SAMBHRAM INSTITUTE OF TECHNOLOGY
BENGALURU

Department of Artificial Intelligence and Machine Learning

GENERATIVE AI
(BAIL657C)
Lab Manual
2024-25

Prepared by,
Shekhar P
Assistant Professor,
Dept of AIML,
SaIT Bangalore

List of Experiments

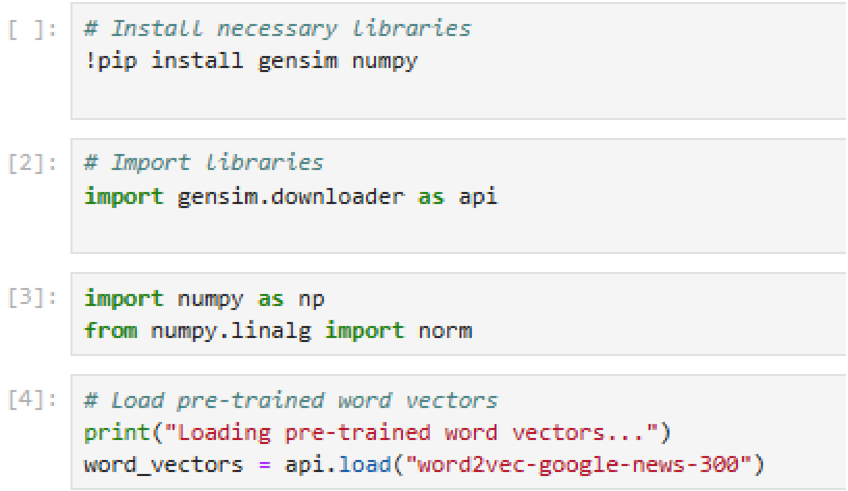
1. Explore pre-trained word vectors. Explore word relationships using vector arithmetic. Perform arithmetic operations and analyze results.

2. Use dimensionality reduction (e.g., PCA or t-SNE) to visualize word embeddings for Q 1. Select 10 words from a specific domain (e.g., sports, technology) and visualize their embeddings. Analyze clusters and relationships. Generate contextually rich outputs using embeddings. Write a program to generate 5 semantically similar words for a given input.

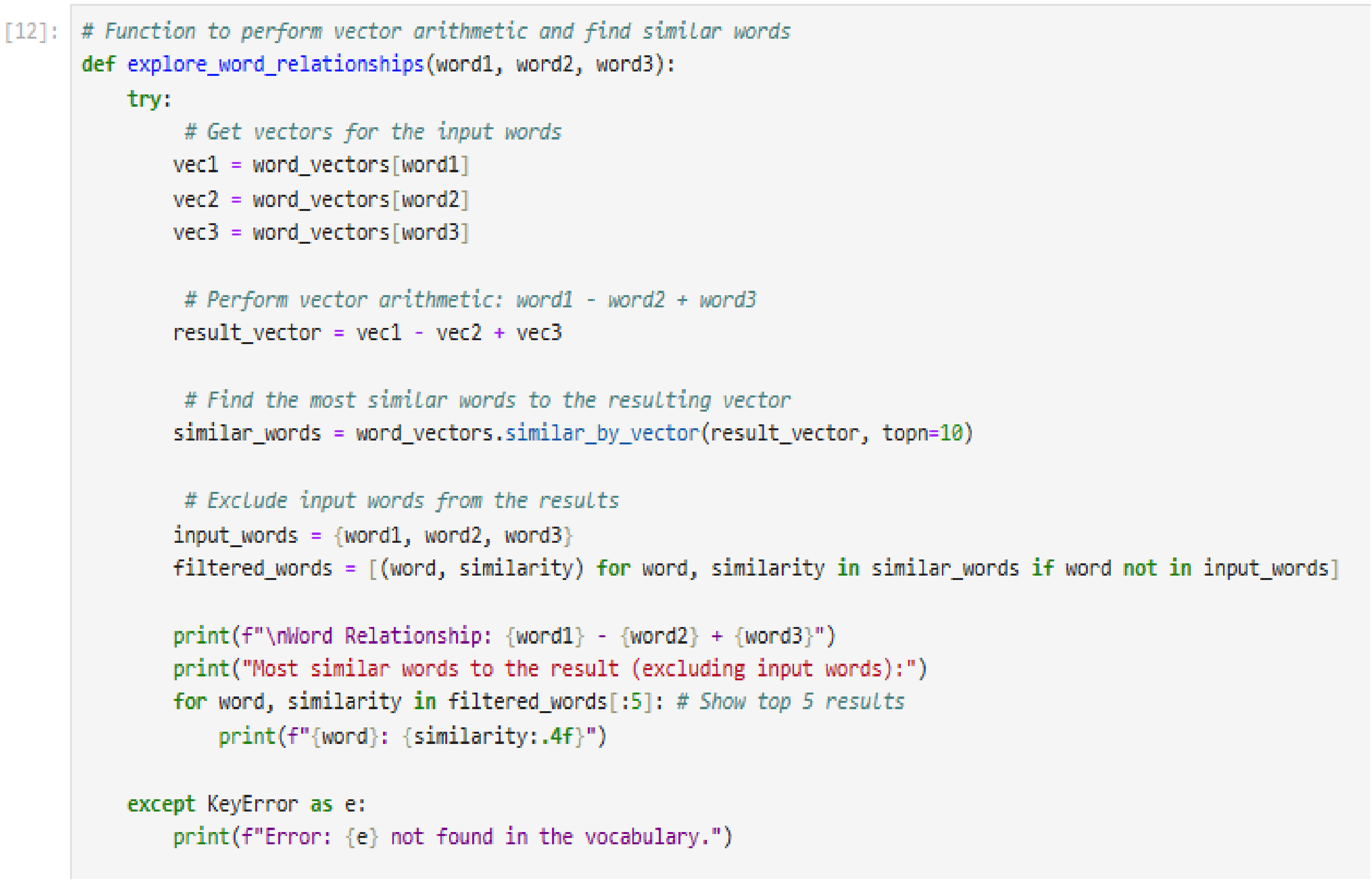
3. Train a custom Word2Vec model on a small dataset. Train embeddings on a domain-specific corpus (e.g., legal, medical) and analyze how embeddings capture domain-specific semantics.
4. Use word embeddings to improve prompts for Generative AI model. Retrieve similar words using word embeddings. Use the similar words to enrich a GenAI prompt. Use the AI model to generate responses for the original and enriched prompts. Compare the outputs in terms of detail.
5. Use word embeddings to create meaningful sentences for creative tasks. Retrieve similar words for a seed word. Create a sentence or story using these words as a starting point. Write a program that: Takes a seed word. Generates similar words. Constructs a short paragraph using these words.
6. Use a pre-trained Hugging Face model to analyze sentiment in text. Assume a real-world application, Load the sentiment analysis pipeline. Analyze the sentiment by giving sentences to input.
7. Summarize long texts using a pre-trained summarization model using Hugging face model. Load the summarization pipeline. Take a passage as input and obtain the summarized text.
8. Install langchain, cohere (for key), langchain-community. Get the api key(By logging into Cohere and obtaining the cohere key).Load a text document from your google drive. Create a prompt template to display the output in a particular manner.
9. Take the Institution name as input. Use Pydantic to define the schema for the desired output and create a custom output parser. Invoke the Chain and Fetch Results. Extract the below Institution related details from Wikipedia: The founder of the Institution. When it was founded. The current branches in the institution. How many employees are working in it? A brief 4-line summary of the institution.
10. Build a chat bot for the Indian Penal Code. We'll start by downloading the official Indian Penal Code document, and then we'll create a chat bot that can interact with it. Users will be able to ask questions about the Indian Penal Code and have a conversation with it.

Experiment No. 1:

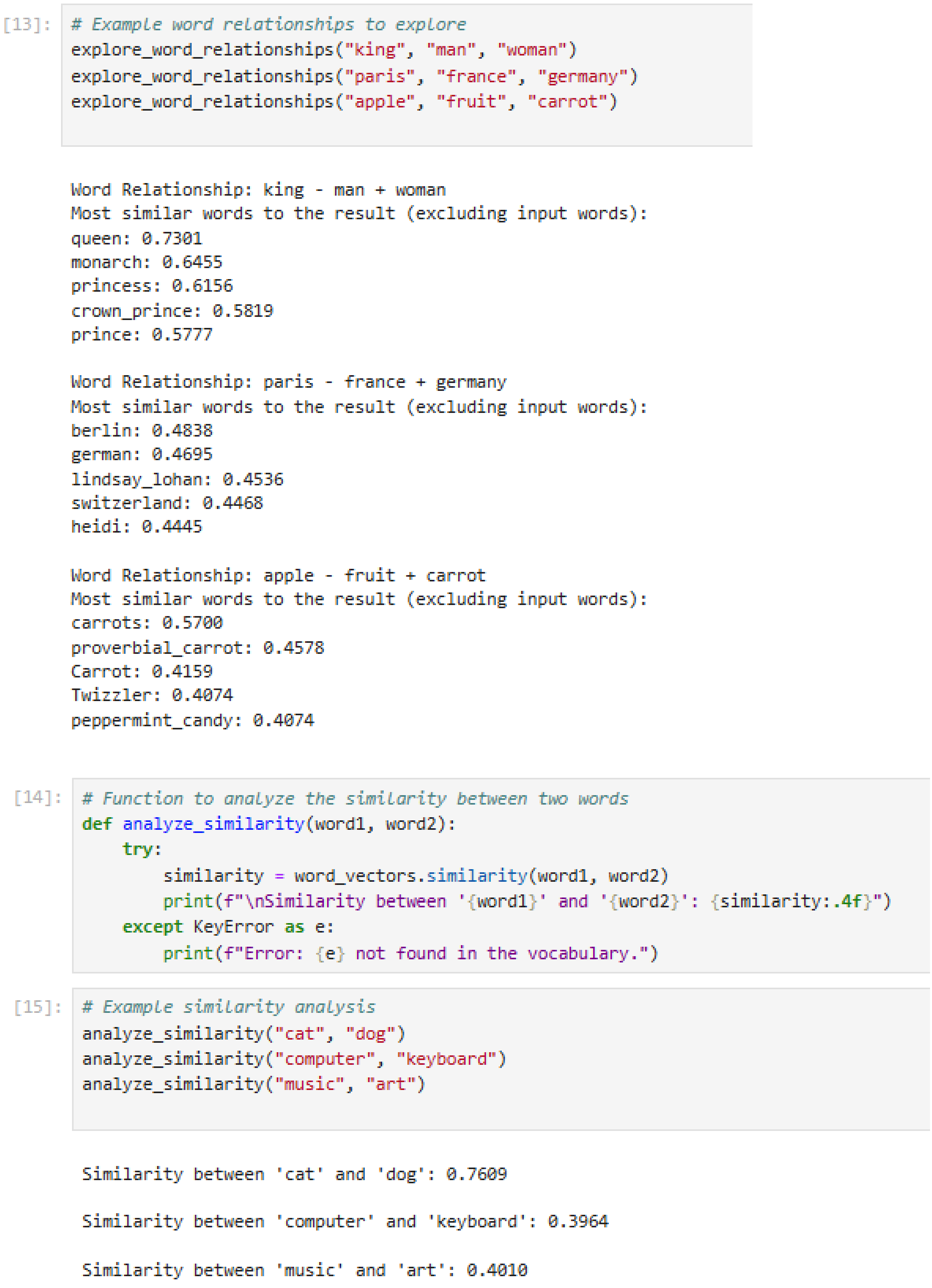
Explore pre-trained word vectors. Explore word relationships using vector arithmetic. Perform arithmetic operations and analyze results.



```
[ ]: # Install necessary libraries
     !pip install gensim numpy

[2]: # Import libraries
     import gensim.downloader as api

[3]: import numpy as np
     from numpy.linalg import norm

[4]: # Load pre-trained word vectors
     print("Loading pre-trained word vectors...")
     word_vectors = api.load("word2vec-google-news-300")
```



```
[12]: # Function to perform vector arithmetic and find similar words
      def explore_word_relationships(word1, word2, word3):
          try:
              # Get vectors for the input words
              vec1 = word_vectors[word1]
              vec2 = word_vectors[word2]
              vec3 = word_vectors[word3]

              # Perform vector arithmetic: word1 - word2 + word3
              result_vector = vec1 - vec2 + vec3

              # Find the most similar words to the resulting vector
              similar_words = word_vectors.similar_by_vector(result_vector, topn=10)

              # Exclude input words from the results
              input_words = {word1, word2, word3}
              filtered_words = [(word, similarity) for word, similarity in similar_words if word not in input_words]

              print(f"\nWord Relationship: {word1} - {word2} + {word3}")
              print("Most similar words to the result (excluding input words):")
              for word, similarity in filtered_words[:5]: # Show top 5 results
                  print(f"{word}: {similarity:.4f}")

          except KeyError as e:
              print(f"Error: {e} not found in the vocabulary.")
```



[13]:

```
# Example word relationships to explore
explore_word_relationships("king", "man", "woman")
explore_word_relationships("paris", "france", "germany")
explore_word_relationships("apple", "fruit", "carrot")
```

```
Word Relationship: king - man + woman
Most similar words to the result (excluding input words):
queen: 0.7301
monarch: 0.6455
princess: 0.6156
crown_prince: 0.5819
prince: 0.5777

Word Relationship: paris - france + germany
Most similar words to the result (excluding input words):
berlin: 0.4838
german: 0.4695
lindsay_lohan: 0.4536
switzerland: 0.4468
heidi: 0.4445

Word Relationship: apple - fruit + carrot
Most similar words to the result (excluding input words):
carrots: 0.5700
proverbial_carrot: 0.4578
Carrot: 0.4159
Twizzler: 0.4074
peppermint_candy: 0.4074
```

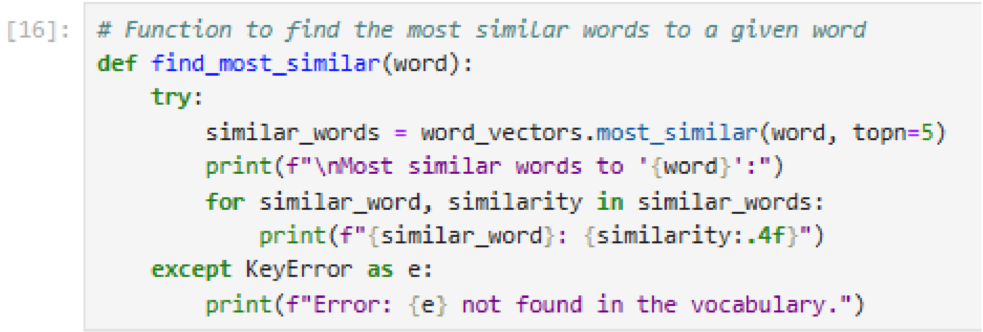
[14]:

```
# Function to analyze the similarity between two words
def analyze_similarity(word1, word2):
    try:
        similarity = word_vectors.similarity(word1, word2)
        print(f"\nSimilarity between '{word1}' and '{word2}': {similarity:.4f}")
    except KeyError as e:
        print(f"Error: {e} not found in the vocabulary.")
```

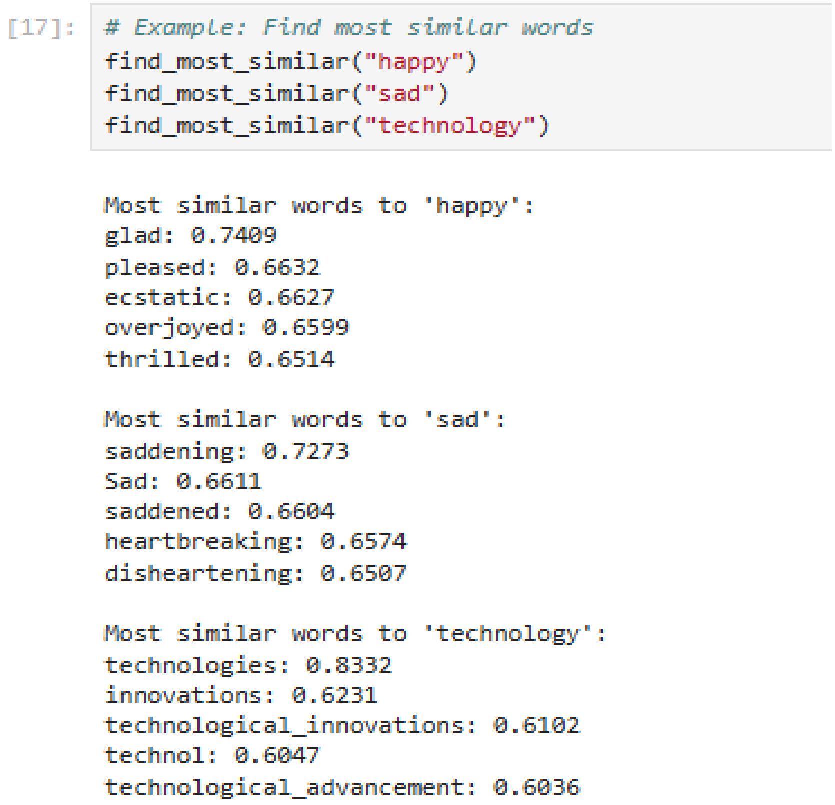
[15]:

```
# Example similarity analysis
analyze_similarity("cat", "dog")
analyze_similarity("computer", "keyboard")
analyze_similarity("music", "art")
```

```
Similarity between 'cat' and 'dog': 0.7609

Similarity between 'computer' and 'keyboard': 0.3964

Similarity between 'music' and 'art': 0.4010
```



```
[16]: # Function to find the most similar words to a given word
      def find_most_similar(word):
          try:
              similar_words = word_vectors.most_similar(word, topn=5)
              print(f"\nMost similar words to '{word}':")
              for similar_word, similarity in similar_words:
                  print(f"{similar_word}: {similarity:.4f}")
          except KeyError as e:
              print(f"Error: {e} not found in the vocabulary.")
```

