notebook

March 6, 2025

Car-ing is sharing, an auto dealership company for car sales and rental, is taking their services to the next level thanks to **Large Language Models (LLMs)**.

As their newly recruited AI and NLP developer, you've been asked to prototype a chatbot app with multiple functionalities that not only assist customers but also provide support to human agents in the company.

The solution should receive textual prompts and use a variety of pre-trained Hugging Face LLMs to respond to a series of tasks, e.g. classifying the sentiment in a car's text review, answering a customer question, summarizing or translating text, etc.

0.1 Before you start

In order to complete the project you may wish to install some Hugging Face libraries such as `transformers` and `evaluate`.

[5]:
```
# !pip install transformers
# !pip install evaluate==0.4.0
# !pip install datasets==2.10.0
# #!pip install sentencepiece==0.1.97
```

```
# !pip install sentencepiece

# from transformers import logging
# logging.set_verbosity(logging.WARNING)
```

[72]:
```
#!pip install pandoc
#!pip install nbconvert
```

[12]:
```
import pandas as pd
import torch, evaluate
from datasets import Dataset
from transformers import pipeline, AutoModel, AutoTokenizer
from transformers import AutoModelForSequenceClassification,␣
 ↪AutoModelForSeq2SeqLM, AutoModelForQuestionAnswering
```

0.2 LOAD DATA

[13]:
```
car_review = pd.read_csv("data/car_reviews.csv", sep=";")
car_review
```

[13]:

	Review	Class
0	I am very satisfied with my 2014 Nissan NV SL…	POSITIVE
1	The car is fine. It's a bit loud and not very …	NEGATIVE
2	My first foreign car. Love it, I would buy ano…	POSITIVE
3	I've come across numerous reviews praising the…	NEGATIVE
4	I've been dreaming of owning an SUV for quite …	POSITIVE

[14]:
```
# Convert Pandas DataFrame to Hugging Face Dataset
test_data = Dataset.from_pandas(car_review)
test_data
```

[14]:
```
Dataset({
    features: ['Review', 'Class'],
    num_rows: 5
})
```

[15]:
```
# from datacamp solution
# Put the car reviews and their associated sentiment labels in two lists
reviews_dc = car_review['Review'].tolist()
real_labels_dc = car_review['Class'].tolist()
real_labels_dc
```

[15]:
```
['POSITIVE', 'NEGATIVE', 'POSITIVE', 'NEGATIVE', 'POSITIVE']
```

0.3 SENTIMENT ANALYSIS MODEL

0.3.1 USING PIPELINE APPROACH

[46]:
```
# My solution using pipeline

# Load the sentiment-analysis pipeline
classifier = pipeline(task="sentiment-analysis",␣
 ↪model="distilbert-base-uncased-finetuned-sst-2-english")

# Run predictions
predicted_labels = classifier(test_data['Review'])

print(predicted_labels)

# Extract predicted labels
predictions = [1 if pred['label'] == 'POSITIVE' else 0 for pred in␣
 ↪predicted_labels]

# Print predictions
print(predictions)
```

```
Device set to use mps:0
```

```
[{'label': 'POSITIVE', 'score': 0.9293984174728394}, {'label': 'POSITIVE',
'score': 0.8654274344444275}, {'label': 'POSITIVE', 'score':
0.9994640946388245}, {'label': 'NEGATIVE', 'score': 0.9935314059257507},
{'label': 'POSITIVE', 'score': 0.9986565113067627}]
[1, 1, 1, 0, 1]
```

[47]:
```
# datacamp solution using pipeline

# Load a sentiment analysis LLM into a pipeline
classifier = pipeline('sentiment-analysis',␣
 ↪model='distilbert-base-uncased-finetuned-sst-2-english')

# Perform inference on the car reviews and display prediction results
predicted_labels_dc = classifier(reviews_dc)

# better way of presenting the prediction output
for review_dc, prediction_dc, label_dc in zip(reviews_dc, predicted_labels_dc,␣
 ↪real_labels_dc):
    print(f"Review: {review_dc}\nActual Sentiment: {label_dc}\nPredicted␣
 ↪Sentiment: {prediction_dc['label']} (Confidence: {prediction_dc['score']:.
 ↪4f})\n")
```

```
Device set to use mps:0
```

```
Review: I am very satisfied with my 2014 Nissan NV SL. I use this van for my
business deliveries and personal use. Camping, road trips, etc. We dont have any
```

children so I store most of the seats in my warehouse. I wanted the passenger van for the rear air conditioning. We drove our van from Florida to California for a Cross Country trip in 2014. We averaged about 18 mpg. We drove thru a lot of rain and It was a very comfortable and stable vehicle. The V8 Nissan Titan engine is a 500k mile engine. It has been tested many times by delivery and trucking companies. This is why Nissan gives you a 5 year or 100k mile bumper to bumper warranty. Many people are scared about driving this van because of its size. But with front and rear sonar sensors, large mirrors and the back up camera. It is easy to drive. The front and rear sensors also monitor the front and rear sides of the bumpers making it easier to park close to objects. Our Nissan NV is a Tow Monster. It pulls our 5000 pound travel trailer like its not even there. I have plenty of power to pass a vehicle if needed. The 5.6 liter engine produces 317 hp. I have owned Chevy and Ford vans and there were not very comfortable and had little cockpit room. The Nissan NV is the only vehicle made that has the engine forward like a pick up truck giving the driver plenty of room and comfort in the cockpit area. I dont have any negatives to say about my NV. This is a wide vehicle. The only modification I would like to see from Nissan is for them to add amber side mirror marker lights.BTW. I now own a 2016 Nissan NVP SL. Love it.
Actual Sentiment: POSITIVE
Predicted Sentiment: POSITIVE (Confidence: 0.9294)

