Big Data Analytics 2024 Project 1

:movie_camera: Welcome to **CineSense**

CineSense is an innovative video-processing startup that extracts valuable insights from social media video content. It uses advanced natural language processing (NLP) and computer vision techniques to analyse videos' sentiments and emotions. This analysis is crucial for businesses seeking to understand their audience, improve customer experiences, and make data-driven decisions.

On your first day at **CineSense** as a data engineer, you were assigned a critical project: **find the best strategy to download and analyse videos**. **CineSense** uses YouTube videos posted by its target audience. Each video must be analysed, and results should be extracted and shared with customers. Usual tasks include extracting video, audio, transcripts, sentiments, and emotions. Since you are paired with a customer from Madrid, you will also need to transcribe videos in Spanish.

Thanks to Python, you can showcase your skills and automate the tasks. Your project leader is particularly looking for a strategy to complete tasks faster! Parallel processing concepts can help you speed up your analysis tasks, contributing to your company's success. Your colleagues prepared a project description (**part 1**) and a set of skeleton scripts for your developments provided (**part 2**).

Good luck :rocket:

Part 1: Project description

* Develop a Python application using multiprocessing, threading or asynchronous programming concepts to download and analyse YouTube videos.

* Compare and contrast different solutions for serial and parallel processing executions.

* You have four weeks to complete all the tasks and document your solutions.

Tasks:

1. Manually retrieve 10-15 random video URLs from YouTube.

   * Save the URLs in a text file called `video_urls.txt`, where each URL should be stored on a separate line.
   * Consider YouTube videos that are 2-3 minutes in duration.

2. Develop a Python script to read the URLs.

   - Assuming you have the text file named `video_urls.txt` containing the URLs of YouTube videos, load it in Python and extract the URLs using your preferred data structure.

   [2.5 marks]

3. Develop a Python script to download the videos using their URLs.

   - Test your solution by downloading the files serially.
   - Use parallel programming such as multiprocessing or threading to handle downloads. Your decision will determine the best strategy.
   - For testing reasons, ensure the script can download up to `5` videos simultaneously to avoid YouTube blocks.
   - You are advised to use `threads` and `semaphores` to control the downloads.
   - Compare serial and parallel executions for your video download script.

~~~
   - Discuss the complexity of your video download scripts' time and space.

   **[15 marks]**

4. Develop a Python script to keep a log for each download.

   - After downloading each video, create a logger to record which video was downloaded by which process or thread.

   - Save the log entries to the same file, e.g., `download_log.txt`.

   - For this script, you have to use `threads` and a `mutex`.

   - The entries could be in the following format:

     - ```json
       "Timestamp": 12:23, 21 May 2024, "URL":"http://www.youtube.com/1234", "Download":True
       "Timestamp": 12:25, 21 May 2024, "URL":"http://www.youtube.com/1235", "Download":True
       ```


   **[12.5 marks]**

5. Develop Python scripts to perform various video analysis tasks.

   - After downloading a video, perform the following tasks.
   - It is preferable to develop a separate script for each functionality.
   - The five analysis subtasks that you have to develop include the following:
     - Extract audio from a video file.
     - Transcribe audio to text.
~~~

    - Perform the sentiment analysis on a video's content, extracting its polarity and sensitivity.
    - Translate the text into another language, e.g. Spanish.
    - Extract the emotions of a text.
  - Each output task should store its results in a dedicated folder designated for each video, using the video title. Feel free to organise your folder structure as you prefer.
  - You can use any library, including `moviepy` for loading video and `speech_recognition` or `textblob` for sentiment analysis.
  - To implement the analysis subtasks, you must use at least one of the following libraries: `multiprocessing`, `threading`, or `asyncio`.
  - You must compare serial, multiprocessing, threading, and concurrency for at least one of the subtasks, such as the extracting audio functionality. You do not have to do it for the rest of the subtasks.

  [30 marks]

6. Document your solutions in a short technical report.

  * This project is open to your interpretations, and you can use any programming strategy that fits better to this project, including processes, threads, or asynchronous programming.
  * Discuss briefly your strategy for developing each solution.
  * A 1-2 page discussion (600-1200 words) will suffice, but you can include more if necessary.
  * You can include text in a `ipynb` file if you prefer.
  * There is no penalty for submitting more or less text.
  * There is no need to include references if you did not use existing resources. You don't have to reference the supporting material or the lab notes.

~~~
If you use external resources, e.g. code from online sources, then provide a reference in the report.
  * You can run your scripts in any Python IDE you prefer, including Visual Studio Code, Colab, etc.

  **[20 marks]**

7. Submit quality scripts following coding practices such as functional or object-oriented programming, separate Python files to break functionality for readability, use of `main`, and other standard practices. Feel free to use Jupyter Notebooks, but provide clear and well-organized code.


   **[20 marks]**



### Part 2: Supporting material

The following supporting scripts can help you kickstart your project. You spend some time familiarising yourself with the suggested libraries and scripts. You can use any library you prefer, not necessarily the ones provided below.

* Install the required libraries using the following command.