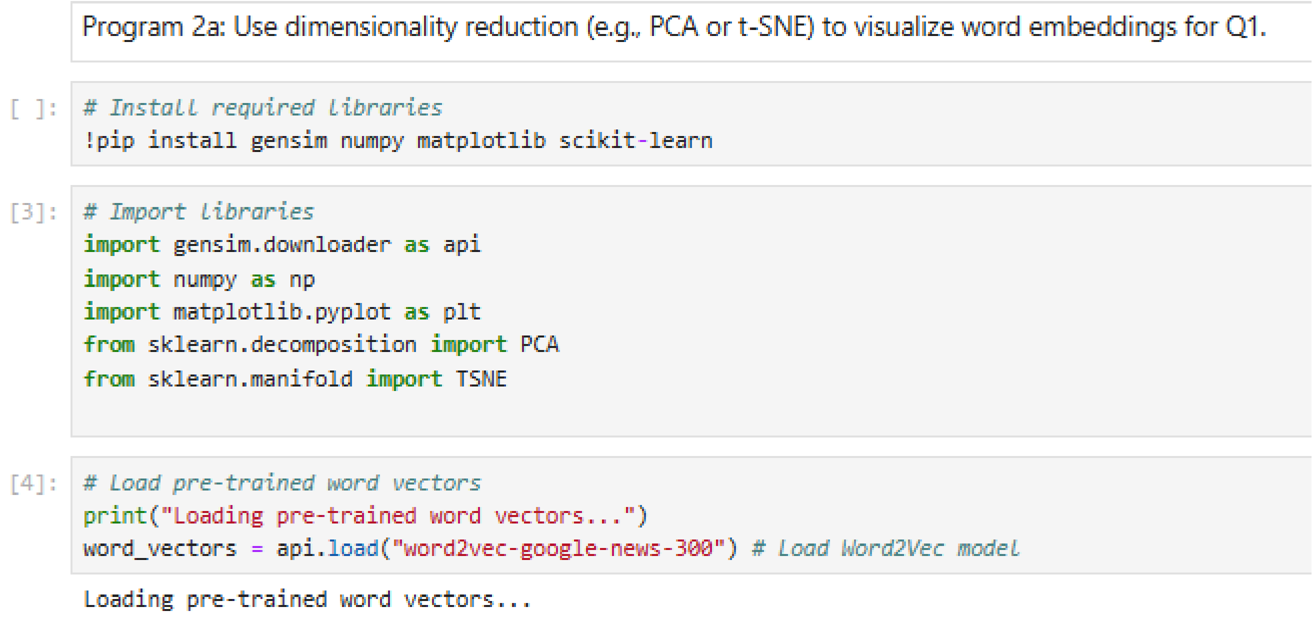


```
[17]: # Example: Find most similar words
      find_most_similar("happy")
      find_most_similar("sad")
      find_most_similar("technology")
```

```
Most similar words to 'happy':
glad: 0.7409
pleased: 0.6632
ecstatic: 0.6627
overjoyed: 0.6599
thrilled: 0.6514

Most similar words to 'sad':
saddening: 0.7273
Sad: 0.6611
saddened: 0.6604
heartbreaking: 0.6574
disheartening: 0.6507

Most similar words to 'technology':
technologies: 0.8332
innovations: 0.6231
technological_innovations: 0.6102
technol: 0.6047
technological_advancement: 0.6036
```

Experiment No. 2:

Use dimensionality reduction (e.g., PCA or t-SNE) to visualize word embeddings for Q 1. Select 10 words from a specific domain (e.g., sports, technology) and visualize their embeddings. Analyze clusters and relationships. Generate contextually rich outputs using embeddings. Write a program to generate 5 semantically similar words for a given input.



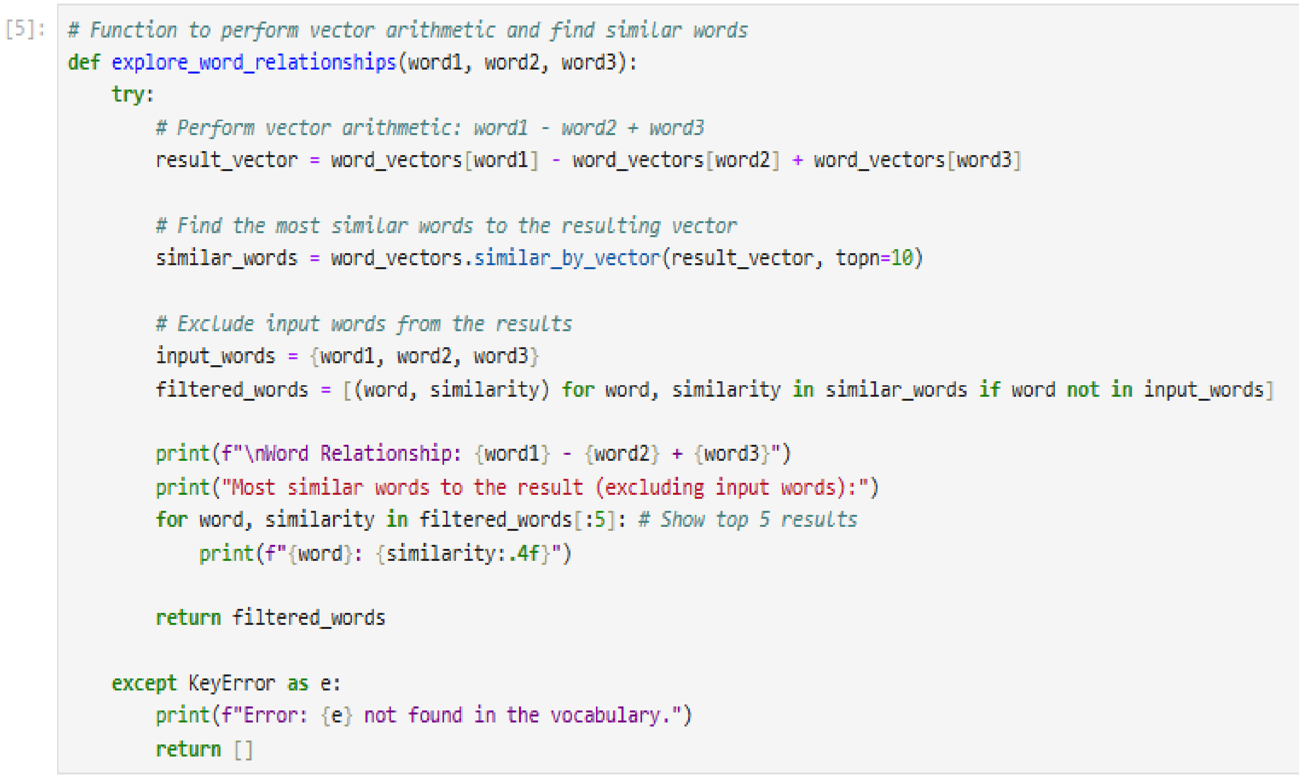
Program 2a: Use dimensionality reduction (e.g., PCA or t-SNE) to visualize word embeddings for Q1.

```
[ ]: # Install required libraries
     !pip install gensim numpy matplotlib scikit-learn
```

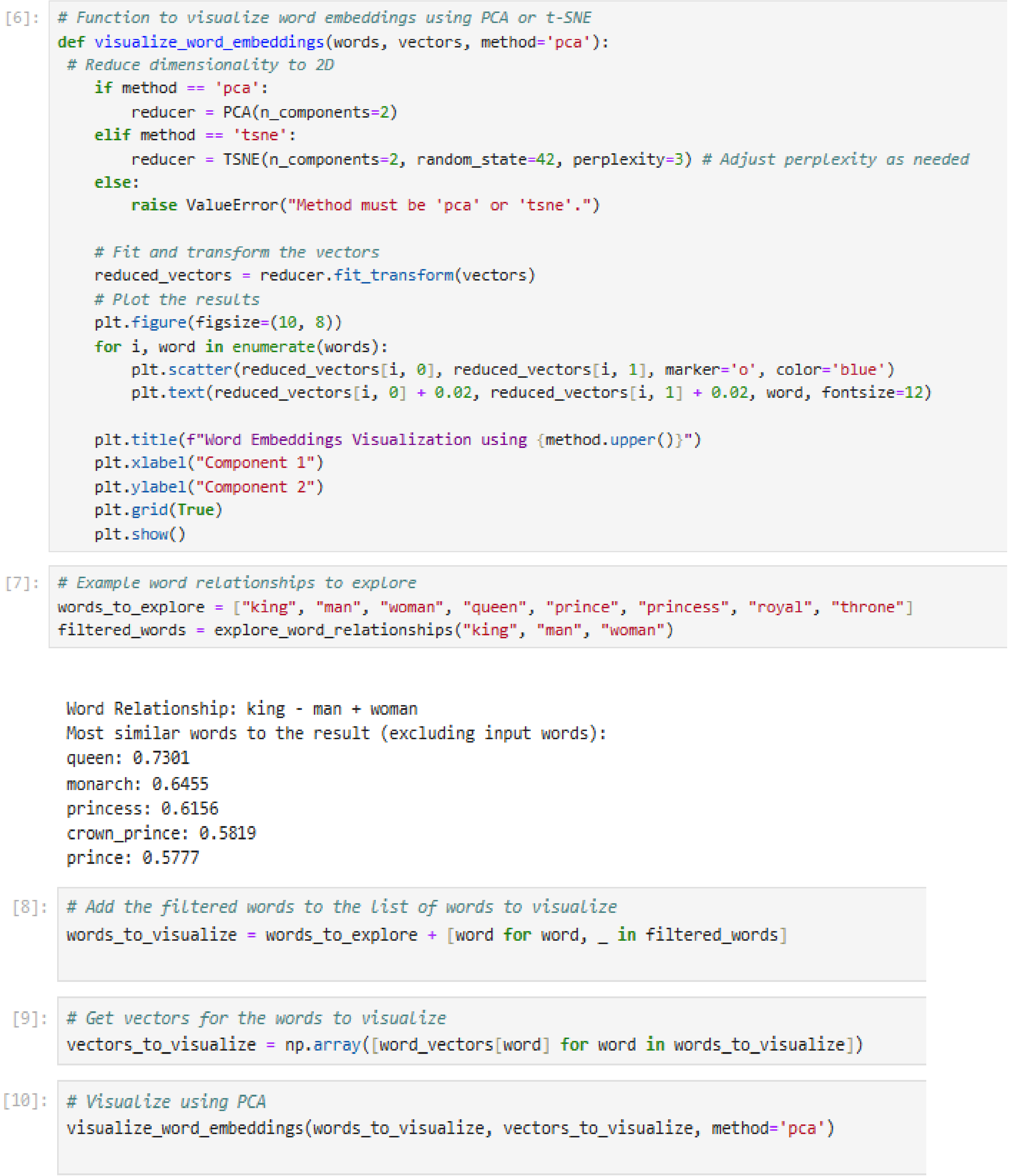
```
[3]: # Import libraries
     import gensim.downloader as api
     import numpy as np
     import matplotlib.pyplot as plt
     from sklearn.decomposition import PCA
     from sklearn.manifold import TSNE
```

```
[4]: # Load pre-trained word vectors
     print("Loading pre-trained word vectors...")
     word_vectors = api.load("word2vec-google-news-300") # Load Word2Vec model
```

```
Loading pre-trained word vectors...
```



```
[5]: # Function to perform vector arithmetic and find similar words
     def explore_word_relationships(word1, word2, word3):
         try:
             # Perform vector arithmetic: word1 - word2 + word3
             result_vector = word_vectors[word1] - word_vectors[word2] + word_vectors[word3]

             # Find the most similar words to the resulting vector
             similar_words = word_vectors.similar_by_vector(result_vector, topn=10)

             # Exclude input words from the results
             input_words = {word1, word2, word3}
             filtered_words = [(word, similarity) for word, similarity in similar_words if word not in input_words]

             print(f"\nWord Relationship: {word1} - {word2} + {word3}")
             print("Most similar words to the result (excluding input words):")
             for word, similarity in filtered_words[:5]: # Show top 5 results
                 print(f"{word}: {similarity:.4f}")

             return filtered_words

         except KeyError as e:
             print(f"Error: {e} not found in the vocabulary.")
             return []
```



```
[6]: # Function to visualize word embeddings using PCA or t-SNE
     def visualize_word_embeddings(words, vectors, method='pca'):
      # Reduce dimensionality to 2D
         if method == 'pca':
             reducer = PCA(n_components=2)
         elif method == 'tsne':
             reducer = TSNE(n_components=2, random_state=42, perplexity=3) # Adjust perplexity as needed
         else:
             raise ValueError("Method must be 'pca' or 'tsne'.")

         # Fit and transform the vectors
         reduced_vectors = reducer.fit_transform(vectors)
         # Plot the results
         plt.figure(figsize=(10, 8))
         for i, word in enumerate(words):
             plt.scatter(reduced_vectors[i, 0], reduced_vectors[i, 1], marker='o', color='blue')
             plt.text(reduced_vectors[i, 0] + 0.02, reduced_vectors[i, 1] + 0.02, word, fontsize=12)

         plt.title(f"Word Embeddings Visualization using {method.upper()}")
         plt.xlabel("Component 1")
         plt.ylabel("Component 2")
         plt.grid(True)
         plt.show()
```

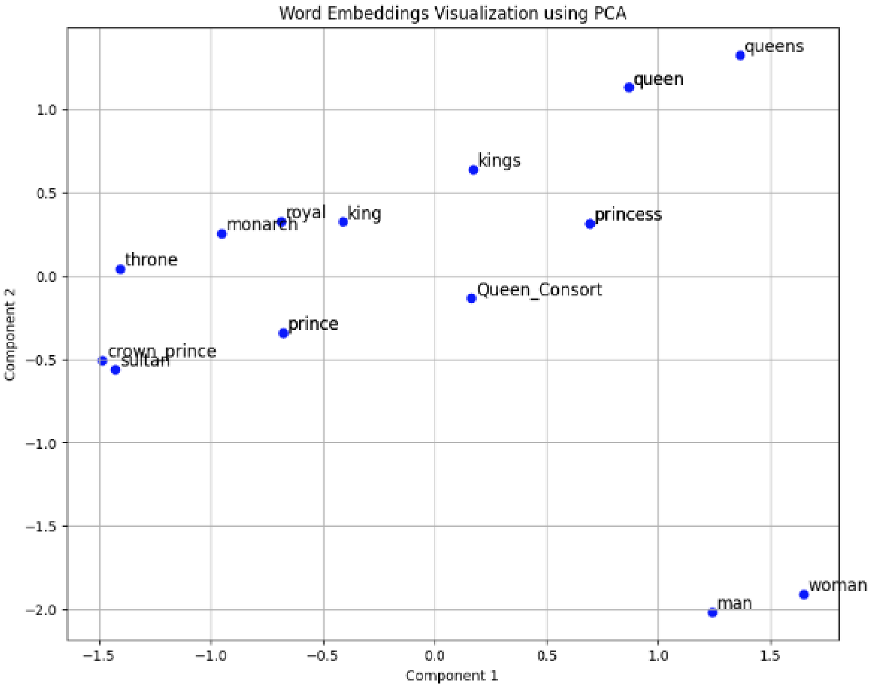
```
[7]: # Example word relationships to explore
     words_to_explore = ["king", "man", "woman", "queen", "prince", "princess", "royal", "throne"]
     filtered_words = explore_word_relationships("king", "man", "woman")
```

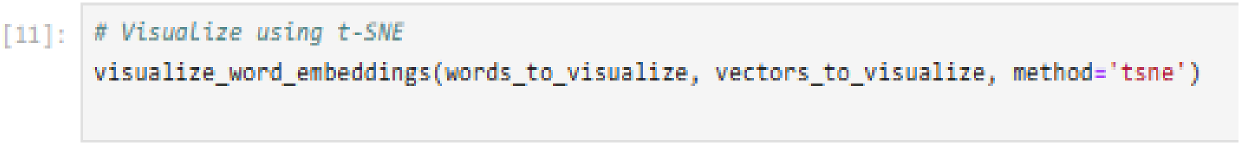
```
Word Relationship: king - man + woman
Most similar words to the result (excluding input words):
queen: 0.7301
monarch: 0.6455
princess: 0.6156
crown_prince: 0.5819
prince: 0.5777
```

```
[8]: # Add the filtered words to the list of words to visualize
     words_to_visualize = words_to_explore + [word for word, _ in filtered_words]
```