Review: The car is fine. It's a bit loud and not very powerful. On one hand, compared to its peers, the interior is well-built. The transmission failed a few years ago, and the dealer replaced it under warranty with no issues. Now, about 60k miles later, the transmission is failing again. It sounds like a truck, and the issues are well-documented. The dealer tells me it is normal, refusing to do anything to resolve the issue. After owning the car for 4 years, there are many other vehicles I would purchase over this one. Initially, I really liked what the brand is about: ride quality, reliability, etc. But I will not purchase another one. Despite these concerns, I must say, the level of comfort in the car has always been satisfactory, but not worth the rest of issues found.
Actual Sentiment: NEGATIVE
Predicted Sentiment: POSITIVE (Confidence: 0.8654)

Review: My first foreign car. Love it, I would buy another.
Actual Sentiment: POSITIVE
Predicted Sentiment: POSITIVE (Confidence: 0.9995)

Review: I've come across numerous reviews praising the Rogue, and I genuinely feel like I might be missing something. It's only been a week since I got the car, and I am genuinely disappointed. I truly wish I could return it. My main concern revolves around what I see as a significant design flaw (which I believe also exists in the Murano, though that wasn't much better and considerably pricier). The rear windshield is just too small. The headrests in the back seat obstruct the sides of the rearview window. This "Crossover" feels more like a cheaply made compact car. My other vehicle is a Sonata, and it provides a significantly quieter and smoother ride. I did not anticipate this car to ride

```
so roughly; my 2006 Pathfinder had a smoother ride! I would rate this car a 5
all around.
Actual Sentiment: NEGATIVE
Predicted Sentiment: NEGATIVE (Confidence: 0.9935)

Review: I've been dreaming of owning an SUV for quite a while, but I've been
driving cars that were already paid for during an extended period. I ultimately
made the decision to transition to a brand-new car, which, of course, involved
taking on new payments. However, given that I don't drive extensively, I was
inclined to avoid a substantial financial commitment. The Nissan Rogue provides
me with the desired SUV experience without burdening me with an exorbitant
payment; the financial arrangement is quite reasonable. Handling and styling are
great; I have hauled 12 bags of mulch in the back with the seats down and could
have held more. I am VERY satisfied overall. I find myself needing to exercise
extra caution when making lane changes, particularly owing to the blind spots
resulting from the small side windows situated towards the rear of the vehicle.
To address this concern, I am actively engaged in making adjustments to my
mirrors and consciously reducing the frequency of lane changes. The engine
delivers strong performance, and the ride is really smooth.
Actual Sentiment: POSITIVE
Predicted Sentiment: POSITIVE (Confidence: 0.9987)
```

0.3.2 USING AUTOMODEL APPROACH

[48]:
```
model = AutoModelForSequenceClassification.
 ↪from_pretrained("distilbert-base-uncased-finetuned-sst-2-english")
tokenizer = AutoTokenizer.
 ↪from_pretrained("distilbert-base-uncased-finetuned-sst-2-english")
```

[49]:
```
# using tokenize function with map()

def tokenize_function(text_data):
    return tokenizer(text_data['Review'],
                     return_tensors="pt",
                     padding=True,
                     truncation=True,
                     max_length=200)

# Tokenize in batches
tokenized_test_data = test_data.map(tokenize_function, batched=True)

print(type(tokenized_test_data))

print(tokenized_test_data)
#tokenized_test_data['input_ids'] # columns are not tensors, will need to␣
 ↪convert to tensor
```

```
Map:   0%|          | 0/5 [00:00<?, ? examples/s]
```

```
<class 'datasets.arrow_dataset.Dataset'>
Dataset({
    features: ['Review', 'Class', 'input_ids', 'attention_mask'],
    num_rows: 5
})
```

[50]:
```
#using direct tokenizer call (output not used in the code below, just for␣
 ↪illustrating the differences)

tokenized_test_data_with_direct_tokenizer_call = tokenizer(test_data['Review'],
                    return_tensors="pt",
                    padding=True,
                    #padding='max_length',
                    truncation=True,
                    max_length=200)

print(type(tokenized_test_data_with_direct_tokenizer_call))

#tokenized_test_data_with_direct_tokenizer_call.input_ids # the columns are␣
 ↪tensors
```

```
<class 'transformers.tokenization_utils_base.BatchEncoding'>
```

0.3.3 Sentiment Analysis Prediction

[51]:
```
# Assuming the model is already on the correct device (CPU or GPU)
device = model.device  # Get the device where the model is located

# Ensure tokenized_test_data['input_ids'] and␣
 ↪tokenized_test_data['attention_mask'] are tensors
input_ids = torch.tensor(tokenized_test_data['input_ids']).to(device)
attention_mask = torch.tensor(tokenized_test_data['attention_mask']).to(device)

# Set model to evaluation mode
model.eval()

# Disable gradient computation to save memory
with torch.no_grad():
    predicted_labels_AM = model(input_ids=input_ids,␣
 ↪attention_mask=attention_mask) #no need to use generate since not generating

# Check the model output structure
print(predicted_labels_AM)

# Extract logits from the model output
```

```
logits = predicted_labels_AM.logits  # This is usually a tensor of shape␣
 ↪(batch_size, num_labels)

# Get the predicted labels by taking the argmax (class with the highest score)
predictions_AM = torch.argmax(logits, dim=1).tolist()

# Now, predicted_labels contains the predicted class indices for each example
print(predictions_AM)
#         0: Negative
#        1: Positive
```

```
SequenceClassifierOutput(loss=None, logits=tensor([[-2.5634,  2.5600],
        [-0.8886,  0.9725],
        [-3.6580,  3.8730],
        [ 2.7306, -2.3037],
        [-1.5670,  1.6940]]), hidden_states=None, attentions=None)
[1, 1, 1, 0, 1]
```

