# IR Assignment 4 Report

**Step-wise Explanation of the code**

**1. Data Source Import Setup:** This section of code sets up the environment for importing data from Kaggle. It defines constants and functions required to download and extract data from URLs provided in the `DATA_SOURCE_MAPPING` variable.

**2. Data Download and Extraction:** The script downloads the specified dataset from Kaggle using the URLs provided in `DATA_SOURCE_MAPPING`. It checks if the file is a ZIP or TAR file and extracts it accordingly.

**3. Data Loading and Preprocessing:** After downloading and extracting the dataset, the script loads it into a Pandas DataFrame. It then selects the first 20,000 rows for further processing.

**4. Text Preprocessing**: The script preprocesses the text data by performing the following steps:
- Lowercasing: Converts all text to lowercase to ensure consistency.
- Tokenization: Splits the text into individual words or tokens.
- Punctuation Removal: Removes punctuation marks from the text.
- Stopword Removal: Removes common stopwords like 'and', 'the', 'is', etc., using NLTK's list of English stopwords.
- Lemmatization: Reduces words to their base or root form using NLTK's WordNet lemmatizer. This step helps in standardizing the words (e.g., "running" becomes "run").

**5. Stemming:** The code also defines a function called `lemmatize_text` that lemmatizes tokens in the text. It then applies this function to the 'processed_text' column of the DataFrame.

**6. Duplicate Removal:** Finally, the script removes duplicate entries based on the preprocessed text column to ensure unique samples in the dataset.

**7. Imports:** Import necessary libraries including pandas for data manipulation, numpy for numerical operations, nltk for natural language processing tasks, and re for regular expressions.

**8. NLTK Resources:** Downloads necessary NLTK resources such as word tokenizer, stopwords, and WordNet corpus.

**9. Data Cleaning:**
- **Drop NaN Rows:** Removes rows with NaN values in the 'Summary' column.
- **Lowercasing**: Converts the 'Summary' text to lowercase to ensure consistency.
- **Tokenization**: Splits the 'Summary' text into individual words or tokens using NLTK's word tokenizer.
- **Removing Punctuation**: Eliminates punctuation marks from the tokenized text.
- Stopword Removal: Filters out common stopwords from the text using NLTK's English stopwords list.

- Join Tokens: Joins the processed tokens back into text form.

**10. Stemming and Lemmatization:**
- Stemming: This section defines a function `lemmatize_text` to lemmatize tokens in the text using NLTK's WordNet lemmatizer. The lemmatized tokens are then joined back into text form.
- Apply Lemmatization: Applies the lemmatization function to the 'processed_summary' column of the DataFrame.

**11. Remove Duplicates:** Removes duplicate entries based on the processed summary text to ensure unique summarization samples.

**12. Splitting into Train and Test Sets:** Splits the data into training and testing sets using a 75-25 split ratio.

**13. Creating DataFrames:** Constructs pandas DataFrames for both the training and testing data, containing the processed text and summaries.

**14. Reset Index:** Optionally resets the index of the DataFrames for consistency.

**15. Text Combination and Formatting:**
- Combining Text: Combines the first 100 words of the processed text with the processed summary, separated by 'TEXT:' and 'SUMMARY:' respectively.
- Adding End Marker: Appends 'END' to the end of each combined text to indicate the end of the sequence.

**16. Mounting Google Drive (Optional):** If running on Google Colab, mounts Google Drive to save or load files.

**17. Writing Samples to Text File:**
- This section opens a text file in write mode and iterates through the samples in the 'processed_text' column of the 'train_data' DataFrame.
- Each sample is written to the file with a prefix "Sample {index}:" followed by the sample text.

**18. Custom Dataset Class:**
- Defines a custom dataset class `CustomDataset` inheriting from `torch.utils.data.Dataset`.
- The class takes the file path, tokenizer, and block size as input parameters.
- In the constructor (`__init__`), it reads the text from the file, tokenizes it using the tokenizer, and creates examples of fixed block size from the tokenized text.
- Implements `__len__` and `__getitem__` methods required for a PyTorch dataset.

**19. Load Dataset Function:**
- Modifies the `load_dataset` function to use the custom dataset class (`CustomDataset`).
- It initializes an instance of `CustomDataset` with the provided parameters and returns it.

**20. Load Data Collator Function:**
- Defines a function `load_data_collator` to create a data collator for language modeling.
- It initializes a `DataCollatorForLanguageModeling` object with the provided tokenizer and whether MLM (masked language modeling) is used.

**21. Training Function:**
- Defines a function `train` to perform fine-tuning of the GPT-2 model on the custom dataset.
- It loads the dataset using the `load_dataset` function and the data collator using the `load_data_collator` function.
- Initializes the GPT-2 model and sets up training arguments including output directory, batch size, number of epochs, etc.
- Instantiates a `Trainer` object with the model, training arguments, data collator, and train dataset.
- Calls the `train` method of the trainer to start the training process.
- Saves the trained model.

**22. Training Parameters Setup:**
- Defines parameters such as the path to the training file, model name, output directory, batch size, number of epochs, and save steps.

**23. Training Invocation:**
- Invokes the `train` function with the provided parameters to start the training process.

**24. Function Definitions:**
- `load_model(model_path)`: Loads the fine-tuned GPT-2 model from the specified `model_path`.
- `load_tokenizer(tokenizer_path)`: Loads the GPT-2 tokenizer from the specified `tokenizer_path`.
- `generate_text(sequence, max_length)`: Generates text (summary in this case) given a starting `sequence` and a maximum length (`max_length`) using the loaded model and tokenizer. It utilizes the `generate` method of the GPT-2 model with specific parameters such as sampling, maximum length, and top-k sampling.

**25. Text Generation Loop:**
- Initializes an empty list `predicted_summaries` to store the generated summaries.
- Initializes an index variable `i` to iterate through the `processed_text` column of the `test_data` DataFrame.
- Enters a while loop that continues as long as the index `i` is less than the length of the `processed_text` column.
- Retrieves a `sequence` from the `processed_text` column at index `i`.
- Calculates the `max_len` for the sequence by adding 100 to the number of words in the sequence.
- Calls the `generate_text` function to generate a summary for the sequence with the specified `max_len`.

- Extracts the generated summary from the output and appends it to the `predicted_summaries` list.
- Increments the index `i` to move to the next sequence in the loop.

**26. Explanation:**
- The code essentially generates summaries for each sequence in the `test_data` DataFrame by iterating through its `processed_text` column.
- For each sequence, it generates a summary using the fine-tuned GPT-2 model, considering a maximum length of the sequence plus 100 additional words.
- The generated summary is then appended to the `predicted_summaries` list.
- Finally, the loop continues until summaries are generated for all sequences in the `test_data` DataFrame.