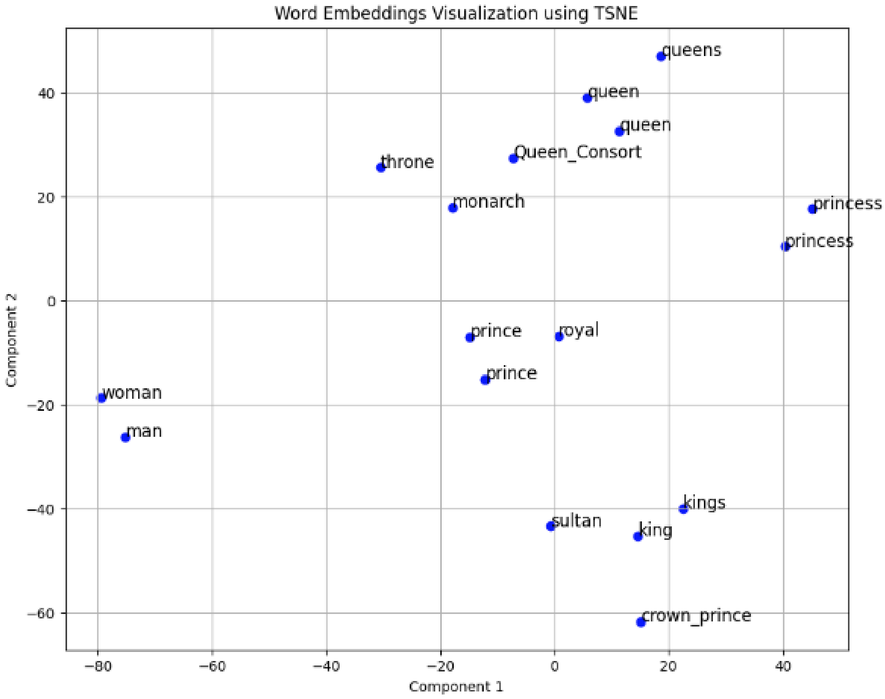
```
[9]: # Get vectors for the words to visualize
     vectors_to_visualize = np.array([word_vectors[word] for word in words_to_visualize])
```

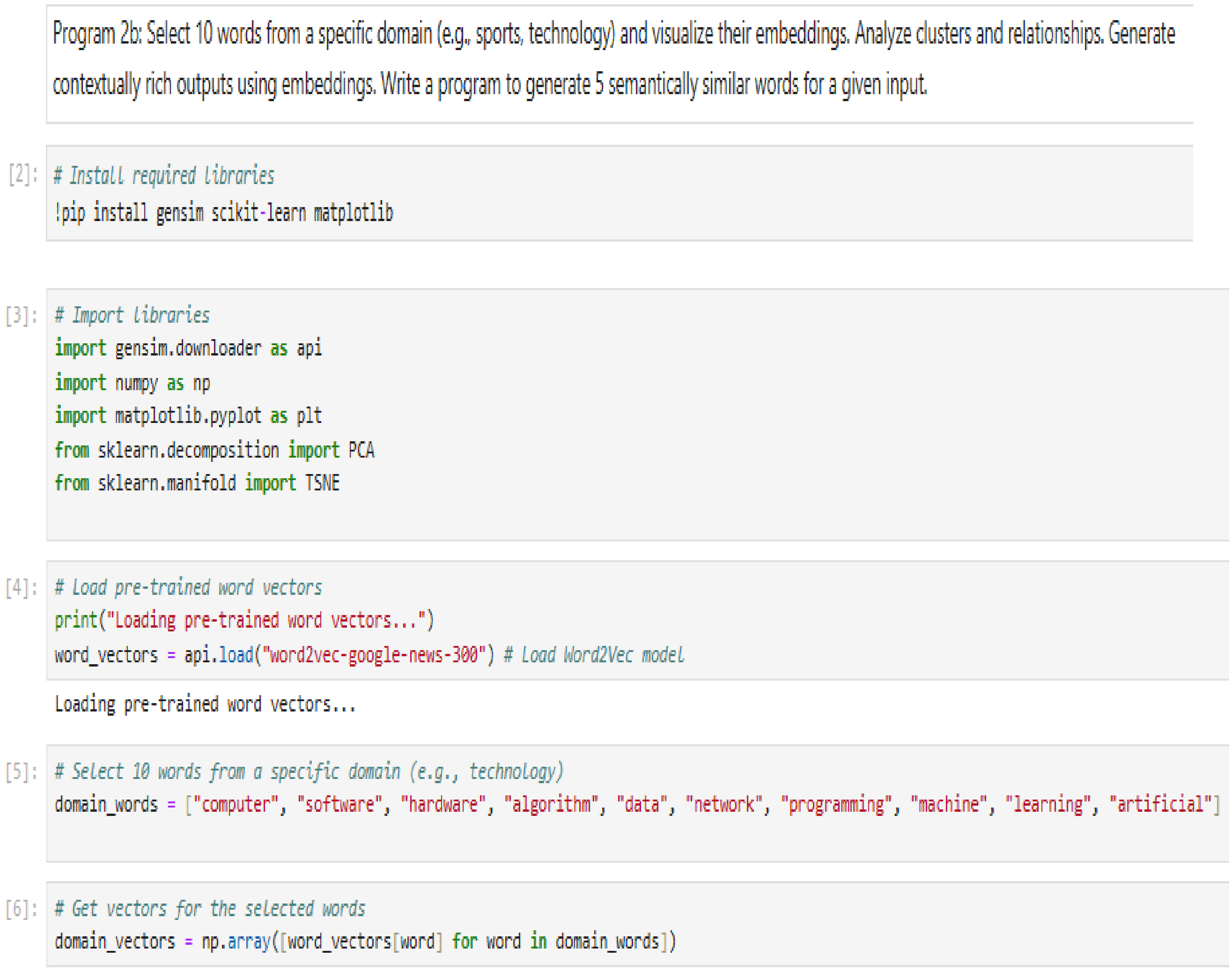
```
[10]: # Visualize using PCA
      visualize_word_embeddings(words_to_visualize, vectors_to_visualize, method='pca')
```





```
[11]: # Visualize using t-SNE
      visualize_word_embeddings(words_to_visualize, vectors_to_visualize, method='tsne')
```





Program 2b: Select 10 words from a specific domain (e.g., sports, technology) and visualize their embeddings. Analyze clusters and relationships. Generate contextually rich outputs using embeddings. Write a program to generate 5 semantically similar words for a given input.

```
[2]: # Install required libraries
     !pip install gensim scikit-learn matplotlib
```

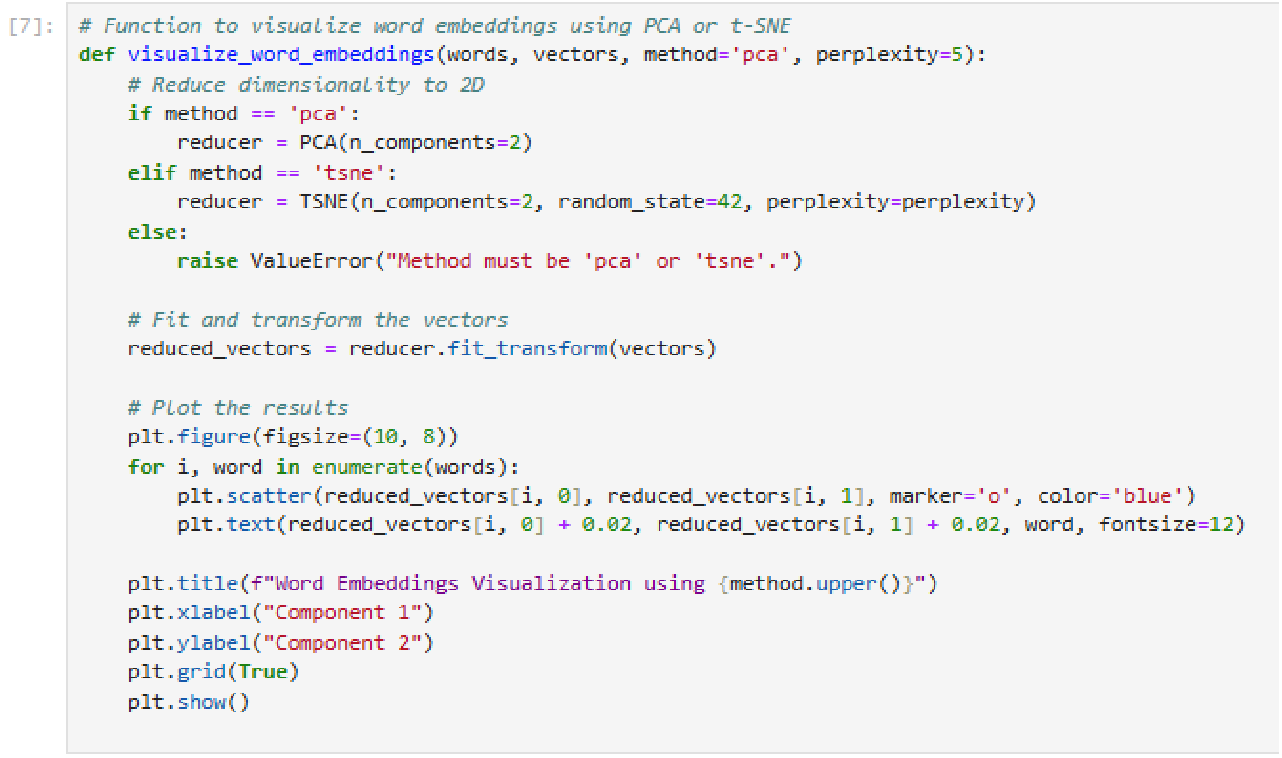
```
[3]: # Import libraries
     import gensim.downloader as api
     import numpy as np
     import matplotlib.pyplot as plt
     from sklearn.decomposition import PCA
     from sklearn.manifold import TSNE
```

```
[4]: # Load pre-trained word vectors
     print("Loading pre-trained word vectors...")
     word_vectors = api.load("word2vec-google-news-300") # Load Word2Vec model
```

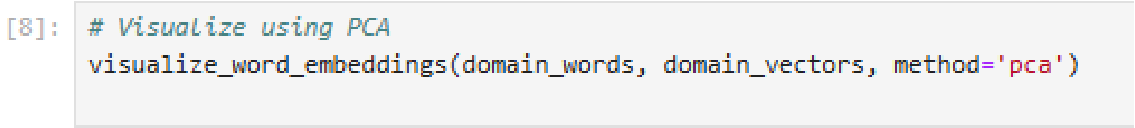
```
Loading pre-trained word vectors...
```

```
[5]: # Select 10 words from a specific domain (e.g., technology)
     domain_words = ["computer", "software", "hardware", "algorithm", "data", "network", "programming", "machine", "learning", "artificial"]
```

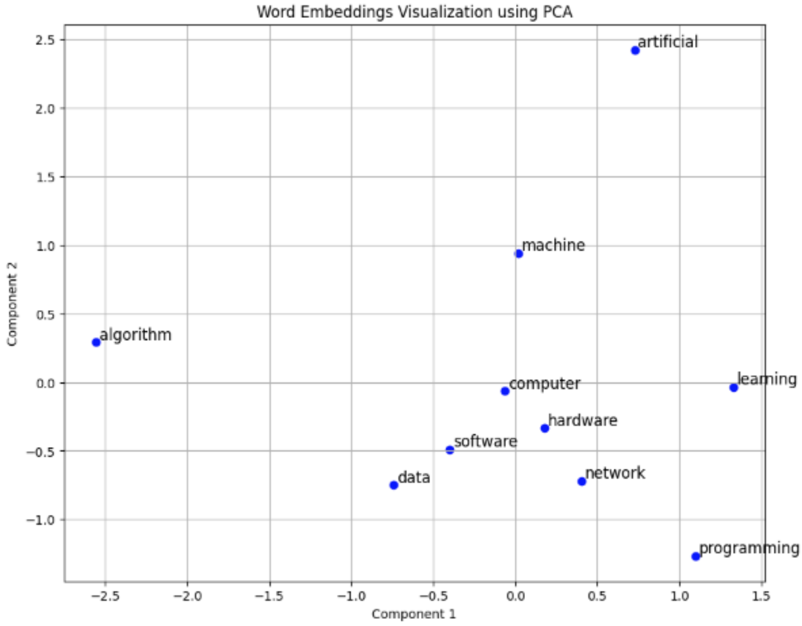
```
[6]: # Get vectors for the selected words
     domain_vectors = np.array([word_vectors[word] for word in domain_words])
```



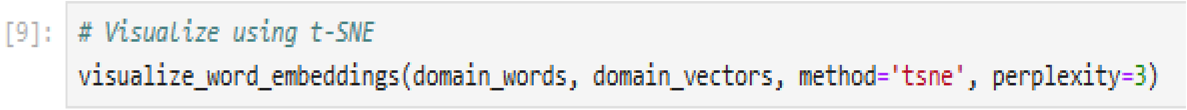
```
[7]: # Function to visualize word embeddings using PCA or t-SNE
     def visualize_word_embeddings(words, vectors, method='pca', perplexity=5):
         # Reduce dimensionality to 2D
         if method == 'pca':
             reducer = PCA(n_components=2)
         elif method == 'tsne':
             reducer = TSNE(n_components=2, random_state=42, perplexity=perplexity)
         else:
             raise ValueError("Method must be 'pca' or 'tsne'.")

         # Fit and transform the vectors
         reduced_vectors = reducer.fit_transform(vectors)

         # Plot the results
         plt.figure(figsize=(10, 8))
         for i, word in enumerate(words):
             plt.scatter(reduced_vectors[i, 0], reduced_vectors[i, 1], marker='o', color='blue')
             plt.text(reduced_vectors[i, 0] + 0.02, reduced_vectors[i, 1] + 0.02, word, fontsize=12)

         plt.title(f"Word Embeddings Visualization using {method.upper()}")
         plt.xlabel("Component 1")
         plt.ylabel("Component 2")
         plt.grid(True)
         plt.show()
```



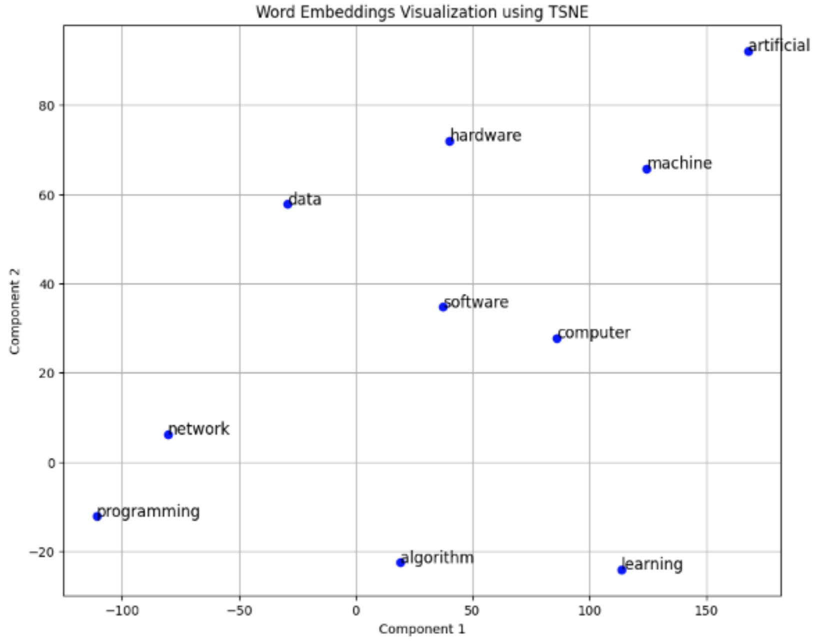
```
[8]: # Visualize using PCA
     visualize_word_embeddings(domain_words, domain_vectors, method='pca')
```



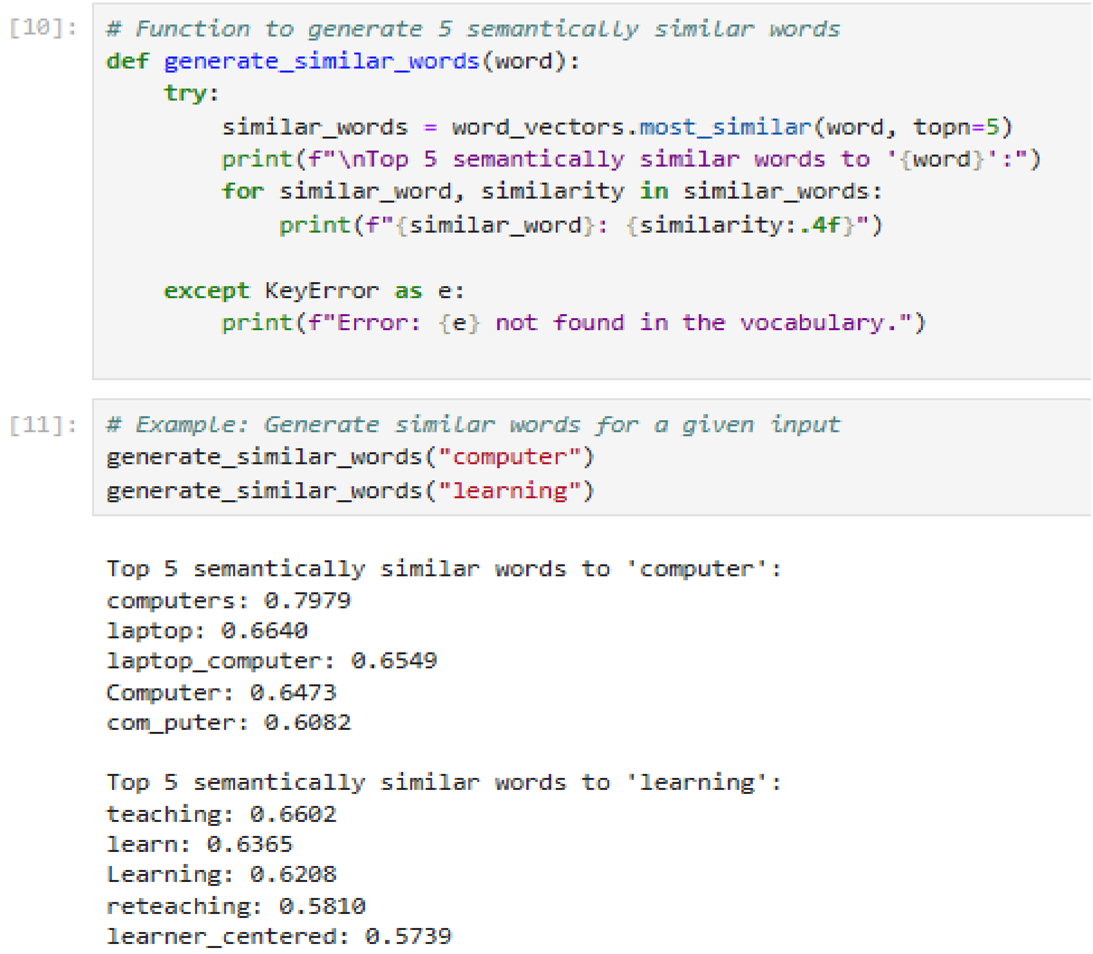
Word Embeddings Visualization using PCA



```
[9]: # Visualize using t-SNE
     visualize_word_embeddings(domain_words, domain_vectors, method='tsne', perplexity=3)
```