0.3.4 Sentiment Analysis Evaluation

[52]:
```
accuracy = evaluate.load("accuracy")
f1 = evaluate.load("f1")
```

[53]:
```
# my solution

real_labels = [1 if data["Class"] == "POSITIVE" else 0 for data in test_data]
print(f"Real Labels: {real_labels}")
print(f"Predicted Labels: {predictions}")

accuracy_result_compute = accuracy.compute(references=real_labels,␣
 ↪predictions=predictions)
accuracy_result = accuracy_result_compute['accuracy']
print(accuracy_result_compute)
f1_result_compute = f1.compute(references=real_labels, predictions=predictions)
f1_result = f1_result_compute['f1']
print(f1_result_compute)
```

```
Real Labels: [1, 0, 1, 0, 1]
Predicted Labels: [1, 1, 1, 0, 1]
{'accuracy': 0.8}
{'f1': 0.8571428571428571}
```

[54]:
```
## datacamp solution

# Map categorical sentiment labels into integer labels
references_dc = [1 if label == "POSITIVE" else 0 for label in real_labels_dc]
predictions_dc = [1 if label['label'] == "POSITIVE" else 0 for label in␣
 ↪predicted_labels_dc]
```

```
print(f"Real Labels: {references_dc}")
print(f"Predicted Labels: {predictions_dc}")

# Calculate accuracy and F1 score
accuracy_result_dict_dc = accuracy.compute(references=references_dc,␣
 ↪predictions=predictions_dc)
accuracy_result_dc = accuracy_result_dict_dc['accuracy']
f1_result_dict_dc = f1.compute(references=references_dc,␣
 ↪predictions=predictions_dc)
f1_result_dc = f1_result_dict_dc['f1']
print(f"Accuracy: {accuracy_result_dc}")
print(f"F1 result: {f1_result_dc}")
```

```
Real Labels: [1, 0, 1, 0, 1]
Predicted Labels: [1, 1, 1, 0, 1]
Accuracy: 0.8
F1 result: 0.8571428571428571
```

0.4 TEXT TRANSLATION MODEL

0.4.1 USING PIPELINE APPROACH

[17]:
```
# my solution usine pipeline

#         •        Here, truncation happens manually before passing the text to␣
 ↪the model.
        # •        The model never sees the truncated part.
        # •        If important context is cut off, the translation might lose␣
 ↪meaning or fluency.

# first 2 sentences of the 1st review
english_data = ['I am very satisfied with my 2014 Nissan NV SL. I use this van␣
 ↪for my business deliveries and personal use.']

translator = pipeline(task="translation_en_to_es", model="Helsinki-NLP/
 ↪opus-mt-en-es")

translated_review_out = translator(english_data,␣
 ↪clean_up_tokenization_spaces=True)

translated_review = translated_review_out[0]['translation_text']
print(translated_review)
# for sentence in translated_review_out:
#     translated_review = sentence['translation_text']
#     print(translated_review)
# #print(output[0]["translation_text"])
```

```
Device set to use mps:0
```

```
Estoy muy satisfecho con mi Nissan NV SL 2014. Uso esta camioneta para mis
entregas de negocios y uso personal.
```

[27]:
```
# datacamp solution using pipeline

# Truncation Inside Translator
# Here, truncation happens inside the translator() function.
#         •       The model processes the entire first_review and only outputs␣
 ↪up to 27 tokens.
#         •       The model determines which part of the text is most␣
 ↪important for translation based on its learned attention mechanism.

# Best Practice: If possible, let the model handle truncation (max_length),␣
 ↪unless you're sure the input is too long for the model.


# Load translation LLM into a pipeline and translate car review
first_review = reviews_dc[0]
translator = pipeline("translation_en_to_es", model="Helsinki-NLP/
 ↪opus-mt-en-es")

# max_length=27 to limit to only the first 2 sentences of the 1st review
translated_review_dc = translator(first_review,␣
 ↪max_length=27)[0]['translation_text']
print(f"Model translation:\n{translated_review_dc}")
```

```
Device set to use mps:0
Your input_length: 365 is bigger than 0.9 * max_length: 27. You might consider
increasing your max_length manually, e.g. translator('…', max_length=400)
```

```
Model translation:
Estoy muy satisfecho con mi 2014 Nissan NV SL. Uso esta furgoneta para mis
entregas de negocios y uso personal.
```

