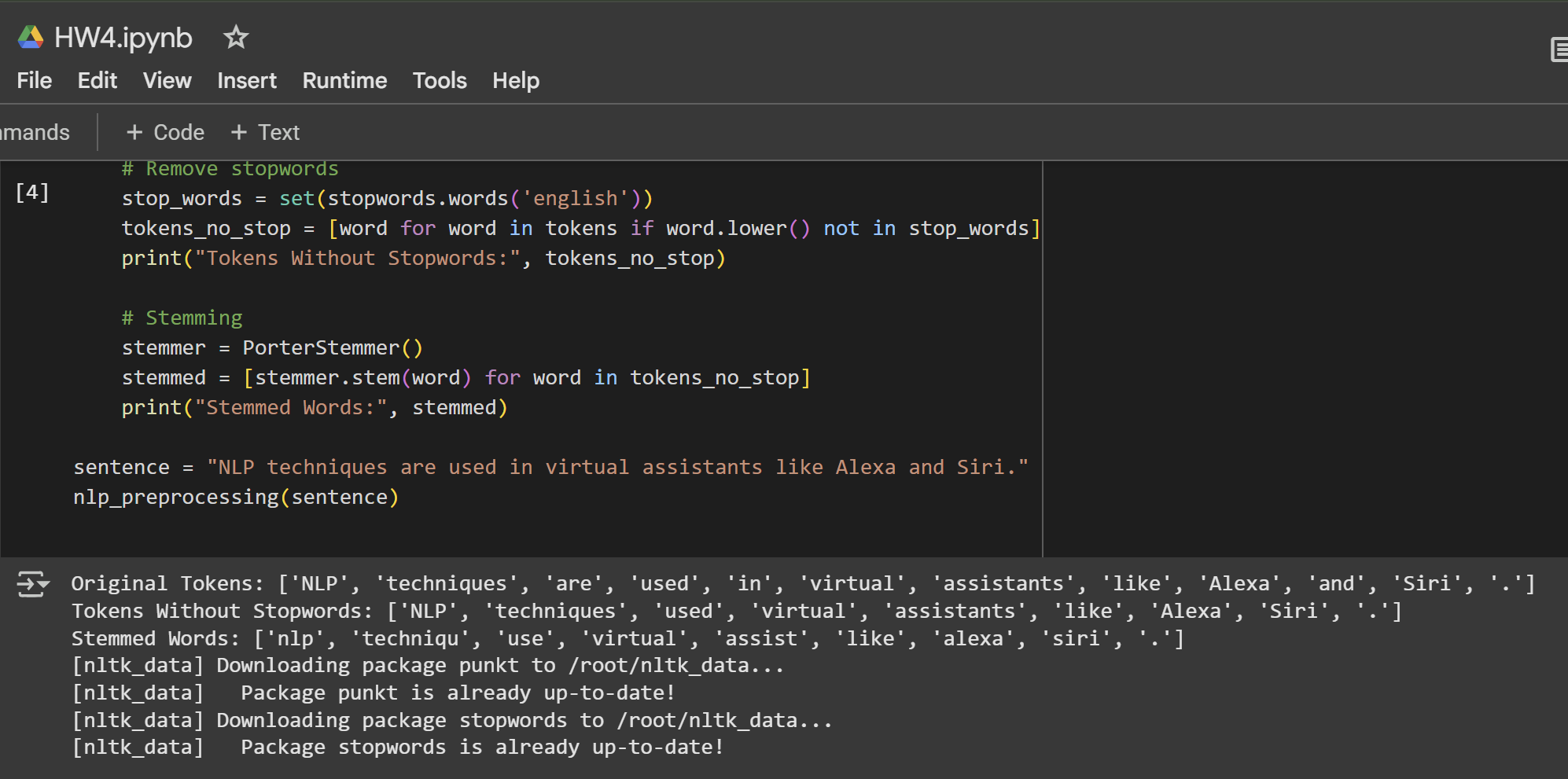
CS5720 Neural Network and Deep Learning - Home Assignment 4 Answers

