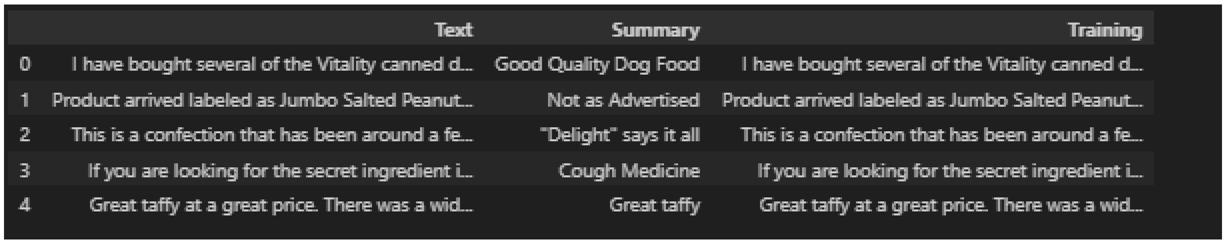
REPORT

Dataset Description

Original size of the dataset=568454



	Text	Summary	Training
0	I have bought several of the Vitality canned d...	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	Product arrived labeled as Jumbo Salted Peanut...	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	This is a confection that has been around a fe...	"Delight" says it all	This is a confection that has been around a fe...
3	If you are looking for the secret ingredient i...	Cough Medicine	If you are looking for the secret ingredient i...
4	Great taffy at a great price. There was a wid...	Great taffy	Great taffy at a great price. There was a wid...

Preprocessing applied:

```
text = re.sub(r'<[^<]+?>', '', text)
    text = re.sub(r'\s+', ' ', text).strip()
    text = re.sub(r'\d', '', text)
    text = re.sub(r'[^\w\s]', '', text)
    text = text.lower()
```

I.e I have removed special characters and done lowercasing

Validation split

```
val=data.sample(frac=0.2,random_state=2000)
```

Sample size for training

```
data = data[:5000]
```

My custom Dataset class

```
class CustomDataset(Dataset):
    def __init__(self, tokenizer, reviews, max_len):
        self.max_len = max_len
        self.tokenizer = tokenizer
        self.reviews = reviews
        self.result = []

        for review in self.reviews:
            # Encode the text using tokenizer.encode(), ensuring the
output is in PyTorch tensors
            tokenized = self.tokenizer.encode(review,
return_tensors='pt').squeeze(0)  # Remove batch dimension

            # Truncate or pad the tokenized tensor to max_len
            padded = self.pad_truncate(tokenized)

            # Store the padded result
            self.result.append(padded)

    def __len__(self):
        return len(self.result)

    def __getitem__(self, idx):
        return self.result[idx]

    def pad_truncate(self, tokens):
        if len(tokens) < self.max_len:
            # Pad with eos_token_id if shorter than max_len
            padded = torch.cat([tokens,
torch.tensor([self.tokenizer.eos_token_id] * (self.max_len -
len(tokens)))])
        else:
            # Truncate and append eos_token_id if longer than max_len
            padded = tokens[:self.max_len - 1]
            padded = torch.cat([padded,
torch.tensor([self.tokenizer.eos_token_id])])
        return padded
```
This class is a custom dataset, meaning it prepares and organizes data (in this case, text reviews) so it can be fed into a model for training or prediction. Here's a breakdown of what's happening in a more human, step-by-step way:

