Roll No. = 2021497
Name = Tanuj Kamboj

Assignment 4, Breakdowns

1.

This code snippet allows you to upload a zip file containing a CSV file to Google Colab, extract the contents of the zip file, read the CSV file into a Pandas DataFrame, and then display the first few rows of the DataFrame. It's a convenient way to handle zip file uploads and extract data within Google Colab.

2.

This code snippet uploads a zip file named 'Reviews.csv.zip' to Google Colab, extracts its contents (assuming it contains a single CSV file), reads the CSV file into a Pandas DataFrame, and then displays the first few rows of the DataFrame. It's a concise way to handle zip file uploads, extract CSV data, and view the data quickly in a Jupyter notebook environment.

3.

This code selects specific columns ('Summary' and 'Text') from the DataFrame 'df' and creates a new DataFrame named 'df_selected' containing only these columns. It then displays the first few rows of the new DataFrame, allowing users to quickly inspect the selected columns.

4.

This code block performs text preprocessing on the 'Summary' and 'Text' columns of a DataFrame named 'df_selected'. The preprocessing steps include converting text to lowercase, removing special characters, tokenizing text, removing stopwords, and joining tokens back into a single string. The preprocessed text is then stored in a new DataFrame called 'df_preprocessed'.

5.

This code block applies the `preprocess_text` function to the 'Text' column of the DataFrame `df_selected`, storing the preprocessed text in the 'Text' column of the DataFrame `df_preprocessed`. Finally, it displays the first few rows of the preprocessed DataFrame using the `head()` function.

6.

This code snippet removes the 'Review' column from the DataFrame `df_preprocessed` using the `drop()` function with `columns=['Review']`. The `inplace=True` argument ensures that the changes are made directly to the DataFrame without the need for reassignment. Finally, it displays the first few rows of the updated DataFrame using the `head()` function.

7.

This command installs the `accelerate` package with a version greater than or equal to `0.21.0`. It's a brief instruction to install a specific version or higher of the `accelerate` package.

8.

This command installs the `transformers` library with the `torch` extra dependency, ensuring it's version `4.12.0` or higher. It's a concise instruction to install the `transformers` library along with the required version of `torch`.

9

This code snippet imports the `drive` module from the `google.colab` library and then mounts Google Drive to the Colab environment, allowing access to files and directories stored in Google Drive. It prompts the user to authenticate and obtain an authorization code to grant access to their Google Drive. Once mounted, the contents of the Google Drive will be accessible within the Colab notebook at the specified path (`/content/drive`).

10

This code saves the DataFrame `df_preprocessed` to a CSV file named `df_preprocessed.csv` located in the specified path (`/content/drive/MyDrive/`). The `index=False` parameter ensures that the DataFrame's index is not included in the CSV file.

11

This code reads the CSV file `df_preprocessed.csv` located at the specified path (`/content/drive/MyDrive/`) into a DataFrame named `df_preprocessed_read`, and then displays the first few rows of the DataFrame using the `head()` function.

12.

This code imports the necessary modules and classes from the `transformers` library, including `Trainer`, `TrainingArguments`, `GPT2Tokenizer`, and `GPT2LMHeadModel`. It also initializes a GPT-2 tokenizer and model using the `from_pretrained` method with the `"gpt2"` pretrained model.

13

This code uses `train_test_split` from `sklearn.model_selection` to split the preprocessed DataFrame (`df_preprocessed_read`) into training and testing sets. It samples 2000 rows from the DataFrame using `sample` and then splits them into training and testing sets with a test size of 25% (`test_size=0.25`). The random state is set to 42 for reproducibility.

After splitting the data, it imports `torch` and `Dataset` from `torch.utils.data`. This suggests that the intention might be to create a custom dataset class for PyTorch.

14

Setting `tokenizer.pad_token` to `tokenizer.eos_token` means that the tokenizer will use the end-of-sequence (EOS) token as the padding token. This configuration tells the tokenizer to pad sequences with the end-of-sequence token when necessary during tokenization. It ensures that all sequences have the same length, which is required for training neural networks efficiently.

15

This code snippet filters out rows from the `train_data` DataFrame where either the "Text" or "Summary" column contains empty values (NaN) or an empty string. After filtering, it prints the shape of the filtered DataFrame to show how many rows remain.

This filtering ensures that only rows with valid text and summary values are kept for training the model, which helps prevent issues during training due to missing or invalid data.

