Documentation

Data Scraping -
The function `scrape_and_split_text(url, max_words=18)` scrapes text from a webpage provided by the `url` parameter, splits it into sentences, and breaks long sentences into shorter ones.

1. Function Parameters:
    - `url`: The URL of the webpage to scrape text from.
    - `max_words`: The maximum number of words allowed in a single sentence after splitting. The default value is set to 18.
2. Function Steps:
    - It starts by defining headers to mimic a web browser using `requests` library.
    - Then, it attempts to fetch the webpage using `requests.get()` method.
    - If successful, it extracts the text from the webpage using BeautifulSoup, removing any script or style tags.
    - The text is split into sentences using regular expressions (`re.split()`).
    - For each sentence, if its length exceeds the `max_words` limit, it is broken into shorter sentences while preserving punctuation boundaries.
    - Finally, it returns the list of split sentences.
3. Error Handling:
    - It handles `requests.RequestException` exceptions that may occur during the HTTP request process.
4. Usage:
    - You can call this function with the URL of a webpage to scrape its text and split it into sentences.
    - You may optionally specify the `max_words` parameter to control the maximum length of each sentence after splitting.
    - The function returns a list of sentences extracted from the webpage, where long sentences are broken into shorter ones.

Refining Parsed Data-
Function `get_entities(sent)` extracts subject and object entities from a given sentence by analyzing its syntactic structure using dependency parsing with spaCy.

1. Functionality:
    - Input: The function takes a single argument `sent`, which represents the input sentence from which subject and object entities are to be extracted.
    - Output: It returns a list containing two elements: the subject entity (`ent1`) and the object entity (`ent2`).

2. Code Explanation:
    - Chunk 1: Initialization of variables such as `ent1`, `ent2`, `prv_tok_dep`, `prv_tok_text`, `prefix`, and `modifier`.
    - Token Iteration (`for tok in nlp(sent)`): It iterates over each token in the processed sentence obtained from spaCy (`nlp(sent)`).
    - Processing Tokens:
        - Chunk 2: Checks if the token is not a punctuation mark (`tok.dep_ != "punct"`).
        - Chunk 2 - Compound Words: Identifies compound words (`tok.dep_ == "compound"`) and combines them with the previous compound word if present.
        - Chunk 2 - Modifiers: Identifies modifier tokens (`tok.dep_.endswith("mod")`) and combines them with the previous compound word if present.
        - Chunk 3: Checks if the token's dependency tag indicates the subject and constructs the subject entity (`ent1`).
        - Chunk 4: Checks if the token's dependency tag indicates the object and constructs the object entity (`ent2`).
        - Chunk 5: Updates variables (`prv_tok_dep`, `prv_tok_text`) for the next iteration.
3. Usage:
    - To use this function, you need to have spaCy installed and have a pre-trained English language model loaded (e.g., `nlp = spacy.load('en_core_web_sm')`).
    - You can call this function with a sentence as input to extract its subject and object entities.
    - It's essential to provide grammatically structured sentences as input to ensure accurate extraction of entities.

Get Relation Function -

This function `get_relation(sent)` is designed to extract the relation between entities from a given sentence by utilizing spaCy's dependency parsing and pattern matching capabilities.

1. Functionality:
    - Input: The function takes a single argument `sent`, which represents the input sentence from which the relation between entities is to be extracted.
    - Output: It returns the relation between entities found in the input sentence.
2. Code Explanation:
    - The function starts by processing the input sentence `sent` using the spaCy model (`nlp(sent)`).
    - It initializes a Matcher object `matcher` using spaCy's vocabulary.

- A pattern is defined to capture the desired structure of the relation between entities. The pattern consists of several dependency tags and part-of-speech (POS) tags that define the expected structure of the relation.
- The pattern is added to the Matcher object with the name `"matching_1"`.
- The Matcher object is then used to find matches in the processed document `doc`.
- If matches are found, the function extracts the last match found (assuming it's the most relevant match) and extracts the text span corresponding to the match.
- Finally, it returns the text of the extracted span, which represents the relation between entities in the input sentence.

3. Usage:
    - To use this function, you need to have spaCy installed and have a pre-trained English language model loaded (e.g., `nlp = spacy.load('en_core_web_sm')`).
    - You can call this function with a sentence as input to extract the relation between entities.
    - It's important to define appropriate patterns based on the expected structure of the relation in your specific use case.

Knowledge Graph-

1. Create a Directed Graph from a DataFrame:
    - It creates a directed graph (`G`) from a pandas DataFrame (`kg_df`).
    - The DataFrame should contain columns named "source" and "target", which represent the source and target nodes of the graph edges, respectively.
    - The `edge_attr=True` parameter indicates that additional attributes associated with the edges are included in the DataFrame.
    - It specifies `create_using=nx.MultiDiGraph()` to create a directed graph allowing multiple parallel edges between nodes.
2. Visualization of the Graph:
    - It sets up a Matplotlib figure for plotting the graph visualization.
    - It calculates the node positions using the spring layout algorithm (`nx.spring_layout(G)`).
    - It draws the graph using NetworkX's `nx.draw()` function.
        - `with_labels=True` adds node labels to the graph.
        - `node_color='red'` sets the color of the nodes to red.
        - `edge_cmap=plt.cm.Blues` sets the colormap for edge colors.
        - `pos=pos` specifies the node positions calculated earlier.
    - Finally, it displays the graph using `plt.show()`.
3. Explanation:
    - The directed graph created from the DataFrame allows for the visualization of relationships between entities represented by nodes in the graph.

