```
import re
import random
import pandas as pd
import numpy as np
import torch
from torch.utils.data import Dataset, DataLoader
from transformers import AutoTokenizer, AutoModelWithLMHead
import torch.optim as optim
from nltk.translate.bleu_score import sentence_bleu
from rouge import Rouge
from tqdm import tqdm
```

```
tokenizer = AutoTokenizer.from_pretrained("gpt2")
model = AutoModelWithLMHead.from_pretrained("gpt2")
```

```
c:\Users\28360\AppData\Local\Programs\Python\Python312\Lib\site-packages\transformers\models\auto\modeling_auto.py:1581: FutureWarnin
  warnings.warn(
```

```
# device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
device = torch.device('cpu')
print(device)
model = model.to(device)
```

```
cpu
```

```
model = model.to(device)
```

```
def topk(probs, n=9):
    # The scores are initially softmaxed to convert to probabilities
    probs = torch.softmax(probs, dim= -1)

    # PyTorch has its own topk method, which we use here
    tokensProb, topIx = torch.topk(probs, k=n)

    # The new selection pool (9 choices) is normalized
    tokensProb = tokensProb / torch.sum(tokensProb)

    # Send to CPU for numpy handling
    tokensProb = tokensProb.cpu().detach().numpy()

    # Make a random choice from the pool based on the new prob distribution
    choice = np.random.choice(n, 1, p = tokensProb)
    tokenId = topIx[choice][0]

    return int(tokenId)
```

```
def model_infer(model, tokenizer, review, max_length=15):
    # Preprocess the init token (task designator)
    review_encoded = tokenizer.encode(review)
    result = review_encoded
    initial_input = torch.tensor(review_encoded).unsqueeze(0).to(device)

    with torch.set_grad_enabled(False):
        # Feed the init token to the model
        output = model(initial_input)

        # Flatten the logits at the final time step
        logits = output.logits[0,-1]

        # Make a top-k choice and append to the result
        result.append(topk(logits))

        # For max_length times:
        for _ in range(max_length):
            # Feed the current sequence to the model and make a choice
            input = torch.tensor(result).unsqueeze(0).to(device)
            output = model(input)
            logits = output.logits[0,-1]
            res_id = topk(logits)

            # If the chosen token is EOS, return the result
            if res_id == tokenizer.eos_token_id:
                return tokenizer.decode(result)
            else: # Append to the sequence
                result.append(res_id)
    # IF no EOS is generated, return after the max_len
    return tokenizer.decode(result)
```

```
path2 = 'C:/Users/28360/Downloads/model/GPT2_summary_model.pt'

model.load_state_dict(torch.load(path2), strict=False)
```

<All keys matched successfully>

```
song_data = pd.read_csv( 'C:/Users/28360/Downloads/songs(2014-23).csv')
song_data = song_data[:8001]
song_data.shape
```

```
(8001, 9)
```

```
result = pd.read_csv('C:/Users/28360/Downloads/gpt2_result.csv')
current_index = result.iloc[-1]['index']

print(song_data.iloc[current_index]['title'] == result.iloc[-1]['title'])
print(current_index)
print(song_data.iloc[current_index]['title'], "|||", result.iloc[-1]['title'])
```

```
True
6830
1984 ||| 1984
```

```
result
```

	index	title	summary
0	0	CA Q1 2015 Earnings Conference Call	NaN
1	1	Dont worry hun	The song "All Good" by The Diplomats portrays ...
2	2	Bukiet smaków	The song "Zodiacz" by Das Racist is a collecti...
3	3	Israel–Palestine Conflict Visionary Stream of ...	The song "Star Wars" by Joe Budden reflects on...
4	4	Who I am?	The song "Kabal finish" by Mabin Family reflec...
...	...	...	...
6831	6826	Where Do We Belong? Anywhere But Here	The song "Do You Love Me" by Gnarls Barkley de...
6832	6827	.smoke rings.	The song "Ain't My Heart" by Atmosphere delves...
6833	6828	Sunny Days	The song "It's Goingober" byuke christopher

```
current_index = 0

for index, row in song_data[current_index+1:].iterrows():
    cleaned_lyrics = row['cleaned_lyrics']
    title = row['title']

    # check if cleaned lyrics exist
    if isinstance(cleaned_lyrics, float):
        print('not exist: ', index)
        summary = ""
    else:
        # gpt2 summary
        try:
            summary = model_infer(model, tokenizer, cleaned_lyrics + " TL;DR ", max_length = 200).split(" TL;DR ")[1].strip()
        except IndexError:
            print("delete: ", index)
            summary = float('nan')

    result_extended = pd.DataFrame([[index, title, summary]], columns=['index', 'title', 'summary'])
    result = pd.concat([result, result_extended], ignore_index=True)

    # successful summarise prompt
    print("finish summarising summary", index)

    # store to csv every 10 summaries
    if index % 10 == 0:
        result.to_csv('C:/Users/28360/Downloads/gpt2_result.csv', index=0)
        print("finish written summary 0 to", index, "to file result.csv")

result
```

```
Token indices sequence length is longer than the specified maximum sequence length for this model (1460 > 1024). Running this sequence
delete:  6831
finish summarising summary 6831
finish summarising summary 6832
finish summarising summary 6833
finish summarising summary 6834
finish summarising summary 6835
finish summarising summary 6836
delete:  6837
finish summarising summary 6837
delete:  6838
finish summarising summary 6838
finish summarising summary 6839
finish summarising summary 6840
finish written summary 0 to 6840 to file result.csv
finish summarising summary 6841
delete:  6842
finish summarising summary 6842
delete:  6843
finish summarising summary 6843
delete:  6844
finish summarising summary 6844
finish summarising summary 6845
finish summarising summary 6846
delete:  6847
finish summarising summary 6847
delete:  6848
finish summarising summary 6848
finish summarising summary 6849
finish summarising summary 6850
finish written summary 0 to 6850 to file result.csv
finish summarising summary 6851
finish summarising summary 6852
finish summarising summary 6853
finish summarising summary 6854
finish summarising summary 6855
delete:  6856
finish summarising summary 6856
finish summarising summary 6857
finish summarising summary 6858
finish summarising summary 6859
delete:  6860
finish summarising summary 6860
finish written summary 0 to 6860 to file result.csv
finish summarising summary 6861
delete:  6862
finish summarising summary 6862
delete:  6863
finish summarising summary 6863
finish summarising summary 6864
finish summarising summary 6865
delete:  6866
finish summarising summary 6866
finish summarising summary 6867
finish summarising summary 6868
finish summarising summary 6869
finish summarising summary 6870
finish written summary 0 to 6870 to file result.csv
not exist:  6871
finish summarising summary 6871
finish summarising summary 6872
finish summarising summary 6873
finish summarising summary 6874
finish summarising summary 6875
delete:  6876
finish summarising summary 6876
finish summarising summary 6877
delete:  6878
finish summarising summary 6878
delete:  6879
finish summarising summary 6879
delete:  6880
finish summarising summary 6880
finish written summary 0 to 6880 to file result.csv
delete:  6881
finish summarising summary 6881
delete:  6882
finish summarising summary 6882
delete:  6883
finish summarising summary 6883
finish summarising summary 6884
delete:  6885
finish summarising summary 6885
delete:  6886
finish summarising summary 6886
delete:  6887
finish summarising summary 6887
finish summarising summary 6888
delete:  6889
finish summarising summary 6889
delete:  6890
finish summarising summary 6890
finish written summary 0 to 6890 to file result.csv
finish summarising summary 6891
finish summarising summary 6892
finish summarising summary 6893
finish summarising summary 6894
```