**import** pandas **as** pd

**import** torch

# Load the car reviews dataset

file\_path = "data/car\_reviews.csv"

df = pd.read\_csv(file\_path, delimiter=";")

# Put the car reviews and their associated sentiment labels in two lists

reviews = df['Review'].tolist()

real\_labels = df['Class'].tolist()

# Instruction 1: sentiment classification

# Load a sentiment analysis LLM into a pipeline

**from** transformers **import** 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 = classifier(reviews)

**for** review, prediction, label **in** zip(reviews, predicted\_labels, real\_labels):

print(f"Review: {review}\nActual Sentiment: {label}\nPredicted Sentiment: {prediction['label']} (Confidence: {prediction['score']:.4f})\n")

# Load accuracy and F1 score metrics

**import** evaluate

accuracy = evaluate.load("accuracy")

f1 = evaluate.load("f1")

# Map categorical sentiment labels into integer labels

references = [1 **if** label == "POSITIVE" **else** 0 **for** label **in** real\_labels]

predictions = [1 **if** label['label'] == "POSITIVE" **else** 0 **for** label **in** predicted\_labels]

# Calculate accuracy and F1 score

accuracy\_result\_dict = accuracy.compute(references=references, predictions=predictions)

accuracy\_result = accuracy\_result\_dict['accuracy']

f1\_result\_dict = f1.compute(references=references, predictions=predictions)

f1\_result = f1\_result\_dict['f1']

print(f"Accuracy: {accuracy\_result}")

print(f"F1 result: {f1\_result}")

# Instruction 2: Translation

# Load translation LLM into a pipeline and translate car review

first\_review = reviews[0]

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

translated\_review = translator(first\_review, max\_length=27)[0]['translation\_text']

print(f"Model translation:\n{translated\_review}")

# Load reference translations from file

**with** open("data/reference\_translations.txt", 'r') **as** file:

lines = file.readlines()

references = [line.strip() **for** line **in** lines]

print(f"Spanish translation references:\n{references}")

# Load and calculate BLEU score metric

bleu = evaluate.load("bleu")

bleu\_score = bleu.compute(predictions=[translated\_review], references=[references])

print(bleu\_score['bleu'])

# Instruction 3: extractive QA

# Import auto classes (optional: can be solved via pipelines too)

**from** transformers **import** AutoTokenizer

**from** transformers **import** AutoModelForQuestionAnswering

# 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[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)

# Instruction 4

# Get original text to summarize upon car review

text\_to\_summarize = reviews[-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}")