- Each edge between nodes represents a relationship, and the direction of the edge indicates the direction of the relationship.
- Node labels are displayed to identify the entities, and the colour of the nodes and edges can be customized for better visualization.
- The layout algorithm (`spring_layout`) is used to position the nodes in the graph in a visually pleasing manner.

Data Filtering-

1. Defined a Function to Check Sentence Length:
    - The function `is_short_sentence(sentence)` takes a sentence as input and returns `True` if the sentence contains two or fewer words, indicating it is a short sentence.
2. Filter the DataFrame:
    - It defines a variable `column_to_check` representing the name of the column in the DataFrame (`kg_df`) where sentence lengths are to be checked. This column is assumed to contain sentences.
    - The DataFrame is filtered using boolean indexing (`kg_df[~kg_df[column_to_check].apply(is_short_sentence)]`) to remove rows containing short sentences in the specified column.
    - The `apply()` method applies the `is_short_sentence()` function to each element in the specified column, returning a boolean mask indicating whether each sentence is short or not. The `~` operator negates this mask, resulting in a filter that keeps only rows with sentences that are not short.
3. Explanation:
    - The purpose of this code is to filter out rows from the DataFrame where the sentences in a specified column (`column_to_check`) are short, defined as having two or fewer words.
    - This filtering is useful for tasks where short sentences might be irrelevant or noise, such as text analysis or natural language processing.

Generate Response-

Function `generate_response(query)` that utilizes a pre-trained sentence transformer model to generate responses based on user queries.

1. Importing Libraries:
    - `pandas` is imported as `pd` for data manipulation.
    - `spacy` is imported for entity recognition and dependency parsing.
    - `SentenceTransformer` and `util` are imported from `sentence_transformers` to work with pre-trained sentence embeddings.

2. Initializing Components:
    - The spaCy model `'en_core_web_sm'` is loaded for natural language processing tasks such as tokenization and part-of-speech tagging.
    - A pre-trained SentenceTransformer model `'all-mpnet-base-v2'` is loaded for encoding sentences into fixed-dimensional vectors.
3. Defining `generate_response()` Function:
    - This function takes a query as input and generates a response based on the highest cosine similarity score between the query and the source or target entity embeddings in the DataFrame.
    - The query is encoded into a fixed-dimensional vector using the loaded SentenceTransformer model.
    - The function iterates over each row in the filtered DataFrame (`kg_df_filtered`) to compute similarity scores between the query embedding and the source and target entity embeddings.
    - If the maximum of the source similarity and target similarity scores exceeds the current maximum score (`max_score`), the response is updated with the corresponding source, edge, and target entities.
    - The function returns the best response found.

4. Functionality:
    - The function utilizes pre-trained models for encoding sentences into fixed-dimensional vectors and computing cosine similarity scores.
    - It generates responses based on the similarity between the user query and the entities stored in the DataFrame, considering both source and target entities.

Summary:

This script showcases a pipeline for extracting information from web page, constructing a knowledge graph, filtering the graph, and using it for generating responses to user queries, all integrated into a conversational interface. The code performs several tasks related to web scraping, natural language processing (NLP), knowledge graph construction, and response generation. Let's break it down step by step:

1. **Importing Libraries:** The code imports necessary libraries for various tasks, such as web scraping (requests, BeautifulSoup), NLP (nltk, spacy), data manipulation (pandas), graph visualization (networkx, matplotlib), and sentence embedding (sentence_transformers).
2. **Setting up NLP Environment:** It loads a pre-trained English language model (en_core_web_sm) from spaCy.

3. **Web Scraping:** The scrape_and_split_text() function is defined as scraping text from a given URL, cleaning it, splitting it into sentences, and breaking long sentences into shorter ones.
4. **Entity Extraction:** The get_entities() function extracts entities (subject and object) from sentences using spaCy's dependency parsing.
5. **Relation Extraction:** The get_relation() function extracts relations between entities based on a predefined pattern using spaCy's dependency parsing and matching capabilities.
6. **Constructing the Knowledge Graph:** Entities and relations extracted from sentences are used to construct a knowledge graph stored in a data frame (kg_df).
7. **Graph Visualization:** The knowledge graph is visualized as a directed graph using NetworkX and Matplotlib.
8. **Data Filtering:** The DataFrame is filtered to remove short sentences using the is_short_sentence() function.
9. **Response Generation:** A pre-trained sentence transformer model (all-mpnet-base-v2) is loaded using the Sentence Transformers library. The generate_response() function generates responses to user queries based on similarity scores between the query and entities in the knowledge graph. It iterates over each row in the filtered DataFrame to compute similarity scores and generate the best response.

Example Usage:
A loop is created to continuously accept user input until the user enters "exit".
For each user input, the generate_response() function is called to generate a response based on the input query.

References-

1. https://docs.streamlit.io/develop/tutorials/llms/build-conversational-apps
2. https://www.kaggle.com/code/pavansanagapati/knowledge-graph-nlp-tutorial-bert-spacy-nltk
3. ChatGPT
4. https://huggingface.co/sentence-transformers/all-mpnet-base-v2