Q1)

Preprocessing of Text and Summary :-
1) HTML Tag Removal: The beautifulSoup library is used to remove any html tags from the text data. html tags are common in web data but are irrelevant for many text analysis tasks.

2) Lowercasing: Convert all text to lowercase. This standardizes the text and ensures that the model treats words with different cases as the same.

3) Punctuation Removal: Regular expressions (re-module) are used to remove any punctuation marks from the text. Punctuation often doesn't carry much meaning in text analysis tasks and can be safely removed.

4) Tokenization: The text is tokenized using the GPT2 tokenizer. Tokenization splits the text into individual words or subwords, which are easier for the model to process.

5) Lemmatization: Lemmatization is the process of reducing words to their base or dictionary form (lemmas). This step helps in standardizing words and reducing the dimensionality of the feature.

6) Joining Tokens: After tokenization and lemmatization, the processed tokens are joined back into a single string, with consecutive spaces removed.

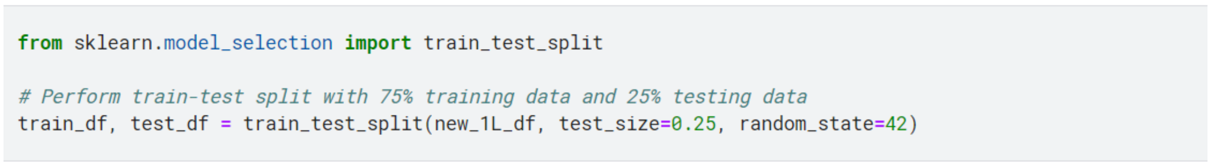
```python
# Initialize GPT2 tokenizer
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

# Preprocess text function with lemmatization and GPT2 tokenizer
def preprocess_text(text):
    # Check if text is not null and is a string
    if isinstance(text, str):
        # Remove HTML tags
        text = BeautifulSoup(text, "html.parser").get_text()

        # Convert text to lowercase
        text = text.lower()

        # Remove punctuations using regular expression
        text = re.sub(r'[^\w\s]', '', text)

        # Tokenization using GPT2 tokenizer
        tokens = tokenizer.tokenize(text)

        # Perform lemmatization
        lemmatized_tokens = [lemmatizer.lemmatize(token) for token in tokens]

        # Join the tokens back into a single string and remove consecutive spaces
        processed_text = ' '.join(lemmatized_tokens).strip()

        return processed_text
    else:
        return ''
```

Model training:-

It begins by importing necessary libraries and defining a custom Dataset class to handle the loading and preprocessing of data. This class tokenizes input text and summaries using a GPT-2 tokenizer and ensures uniform batch sizes through padding. The dataset is prepared by loading and preprocessing training and test data from CSV files, dropping rows with missing text or summaries.

The code initializes the GPT-2 tokenizer and language model. It then sets hyperparameters such as learning rate, batch size, and maximum sequence length, and creates datasets and data loaders for batch processing. Additionally, it defines an optimizer (AdamW) and a learning rate scheduler to adjust learning rates during training. The model is trained over multiple epochs using a training loop, where batches of data are iterated through, and model parameters are updated based on computed loss.