Word Embeddings Visualization using TSNE



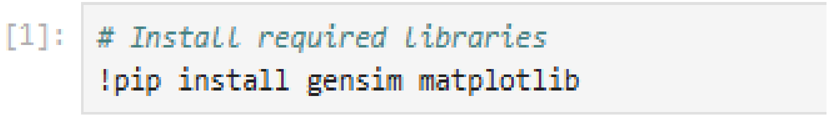
```
[10]: # Function to generate 5 semantically similar words
      def generate_similar_words(word):
          try:
              similar_words = word_vectors.most_similar(word, topn=5)
              print(f"\nTop 5 semantically similar words to '{word}':")
              for similar_word, similarity in similar_words:
                  print(f"{similar_word}: {similarity:.4f}")

          except KeyError as e:
              print(f"Error: {e} not found in the vocabulary.")
```

```
[11]: # Example: Generate similar words for a given input
      generate_similar_words("computer")
      generate_similar_words("learning")
```

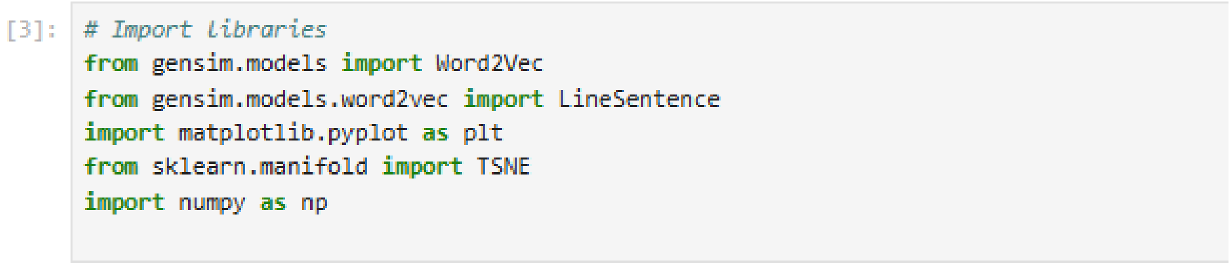
```
Top 5 semantically similar words to 'computer':
computers: 0.7979
laptop: 0.6640
laptop_computer: 0.6549
Computer: 0.6473
com_puter: 0.6082

Top 5 semantically similar words to 'learning':
teaching: 0.6602
learn: 0.6365
Learning: 0.6208
reteaching: 0.5810
learner_centered: 0.5739
```

Experiment No.3:

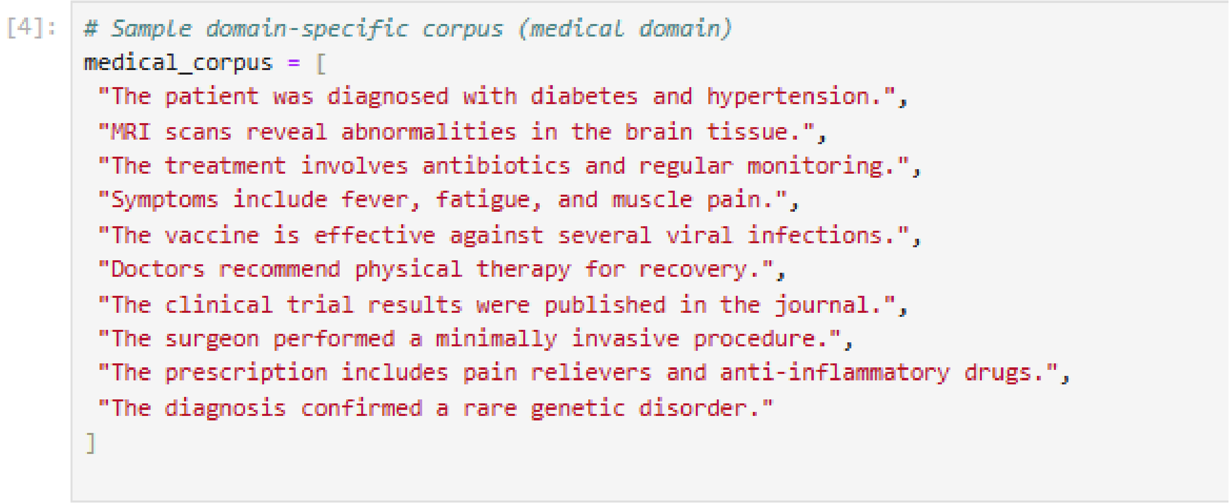
Train a custom Word2Vec model on a small dataset. Train embeddings on a domain-specific corpus (e.g., legal, medical) and analyze how embeddings capture domain-specific semantics.



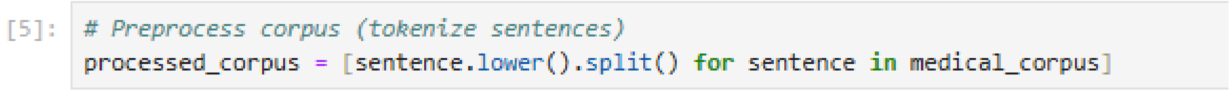
```
[1]: # Install required libraries
     !pip install gensim matplotlib
```



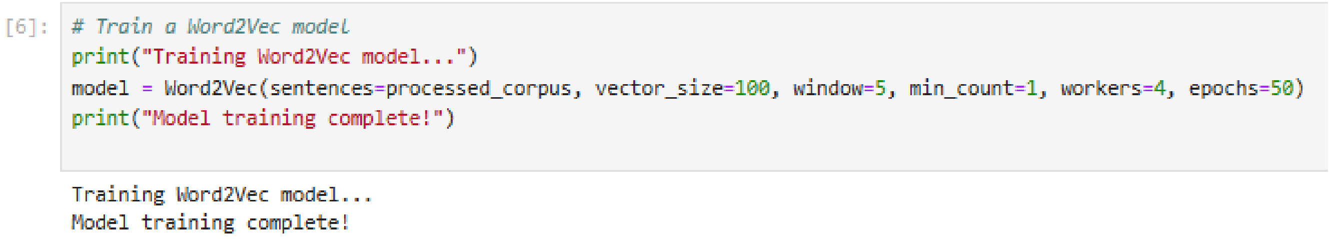
```
[3]: # Import libraries
     from gensim.models import Word2Vec
     from gensim.models.word2vec import LineSentence
     import matplotlib.pyplot as plt
     from sklearn.manifold import TSNE
     import numpy as np
```



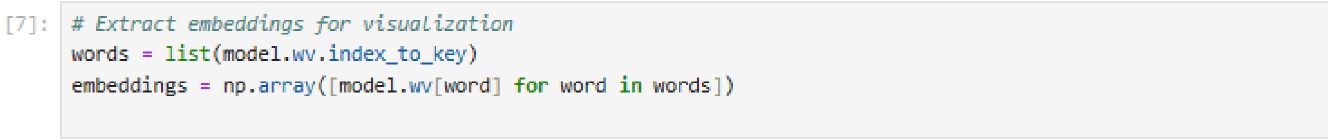
```
[4]: # Sample domain-specific corpus (medical domain)
     medical_corpus = [
      "The patient was diagnosed with diabetes and hypertension.",
      "MRI scans reveal abnormalities in the brain tissue.",
      "The treatment involves antibiotics and regular monitoring.",
      "Symptoms include fever, fatigue, and muscle pain.",
      "The vaccine is effective against several viral infections.",
      "Doctors recommend physical therapy for recovery.",
      "The clinical trial results were published in the journal.",
      "The surgeon performed a minimally invasive procedure.",
      "The prescription includes pain relievers and anti-inflammatory drugs.",
      "The diagnosis confirmed a rare genetic disorder."
     ]
```



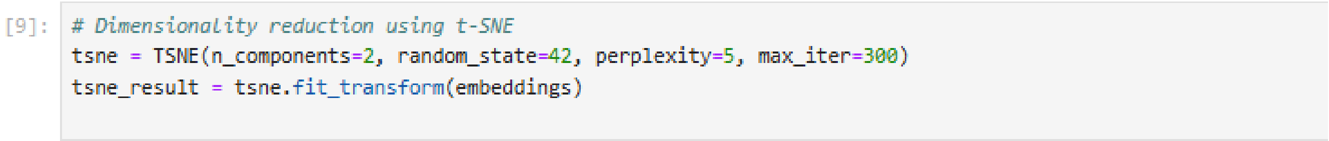
```
[5]: # Preprocess corpus (tokenize sentences)
     processed_corpus = [sentence.lower().split() for sentence in medical_corpus]
```



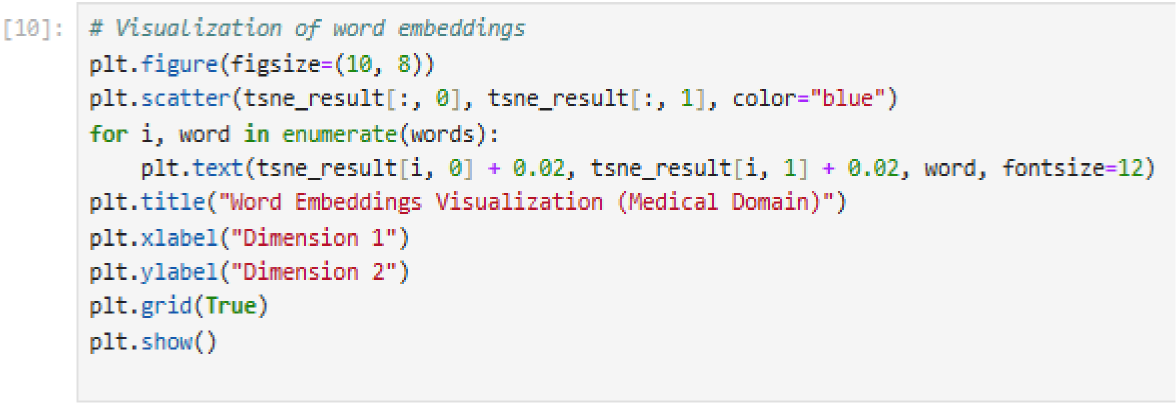
```
[6]: # Train a Word2Vec model
     print("Training Word2Vec model...")
     model = Word2Vec(sentences=processed_corpus, vector_size=100, window=5, min_count=1, workers=4, epochs=50)
     print("Model training complete!")

     Training Word2Vec model...
     Model training complete!
```



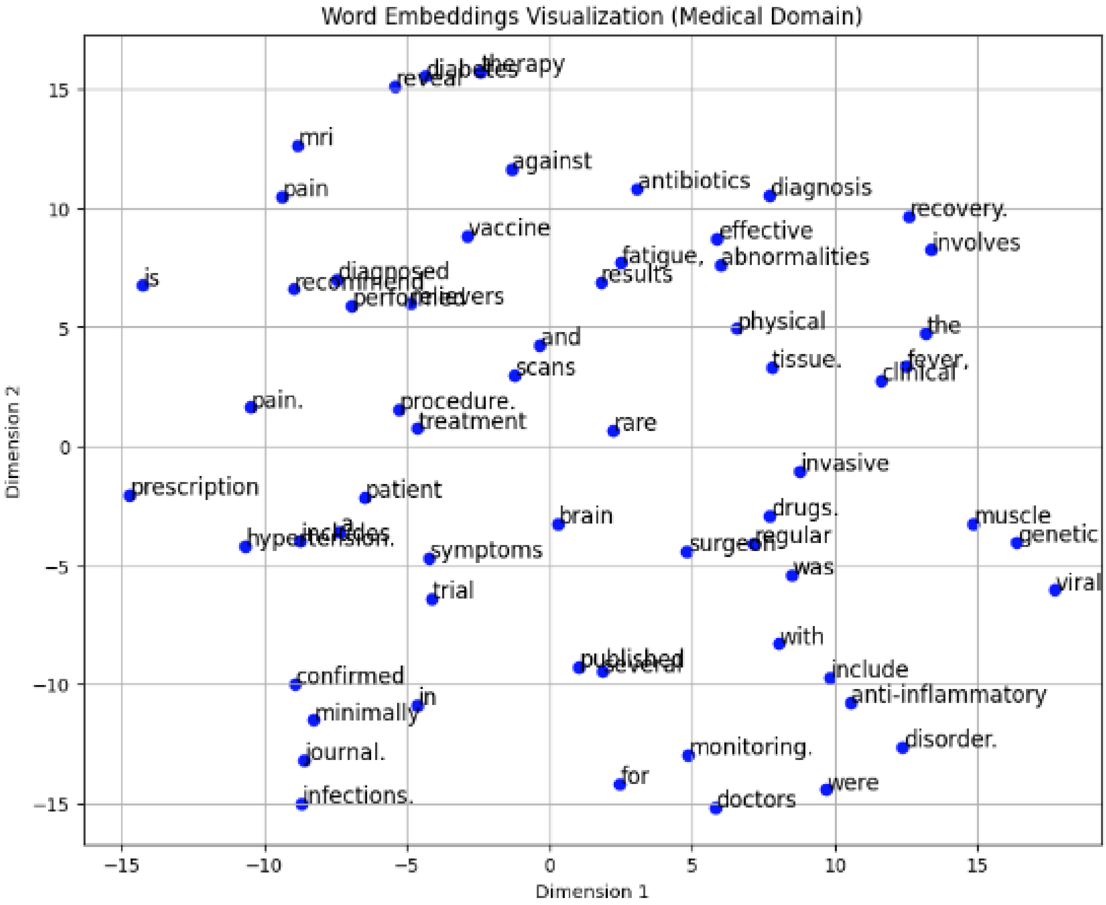
```
[7]: # Extract embeddings for visualization
     words = list(model.wv.index_to_key)
     embeddings = np.array([model.wv[word] for word in words])
```

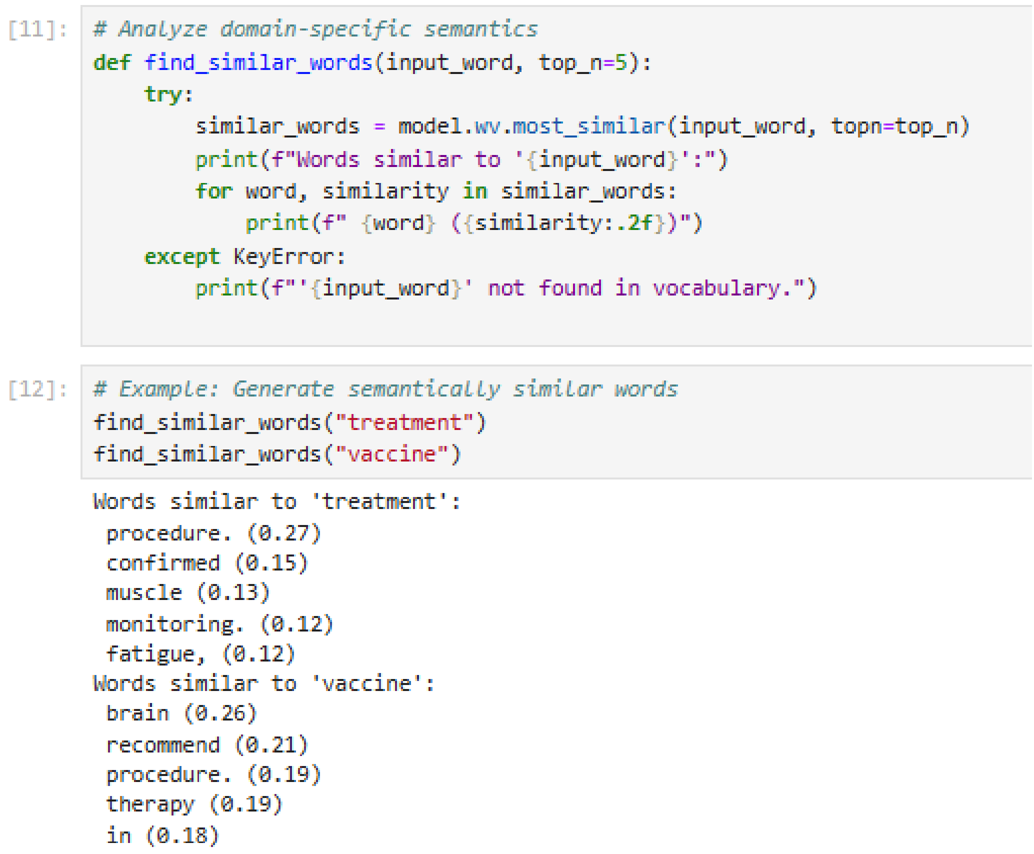


```
[9]: # Dimensionality reduction using t-SNE
     tsne = TSNE(n_components=2, random_state=42, perplexity=5, max_iter=300)
     tsne_result = tsne.fit_transform(embeddings)
```



```
[10]: # Visualization of word embeddings
      plt.figure(figsize=(10, 8))
      plt.scatter(tsne_result[:, 0], tsne_result[:, 1], color="blue")
      for i, word in enumerate(words):
          plt.text(tsne_result[i, 0] + 0.02, tsne_result[i, 1] + 0.02, word, fontsize=12)
      plt.title("Word Embeddings Visualization (Medical Domain)")
      plt.xlabel("Dimension 1")
      plt.ylabel("Dimension 2")
      plt.grid(True)
      plt.show()
```