# Q1: NLP Preprocessing Pipeline

Python Function:

import nltk  
from nltk.tokenize import word\_tokenize  
from nltk.corpus import stopwords  
from nltk.stem import PorterStemmer  
  
# Download resources if not already present  
nltk.download('punkt')  
nltk.download('stopwords')  
  
def nlp\_preprocessing(sentence):  
 # Tokenize  
 tokens = word\_tokenize(sentence)  
 print("Original Tokens:", tokens)  
  
 # Remove stopwords  
 stop\_words = set(stopwords.words('english'))  
 tokens\_no\_stop = [word for word in tokens if word.lower() not in stop\_words]  
 print("Tokens Without Stopwords:", tokens\_no\_stop)  
  
 # Stemming  
 stemmer = PorterStemmer()  
 stemmed = [stemmer.stem(word) for word in tokens\_no\_stop]  
 print("Stemmed Words:", stemmed)  
  
sentence = "NLP techniques are used in virtual assistants like Alexa and Siri."  
nlp\_preprocessing(sentence)



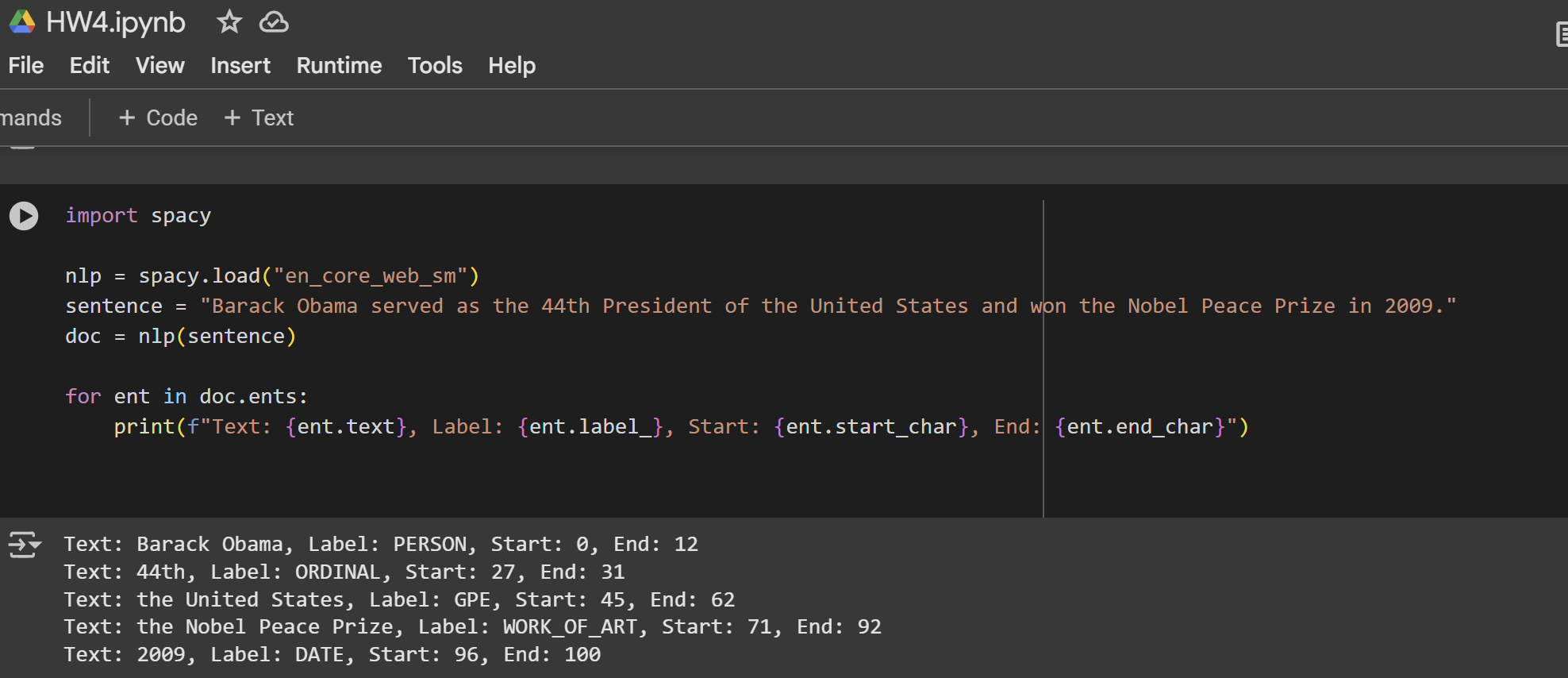
Short Answer Questions:

1. Difference between stemming and lemmatization (example: “running”):  
Stemming reduces a word to its root by removing suffixes, often producing non-words (e.g., “running” → “run”). Lemmatization reduces a word to its base or dictionary form, considering context and grammar (e.g., “running” → “run” if verb, “running” → “running” if noun).

2. Why remove stop words? When might it be harmful?  
Removing stop words helps focus on meaningful words, improving efficiency and sometimes accuracy in tasks like topic modeling. However, it can be harmful in tasks where stop words carry important meaning, such as sentiment analysis (“not good” vs. “good”).

# Q2: Named Entity Recognition with SpaCy

Python Code:

import spacy  
  
nlp = spacy.load("en\_core\_web\_sm")  
sentence = "Barack Obama served as the 44th President of the United States and won the Nobel Peace Prize in 2009."  
doc = nlp(sentence)  
  
for ent in doc.ents:  
 print(f"Text: {ent.text}, Label: {ent.label\_}, Start: {ent.start\_char}, End: {ent.end\_char}")  


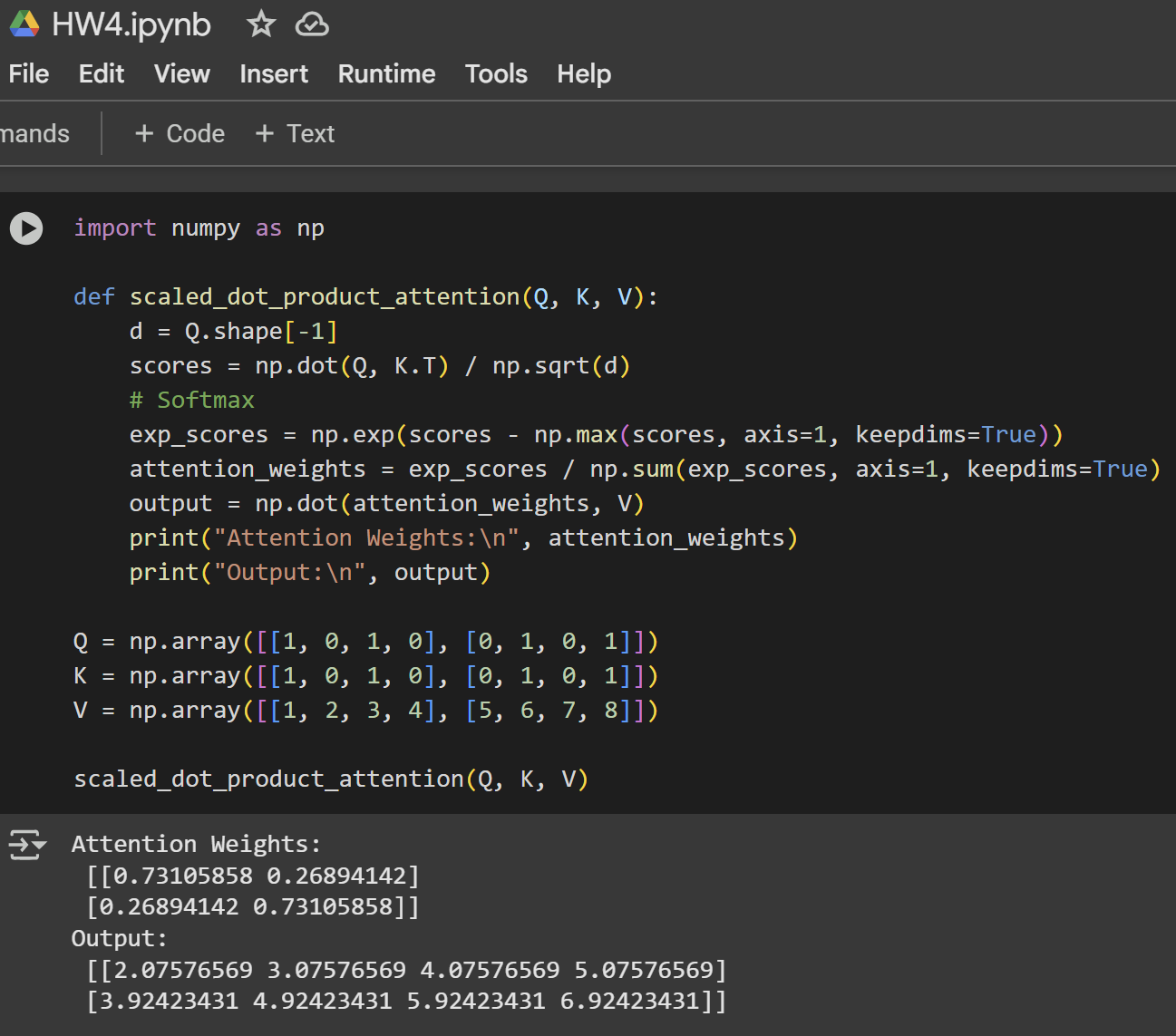
Short Answer Questions:

1. How does NER differ from POS tagging?  
NER identifies and classifies named entities (like people, organizations, dates), while POS tagging labels each word with its part of speech (noun, verb, etc.).

2. Two real-world NER applications:  
- Financial news: Extracting company names, monetary values, and dates for market analysis.  
- Search engines: Recognizing entities in queries to improve search relevance.

# Q3: Scaled Dot-Product Attention

Python Code:

import numpy as np  
  
def scaled\_dot\_product\_attention(Q, K, V):  
 d = Q.shape[-1]  
 scores = np.dot(Q, K.T) / np.sqrt(d)  
 # Softmax  
 exp\_scores = np.exp(scores - np.max(scores, axis=1, keepdims=True))  
 attention\_weights = exp\_scores / np.sum(exp\_scores, axis=1, keepdims=True)  
 output = np.dot(attention\_weights, V)  
 print("Attention Weights:  
", attention\_weights)  
 print("Output:  
", output)  
  
Q = np.array([[1, 0, 1, 0], [0, 1, 0, 1]])  
K = np.array([[1, 0, 1, 0], [0, 1, 0, 1]])  
V = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])  
  
scaled\_dot\_product\_attention(Q, K, V)  


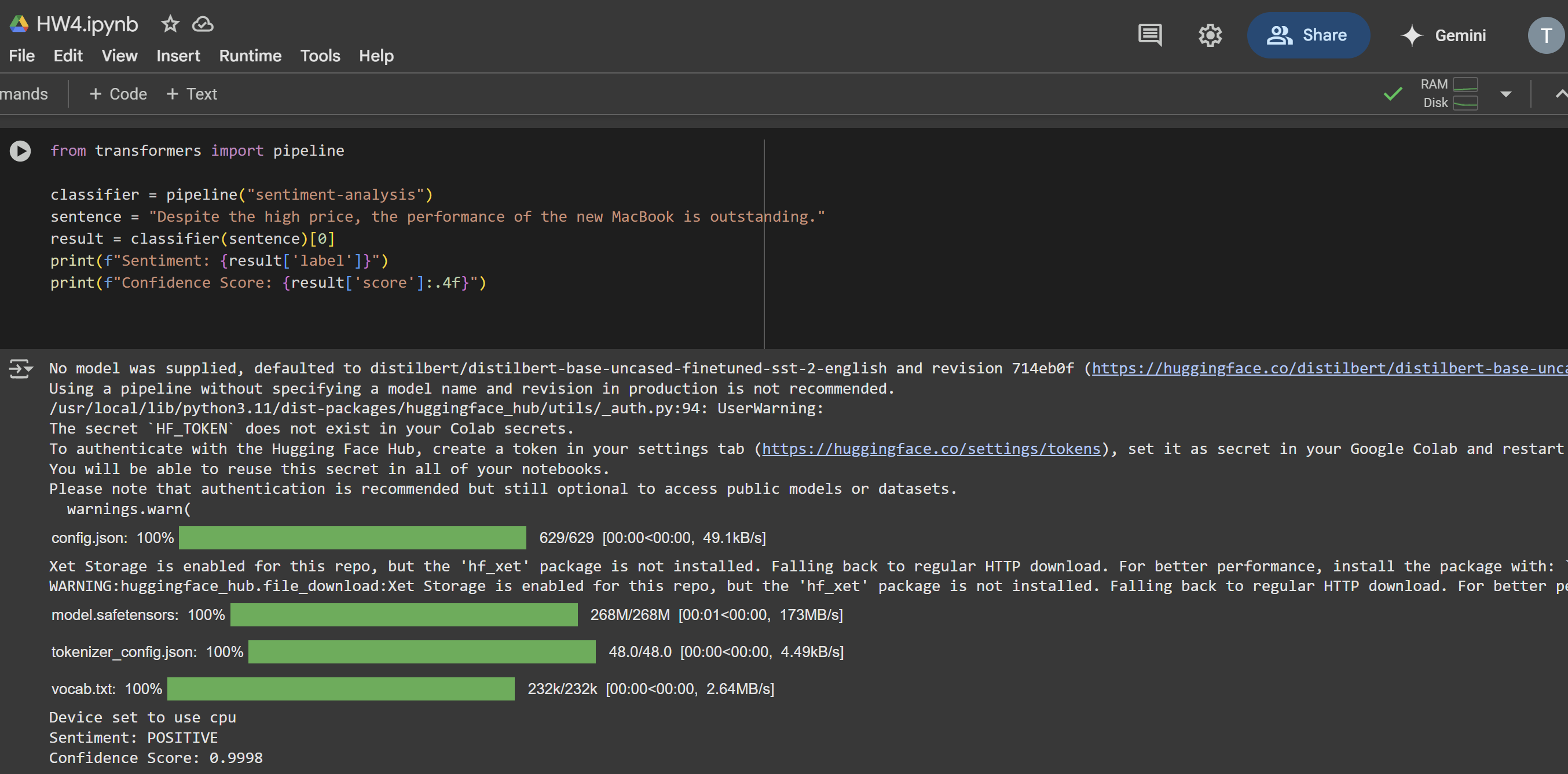
Short Answer Questions:

1. Why divide by √d in scaled dot-product attention?  
Dividing by √d prevents the dot products from growing too large, which stabilizes gradients and makes the softmax function more effective.

2. How does self-attention help models?  
Self-attention allows the model to weigh the importance of each word in a sentence relative to others, capturing dependencies regardless of their distance.

# Q4: Sentiment Analysis using HuggingFace Transformers

Python Code:

from transformers import pipeline  
  
classifier = pipeline("sentiment-analysis")  
sentence = "Despite the high price, the performance of the new MacBook is outstanding."  
result = classifier(sentence)[0]  
print(f"Sentiment: {result['label']}")  
print(f"Confidence Score: {result['score']:.4f}")  


Short Answer Questions

1. Main architectural difference between BERT and GPT:

BERT uses only the encoder part of the transformer (bidirectional), while GPT uses only the decoder (unidirectional).

2. Why use pre-trained models?

Pre-trained models save time and resources, leverage large-scale data, and provide strong performance even with limited task-specific data.