0.4.2 USING AUTOMODEL APPROACH

[55]:
```
translation_model = AutoModelForSeq2SeqLM.from_pretrained("Helsinki-NLP/
 ↪opus-mt-en-es")
translation_tokenizer = AutoTokenizer.from_pretrained("Helsinki-NLP/
 ↪opus-mt-en-es")

english_input = translation_tokenizer(english_data,
                      return_tensors="pt",
                      padding=True,
                      truncation=True,
                      max_length=100)
```

```
# Set model to evaluation mode
translation_model.eval()

# Disable gradient computation to save memory
with torch.no_grad():
    translated_review_out = translation_model.generate(**english_input) # using␣
 ↪generate to generate translation

print(translated_review_out)

# Decode the translated tokens to get the translated text
translated_review_decoded = [translation_tokenizer.decode(tokens,␣
 ↪skip_special_tokens=True) for tokens in translated_review_out]

#print(translated_review_decoded)

translated_review = translated_review_decoded[0]
print(translated_review)
# Print the translated sentences
# for sentence in translated_review_out:
#     translated_review = sentence
#     print(translated_review)
```

```
/Users/sumitsinha/miniconda3/envs/pytorch/lib/python3.11/site-
packages/transformers/models/marian/tokenization_marian.py:175: UserWarning:
Recommended: pip install sacremoses.
  warnings.warn("Recommended: pip install sacremoses.")
```

```
tensor([[65000,  1686,   239, 22669,    29,   155, 40906,   716,  1239,   175,
           399,  8356,  9652,   135, 26327,    24,  1071,  3678,     9,     4,
          4553,    11,   694,   429,     3,     0]])
Estoy muy satisfecho con mi Nissan NV SL 2014. Uso esta camioneta para mis
entregas de negocios y uso personal.
```

0.4.3 Text Translation Evaluation

[56]:
```
# Open and read the file into a list
with open("data/reference_translations.txt", "r") as file:
    reference_translations = file.readlines()

# Strip newlines and any extra spaces
reference_translations = [line.strip() for line in reference_translations] #␣
 ↪each line is a list

# Print the list
print(f"Spanish translation references:\n{reference_translations}")

print(f"English Text:\n{translated_review}")
```

```
Spanish translation references:
['Estoy muy satisfecho con mi Nissan NV SL 2014. Utilizo esta camioneta para mis
entregas comerciales y uso personal.', 'Estoy muy satisfecho con mi Nissan NV SL
2014. Uso esta furgoneta para mis entregas comerciales y uso personal.']
English Text:
Estoy muy satisfecho con mi Nissan NV SL 2014. Uso esta camioneta para mis
entregas de negocios y uso personal.
```

[57]:
```
bleu = evaluate.load("bleu")
bleu_score = bleu.compute(predictions=[translated_review],␣
 ↪references=[reference_translations])
print(bleu_score)
```

```
{'bleu': 0.7794483794144497, 'precisions': [0.9090909090909091,
0.8571428571428571, 0.75, 0.631578947368421], 'brevity_penalty': 1.0,
'length_ratio': 1.0476190476190477, 'translation_length': 22,
'reference_length': 21}
```

[25]:
```
# datacamp solution (different blue score from my solution because the datacamp␣
 ↪solution translated text is slight different)

# Load and calculate BLEU score metric
bleu_score = bleu.compute(predictions=[translated_review_dc],␣
 ↪references=[reference_translations])
print(bleu_score['bleu'])
```

```
0.6022774485691839
```

0.5 EXTRACTIVE Q&A MODEL

[56]:
```
test_data['Review'][1]
```

[56]:
```
"The car is fine. It's a bit loud and not very powerful. On one hand, compared
to its peers, the interior is well-built. The transmission failed a few years
ago, and the dealer replaced it under warranty with no issues. Now, about 60k
miles later, the transmission is failing again. It sounds like a truck, and the
issues are well-documented. The dealer tells me it is normal, refusing to do
anything to resolve the issue. After owning the car for 4 years, there are many
other vehicles I would purchase over this one. Initially, I really liked what
the brand is about: ride quality, reliability, etc. But I will not purchase
another one. Despite these concerns, I must say, the level of comfort in the car
has always been satisfactory, but not worth the rest of issues found."
```

0.5.1 USING PIPELINE APPROACH

[21]:
```
# Using Pipeline - this is actually extractive Q&A Model

question = "What did he like about the brand?"
```

```
context = test_data['Review'][1]

# Define the appropriate model
qa = pipeline(task="question-answering", model='deepset/minilm-uncased-squad2')

output = qa(question=question, context=context)
print(output['answer'])
```

```
Some weights of the model checkpoint at deepset/minilm-uncased-squad2 were not
used when initializing BertForQuestionAnswering: ['bert.pooler.dense.bias',
'bert.pooler.dense.weight']
- This IS expected if you are initializing BertForQuestionAnswering from the
checkpoint of a model trained on another task or with another architecture (e.g.
initializing a BertForSequenceClassification model from a BertForPreTraining
model).
- This IS NOT expected if you are initializing BertForQuestionAnswering from the
checkpoint of a model that you expect to be exactly identical (initializing a
BertForSequenceClassification model from a BertForSequenceClassification model).
Device set to use mps:0
```

```
ride quality, reliability
```

