata` orchestrates the full process: it downloads audio from a YouTube URL, extracts its metadata, saves that metadata to a JSON file, and prints the status. If any error occurs, it catches and reports it.

```python
def download_youtube_audio_with_metadata(url: str):
    """Main function to download audio and save metadata."""
    print(f"\n🎵 Downloading: {url}")
    try:
        info = get_video_info(url)
        metadata = extract_metadata(info)
        json_path = save_metadata_to_file(metadata, metadata["title"])
        print(f"✅ Done: {metadata['title']}\n📄 Metadata: {json_path}")
    except Exception as e:
        print(f"❌ Failed to download: {url}\n   Error: {e}")
```

- The `main` block runs the script if it's executed directly. It loops through a list of YouTube URLs, downloads each audio file, extracts its metadata, and saves it—calling the main function `download_youtube_audio_with_metadata(url)` for each one.

```python
if __name__ == "__main__":
    youtube_urls = [
        "https://youtu.be/4D7u5KF7SP8?si=Y_S0k_5MbdW5mKI2",
        "https://youtu.be/Q3Bp1QVVieM?si=4BlPs5dpK3Y-iPYr",
        "https://youtu.be/5m4ZkEqQrn0?si=7DfUDfMGKkbktUDi"
    ]
    for url in youtube_urls:
        download_youtube_audio_with_metadata(url)
```

---

4. Run `bda_boilerplate.py` and examine the output file structure. This is provided as example code—you can start modifying it to suit your project's requirements.

- Here is an example of the output.

```
# (venv) stelios@pc bda-capstone % python3 bda_boilerplate.py

🎵 Downloading: https://youtu.be/4D7u5KF7SP8?si=Y_S0k_5MbdW5mKI2
✅ Done: Get Lucky (feat. Pharrell Williams and Nile Rodgers)
📄 Metadata: audio_output/Get Lucky feat Pharrell Williams and Nile Rodgers.json

🎵 Downloading: https://youtu.be/Q3Bp1QVVieM?si=4BlPs5dpK3Y-iPYr
✅ Done: Daft Punk - Lose Yourself To Dance (Feat. Pharrell Williams)
📄 Metadata: audio_output/Daft Punk - Lose Yourself To Dance Feat Pharrell Williams.json

🎵 Downloading: https://youtu.be/5m4ZkEqQrn0?si=7DfUDfMGKkbktUDi
✅ Done: Daft Punk - Giorgio By Moroder
📄 Metadata: audio_output/Daft Punk - Giorgio By Moroder.json
```

- Here is an example of the metadata extracted from a YouTube url: `https://www.youtube.com/watch?v=Q3Bp1QVVieM`

```json
{
  "id": "Q3Bp1QVVieM",
  "title": "Daft Punk - Lose Yourself To Dance (Feat. Pharrell Williams)",
  "uploader": "EXPO STORY",
  "uploader_id": "@expostory",
  "channel": "EXPO STORY",
  "track": null,
  "artist": null,
  "album": null,
  "description": "",
  "tags": [
    "Pharrell Williams (Rapper)",
    "Daft Punk (Musical Group)",
    "Dance (song)",
    "Song (album)",
    "New",
    "Singing",
    "Get Lucky",
    "Random Access Memories",
    "2013",
    "Album",
    "Disco"
  ],
  "duration_seconds": 350,
  "upload_date": "20130516",
  "view_count": 13241154,
  "like_count": 61654,
  "webpage_url": "https://www.youtube.com/watch?v=Q3Bp1QVVieM",
  "year_uploaded": 2013,
  "tag_count": 11
}
```

Part 1 — Audio Download Pipeline & Logging

Your first goal is to build a working pipeline to download audio content and metadata from a list of YouTube videos.

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.

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.

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:
  - Audio files to a folder called `audio_output`
  - Metadata files to the same folder.
  - Logs to a folder called `logs`. You can use the `os.makedirs("logs", exist_ok=True)` to create the folder.
- Test your script in two modes:
  1. Serial mode — download one video at a time.
  2. Parallel mode — download multiple videos at once using threads.

Requirements:

- Implement a parallel version that can download up to 5 videos simultaneously.
- After both versions run, compare their performance and explain:
  - Which one is faster?
  - What are the trade-offs (e.g., complexity, system load)?
  - How do time and space complexity change?

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. The following example is in a JSON-like format.

```
[
  {
  "timestamp": "2025-05-21T12:23:00",
  "url": "https://www.youtube.com/watch?v=1234",
  "download": true
  },
  ...
]
```

> **Tip**
>
> Feel free to use any library to extract the timestamp, or use the following snippet:
>
> ```
> from datetime import datetime
> timestamp = datetime.now().isoformat(timespec='seconds')
> ```

- Save all logs in the `logs` folder.

Step 5: Handle Errors Gracefully

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

Part 2 — Audio Data Extraction

1. Write a Python script to:
    - Read the `.json` metadata files from the `audio_output` directory.
    - Check the following example before you proceed to familiarize with the `json` library.
        - Consider the following audio files `audio1.json` and `audio2.json` stored in the `audio_output` folder.

```
# audio1.json
  {
    "title": "How to Bake Bread",
    "duration_seconds": 480,
    "view_count": 15000
  }
```

```
# audio2.json
  {
    "title": "Jazz Piano Session",
    "duration_seconds": 720,
    "view_count": 82000
  }
```

    - This is the script to construct the Python dataframe and save it to a file called `example.csv`.

```python
import pandas as pd
import glob
import json

# Read all .json files in the folder
json_files = glob.glob("audio_output/*.json")

# Load all JSON files into one DataFrame
df = pd.DataFrame()

for file in json_files:
    with open(file, "r", encoding="utf-8") as f:
        data = json.load(f)
        # Ensure it’s always a list of dicts
        records = data if isinstance(data, list) else [data]
        df = pd.concat([df, pd.DataFrame(records)], ignore_index=True)

print(df.head())
df.to_csv("combined_metadata.csv", index=False)
```