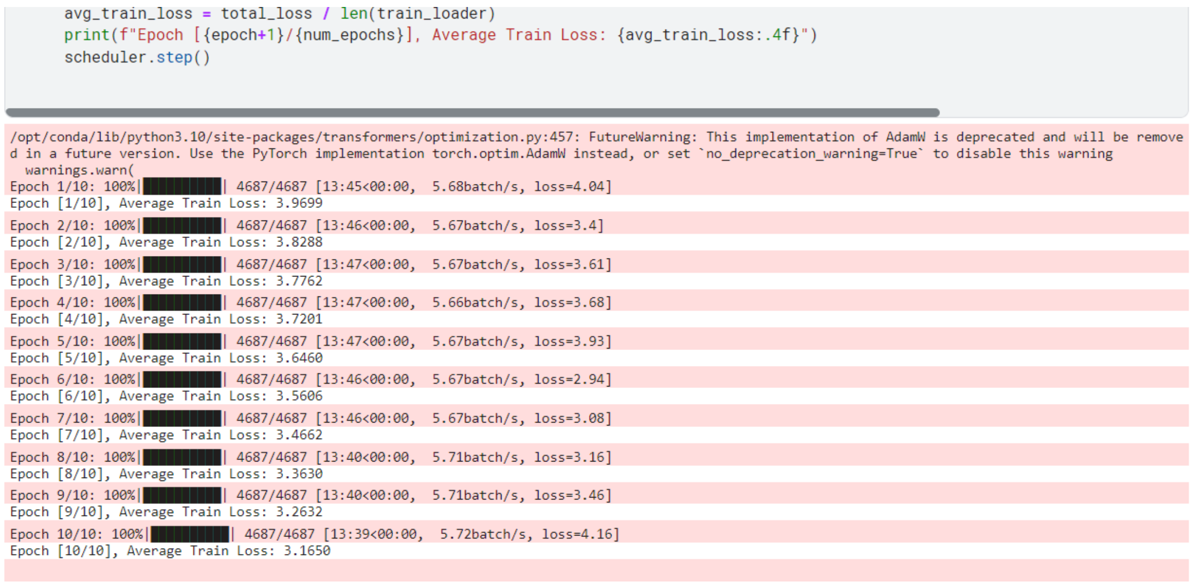
Split the dataset into 75-25 :-



```python
from sklearn.model_selection import train_test_split

# Perform train-test split with 75% training data and 25% testing data
train_df, test_df = train_test_split(new_1L_df, test_size=0.25, random_state=42)
```

Throughout the training process, the model is fine-tuned to generate concise summaries by iteratively adjusting its parameters to minimize the loss function. The training loop executes on either CPU or GPU, depending on availability, to efficiently utilize computational resources. This code provides a comprehensive framework for training a text summarization model using GPT-2, allowing for further experimentation and optimization to enhance summarization performance for various applications.

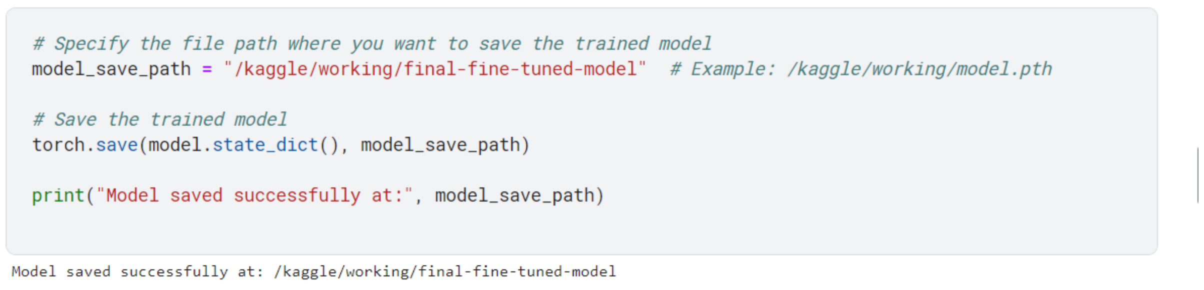
Average loss for first time training on 50000 dataset size:-



```
    avg_train_loss = total_loss / len(train_loader)
    print(f"Epoch [{epoch+1}/{num_epochs}], Average Train Loss: {avg_train_loss:.4f}")
    scheduler.step()
```

```
/opt/conda/lib/python3.10/site-packages/transformers/optimization.py:457: FutureWarning: This implementation of AdamW is deprecated and will be remove
d in a future version. Use the PyTorch implementation torch.optim.AdamW instead, or set `no_deprecation_warning=True` to disable this warning
  warnings.warn(
Epoch 1/10: 100%|██████████| 4687/4687 [13:45<00:00,  5.68batch/s, loss=4.04]
Epoch [1/10], Average Train Loss: 3.9699
Epoch 2/10: 100%|██████████| 4687/4687 [13:46<00:00,  5.67batch/s, loss=3.4]
Epoch [2/10], Average Train Loss: 3.8288
Epoch 3/10: 100%|██████████| 4687/4687 [13:47<00:00,  5.67batch/s, loss=3.61]
Epoch [3/10], Average Train Loss: 3.7762
Epoch 4/10: 100%|██████████| 4687/4687 [13:47<00:00,  5.66batch/s, loss=3.68]
Epoch [4/10], Average Train Loss: 3.7201
Epoch 5/10: 100%|██████████| 4687/4687 [13:47<00:00,  5.67batch/s, loss=3.93]
Epoch [5/10], Average Train Loss: 3.6460
Epoch 6/10: 100%|██████████| 4687/4687 [13:46<00:00,  5.67batch/s, loss=2.94]
Epoch [6/10], Average Train Loss: 3.5606
Epoch 7/10: 100%|██████████| 4687/4687 [13:46<00:00,  5.67batch/s, loss=3.08]
Epoch [7/10], Average Train Loss: 3.4662
Epoch 8/10: 100%|██████████| 4687/4687 [13:40<00:00,  5.71batch/s, loss=3.16]
Epoch [8/10], Average Train Loss: 3.3630
Epoch 9/10: 100%|██████████| 4687/4687 [13:40<00:00,  5.71batch/s, loss=3.46]
Epoch [9/10], Average Train Loss: 3.2632
Epoch 10/10: 100%|██████████| 4687/4687 [13:39<00:00,  5.72batch/s, loss=4.16]
Epoch [10/10], Average Train Loss: 3.1650
```