[11]:

```
# Analyze domain-specific semantics
def find_similar_words(input_word, top_n=5):
    try:
        similar_words = model.wv.most_similar(input_word, topn=top_n)
        print(f"Words similar to '{input_word}':")
        for word, similarity in similar_words:
            print(f" {word} ({similarity:.2f})")
    except KeyError:
        print(f"'{input_word}' not found in vocabulary.")
```

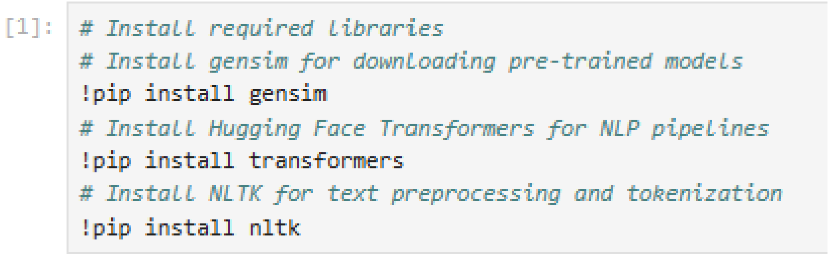
[12]:

```
# Example: Generate semantically similar words
find_similar_words("treatment")
find_similar_words("vaccine")
```

```
Words similar to 'treatment':
 procedure. (0.27)
 confirmed (0.15)
 muscle (0.13)
 monitoring. (0.12)
 fatigue, (0.12)
Words similar to 'vaccine':
 brain (0.26)
 recommend (0.21)
 procedure. (0.19)
 therapy (0.19)
 in (0.18)
```

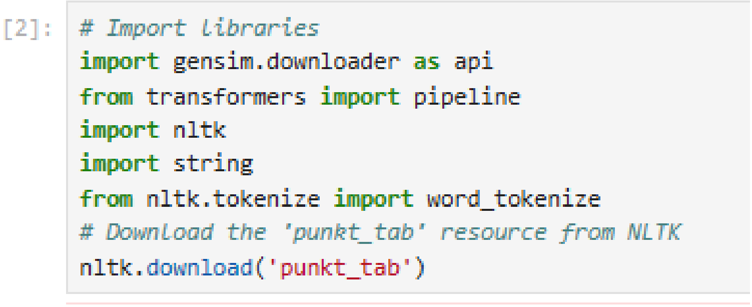
Experiment No.4:

Use word embeddings to improve prompts for Generative AI model. Retrieve similar words using word embeddings. Use the similar words to enrich a GenAI prompt. Use the AI model to generate responses for the original and enriched prompts. Compare the outputs in terms of detail.



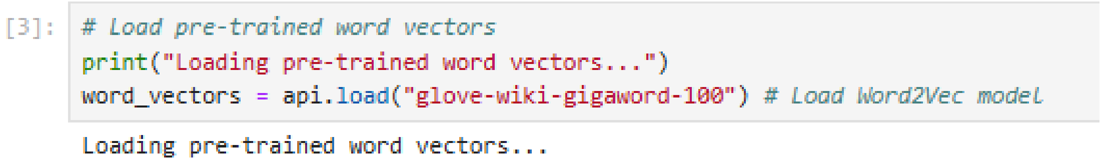
[1]:

```
# Install required libraries
# Install gensim for downloading pre-trained models
!pip install gensim
# Install Hugging Face Transformers for NLP pipelines
!pip install transformers
# Install NLTK for text preprocessing and tokenization
!pip install nltk
```



[2]:

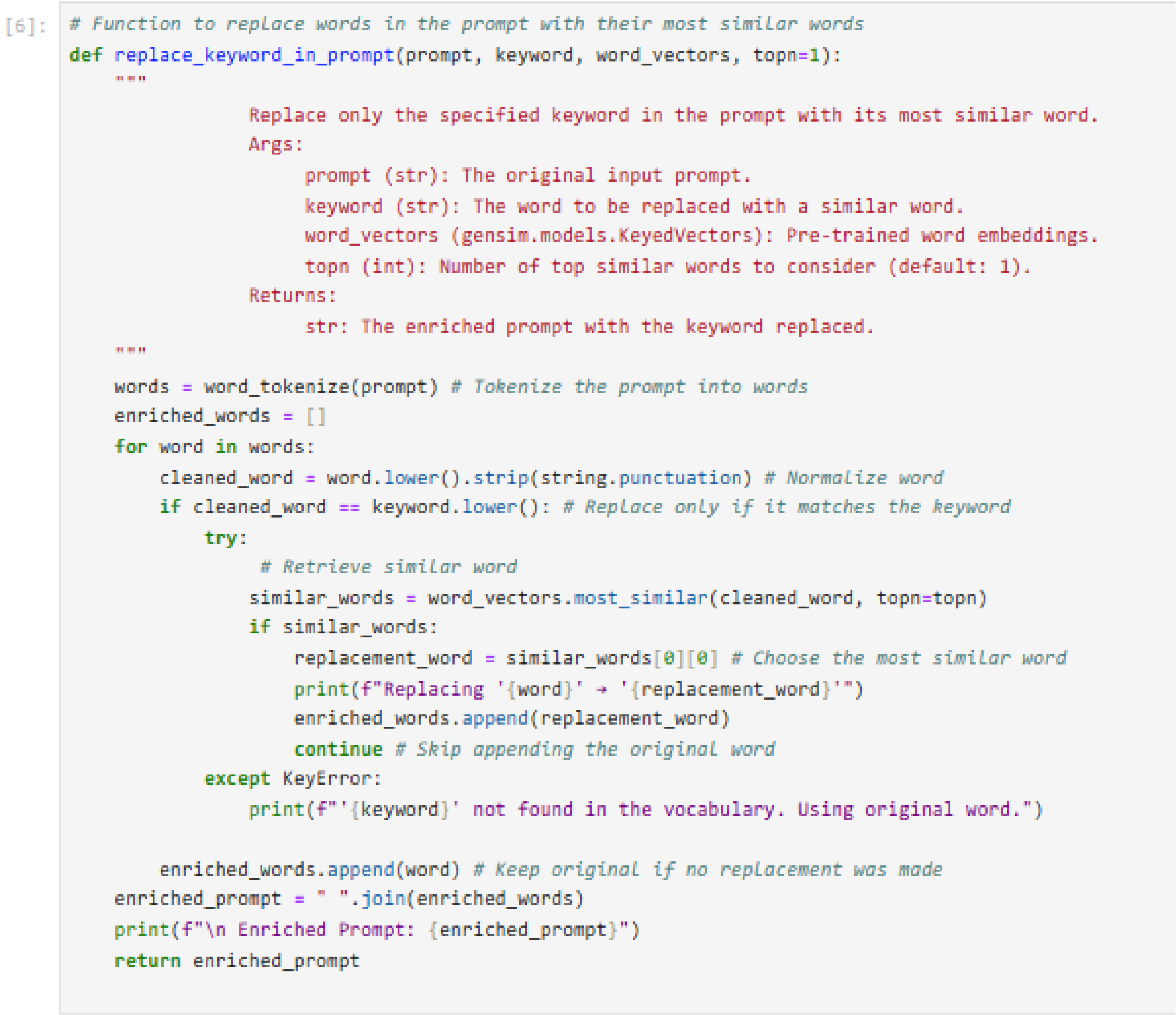
```
# Import libraries
import gensim.downloader as api
from transformers import pipeline
import nltk
import string
from nltk.tokenize import word_tokenize
# Download the 'punkt_tab' resource from NLTK
nltk.download('punkt_tab')
```



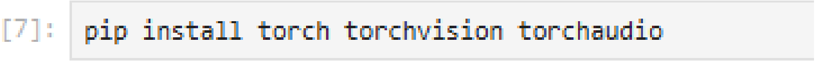
[3]:

```
# Load pre-trained word vectors
print("Loading pre-trained word vectors...")
word_vectors = api.load("glove-wiki-gigaword-100") # Load Word2Vec model
```

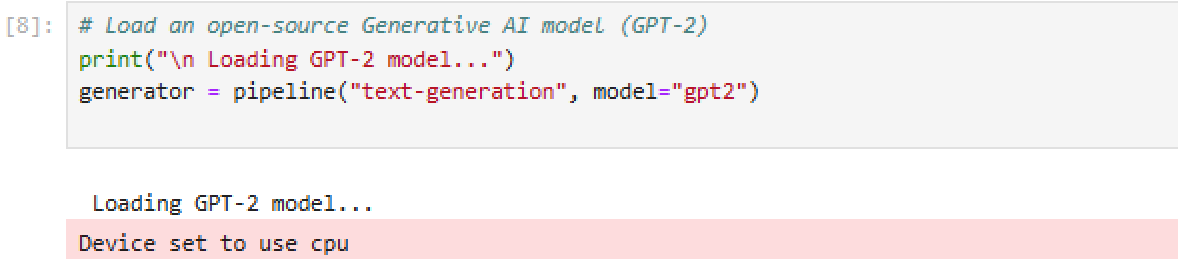
```
Loading pre-trained word vectors...
```



```
[6]: # Function to replace words in the prompt with their most similar words
def replace_keyword_in_prompt(prompt, keyword, word_vectors, topn=1):
    """
                Replace only the specified keyword in the prompt with its most similar word.
                Args:
                     prompt (str): The original input prompt.
                     keyword (str): The word to be replaced with a similar word.
                     word_vectors (gensim.models.KeyedVectors): Pre-trained word embeddings.
                     topn (int): Number of top similar words to consider (default: 1).
                Returns:
                     str: The enriched prompt with the keyword replaced.
    """
    words = word_tokenize(prompt) # Tokenize the prompt into words
    enriched_words = []
    for word in words:
        cleaned_word = word.lower().strip(string.punctuation) # Normalize word
        if cleaned_word == keyword.lower(): # Replace only if it matches the keyword
            try:
                 # Retrieve similar word
                similar_words = word_vectors.most_similar(cleaned_word, topn=topn)
                if similar_words:
                    replacement_word = similar_words[0][0] # Choose the most similar word
                    print(f"Replacing '{word}' → '{replacement_word}'")
                    enriched_words.append(replacement_word)
                    continue # Skip appending the original word
            except KeyError:
                print(f"'{keyword}' not found in the vocabulary. Using original word.")

        enriched_words.append(word) # Keep original if no replacement was made
    enriched_prompt = " ".join(enriched_words)
    print(f"\n Enriched Prompt: {enriched_prompt}")
    return enriched_prompt
```

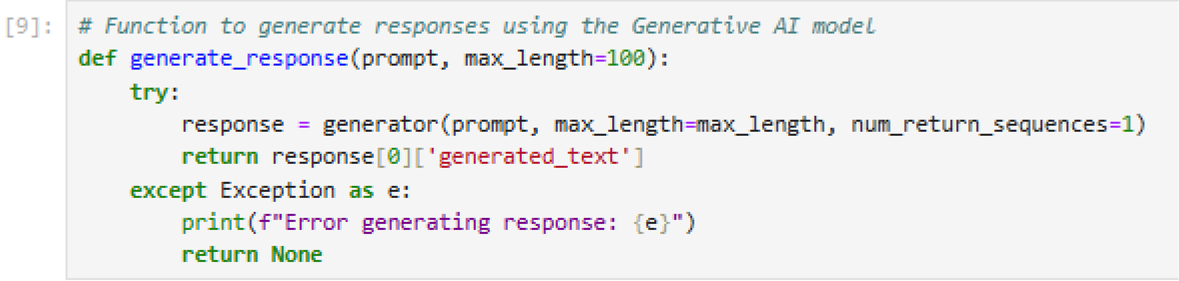


```
[7]: pip install torch torchvision torchaudio
```

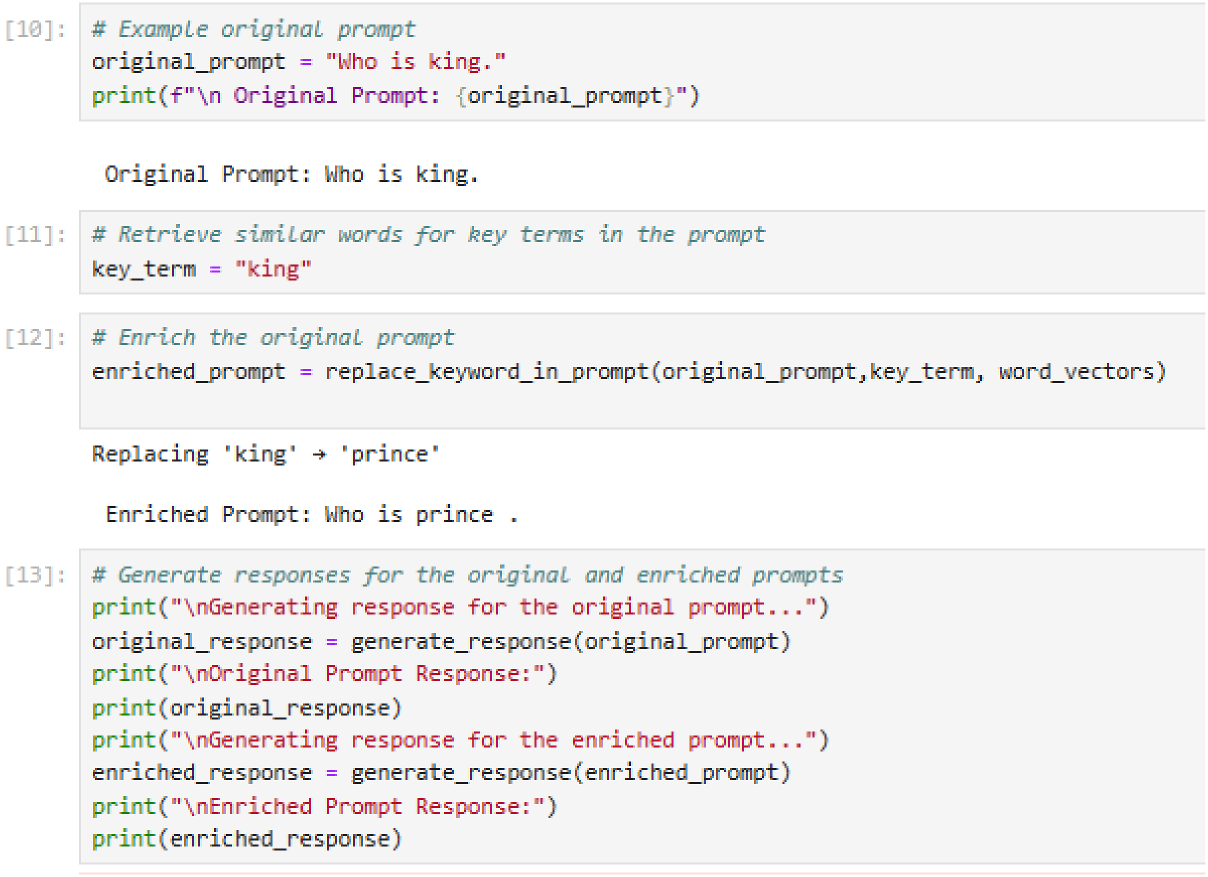


```
[8]: # Load an open-source Generative AI model (GPT-2)
print("\n Loading GPT-2 model...")
generator = pipeline("text-generation", model="gpt2")
```