0.5.2 USING AUTOMODEL APPROACH

[63]:
```
# My solution using AutoModel

# Load model and tokenizer
QAmodel = AutoModelForQuestionAnswering.from_pretrained('deepset/
 ↪minilm-uncased-squad2')
QAtokenizer = AutoTokenizer.from_pretrained('deepset/minilm-uncased-squad2')

# Define question and context
question = "What did he like about the brand?"
context = test_data['Review'][1]

# Tokenize input
inputs = QAtokenizer(question, context, return_tensors="pt", truncation=True,␣
 ↪padding=True)

# Perform inference
QAmodel.eval()
with torch.no_grad():
    outputs = QAmodel(**inputs)

print(outputs)

# The model returns two sets of scores:
```

```
#         •       start_logits: Scores indicating the probability of each␣
 ↪token being the start of the answer.
#         •       end_logits: Scores indicating the probability of each token␣
 ↪being the end of the answer.
# These scores are logits (raw scores before applying softmax).

# Extract answer span
start_scores = outputs.start_logits
end_scores = outputs.end_logits

# •       torch.argmax(start_scores): Finds the index of the token most likely␣
 ↪to be the start of the answer.
# •       torch.argmax(end_scores): Finds the index of the token most likely␣
 ↪to be the end of the answer.
# •       +1: Since Python slicing is exclusive on the right, adding 1 ensures␣
 ↪we include the last token.
start_index = torch.argmax(start_scores)
end_index = torch.argmax(end_scores) + 1  # +1 to include the last token

print(start_index, end_index)

# Decode answer
# •       inputs["input_ids"][0]: Retrieves the token IDs corresponding to the␣
 ↪input text.
# •       convert_ids_to_tokens(...): Converts token IDs back into words␣
 ↪(subword tokens).
# •       convert_tokens_to_string(...): Joins the tokens into a␣
 ↪human-readable sentence.
#answer = QAtokenizer.convert_tokens_to_string(QAtokenizer.
 ↪convert_ids_to_tokens(inputs["input_ids"][0][start_index:end_index]))
#OR
answer = QAtokenizer.decode(inputs["input_ids"][0][start_index:end_index])

print("Answer:", answer)
```

```
Some weights of the model checkpoint at deepset/minilm-uncased-squad2 were not
used when initializing BertForQuestionAnswering: ['bert.pooler.dense.bias',
'bert.pooler.dense.weight']
- This IS expected if you are initializing BertForQuestionAnswering from the
checkpoint of a model trained on another task or with another architecture (e.g.
initializing a BertForSequenceClassification model from a BertForPreTraining
model).
- This IS NOT expected if you are initializing BertForQuestionAnswering from the
checkpoint of a model that you expect to be exactly identical (initializing a
BertForSequenceClassification model from a BertForSequenceClassification model).
Asking to truncate to max_length but no maximum length is provided and the model
has no predefined maximum length. Default to no truncation.
```

```
QuestionAnsweringModelOutput(loss=None, start_logits=tensor([[ 2.2762, -6.2343,
-6.0321, -6.0678, -6.0882, -6.0398, -6.0916, -6.3243,
         -6.6169,  2.2762, -4.1784, -5.3074, -6.0259, -5.0759, -6.3563, -4.7436,
         -6.3526, -6.4298, -5.7884, -5.7856, -5.1914, -6.4162, -4.9813, -5.8880,
         -6.1990, -6.5654, -5.0210, -6.2009, -6.6289, -6.4944, -5.6861, -6.2675,
         -6.1552, -6.4367, -6.4137, -5.1342, -5.3643, -6.2586, -5.5127, -6.2846,
         -6.5086, -6.5867, -5.4555, -5.4227, -6.3117, -6.0545, -6.1566, -6.4791,
         -6.6384, -6.8284, -6.2884, -5.4548, -5.0252, -6.0445, -6.5075, -5.9411,
         -6.0409, -6.8019, -6.3508, -5.8416, -6.7474, -6.7375, -6.0882, -6.3069,
         -5.2681, -5.4771, -6.5445, -6.5645, -6.5778, -6.5040, -5.7578, -5.9364,
         -6.3930, -6.5072, -6.7121, -6.6489, -5.7286, -6.0741, -6.1914, -5.8883,
         -6.2878, -6.8106, -6.4051, -6.0894, -6.1538, -6.4427, -6.1129, -6.5051,
         -6.7101, -6.6582, -3.8438, -4.4204, -5.9236, -5.8944, -5.1940, -6.0101,
         -5.1710, -6.7102, -5.3207, -6.3170, -6.3437, -6.6149, -6.3897, -6.2319,
         -6.2741, -6.7939, -6.3751, -3.2043, -4.9956, -5.6109, -5.8733, -5.5840,
         -4.4363, -6.3063, -6.0574, -4.4409, -5.7468, -5.0566, -5.6441, -5.5577,
         -4.8432, -5.9208, -5.9335, -5.6904, -5.5082, -6.2614, -4.3558, -0.1342,
         -4.2098,  1.3126, -0.0211, -2.1085,  0.5149, -0.6549, -1.4021, -5.1147,
         -4.0444,  1.1624,  6.5843, -0.4191, -4.1999,  0.0190, -5.4061, -3.7833,
         -3.0379, -2.7572, -4.7838, -5.2645, -5.0845, -6.1400, -6.0869, -6.1631,
         -6.0246, -5.2798, -6.0814, -6.0982, -6.5720, -5.8292, -6.1484, -6.5104,
         -6.3216, -4.4269, -3.7098, -5.9750, -2.5570, -6.1964, -5.7527, -6.2070,
         -6.1268, -5.2331, -6.1594, -5.1747, -6.8336, -6.0148, -4.8801, -6.1285,
         -6.1785, -6.4428, -6.5086, -6.2561, -6.7754, -6.7826,  2.2762]]),
end_logits=tensor([[ 1.7217, -5.5849, -5.7638, -5.7849, -5.7597, -5.9993,
-6.0658, -5.7188,
         -5.3955,  1.7217, -6.4926, -5.5980, -6.0494, -3.9630, -3.8977, -6.5789,
         -5.9050, -5.7757, -5.6061, -6.1973, -5.4223, -6.0774, -6.5524, -6.3096,
         -3.7424, -4.1772, -6.8504, -6.1628, -5.5142, -5.6960, -6.6004, -6.4772,
         -6.4225, -5.5627, -5.8090, -6.6598, -6.0527, -6.1480, -6.3114, -6.1945,
         -4.6297, -4.6909, -6.5800, -5.8811, -5.8286, -6.1960, -6.3269, -5.9047,
         -5.5221, -5.1942, -6.2089, -6.5062, -4.7839, -6.3639, -5.9842, -6.4674,
         -6.3350, -5.2968, -6.2943, -6.3907, -5.1605, -5.3773, -6.3443, -6.1900,
         -6.7503, -6.4986, -5.6607, -5.6428, -5.8623, -6.0834, -6.6467, -6.1517,
         -6.2530, -5.6364, -5.6786, -5.2648, -6.5790, -6.2122, -6.3331, -6.5364,
         -5.1746, -5.4133, -6.2698, -6.5477, -6.2277, -6.2842, -6.4320, -6.1782,
         -5.4090, -5.1533, -6.2739, -4.5883, -6.2048, -5.5660, -6.5231, -6.2561,
         -4.0107, -4.7839, -6.1106, -6.0624, -6.2059, -5.4922, -5.8332, -6.1128,
         -6.2036, -4.6374, -5.0095, -6.3997, -6.4959, -6.2846, -4.9585, -6.5286,
         -5.9795, -4.9209, -3.8251, -6.6017, -6.3675, -6.3518, -6.1716, -3.9460,
         -5.4405, -6.1387, -5.3452, -6.0140, -6.2078, -3.3367, -2.8690, -2.9335,
         -4.3813, -3.2139, -3.7768, -3.6477, -4.3307, -5.9965, -4.2802, -5.3511,
         -0.8512, -1.1055,  0.1324,  2.7504, -1.1518,  4.6218,  0.1214,  2.6589,
          4.1670, -4.1144, -5.8860, -6.2337, -5.9412, -5.8832, -5.9582, -1.9777,
         -2.0568, -6.7672, -6.4071, -5.4718, -5.6850, -5.9542, -6.3319, -5.8220,
         -5.9721, -6.8000, -6.4552, -6.4716, -2.5971, -6.2833, -6.5310, -5.0169,
         -6.2512, -6.3676, -6.3587, -4.4565, -4.9604, -6.3348, -6.5119, -6.0018,
         -6.1153, -5.8433, -6.1410, -5.7085, -4.6369, -4.4519,  1.7217]]),
```

```
hidden_states=None, attentions=None)
tensor(138) tensor(142)
Answer: ride quality, reliability
```