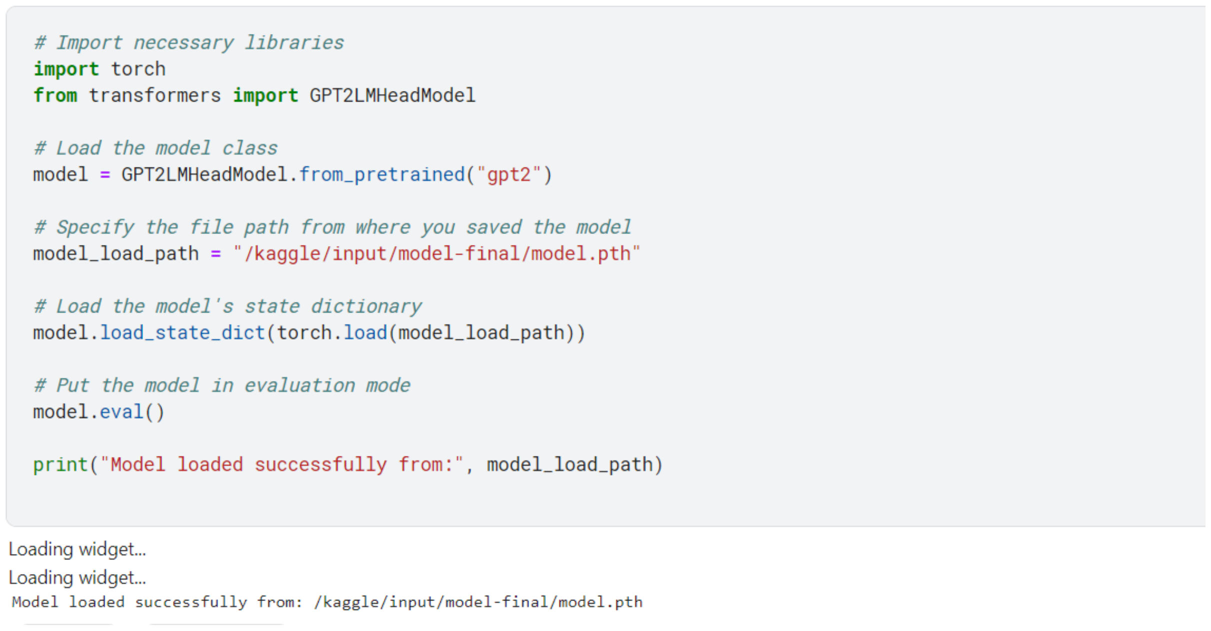
Saving the model after training :-



```
# Specify the file path where you want to save the trained model
model_save_path = "/kaggle/working/final-fine-tuned-model"  # Example: /kaggle/working/model.pth

# Save the trained model
torch.save(model.state_dict(), model_save_path)

print("Model saved successfully at:", model_save_path)
```

```
Model saved successfully at: /kaggle/working/final-fine-tuned-model
```

