Library	Usage	How It Is Used
librosa	Audio file loading, waveform display, MFCC extraction	- librosa.load(audio_path_1): Loads the first audio file, returning audio samples (y_1) and sampling rate (sr_1).
		- librosa.load(audio_path_2): Loads the second audio file, returning audio samples (y_2) and sampling rate (sr_2).
		- librosa.display.waveshow(y_1, sr=sr_1): Displays the waveform of the first audio file.
		- librosa.display.waveshow(y_2, sr=sr_2): Displays the waveform of the second audio file.
		- librosa.feature.mfcc(y=y_1, sr=sr_1): Extracts Mel-frequency cepstral coefficients (MFCC) from the first audio file (y_1).
		- librosa.feature.mfcc(y=y_2, sr=sr_2): Extracts MFCC from the second audio file (y_2).
		- librosa.display.specshow(mfcc_1, x_axis='time'): Displays the MFCC of the first audio file as a spectrogram.
		- sr: Displays the sampling rate after the audio file is loaded.
		- librosa.display.specshow(mfcc_2, x_axis='time'): Displays the MFCC of the second audio file as a spectrogram. - librosa.load(audio_file, sr=None): Loads an audio file and returns audio samples (y) and the sampling rate (sr).
matplotlib.pyplot	Plotting and visualizing waveforms and MFCCs	- plt.figure(figsize=(12, 6)): Defines the size of the figure for the plot.
		- plt.subplot(2, 1, 1): Creates the first subplot for visualizing the waveform or MFCC of the first audio file.
		- plt.subplot(2, 1, 2): Creates the second subplot for visualizing the waveform or MFCC of the second audio file.
		- plt.tight_layout(): Adjusts the space between subplots to prevent overlap.
		- plt.show(): Displays the plot to the user.

		- plt.plot(): Creates a line plot (used for the number of lines per character over seasons).
		plt.bar(): Creates a bar plot (used for the number of lines per character).
		- plt.show(): Displays the plot.
		- plt.xlabel(), plt.ylabel(), plt.title(): Add labels and title to the plot.
matplotlib.font_manager	Font management for matplotlib	- fm.findSystemFonts(fontpaths=None, fontext='ttf'): Searches for system fonts on the machine.
		- fm.findfont('serif'): Finds a specific font ('serif') on the system.
matplotlib	Plotting and visualizing data	- set_matplotlib_formats('retina', quality=100): Configures the display format for IPython to provide high-quality output (useful for Jupyter notebooks).
		- plt.rcParams['figure.figsize']: Sets the figure size for the plot.
		- plt.subplots(): Creates a figure and axes for plotting.
		- plt.bar(): Creates a bar chart.
		- plt.plot(): Creates a line plot.
		- plt.xlabel(), plt.ylabel(), plt.title(): Add labels and title to the plot.
		- plt.legend(): Adds a legend to the plot.
		- plt.show(): Displays the plot.
os	Path manipulation for JSON files	os.path.expanduser("~/Documents/BrainHack/BrainHack_projects/data/json/json_aa/json_aa"): Expands the user path to the JSON directory.
		- os.path.join(json_dir, "friends_s01e01a_aa.json"): Joins the directory path with the JSON filename to create the full path.
		- os.path.join(matplotlib.get_cachedir(), 'fontlist-v330.json'): Defines the path to the font cache file.
		- os.remove(font_cache_dir): Deletes the font cache file if it exists.
json	Loading and working	- json.load(file): Loads the content of the JSON file into the data variable.
		- data.keys(): Displays the keys of the loaded JSON data (as a dictionary).

	with JSON data	data["results"]["channels"][0]['alternatives'][0]["words"]: Accesses the transcribed words from the JSON data.
		- json.load(): Reads and parses JSON data.
		- json.dumps(): Converts data to JSON format
assemblyai (aai)	Audio transcription using AssemblyAI API	- aai.settings.api_key = "API_KEY": Configures the API key for accessing the AssemblyAI service.
		- aai.TranscriptionConfig(...): Creates a configuration for the transcription, specifying various options like sentiment analysis, speaker labels, etc.
		- aai.Transcriber(config=config): Creates a transcriber instance with the specified configuration.
		- transcriber.transcribe(FILE_URL): Transcribes the audio file located at the specified URL.
		- transcript.status == aai.TranscriptStatus.error: Checks if the transcription failed and prints any errors.
		- transcript.text: Displays the transcribed text.
		- transcript.utterances: Iterates through the utterances in the transcription, displaying speaker labels and text if available.
		- transcript.entities: Displays detected entities (such as people, locations, etc.) from the transcription.
warnings	Suppressing warnings	- warnings.filterwarnings('ignore'): Ignores all warnings that might appear during code execution.
sqlite3	SQLite database interaction	- sqlite3.connect('friends_script.db'): Creates a connection to an SQLite database (or opens it if it exists).
		- cur = conn.cursor(): Creates a cursor object for executing SQL queries.
		- cur.execute('CREATE TABLE IF NOT EXISTS Friends(...)'): Creates a table in the SQLite database if it does not already exist.
		- df.to_sql('Friends', conn, if_exists='replace', index=False): Saves the DataFrame into the SQLite database table.