[33]:
```python
# datacamp solution using AutoModel - similar to my solution

# Instantiate model and tokenizer
model_ckp = "deepset/minilm-uncased-squad2"
tokenizer = AutoTokenizer.from_pretrained(model_ckp)
model = AutoModelForQuestionAnswering.from_pretrained(model_ckp)

# Define context and question, and tokenize them
context = reviews_dc[1]
print(f"Context:\n{context}")
question = "What did he like about the brand?"
inputs = tokenizer(question, context, return_tensors="pt")

# Perform inference and extract answer from raw outputs
with torch.no_grad():
  outputs = model(**inputs)
start_idx = torch.argmax(outputs.start_logits)
end_idx = torch.argmax(outputs.end_logits) + 1
answer_span = inputs["input_ids"][0][start_idx:end_idx]

# Decode and show answer
answer = tokenizer.decode(answer_span)
print("Answer: ", answer)
```

```
Some weights of the model checkpoint at deepset/minilm-uncased-squad2 were not
used when initializing BertForQuestionAnswering: ['bert.pooler.dense.bias',
'bert.pooler.dense.weight']
- This IS expected if you are initializing BertForQuestionAnswering from the
checkpoint of a model trained on another task or with another architecture (e.g.
initializing a BertForSequenceClassification model from a BertForPreTraining
model).
- This IS NOT expected if you are initializing BertForQuestionAnswering from the
checkpoint of a model that you expect to be exactly identical (initializing a
BertForSequenceClassification model from a BertForSequenceClassification model).

Context:
The car is fine. It's a bit loud and not very powerful. On one hand, compared to
its peers, the interior is well-built. The transmission failed a few years ago,
and the dealer replaced it under warranty with no issues. Now, about 60k miles
later, the transmission is failing again. It sounds like a truck, and the issues
are well-documented. The dealer tells me it is normal, refusing to do anything
to resolve the issue. After owning the car for 4 years, there are many other
vehicles I would purchase over this one. Initially, I really liked what the
brand is about: ride quality, reliability, etc. But I will not purchase another
one. Despite these concerns, I must say, the level of comfort in the car has
```

always been satisfactory, but not worth the rest of issues found.
Answer: ride quality, reliability

0.6 TEXT SUMMARIZATION MODEL

[65]:
```
#last review
#test_data['Review'][4]
#OR
test_data['Review'][-1]  # much better way of finding last review
```

[65]: "I've been dreaming of owning an SUV for quite a while, but I've been driving cars that were already paid for during an extended period. I ultimately made the decision to transition to a brand-new car, which, of course, involved taking on new payments. However, given that I don't drive extensively, I was inclined to avoid a substantial financial commitment. The Nissan Rogue provides me with the desired SUV experience without burdening me with an exorbitant payment; the financial arrangement is quite reasonable. Handling and styling are great; I have hauled 12 bags of mulch in the back with the seats down and could have held more. I am VERY satisfied overall. I find myself needing to exercise extra caution when making lane changes, particularly owing to the blind spots resulting from the small side windows situated towards the rear of the vehicle. To address this concern, I am actively engaged in making adjustments to my mirrors and consciously reducing the frequency of lane changes. The engine delivers strong performance, and the ride is really smooth."