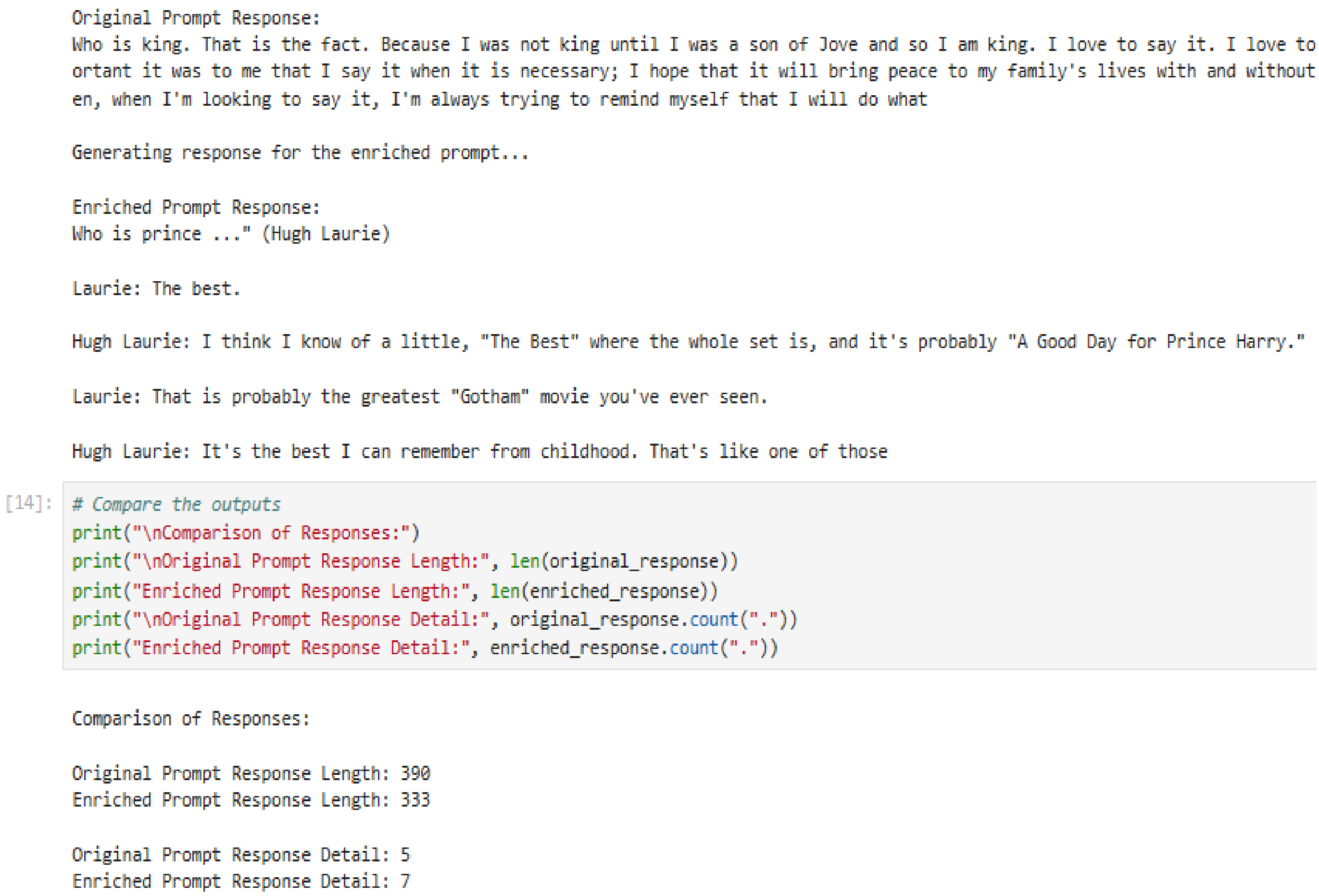
```

 Loading GPT-2 model...
Device set to use cpu
```



```
[9]: # Function to generate responses using the Generative AI model
def generate_response(prompt, max_length=100):
    try:
        response = generator(prompt, max_length=max_length, num_return_sequences=1)
        return response[0]['generated_text']
    except Exception as e:
        print(f"Error generating response: {e}")
        return None
```



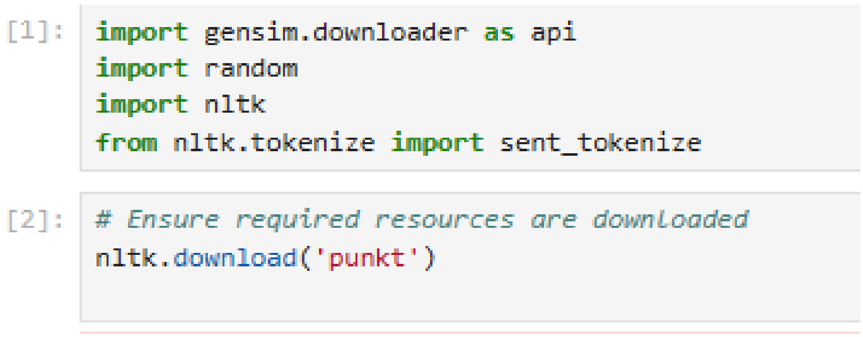
```
[10]: # Example original prompt
      original_prompt = "Who is king."
      print(f"\n Original Prompt: {original_prompt}")

       Original Prompt: Who is king.

[11]: # Retrieve similar words for key terms in the prompt
      key_term = "king"

[12]: # Enrich the original prompt
      enriched_prompt = replace_keyword_in_prompt(original_prompt,key_term, word_vectors)

      Replacing 'king' → 'prince'

       Enriched Prompt: Who is prince .

[13]: # Generate responses for the original and enriched prompts
      print("\nGenerating response for the original prompt...")
      original_response = generate_response(original_prompt)
      print("\nOriginal Prompt Response:")
      print(original_response)
      print("\nGenerating response for the enriched prompt...")
      enriched_response = generate_response(enriched_prompt)
      print("\nEnriched Prompt Response:")
      print(enriched_response)
```



```
Original Prompt Response:
Who is king. That is the fact. Because I was not king until I was a son of Jove and so I am king. I love to say it. I love to
ortant it was to me that I say it when it is necessary; I hope that it will bring peace to my family's lives with and without
en, when I'm looking to say it, I'm always trying to remind myself that I will do what

Generating response for the enriched prompt...

Enriched Prompt Response:
Who is prince ..." (Hugh Laurie)

Laurie: The best.

Hugh Laurie: I think I know of a little, "The Best" where the whole set is, and it's probably "A Good Day for Prince Harry."

Laurie: That is probably the greatest "Gotham" movie you've ever seen.

Hugh Laurie: It's the best I can remember from childhood. That's like one of those

[14]: # Compare the outputs
      print("\nComparison of Responses:")
      print("\nOriginal Prompt Response Length:", len(original_response))
      print("Enriched Prompt Response Length:", len(enriched_response))
      print("\nOriginal Prompt Response Detail:", original_response.count("."))
      print("Enriched Prompt Response Detail:", enriched_response.count("."))

Comparison of Responses:

Original Prompt Response Length: 390
Enriched Prompt Response Length: 333

Original Prompt Response Detail: 5
Enriched Prompt Response Detail: 7
```

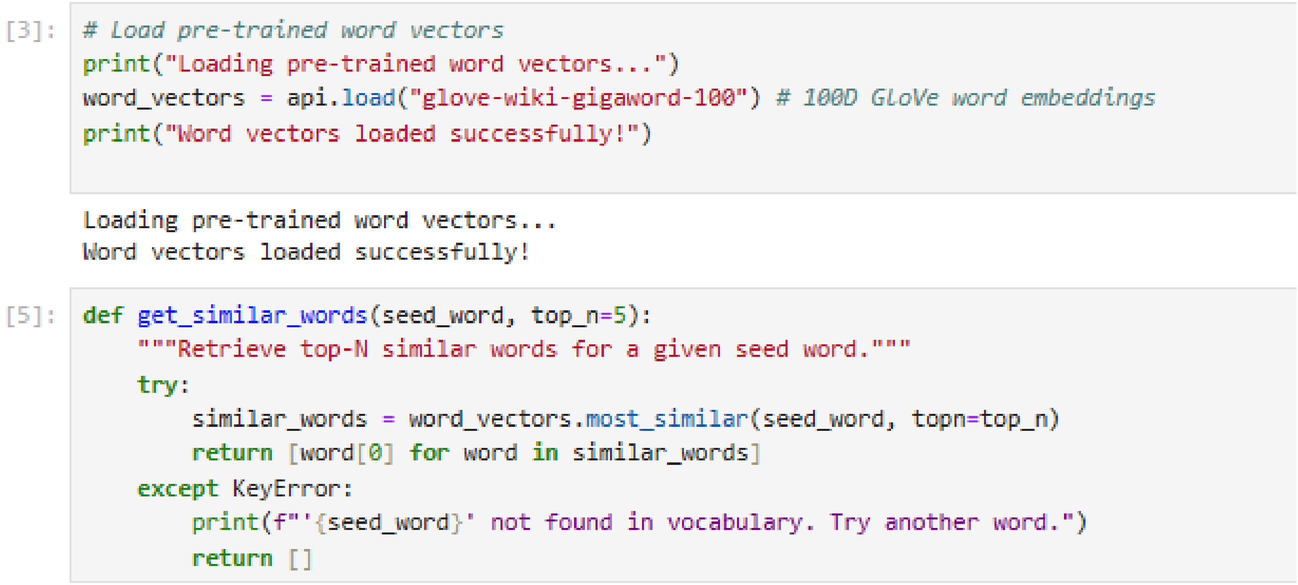
Experiment No.5:

Use word embeddings to create meaningful sentences for creative tasks. Retrieve similar words for a seed word. Create a sentence or story using these words as a starting point. Write a program that: Takes a seed word. Generate similar words. Constructs a short paragraph using these words.



```
[1]: import gensim.downloader as api
     import random
     import nltk
     from nltk.tokenize import sent_tokenize
```

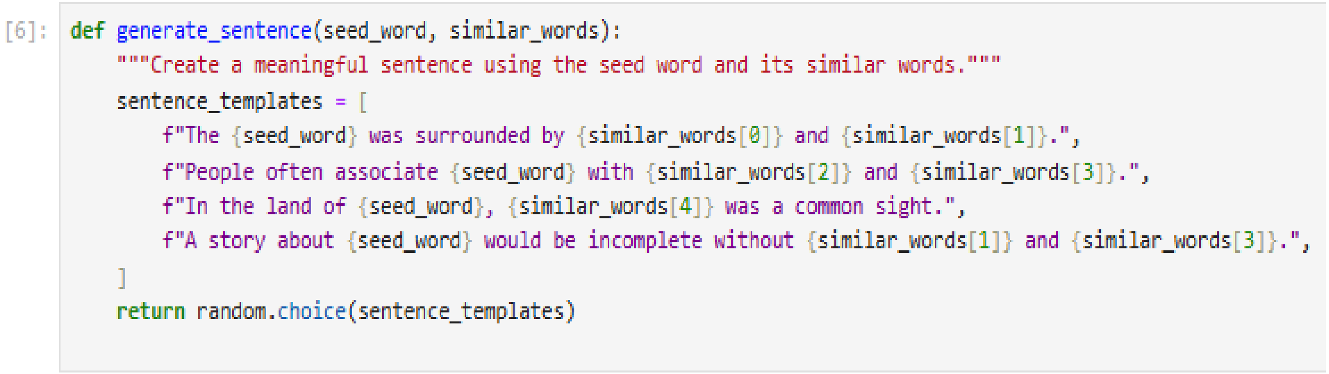
```
[2]: # Ensure required resources are downloaded
     nltk.download('punkt')
```



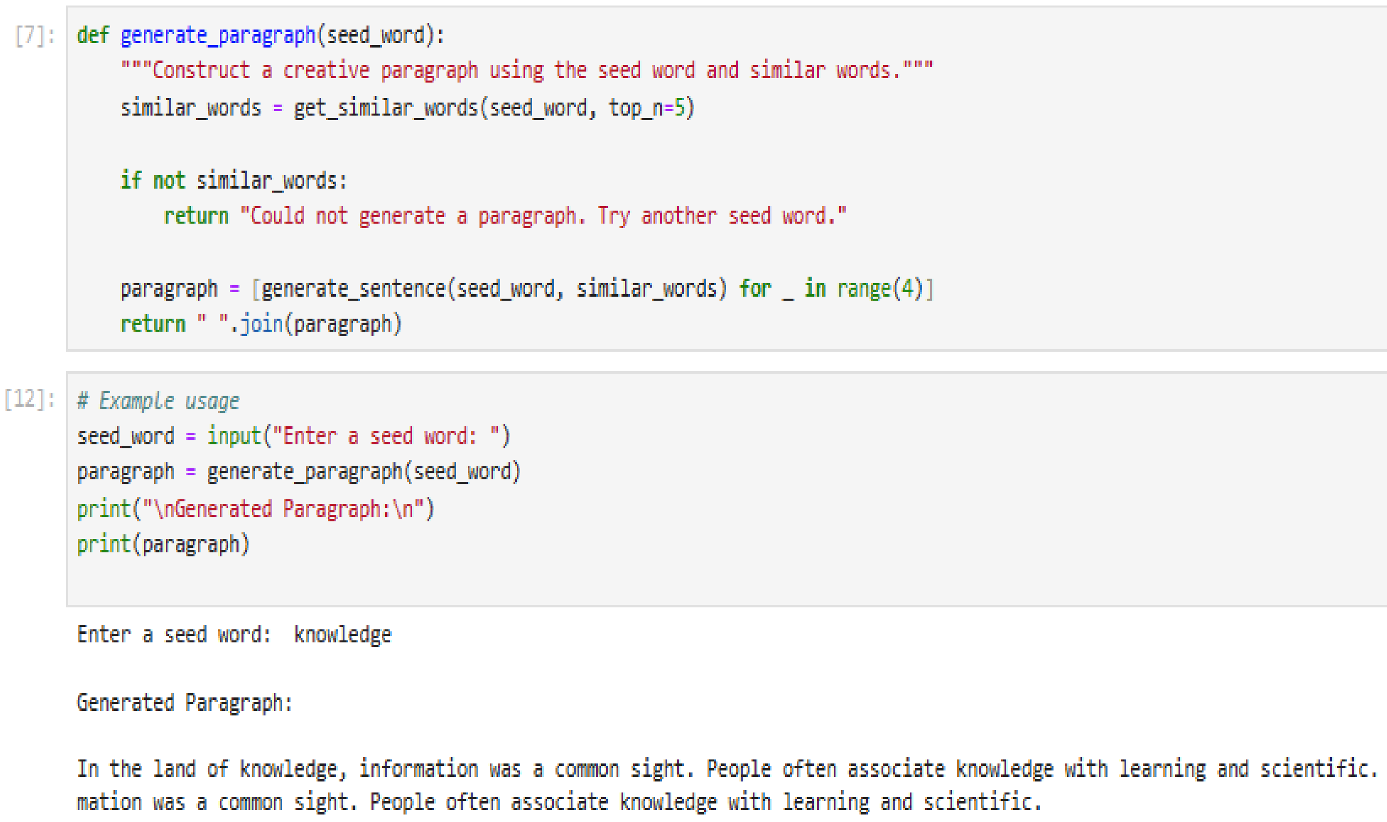
```
[3]: # Load pre-trained word vectors
     print("Loading pre-trained word vectors...")
     word_vectors = api.load("glove-wiki-gigaword-100") # 100D GloVe word embeddings
     print("Word vectors loaded successfully!")
```

```
Loading pre-trained word vectors...
Word vectors loaded successfully!
```

```
[5]: def get_similar_words(seed_word, top_n=5):
         """Retrieve top-N similar words for a given seed word."""
         try:
             similar_words = word_vectors.most_similar(seed_word, topn=top_n)
             return [word[0] for word in similar_words]
         except KeyError:
             print(f"'{seed_word}' not found in vocabulary. Try another word.")
             return []
```



```
[6]: def generate_sentence(seed_word, similar_words):
         """Create a meaningful sentence using the seed word and its similar words."""
         sentence_templates = [
             f"The {seed_word} was surrounded by {similar_words[0]} and {similar_words[1]}.",
             f"People often associate {seed_word} with {similar_words[2]} and {similar_words[3]}.",
             f"In the land of {seed_word}, {similar_words[4]} was a common sight.",
             f"A story about {seed_word} would be incomplete without {similar_words[1]} and {similar_words[3]}.",
         ]
         return random.choice(sentence_templates)
```



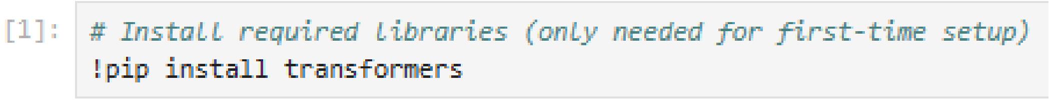
```
[7]:  def generate_paragraph(seed_word):
          """Construct a creative paragraph using the seed word and similar words."""
          similar_words = get_similar_words(seed_word, top_n=5)

          if not similar_words:
              return "Could not generate a paragraph. Try another seed word."

          paragraph = [generate_sentence(seed_word, similar_words) for _ in range(4)]
          return " ".join(paragraph)
```

```
[12]: # Example usage
      seed_word = input("Enter a seed word: ")
      paragraph = generate_paragraph(seed_word)
      print("\nGenerated Paragraph:\n")
      print(paragraph)
```

```
Enter a seed word:  knowledge

Generated Paragraph:

In the land of knowledge, information was a common sight. People often associate knowledge with learning and scientific.
mation was a common sight. People often associate knowledge with learning and scientific.
```

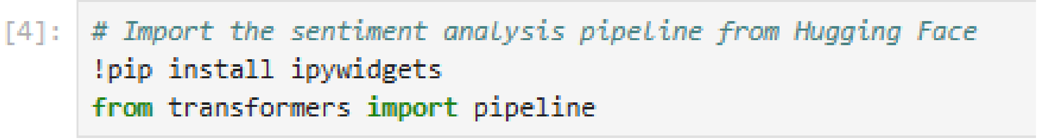
Experiment No.6:

Use a pre-trained Hugging Face model to analyze sentiment in text. Assume a real-world application, Load the sentiment analysis pipeline. Analyze the sentiment by giving sentences to input.