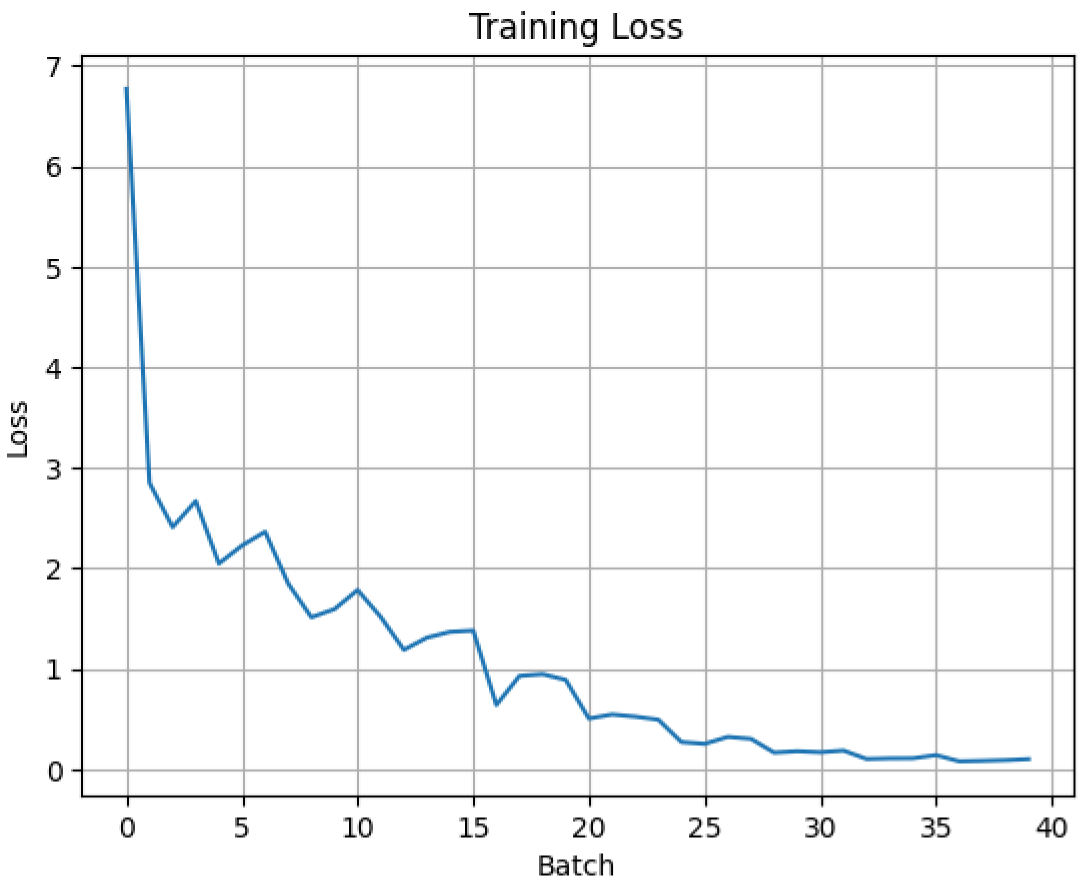
Initialization: When you create an instance of CustomDataset, you need to give it a tokenizer, a list of text reviews, and a maximum length (max_len) for the tokens. The tokenizer is a tool that converts text into a numeric format that the model can understand, while max_len helps ensure all text inputs are of a uniform size.
Tokenizing and Adjusting Text Size:
Tokenizing: Each review is converted into a sequence of numbers (tokens) using the tokenizer. This is like translating words into a secret code that only the model can understand.
Adjusting Text Size (Padding or Truncating): Each tokenized review is adjusted to match the maximum length:
If a review is shorter than max_len, it's padded. This means adding extra tokens to make it longer. Imagine if you had to write a sentence with exactly 50 characters, but your sentence is only 40 characters long, so you add spaces at the end to reach 50.
If a review is too long, it's truncated. This means cutting it short and adding a special end-of-sentence token at the end, much like summarizing a long story to fit within a word limit and then ending with a period.
Storing the Results: Once each review is tokenized and adjusted to the right size, it's stored in a list within the dataset.
Using the Dataset:
Length: If you ask for the length of the dataset, it tells you how many reviews are in it.
Get an Item: If you want a specific processed review, you just need to specify which one you want by its index (like picking a book from a shelf using its position).