Loading the model again to train the model on different dataset:-

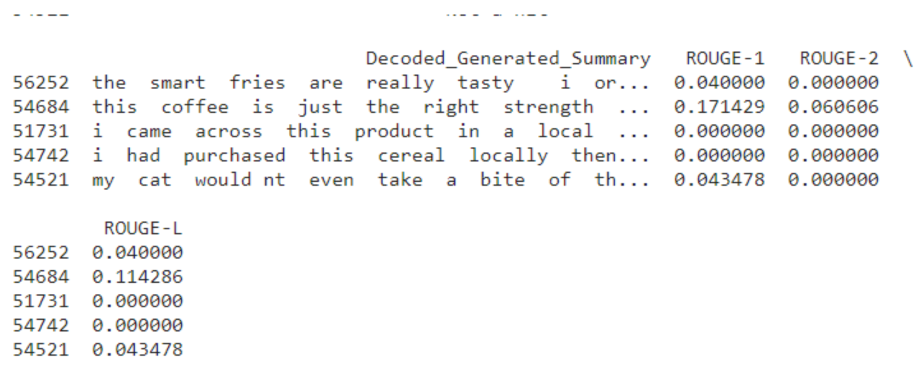


```
# Import necessary libraries
import torch
from transformers import GPT2LMHeadModel

# Load the model class
model = GPT2LMHeadModel.from_pretrained("gpt2")

# Specify the file path from where you saved the model
model_load_path = "/kaggle/input/model-final/model.pth"

# Load the model's state dictionary
model.load_state_dict(torch.load(model_load_path))

# Put the model in evaluation mode
model.eval()

print("Model loaded successfully from:", model_load_path)
```

```
Loading widget...
Loading widget...
Model loaded successfully from: /kaggle/input/model-final/model.pth
```

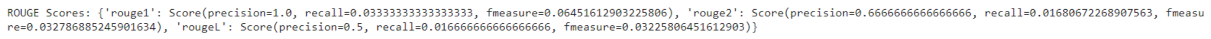
After training save the model again.

Generated summary and rouge score for test dataset :-



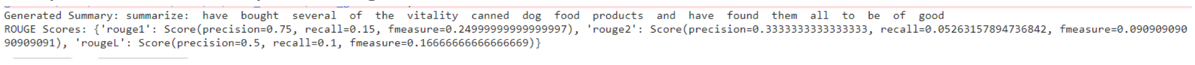
```
                                Decoded_Generated_Summary   ROUGE-1   ROUGE-2  \
56252  the  smart  fries  are  really  tasty    i  or...  0.040000  0.000000
54684  this  coffee  is  just  the  right  strength  ...  0.171429  0.060606
51731  i  came  across  this  product  in  a  local  ...  0.000000  0.000000
54742  i  had  purchased  this  cereal  locally  then...  0.000000  0.000000
54521  my  cat  would nt  even  take  a  bite  of  th...  0.043478  0.000000

        ROUGE-L
56252  0.040000
54684  0.114286
51731  0.000000
54742  0.000000
54521  0.043478
```

For input output:-



```
ROUGE Scores: {'rouge1': Score(precision=1.0, recall=0.03333333333333333, fmeasure=0.06451612903225806), 'rouge2': Score(precision=0.6666666666666666, recall=0.01680672268907563, fmeasu
re=0.032786885245901634), 'rougeL': Score(precision=0.5, recall=0.016666666666666666, fmeasure=0.03225806451612903)}
```

Output with summary and rouge score:-



```
Generated Summary: summarize:  have  bought  several  of  the  vitality  canned  dog  food  products  and  have  found  them  all  to  be  of  good
ROUGE Scores: {'rouge1': Score(precision=0.75, recall=0.15, fmeasure=0.24999999999999997), 'rouge2': Score(precision=0.3333333333333333, recall=0.05263157894736842, fmeasure=0.090909090
90909091), 'rougeL': Score(precision=0.5, recall=0.1, fmeasure=0.16666666666666669)}
```