0.6.1 USING PIPELINE APPROACH

[66]:
```
# My solution using pipeline

summarizer = pipeline(task="summarization", model="facebook/bart-large-cnn")

summarized_text = summarizer(test_data['Review'][-1], max_length=51,␣
 ↪clean_up_tokenization_spaces=True)
print(summarized_text[0]["summary_text"])
```

```
Device set to use mps:0
Your min_length=56 must be inferior than your max_length=51.
/Users/sumitsinha/miniconda3/envs/pytorch/lib/python3.11/site-
packages/transformers/generation/utils.py:1432: UserWarning: Unfeasible length
constraints: `min_length` (56) is larger than the maximum possible length (51).
Generation will stop at the defined maximum length. You should decrease the
minimum length and/or increase the maximum length.
  warnings.warn(
```

Nissan Rogue provides me with the desired SUV experience without burdening me with an exorbitant payment. Handling and styling are great; I have hauled 12 bags of mulch in the back with the seats down and could have held more.

[40]:
```
# datacamp solution using pipeline (uses a different model)

# Get original text to summarize upon car review
text_to_summarize = reviews_dc[-1]
print(f"Original text:\n{text_to_summarize}")

# Load summarization pipeline and perform inference
model_name = "cnicu/t5-small-booksum"
summarizer = pipeline("summarization", model=model_name)

outputs = summarizer(text_to_summarize, max_length=53)
summarized_text = outputs[0]['summary_text']
print(f"Summarized text:\n{summarized_text}")
```

```
Original text:
I've been dreaming of owning an SUV for quite a while, but I've been driving
cars that were already paid for during an extended period. I ultimately made the
decision to transition to a brand-new car, which, of course, involved taking on
new payments. However, given that I don't drive extensively, I was inclined to
avoid a substantial financial commitment. The Nissan Rogue provides me with the
desired SUV experience without burdening me with an exorbitant payment; the
financial arrangement is quite reasonable. Handling and styling are great; I
have hauled 12 bags of mulch in the back with the seats down and could have held
more. I am VERY satisfied overall. I find myself needing to exercise extra
caution when making lane changes, particularly owing to the blind spots
resulting from the small side windows situated towards the rear of the vehicle.
To address this concern, I am actively engaged in making adjustments to my
mirrors and consciously reducing the frequency of lane changes. The engine
delivers strong performance, and the ride is really smooth.

Device set to use mps:0

Summarized text:
the Nissan Rogue provides me with the desired SUV experience without burdening
me with an exorbitant payment; the financial arrangement is quite reasonable. I
have hauled 12 bags of mulch in the back with the seats down and could have held
more.
```

0.6.2 USING AUTOMODEL APPROACH

[67]:
```
# My solution using AutoModel

# Load model and tokenizer
model_name = "facebook/bart-large-cnn"
summarytokenizer = AutoTokenizer.from_pretrained(model_name)
summarymodel = AutoModelForSeq2SeqLM.from_pretrained(model_name)

# Select the text to summarize
input_text = test_data['Review'][-1]
```

```
# Tokenize input
inputs = summarytokenizer(input_text, return_tensors="pt", max_length=1024,␣
 ↪truncation=True)

# Generate summary
summarymodel.eval()

# The length_penalty parameter in generate() controls the preference for longer␣
 ↪or shorter sequences in text generation.
# length_penalty=2.0 encourages shorter outputs because a higher penalty (>1.0)␣
 ↪discourages longer sequences.
# •        If length_penalty < 1.0 → Favors longer outputs.
# •        If length_penalty > 1.0 → Favors shorter outputs.
# •        If length_penalty = 1.0 → No length penalty applied (default␣
 ↪behavior).

with torch.no_grad():
    summary_ids = summarymodel.generate(**inputs, max_length=51,␣
 ↪length_penalty=2.0)

# Decode summary
summarized_text = summarytokenizer.decode(summary_ids[0],␣
 ↪skip_special_tokens=True, clean_up_tokenization_spaces=True)

print("Summary:", summarized_text)
```

```
Summary: The Nissan Rogue provides me with the desired SUV experience without
burdening me with an exorbitant payment. Handling and styling are great; I have
hauled 12 bags of mulch in the back with the seats down and could have held
more.
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

0.7 TEXT GENERATION MODEL

[7]:
```
# Using Pipeline

from transformers import pipeline

generator = pipeline(task="text-generation", model="distilgpt2")
prompt = "The Gion neighborhood in Kyoto is famous for"
output = generator(prompt, max_length=100, pad_token_id=generator.tokenizer.
 ↪eos_token_id)
print(" ".join(output[0]["generated_text"].split()))
```

```
Device set to use mps:0
Truncation was not explicitly activated but `max_length` is provided a specific
value, please use `truncation=True` to explicitly truncate examples to max
length. Defaulting to 'longest_first' truncation strategy. If you encode pairs
of sequences (GLUE-style) with the tokenizer you can select this strategy more
precisely by providing a specific strategy to `truncation`.