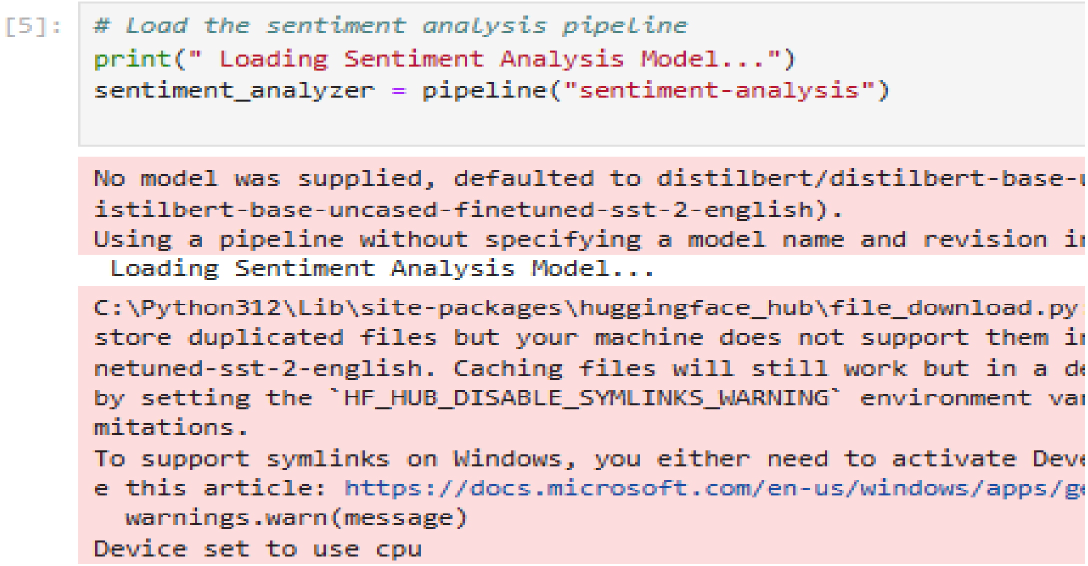
[1]:

```
# Install required libraries (only needed for first-time setup)
!pip install transformers
```



[4]:

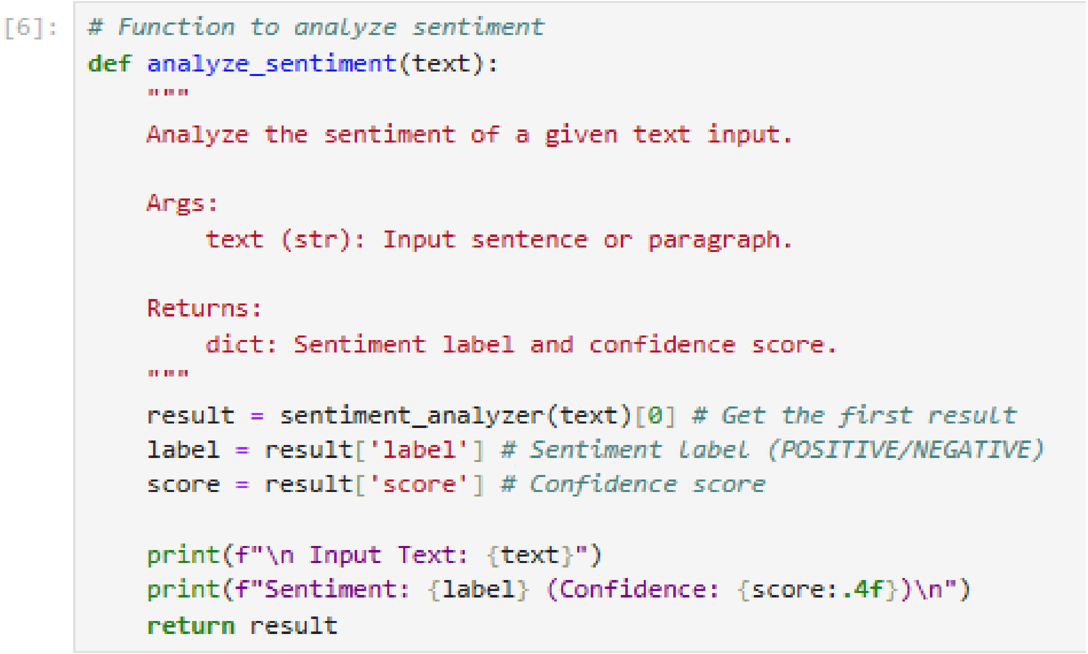
```
# Import the sentiment analysis pipeline from Hugging Face
!pip install ipywidgets
from transformers import pipeline
```



[5]:

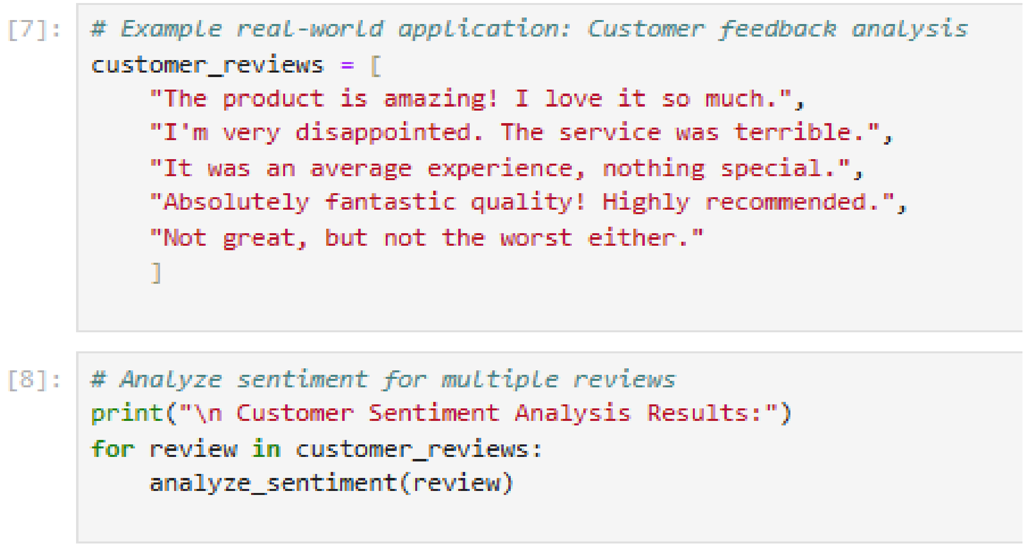
```
# Load the sentiment analysis pipeline
print(" Loading Sentiment Analysis Model...")
sentiment_analyzer = pipeline("sentiment-analysis")
```

```
No model was supplied, defaulted to distilbert/distilbert-base-u
istilbert-base-uncased-finetuned-sst-2-english).
Using a pipeline without specifying a model name and revision in
 Loading Sentiment Analysis Model...
C:\Python312\Lib\site-packages\huggingface_hub\file_download.py
store duplicated files but your machine does not support them in
netuned-sst-2-english. Caching files will still work but in a de
by setting the `HF_HUB_DISABLE_SYMLINKS_WARNING` environment var
mitations.
To support symlinks on Windows, you either need to activate Deve
e this article: https://docs.microsoft.com/en-us/windows/apps/ge
  warnings.warn(message)
Device set to use cpu
```



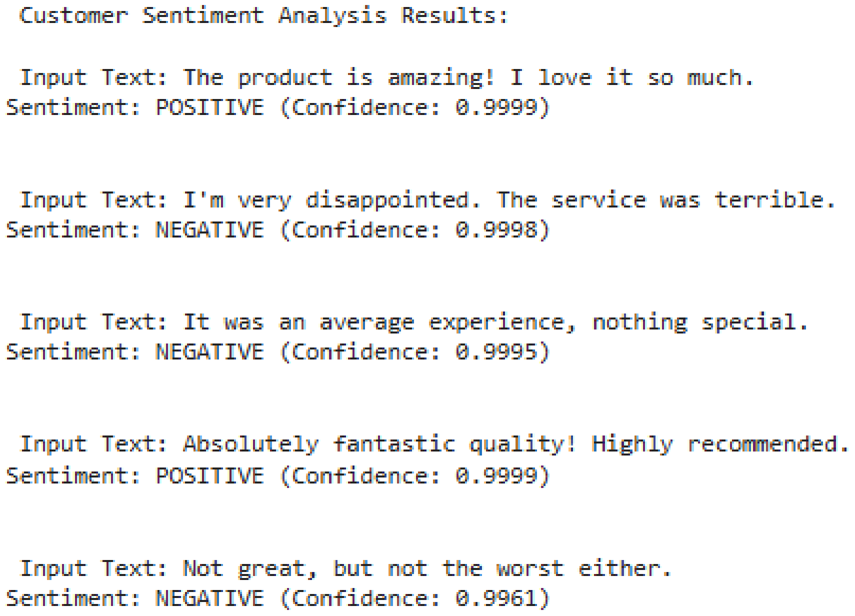
[6]:

```
# Function to analyze sentiment
def analyze_sentiment(text):
    """
    Analyze the sentiment of a given text input.

    Args:
        text (str): Input sentence or paragraph.

    Returns:
        dict: Sentiment label and confidence score.
    """
    result = sentiment_analyzer(text)[0] # Get the first result
    label = result['label'] # Sentiment label (POSITIVE/NEGATIVE)
    score = result['score'] # Confidence score

    print(f"\n Input Text: {text}")
    print(f"Sentiment: {label} (Confidence: {score:.4f})\n")
    return result
```



```
[7]: # Example real-world application: Customer feedback analysis
     customer_reviews = [
         "The product is amazing! I love it so much.",
         "I'm very disappointed. The service was terrible.",
         "It was an average experience, nothing special.",
         "Absolutely fantastic quality! Highly recommended.",
         "Not great, but not the worst either."
         ]
```

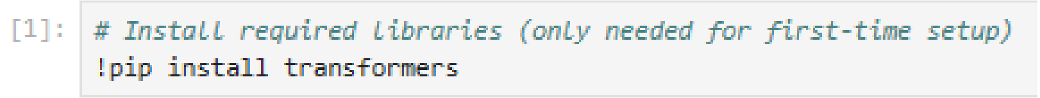
```
[8]: # Analyze sentiment for multiple reviews
     print("\n Customer Sentiment Analysis Results:")
     for review in customer_reviews:
         analyze_sentiment(review)
```



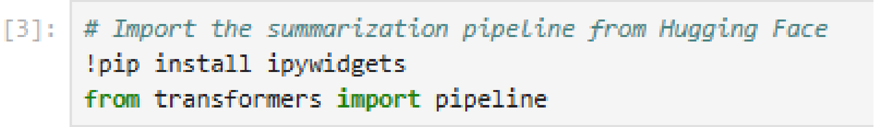
```
 Customer Sentiment Analysis Results:

 Input Text: The product is amazing! I love it so much.
Sentiment: POSITIVE (Confidence: 0.9999)


 Input Text: I'm very disappointed. The service was terrible.
Sentiment: NEGATIVE (Confidence: 0.9998)


 Input Text: It was an average experience, nothing special.
Sentiment: NEGATIVE (Confidence: 0.9995)


 Input Text: Absolutely fantastic quality! Highly recommended.
Sentiment: POSITIVE (Confidence: 0.9999)


 Input Text: Not great, but not the worst either.
Sentiment: NEGATIVE (Confidence: 0.9961)
```

Experiment No.7:

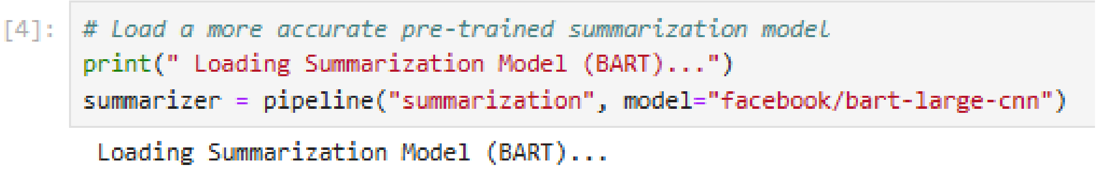
Summarize long texts using a pre-trained summarization model using Hugging face model. Load the summarization pipeline. Take a passage as input and obtain the summarized text.



```
[1]: # Install required libraries (only needed for first-time setup)
     !pip install transformers
```

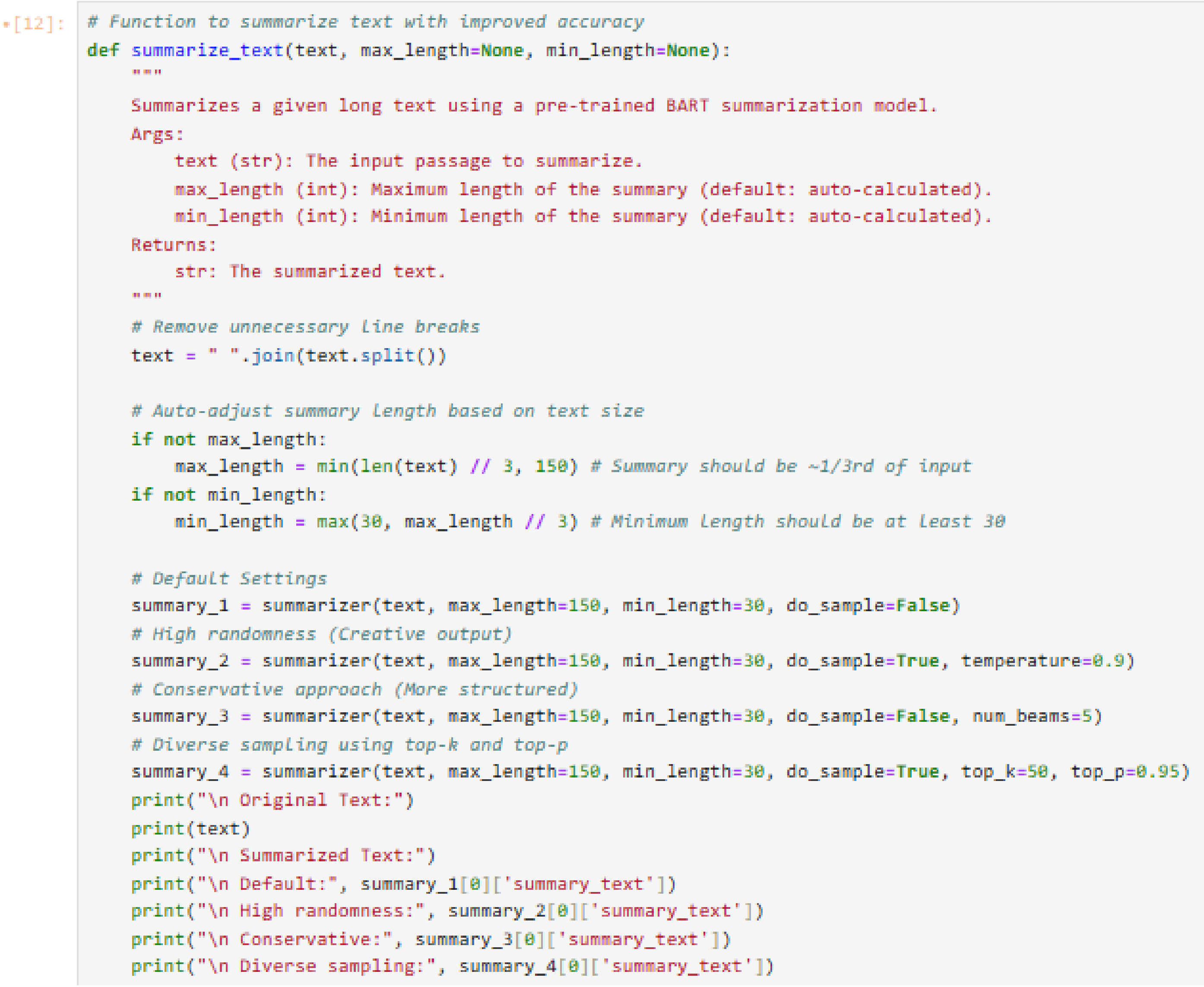


```
[3]: # Import the summarization pipeline from Hugging Face
     !pip install ipywidgets
     from transformers import pipeline
```