```bash
pip3 install pytube moviepy speechrecognition spacy nltk NRCLex Path TextBlob
```

> [!TIP]
>
> Depending on your Python installation, you might need to use `pip3` instead of `pip` .

* Explore the following script that downloads a
~~~

~~~markdown
video in a `video_output folder.

```python
from pytube import YouTube

url = "https://www.youtube.com/watch?v=jNQXAC9IVRw"
yt = YouTube(url)
stream = yt.streams.get_highest_resolution()
print(f"Downloading video: {yt.title}")
stream.download(output_path="video_output")
print(f"Download completed: {yt.title}")
```

> The output is as follows:

```bash
Downloading video: Me at the zoo
Download completed: Me at the zoo
```

> [!NOTE]
>
> The files are stored as `mp4`; you can use the following script to create a folder.
>
> ```python
> from pathlib import Path
> Path("output").mkdir(parents=True, exist_ok=True)
> ```

* The following script extracts the audio file `wav` from a video clip `mp4`.

```python
import moviepy.editor as mp

video = mp.VideoFileClip("Me at the zoo.mp4")
  video.audio.write_audiofile("Me at the zoo.wav")
```

> [!NOTE]
~~~

~~~markdown
>
> The script extracts the audio files in `wav` format in your local directory.

* If you like to store outputs in a folder you can use the following script:

```python
import os

output_folder = "output"
output_filename = "myfile.txt"
output_folder = os.path.join(output_folder, output_filename)
```

* The following script extracts the text from the audio file.

```python
import speech_recognition as sr
recognizer = sr.Recognizer()
with sr.AudioFile("Me at the zoo.wav") as source:
    audio = recognizer.record(source)
text = recognizer.recognize_google(audio)
print(text)
```

> [!NOTE]
>
> The scripts extract the audio as a string. The following script saves the string to a text file.
>
> ```python
> file_path = 'Me at the zoo.txt'
> with open(file_path, 'w') as file:
>     file.write(text)
> print(f'Text has been written to {file_path}')
> ```

* The following script extracts the sentiment from a
~~~

text file.

```python
from textblob import TextBlob

text = "all right so here we are one of the elephants and cool thing about these guys today is that they have really really really really really really long trunks and that's that's cool"

blob = TextBlob(text)
print(blob.sentiment)
print(blob.sentiment.polarity)
print(blob.sentiment.subjectivity)
```

> [!NOTE]
>
> * Polarity measures the text's positive or negative tone. It ranges from `-1.0` (very negative) to `1.0` (very positive). A value of `0` indicates a neutral sentiment.
>
> * Subjectivity measures how subjective or objective the text is. It ranges from `0.0` (very objective) to `1.0` (very subjective). A subjective text contains personal opinions, emotions, or judgments, whereas an objective text is based on factual information.

* Extract the video subtitles in Spanish - or any other language you prefer.

```python
from textblob import TextBlob

text = "all right so here we are one of the elephants and cool thing about these guys today is that they have really really really really really really long trunks and that's that's cool"
```

```python
blob = TextBlob(text)
blob_translated = blob.translate(from_lang='en', to='es')
print(blob_translated)
```

> [!NOTE]
>
> The output will be stored in the `blob_translated` in Spanish.

* Extract the emotions of the text using the `spacy`, `nltk` and the `NRCLex` libraries.

```python
import spacy,nltk
from nrclex import NRCLex
nlp = spacy.load('en_core_web_sm')
nltk.download('punkt')

text = "all right so here we are one of the elephants and cool thing about these guys today is that they have really really really really really really long trunks and that's that's cool"

doc = nlp(text)

full_text = ' '.join([sent.text for sent in doc.sents])

emotion = NRCLex(text)

print("Detected Emotions and Frequencies:")
print(emotion.affect_frequencies)
```

> [!NOTE]
>
> The output will be the video emotions in a dictionary, such as:
>

```
> ```python
> Detected Emotions and Frequencies:
> {'fear': 0.0, 'anger': 0.0, 'anticip': 0.0, 'trust': 0.0, 'surprise': 0.0, 'positive': 0.6666666666666666, 'negative': 0.0, 'sadness': 0.0, 'disgust': 0.0, 'joy': 0.0, 'anticipation': 0.3333333333333333}
> ```
```