		- cur.execute("SELECT char, COUNT(line) AS 'spoken_lines' FROM Friends GROUP BY char ORDER BY spoken_lines DESC"): Executes an SQL query to count the number of lines spoken by each character.
		- most_lines = [c for c in cur.fetchall()]: Fetches and stores the results of the SQL query in a list.
		- sqlite3.connect('friends_script.db'): Creates a connection to an SQLite database (or opens it if it exists).
		- cur.execute(): Executes SQL queries (e.g., to retrieve data from the database).
		- cur.fetchall(): Fetches the result of the query.
		- conn.commit(): Commits changes to the database.
		- df.to_sql(): Stores a DataFrame in a database table.
		cur = conn.cursor(): Creates a cursor object for executing SQL queries.
		- cur.execute(...): Executes a SQL query to retrieve data (e.g., sentiment of Ross in specific episodes).
		- cur.fetchall(): Fetches the results from the query.
TextBlob	Sentiment analysis	- TextBlob(x).sentiment[0]: Applies **TextBlob** to analyze the sentiment of each line of dialogue and extracts the sentiment polarity score.
		- TextBlob(x).sentiment[0]: Extracts the sentiment polarity (how positive or negative the sentence is) for each line.
		- TextBlob(x).sentiment.polarity: Calculates the sentiment polarity score of each dialogue line.
pandas (pd)	Data manipulation and analysis	- pd.DataFrame(master_array, columns=['season', 'episode', 'char', 'line']): Creates a DataFrame from a list (master_array) with specified column names.
		- df['char'].unique()[20:30]: Displays unique characters in the 'char' column.
		- df['char'].replace(...): Replaces variations of character names with a standardized format.
		- df[df['char'].isin(char)]: Filters the DataFrame to keep only specified characters (e.g., Chandler, Joey, Monica, etc.).
		- df['sentiment'] = df['line'].apply(lambda x: TextBlob(x).sentiment[0]): Applies sentiment analysis to each line using **TextBlob** and stores the result in a new column.
		- df['season'] = df['season'].apply(lambda x: int(x)): Converts 'season' column values to integers.

		- df['episode'] = df['episode'].apply(lambda x: int(x)): Converts 'episode' column values to integers.
		- pd.DataFrame(master_array, columns=['season', 'episode', 'char', 'line']): Creates a DataFrame from a list (master_array) with specified column names.
		- df['char'].unique()[20:30]: Displays unique characters in the 'char' column.
		- df['char'].replace(...): Replaces variations of character names with a standardized format.
		- pd.read_sql(): Executes a SQL query directly and loads the results into a DataFrame.
		- df.to_sql(): Saves a DataFrame to an SQLite table.
		- df.groupby(['char']).size(): Groups the df DataFrame by character ('char') and counts the occurrences.
		- df.iterrows(): Iterates over each row of the result_df DataFrame.
		- sum(shared_vocabulary[char1].values()): Sums the values (word counts) for each character to calculate total vocabulary.
		- result_df['Percentage']: Adds the calculated percentage to the DataFrame.
		- max_percentage: Finds the maximum percentage to normalize the data.
		- result_df['Percentage'] = (result_df['Percentage'] / max_percentage) * 100: Normalizes the percentages to a 100% scale.
		- df['sentiment']: Adds a sentiment score to the DataFrame.
		- df.groupby(): Groups dialogue by character for various analyses.
		- df['char'].value_counts(): Counts lines by character.
		- df.iterrows(): Iterates through rows for line analysis.
		- df.apply(): Applies functions (e.g., keyword extraction).
sklearn.feature_extraction.text	Text vectorization (TF-IDF)	- TfidfVectorizer(): Converts text data into a matrix of TF-IDF features (Term Frequency-Inverse Document Frequency).
		- vectorizer.fit_transform(): Fits and transforms the text data to TF-IDF vectors.
sklearn. metrics.pairwise	Similarity calculation (cosine similarity)	- cosine_similarity(tfidf_matrix): Calculates the cosine similarity between two TF-IDF vectors to measure the similarity between two characters' speech styles.
re	Regular expression for text cleaning	- re.compile(): Compiles a regular expression for use in string manipulation.
		- regex.sub(): Removes non-alphabetical characters from text.
itertools	Iteration over pairs of elements	- itertools.combinations(): Generates all possible pairs of characters to compare their language style matching.

collections.Counter	Counting word occurrences	- Counter(): Counts the frequency of words, used in the keyword extraction function.