Loading model and tokenizer

```
    # Get the tokenizer:
    tokenizer = GPT2Tokenizer.from_pretrained('gpt2')

    # Load pretrained model from Hugging face
    model = AutoModelWithLMHead.from_pretrained("gpt2")
```

THE LOSS PLOT FOR TRAINNG



Testing Dataset

```
reviews=reviews[8000:9000]
```

Note that it does not include any of the training data.

Final Rouge Score: 0.13209081387137114

Overall Rouge Score

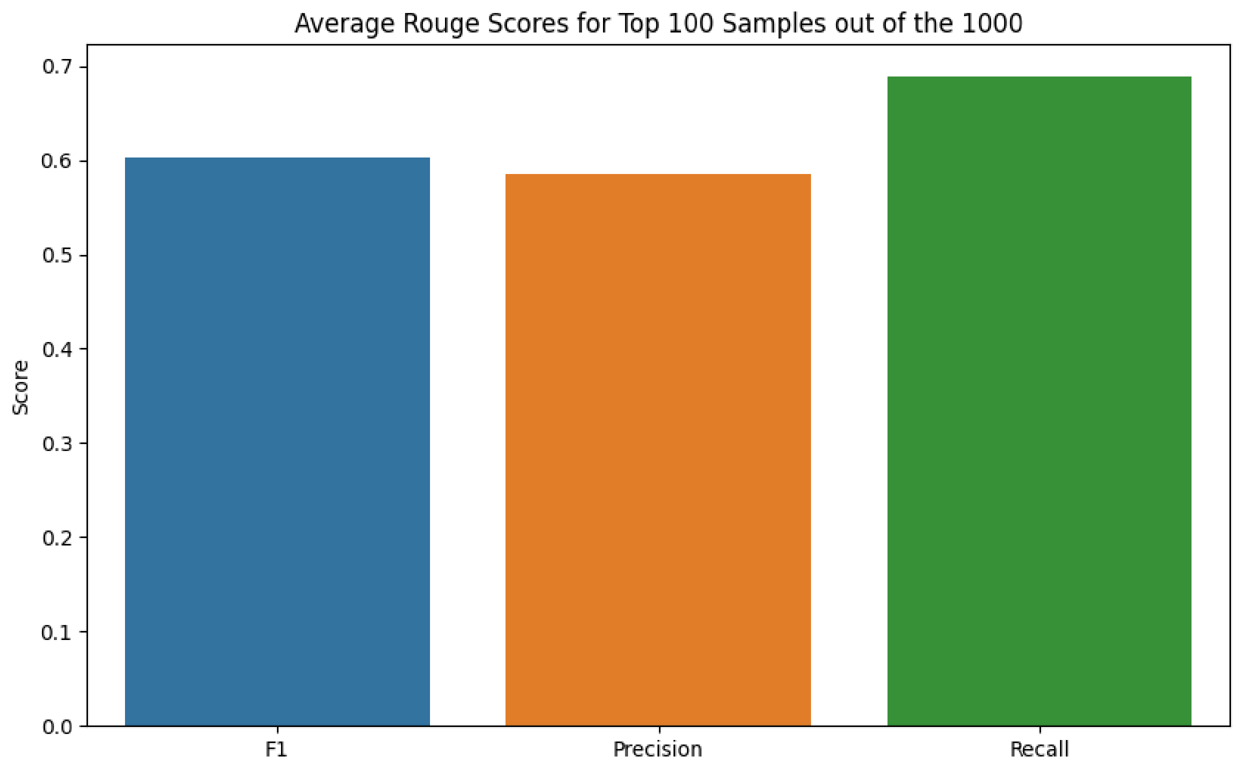
Sample article:
My chihuahuas like this treat, but they do not eat it with enthusiasm. They hesitate and let it sit quite often.

Sample summary:My dog does not like it

Metric	Average	Minimum	Maximum
F1 Score	0.6032	0.4	1.0
Precision Score	0.5859	0.2857	1.0
Recall Score	0.688	0.3333	1.0

Table for top 10% summaries.

Rouge Scores


Average Rouge Scores for Top 100 Samples out of the 1000
Score
F1
Precision
Recall