16
This code checks if a CUDA-enabled GPU is available for use with PyTorch. If a GPU is available, it assigns the device name to "cuda"; otherwise, it assigns it to "cpu". Finally, it prints the device being used.

This is a common approach to dynamically determine whether to use GPU or CPU resources based on availability. Using a GPU can significantly accelerate computations, especially for deep learning tasks.

17

The `model.to(device_name)` method is used to move the model parameters and buffers to the specified device, either GPU or CPU. This is a common step when working with PyTorch models, especially when training on a GPU. By moving the model to the specified device, all subsequent operations involving the model will be performed on that device.

In the context of deep learning, training models on a GPU can often result in faster training times compared to training on a CPU, especially for large models and datasets. Therefore, it's common practice to move the model to the GPU if one is available.

18

Filtering out rows with empty or unexpected values in the "Text" and "Summary" columns ensures that the test dataset contains only valid data for evaluation. This preprocessing step helps maintain the integrity of the data used for testing and ensures that the evaluation metrics are calculated accurately.

After filtering the test data, you can print the shape of the filtered DataFrame to verify how many rows remain for evaluation. This allows you to understand the size of the test dataset after preprocessing and ensures that it meets your expectations before proceeding with further evaluation steps.

19

It's using the Hugging Face's Transformers library to train a GPT-2 model for a text generation task. Here's a breakdown of the different parts:

1. **TextDataset Class**: This class is a custom PyTorch Dataset. It takes the tokenized encodings of the text data as input and formats them for use in a PyTorch model. The `__getitem__` method returns the input and label tensors for a given index.
2. **DataProcessor Class**: This class is used to process the data. It takes a tokenizer and uses it to tokenize the data in a DataFrame. The `process_data` method tokenizes the data and returns a `TextDataset` object.
3. **Tokenizer**: The tokenizer is responsible for converting the input text into a format that the model can understand. In this case, the GPT-2 tokenizer is used. The `pad_token` is set to the `eos_token` because GPT-2 does not have a default padding token.
4. **TrainingArguments**: This is a class provided by the Transformers library that contains various settings for training the model, such as the number of epochs, batch size, learning rate scheduler settings, and more.
5. **Trainer**: The Trainer class is used to train the model. It takes the model, training arguments, and the tokenized datasets as input. The `train` method is used to train the model, and the `evaluate` method is used to evaluate the model's performance on the evaluation dataset.

In this code, the model is being trained to generate the same text as the input, as the labels are set to the input IDs. This is a form of autoencoding.

20

The `rouge_score` package is a useful library for evaluating the quality of text summarization models using the ROUGE (Recall-Oriented Understudy for Gisting Evaluation) metric.
Once installed, you can import and use the ROUGE metric from the `rouge_score` package to evaluate the performance of your text summarization model by comparing the generated summaries with reference summaries. This evaluation helps quantify how well your model captures the essential information from the input text.

21

It's using the Hugging Face's Transformers library to generate summaries for a test dataset and then compute ROUGE scores for the generated summaries. Here's a breakdown of the different parts:

1. **ROUGE Scorer Initialization**: The `rouge_scorer.RougeScorer` is initialized with the types of ROUGE scores you want to compute: ROUGE-1, ROUGE-2, and ROUGE-L. The `use_stemmer=True` argument means that it will use stemming when computing the scores.
2. **Test Data Iteration**: The code iterates over each row in the test data. For each row, it checks if either the "Text" or "Summary" is NaN or not a string. If so, it skips that row.
3. **Text Tokenization and Conversion to Tensor**: The "Text" for each row is tokenized using the provided tokenizer, and the tokenized text is converted to a PyTorch tensor. This tensor is then moved to the GPU with `.to("cuda")`.
4. **Summary Generation**: The model generates a summary for the tokenized text using the `model.generate()` function. The `max_length=512` argument specifies the maximum length of the generated summary.
5. **Prediction Decoding**: The generated summary, which is in the form of token IDs, is decoded back into text using the `tokenizer.decode()` function.
6. **ROUGE Score Computation**: The ROUGE score between the generated summary and the actual summary ("Summary" column in the test data) is computed using the `scorer.score()` function.
7. **Score Storage**: The computed ROUGE score is appended to the `scores` list.
8. **Score Printing**: After all the summaries have been generated and scores computed, the ROUGE scores are printed.