The Gion neighborhood in Kyoto is famous for its art and its colorful landscape.
It's not the only place to have a family or a friend in Gion, Japan. In the
city's northern suburbs, it is a unique and unique place to have a family or a
friend anywhere. The name Gion refers to Gionshi Shiii in Japan. In the city's
central city, it's the second-highest ranking family in the country. Other
cities have the highest
```

[29]:
```
# Using AutoModel - takes too much memory - times out without torch_dtype=torch.
 ↪float16
# torch.float16: load the model in half-precision (fp16). This significantly␣
 ↪reduces memory consumption.
# But not very good text generations

from transformers import AutoModelForCausalLM, AutoTokenizer
import torch

# Load model and tokenizer
model_name = "distilgpt2"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name, torch_dtype=torch.
 ↪float16)

# Define prompt
prompt = "The Gion neighborhood in Kyoto is famous for"
#input_ids = tokenizer(prompt, return_tensors="pt").input_ids
# OR
input_ids = tokenizer.encode(prompt, return_tensors="pt")
print(input_ids)
```

```
device = torch.device("cpu")
model.to(device)
input_ids = input_ids.to(device)

model.eval()

# Generate text
with torch.no_grad():
    output_ids = model.generate(input_ids, max_length=100,␣
 ↪pad_token_id=tokenizer.eos_token_id)
print(output_ids)
output_text = tokenizer.decode(output_ids[0], skip_special_tokens=True)

print(output_text.strip())
```

```
tensor([[  464,   402,   295,  6232,   287, 36298,   318,  5863,   329]])
tensor([[  464,   402,   295,  6232,   287, 36298,   318,  5863,   329,   663,
          1029,    12, 17163,    11,  1029,    12, 17163,    11,   290,  1029,
            12, 17163,  6832,    13,   198,   198,   198,   198,   198,   198,
           198,   198,   198,   198,   198,   198,   198,   198,   198,   198,
           198,   198,   198,   198,   198,   198,   198,   198,   198,   198,
           198,   198,   198,   198,   198,   198,   198,   198,   198,   198,
           198,   198,   198,   198,   198,   198,   198,   198,   198,   198,
           198,   198,   198,   198,   198,   198,   198,   198,   198,   198,
           198,   198,   198,   198,   198,   198,   198,   198,   198,   198]])
The Gion neighborhood in Kyoto is famous for its high-rise, high-rise, and high-
rise buildings.
```

0.8 GENERATIVE Q&A MODEL

[30]:
```
# Using Pipeline - CORRECT extractive Q&A Model with using deepset/
 ↪minilm-uncased-squad2

question = "What did he like about the brand?"
context = test_data['Review'][1]

# Define the appropriate model
qa = pipeline(task="question-answering", model='deepset/minilm-uncased-squad2')

output = qa(question=question, context=context)
print(output['answer'])
```

```
Some weights of the model checkpoint at deepset/minilm-uncased-squad2 were not
used when initializing BertForQuestionAnswering: ['bert.pooler.dense.bias',
'bert.pooler.dense.weight']
- This IS expected if you are initializing BertForQuestionAnswering from the
checkpoint of a model trained on another task or with another architecture (e.g.
```

```
initializing a BertForSequenceClassification model from a BertForPreTraining
model).
- This IS NOT expected if you are initializing BertForQuestionAnswering from the
checkpoint of a model that you expect to be exactly identical (initializing a
BertForSequenceClassification model from a BertForSequenceClassification model).
Device set to use mps:0
```

```
ride quality, reliability
```

[32]:
```
# Using pipeline - CORRECT generative Q&A Model with using gpt2

question = "What did he like about the brand?"
context = test_data['Review'][1]

# Define the appropriate model
qa = pipeline(task="question-answering", model='gpt2')

input_text = f"Context: {context}\n\nQuestion: {question}\n\nAnswer:"

output = qa({"context": context, "question": question}, max_length=150)
answer = output['answer']

print(answer)
```

```
Some weights of GPT2ForQuestionAnswering were not initialized from the model
checkpoint at gpt2 and are newly initialized: ['qa_outputs.bias',
'qa_outputs.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
Device set to use mps:0
```

```
 rest of issues found.
```

[35]:
```
# Using AutoModel APproach for generative QA

from transformers import AutoModelForCausalLM, AutoTokenizer
import torch

# Define the question and context
question = "What did he like about the brand?"
context = test_data['Review'][1]

# Load the model and tokenizer (using GPT-2 for generative Q&A)
model_name = "gpt2"  # Or use a larger model like "gpt-3.5-turbo" if available
model = AutoModelForCausalLM.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)

# Set the pad_token to be the eos_token for GPT-2
tokenizer.pad_token = tokenizer.eos_token
```

```
# Prepare the input text combining question and context
input_text = f"Context: {context}\n\nQuestion: {question}\n\nAnswer:"

# Tokenize the input
inputs = tokenizer(input_text, return_tensors="pt", truncation=True,␣
 ↪padding=True, max_length=1024)

# Generate the answer (sampling or beam search can be added for diversity)
# max_new_tokens ensures that the model generates no more than the specified␣
 ↪number of tokens.
# num_return_sequences=1: This specifies the number of different sequences the␣
 ↪model should generate for the given input.
# In this case, it will generate 1 sequence (one possible answer).
# If you set it to num_return_sequences=3, the model will generate 3 distinct␣
 ↪answers.
# no_repeat_ngram_size=2: you are asking the model not to repeat any bigrams␣
 ↪(two consecutive tokens) during the generation process.

model.eval()
with torch.no_grad():
    outputs = model.generate(inputs['input_ids'], max_new_tokens=50,␣
 ↪num_return_sequences=1, no_repeat_ngram_size=2)

# Decode the generated answer
generated_answer = tokenizer.decode(outputs[0], skip_special_tokens=True)

# Extract only the answer part (after "Answer:")
answer_start = generated_answer.find("Answer:") + len("Answer:")
answer = generated_answer[answer_start:].strip()

print("Answer:", answer)
```

```
The attention mask and the pad token id were not set. As a consequence, you may
observe unexpected behavior. Please pass your input's `attention_mask` to obtain
reliable results.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.

Answer: I like the way the company is handling the situation. I think the
problem is that the dealership is not doing enough to address the problems. They
are not taking the time to fix the cars. There are a lot of problems with the
vehicles, so
```

[ ]: