Criterion A

Scenario
My client is the founder of the podcast club in our school. One of his main objectives involves coordinating and helping students make their own podcast episodes.

However, my client highlights the problem with “scouting for relevant podcasts” when they have more than one topic with which they want a specific podcast to address. Searching for relevant podcasts that merge various topics becomes difficult especially within individual episodes as they don’t always offer the most relevant discussions, making the process inefficient.

He specifies how the club is in need of a podcast recommendation system that can return “a podcast most relevant to all the search [topics]” that the user specifies.

Therefore, I suggested a program with a user interface that allows the user to input specific words related to their topics of interest and generate a keyword that is able to encapsulate all the topical terms which can be used to then return the most relevant podcasts for the user to explore.

Rationale for the Chosen Solution

I will use Python for the backend of my program as it offers various modules. Since my program aims to find different podcast episodes relating to user-specified inputs/topics, I would require the use of NLP libraries such as NLTK to extract keywords and compare their similarities. Using deep NLP models like Word2Vector, I can use vectorization to calculate the most relevant keyword to the user inputs from large datasets. Other packages like Numpy and Pandas can be used to create dataframes to organise information about relevant podcasts, making the cleaning and processing of data very efficient. Lastly, Python is an easy language to code in as it makes use of natural language, which is intuitive and efficient.

I chose to create the frontend of my program using Flask, HTML and CSS. Flask allows storage and fetching of previous search inputs so relevant results can be directly returned rather than being web-scraped and processed redundantly. Flask is also directly compatible with HTML, using SQLAlchemy to connect my relational database to user interactions and backend data processing. The Bootstrap framework using HTML and CSS can be used to create a more visually pleasing interface.

I can explore alternative backend designs through the use of different NLP libraries like spaCy. It offers higher speed and efficiency than NLTK in processing large datasets. Instead of Word2Vec models, I can use GloVe embeddings due to their fast training and scalability with larger datasets. It may also more accurately capture the semantic meaning of user inputs due to the nature of its model. For frontend development, I can explore Vue.js as it is a component-based architecture so it will allow me to reuse modules, making the coding of the UI more efficient.

Success Criteria

1) Minimal client input:
    a) Input topics limited to 3 keywords
    b) Use a simple bootstrap template to create a user-friendly interface where the user can easily navigate to the three input boxes for inputting keywords.
    c) Input processing using Flask to define routes that can handle the inputs
    d) Ensure quality assurance for user input
        i) Increase efficiency by storing previous input in the database as cache
        ii) Implement auto-suggestions to assist users in inputting meaningful keywords
        iii) Input validation:
            (1) Data cleaning techniques include converting plural words to their singular form and removing stopwords, etc.
            (2) Check if search results are available for user input in Google Podcast
            (3) Return to the homepage if either (1) or (2) fails
            (4) Provide clear error message when input is not valid

2) Web crawling:
    a) Obtain API key to authenticate request to Google Podcast API
    b) Use client input to connect to Google Podcast API
    c) For each user input, get URLs of relevant podcast episodes
    d) For each URL of relevant podcast episodes, get its homepage URL
    e) Scrape the RSS feed from each homepage URL
    f) Extract relevant information from the RSS feed including podcast titles and descriptions
        i) Data cleaning techniques include converting plural words to their singular form and removing stopwords, etc.
        ii) Reference NLP a)
    g) Handle errors where podcast URL is not found
    h) Handle errors where API request fails

3) NLP:
    a) Create keywords from the description of each podcast related to the user inputs using keyBERT
    b) Quality Assurance of model output:
        i) Experiment 1: Use of relevant and irrelevant inputs → Conclusion 1: distance can be used as a measurement of the degree of input relevance; Conclusion 2: The centroid should be the point with the smallest distance to other keyword vectors, thus it holds the highest representative power to all input. For example (tea, god, mountain → low confidence score | cat, dog, horse → animal → high confidence)
        ii) Use the Word2Vector model to find a centroid from all of the extracted keywords processed
        iii) Find the most relevant recommended keyword based on Euler distance to the centroid
    c) Feed centroid word back to API to find relevant podcast episodes to be displayed
4) Previous inputs:
    a) Sort inputs and keywords alphabetically
    b) Fetch corresponding keywords when the user enters input in datatable search box of the previous inputs page
5) Displays the following:
    a) Podcast recommendation page with user inputs, relevancy of recommended result, connection to API and word cloud of relevant keywords
    b) Database of previous inputs and relevant keywords
    c) Word cloud of keywords when the user enters input in the previous inputs page
    d) A uniform simple bootstrap template across all pages with button that can redirect user back to homepage