- Extract the following fields from each JSON file:

Column Name	Description
`id`	YouTube video ID
`title`	Video title
`uploader`	Channel or uploader name
`artist`	Artist name (if available; can be `null`)
`tags`	`tags` — Tags as a list of strings (e.g., `["music", "podcast", "2023"]`). Must remain a list for consistency and compatibility with JSON and Spark.
`duration_seconds`	Duration in seconds
`upload_date`	Original upload date in format `YYYYMMDD`. I suggest converting it to a proper `datetime` object or extracting the year with slicing.
`view_count`	Number of views
`like_count`	Number of likes
`year_uploaded`	Extracted from `upload_date` (e.g., `2020`) - You can extract `year_uploaded` from the first 4 characters of `upload_date`.
`tag_count`	Number of tags present

2. Save the final DataFrame as a `combined_metadata.csv` in your preferred directory.

Part 3 — Data Analysis

Use your `combined_metadata.csv` file to answer the following questions using the `Pandas` and/or the `Spark` libraries. Please note that you should not expect any significant speed-up as the dataset is currently small

> **Important**

> - *If your dataset is missing any of the specified columns, feel free to adapt the structure based on the columns available to you. You may substitute any missing column with a suitable alternative — for example, if a question asks for the average of `year` but it's unavailable, use `duration_seconds` instead. Include a brief comment in your report explaining the substitution.*
> - *If your dataset is missing all values for a key field (e.g. `duration` is `None` for all entries), feel free to manually adapt the dataset so that your queries can run correctly. However, if only a few entries are missing data, simply exclude those rows from your analysis.*

3.1 Descriptive Statistics

1. What is the average duration (in seconds) of all videos in the dataset?
   - Provide both the `pandas` and `spark` query.
2. Which uploader appears most frequently in the dataset?
   - Provide both the `pandas` and `spark` query.
3. Which five videos have the highest number of views? List their titles and view counts.
   - Provide both the `pandas` and `spark` query.
4. For each upload year, what is the average number of likes?
   - Provide both the `pandas` and `spark` query.
5. How many videos are missing artist information?
   - Provide both the `pandas` and `spark` query.

3.2 Tag and Content Characteristics

1. How many tags does each video have? Visualize the distribution using a histogram.
2. What is the total number of views per uploader? Rank the results in descending order.
3. Which video has the longest duration? List the title and its duration.
4. How many videos were uploaded in each year? Present the results sorted by year.
5. Is there a correlation between the number of views and the number of likes? Feel free to drop or filter rows with missing or zero values before computing correlation.

3.3 Derived Metrics & Custom Analysis

1. Which video has the highest number of likes per second of duration?
2. Which uploader has the longest total duration of all their uploaded videos combined?
3. What is the ratio of views to likes for each video?

Part 4 — Document your solution

You are expected to submit a short report summarizing your work for Parts 1–3 of the assignment. Your report should explain your process, include code snippets, outputs, and briefly reflect on what you learned or found challenging.

- In **Part 1**, describe how your pipeline works from start to finish. Compare the performance of both serial and parallel downloading. Report which approach was faster, and explain any trade-offs you observed, such as added complexity or system load. Provide script snippets as needed.
- In **Part 2**, provide comments on the code regarding your implementation. If needed, provide details in the report.
- In **Part 3**, provide comments on the code regarding your implementation. If needed, provide details in the report.

What to Submit

1. Include all scripts you used:

- Audio download (serial & parallel)
- Metadata extraction
- Data analysis

2. Report (PDF, Word, or Markdown):

- Summary of Parts 1–3
- Key code snippets and outputs
- Answers to analysis questions
- Optional short reflection

3. Data Files:

- `video_urls.txt` — list of YouTube links
- `audio_output/` — Only the metadata JSON files - **DO NOT INCLUDE AUDIO FILES**
- `logs` — `download_log.txt`
- `combined_metadata.csv` — final dataset for analysis

Marking Rubric (Total: 100 points)

Category	Criteria	Points
Part 1 — Audio Download Pipeline & Logging		**/40**
URL Collection	A text file includes 10–15 valid YouTube URLs, each on a separate line	5
URL Loading	The script correctly loads URLs from the file into a suitable Python data structure	5
Serial Download	Downloads audio and metadata sequentially for each URL	8
Parallel Download	Implements parallel downloading with thread control and resource limits	8
Logging	A thread-safe logging system records each download attempt to a file	7
Error Handling	Errors are managed with appropriate try/except handling and optional retry logic	7
Part 2 — Metadata Aggregation		**/25**
JSON Reading	Metadata is read successfully from all relevant JSON files	5
DataFrame Construction	Metadata fields are correctly extracted and structured into a DataFrame	8
Derived Fields	New fields are added from existing data (e.g., tag counts, upload year)	6
CSV Export	The combined metadata is saved correctly to a CSV file	6
Part 3 — Data Analysis		**/25**
Descriptive Statistics	Core statistics are calculated and presented using the metadata	7
Tag & Content Insights	Content characteristics are analyzed and visualized appropriately	7.5
Derived Metrics	Additional metrics are computed based on the dataset	8
Spark queries	Correct implementation of the Spark queries	2.5
Report & Documentation		**/10**
	The report clearly describes steps, methods, and	

Clarity and Structure	structure of the solution	4
Code Commentary	The code is well-organized and commented, or explanations are provided in the report	4
Reflection	Includes a brief reflection on challenges or insights from the project	2