```
'''
@InProceedings{maas-EtAl:2011:ACL-HLT2011,
  author    = {Maas, Andrew L.  and  Daly, Raymond E.  and  Pham, Peter T.  and  Huang, Dan  and  Ng, Andrew Y.  and  Potts, Christopher},
  title     = {Learning Word Vectors for Sentiment Analysis},
  booktitle = {Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies},
  month     = {June},
  year      = {2011},
  address   = {Portland, Oregon, USA},
  publisher = {Association for Computational Linguistics},
  pages     = {142--150},
  url       = {http://www.aclweb.org/anthology/P11-1015}
}
'''
```

```
'Wn@InProceedings{maas-EtAl:2011:ACL-HLT2011,Wn  author    = {Maas, Andrew L.  and  Daly, Raymond E.  and  Pham, Peter T.  and  Huang, Dan  and  Ng, Andrew Y.  and  Potts, Christopher},Wn  title     = {Learning Word Vectors for Sentiment Analysis},Wn  booktitle = {Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies},Wn  month     = {June},Wn  year      = {2011},Wn  address   = {Portland, Oregon, USA},Wn  publisher = {Association for Computational Linguistics},Wn  pages     = {142--150},Wn  url       = {http://www.aclweb.org/anthology/P11-1015}Wn}Wn'
```

```
import torch
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
import os

from torch.utils.data import Dataset, DataLoader, random_split
from transformers import AutoTokenizer
from transformers import BertForSequenceClassification
from torch.optim import AdamW
from torch.optim.lr_scheduler import StepLR
from tqdm import tqdm
```

```
from google.colab import drive
drive.mount('/content/drive')

path = "/content/drive/MyDrive/Colab_Notebooks/IIPL/week10"
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
file_name = "{}/IMDB_Dataset.csv.zip".format(path)
os.system("unzip "+file_name+" -d "+path)
```

```
256
```

```
df = pd.read_csv("{}/IMDB Dataset.csv".format(path))
df.head()
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

다음 단계: df 변수로 코드 생성 추천 차트 보기 New interactive sheet

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   review     50000 non-null  object
 1   sentiment  50000 non-null  object
dtypes: object(2)
memory usage: 781.4+ KB
```

```
df.isnull().sum()
```

	0
review	0
sentiment	0

dtype: int64

```
df['sentiment'] = df['sentiment'].map({"positive": 1, "negative": 0}).astype(int)
print(df.dtypes)
```

```
review       object
sentiment     int64
dtype: object
```

```
# texts = df['review'].tolist()
# tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")
# token_lengths = [len(tokenizer.encode(text, truncation=False)) for text in texts]
```

```
# token_lengths_lst = np.array(token_lengths)

# under_128 = np.mean(token_lengths_lst <= 128)
# under_256 = np.mean(token_lengths_lst <= 256)
# under_512 = np.mean(token_lengths_lst <= 512)

# print(f"under 128: {under_128:.2%}")
# print(f"under 256: {under_256:.2%}")
# print(f"under 512: {under_512:.2%}")
```

```
# visualize distribution of token lengths (written by ChatGPT 4o)
# plt.figure(figsize=(10, 6))
# plt.hist(token_lengths, bins=50, color='skyblue', edgecolor='black')
# plt.title("Distribution of IMDB token length (BERT Tokenizer)")
# plt.xlabel("Token Length")
# plt.ylabel("Number of Review")
# plt.axvline(x=128, color='red', linestyle='--', label='max_len=128')
# plt.axvline(x=256, color='orange', linestyle='--', label='max_len=256')
# plt.axvline(x=512, color='green', linestyle='--', label='max_len=512')
# plt.legend()
# plt.grid(True)
# plt.show()
```

```
class IMDBdataset(Dataset): # custom dataset
  def __init__(self, df, tokenizer, max_length):
    self.texts = df['review'].values
    self.labels = df['sentiment'].values
    self.tokenizer = tokenizer
    self.max_length = max_length

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

  def __getitem__(self, idx):
    text = self.texts[idx]
    label = self.labels[idx]

    encoded = self.tokenizer(
        text,
        max_length=self.max_length,
        padding='max_length',
        truncation=True,
        return_tensors='pt'
    )

    return{
        'input_ids': encoded['input_ids'].squeeze(),
        'attention_mask': encoded['attention_mask'].squeeze(),
        'labels': torch.tensor(label)
    }
```

```
def train(model, device, train_loader, optimizer, epoch):
  model.train()
  total_loss = 0

  for batch in tqdm(train_loader):
    input_ids = batch['input_ids'].to(device)
    attention_mask = batch['attention_mask'].to(device)
    labels = batch['labels'].to(device)
```

```
    outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
    loss = outputs.loss

    loss.backward()
    torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
    optimizer.step()
    optimizer.zero_grad()

    total_loss += loss.item()

  return total_loss / len(train_loader)
```

```
def test(model, device, test_loader):
  model.eval()
  correct = 0
  total = 0

  with torch.no_grad():
    for batch in test_loader:
      input_ids = batch['input_ids'].to(device)
      attention_mask = batch['attention_mask'].to(device)
      labels = batch['labels'].to(device)

      outputs = model(input_ids, attention_mask=attention_mask)
      preds = torch.argmax(outputs.logits, dim=1)

      correct += (preds == labels).sum().item()
      total += labels.size(0)

  accuracy = correct / total
  print(f"Test Accuracy: {accuracy:.4f}")

  return accuracy
```

```
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

df_size = len(df)
train_size = int(df_size * 0.8)
val_size = int(df_size * 0.1)
test_size = df_size - train_size - val_size

dataset = IMDBdataset(df, tokenizer, max_length=512)
train_dataset, val_dataset, test_dataset = random_split(dataset, [train_size, val_size, test_size])

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32)
test_loader = DataLoader(test_dataset, batch_size=32)
```

```
tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 5.79kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 15.8MB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 22.1MB/s]
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
model = BertForSequenceClassification.from_pretrained('bert-base-uncased')
model.to(device)

optimizer = AdamW(model.parameters(), lr=2e-5)
scheduler = StepLR(optimizer, step_size=10, gamma=0.7)
```

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['c
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
train_losses = []
val_accuracies = []

print("=== Train accuracy ===")
for epoch in range(1, 6):
  loss = train(model, device, train_loader, optimizer, epoch)
  val_accuracy = test(model, device, val_loader)

  train_losses.append(loss)
  val_accuracies.append(val_accuracy)

  scheduler.step()
```

```
print("=== Test accuracy ===")
test_accuracy = test(model, device, test_loader)
```

```
=== Train accuracy ===
100%|██████████| 1250/1250 [12:56<00:00,  1.61it/s]
Test Accuracy: 0.9434
100%|██████████| 1250/1250 [12:56<00:00,  1.61it/s]
Test Accuracy: 0.9414
100%|██████████| 1250/1250 [12:55<00:00,  1.61it/s]
Test Accuracy: 0.9404
100%|██████████| 1250/1250 [12:55<00:00,  1.61it/s]
Test Accuracy: 0.9376
100%|██████████| 1250/1250 [12:55<00:00,  1.61it/s]
Test Accuracy: 0.9410
=== Test accuracy ===
Test Accuracy: 0.9452
```

코딩을 시작하거나 AI로 코드를 생성하세요.