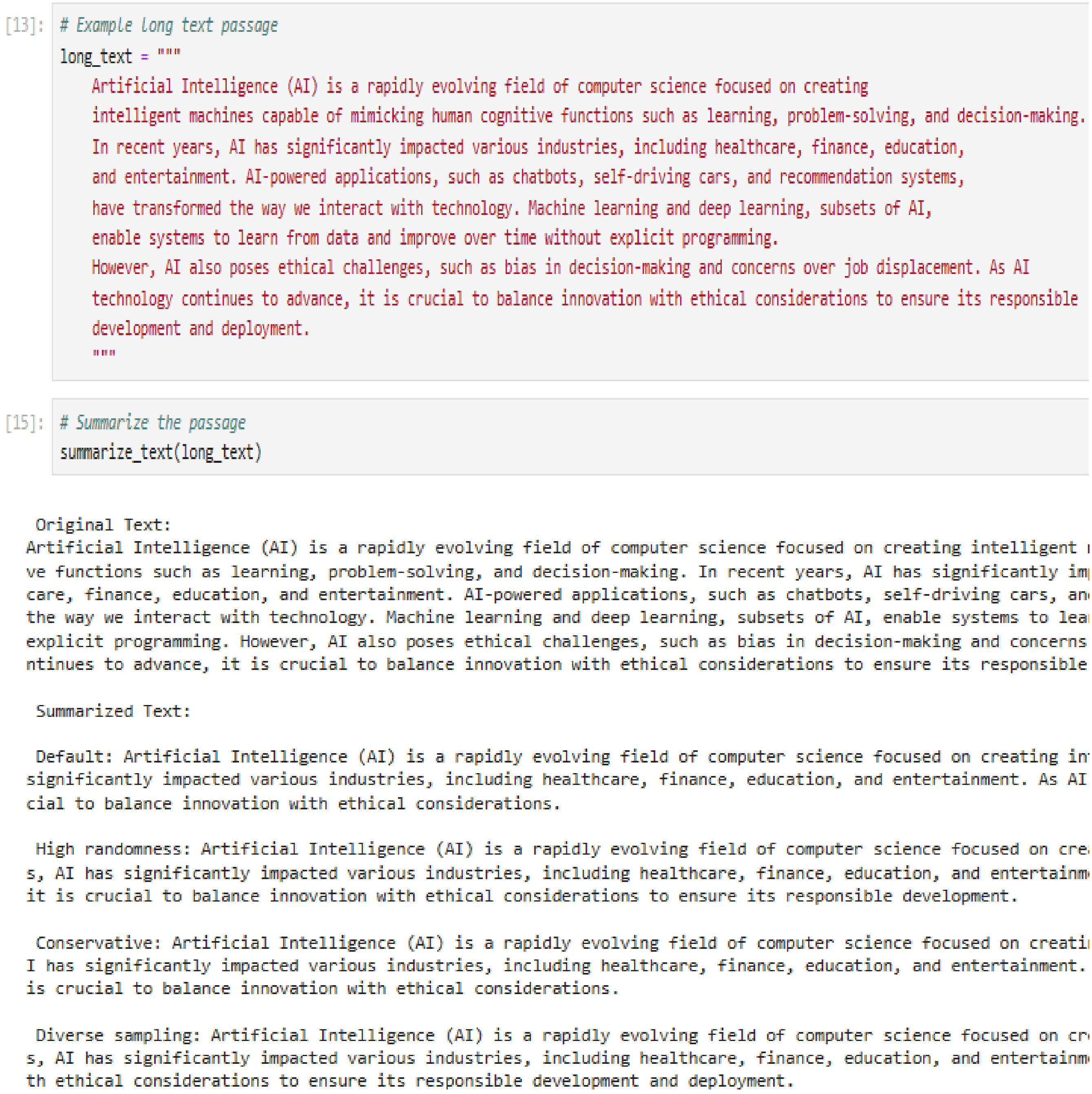


```
[4]: # Load a more accurate pre-trained summarization model
     print(" Loading Summarization Model (BART)...")
     summarizer = pipeline("summarization", model="facebook/bart-large-cnn")
```

```
Loading Summarization Model (BART)...
```



```
*[12]: # Function to summarize text with improved accuracy
def summarize_text(text, max_length=None, min_length=None):
    """
    Summarizes a given long text using a pre-trained BART summarization model.
    Args:
        text (str): The input passage to summarize.
        max_length (int): Maximum length of the summary (default: auto-calculated).
        min_length (int): Minimum length of the summary (default: auto-calculated).
    Returns:
        str: The summarized text.
    """
    # Remove unnecessary line breaks
    text = " ".join(text.split())

    # Auto-adjust summary length based on text size
    if not max_length:
        max_length = min(len(text) // 3, 150) # Summary should be ~1/3rd of input
    if not min_length:
        min_length = max(30, max_length // 3) # Minimum length should be at least 30

    # Default Settings
    summary_1 = summarizer(text, max_length=150, min_length=30, do_sample=False)
    # High randomness (Creative output)
    summary_2 = summarizer(text, max_length=150, min_length=30, do_sample=True, temperature=0.9)
    # Conservative approach (More structured)
    summary_3 = summarizer(text, max_length=150, min_length=30, do_sample=False, num_beams=5)
    # Diverse sampling using top-k and top-p
    summary_4 = summarizer(text, max_length=150, min_length=30, do_sample=True, top_k=50, top_p=0.95)
    print("\n Original Text:")
    print(text)
    print("\n Summarized Text:")
    print("\n Default:", summary_1[0]['summary_text'])
    print("\n High randomness:", summary_2[0]['summary_text'])
    print("\n Conservative:", summary_3[0]['summary_text'])
    print("\n Diverse sampling:", summary_4[0]['summary_text'])
```



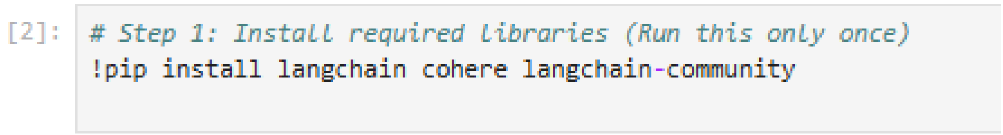
```
[13]: # Example long text passage
      long_text = """
          Artificial Intelligence (AI) is a rapidly evolving field of computer science focused on creating
          intelligent machines capable of mimicking human cognitive functions such as learning, problem-solving, and decision-making.
          In recent years, AI has significantly impacted various industries, including healthcare, finance, education,
          and entertainment. AI-powered applications, such as chatbots, self-driving cars, and recommendation systems,
          have transformed the way we interact with technology. Machine learning and deep learning, subsets of AI,
          enable systems to learn from data and improve over time without explicit programming.
          However, AI also poses ethical challenges, such as bias in decision-making and concerns over job displacement. As AI
          technology continues to advance, it is crucial to balance innovation with ethical considerations to ensure its responsible
          development and deployment.
          """
```

```
[15]: # Summarize the passage
      summarize_text(long_text)
```

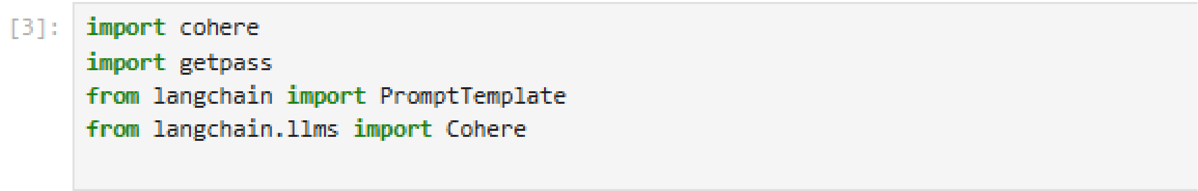
```
 Original Text:
Artificial Intelligence (AI) is a rapidly evolving field of computer science focused on creating intelligent
ve functions such as learning, problem-solving, and decision-making. In recent years, AI has significantly im
care, finance, education, and entertainment. AI-powered applications, such as chatbots, self-driving cars, an
the way we interact with technology. Machine learning and deep learning, subsets of AI, enable systems to lea
explicit programming. However, AI also poses ethical challenges, such as bias in decision-making and concerns
ntinues to advance, it is crucial to balance innovation with ethical considerations to ensure its responsible

 Summarized Text:

 Default: Artificial Intelligence (AI) is a rapidly evolving field of computer science focused on creating in
significantly impacted various industries, including healthcare, finance, education, and entertainment. As AI
cial to balance innovation with ethical considerations.

 High randomness: Artificial Intelligence (AI) is a rapidly evolving field of computer science focused on cre
s, AI has significantly impacted various industries, including healthcare, finance, education, and entertainm
it is crucial to balance innovation with ethical considerations to ensure its responsible development.

 Conservative: Artificial Intelligence (AI) is a rapidly evolving field of computer science focused on creati
I has significantly impacted various industries, including healthcare, finance, education, and entertainment.
is crucial to balance innovation with ethical considerations.

 Diverse sampling: Artificial Intelligence (AI) is a rapidly evolving field of computer science focused on cr
s, AI has significantly impacted various industries, including healthcare, finance, education, and entertainm
th ethical considerations to ensure its responsible development and deployment.
```

Experiment No.8:

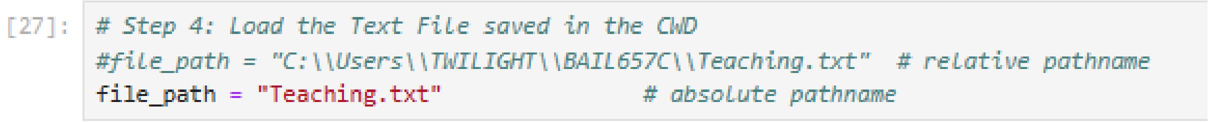
Install langchain, cohere (for key), langchain-community. Get the api key(By logging into Cohere and obtaining the cohere key).Load a text document from your google drive. Create a prompt template to display the output in a particular manner.



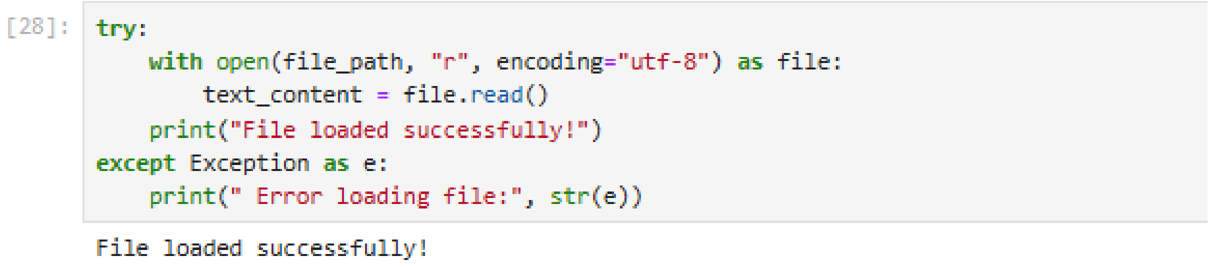
```
[2]: # Step 1: Install required libraries (Run this only once)
!pip install langchain cohere langchain-community
```



```
[3]: import cohere
import getpass
from langchain import PromptTemplate
from langchain.llms import Cohere
```

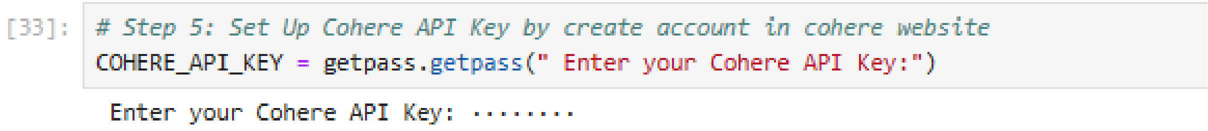


```
[27]: # Step 4: Load the Text File saved in the CWD
#file_path = "C:\\Users\\TWILIGHT\\BAIL657C\\Teaching.txt"  # relative pathname
file_path = "Teaching.txt"                              # absolute pathname
```



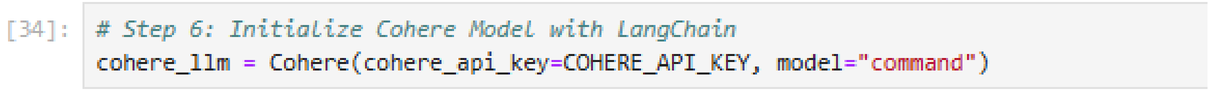
```
[28]: try:
    with open(file_path, "r", encoding="utf-8") as file:
        text_content = file.read()
    print("File loaded successfully!")
except Exception as e:
    print(" Error loading file:", str(e))
```

```
File loaded successfully!
```

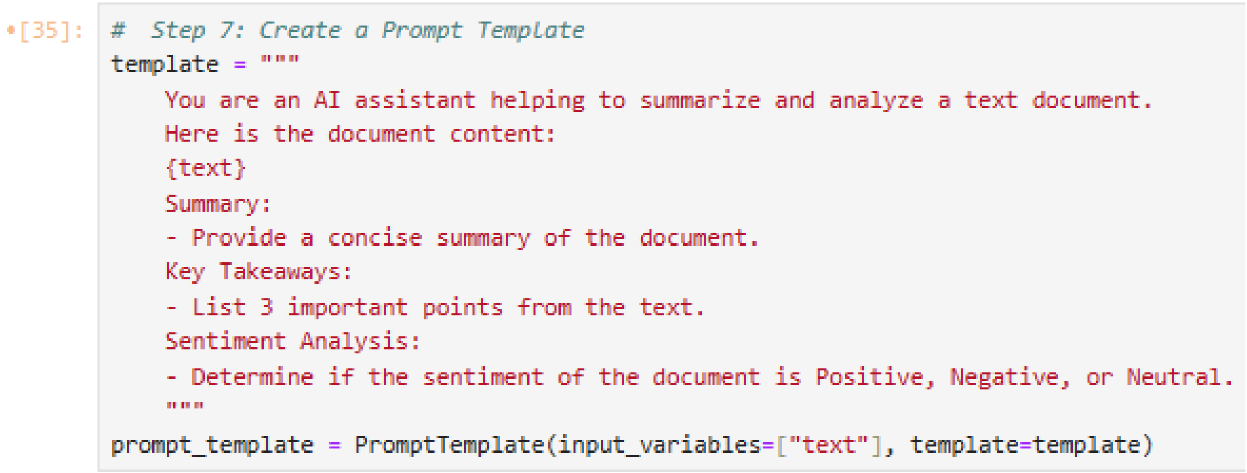


```
[33]: # Step 5: Set Up Cohere API Key by create account in cohere website
COHERE_API_KEY = getpass.getpass(" Enter your Cohere API Key:")
```

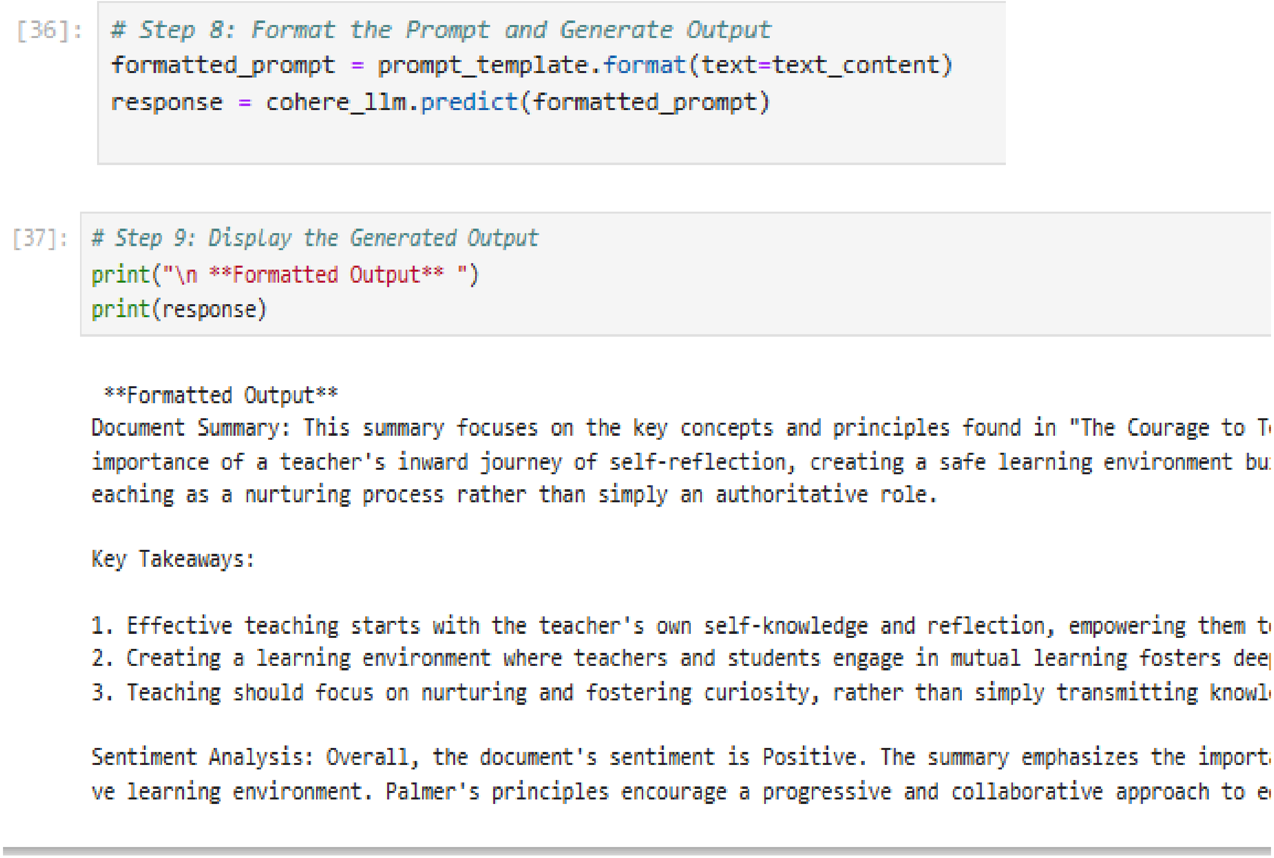
```
 Enter your Cohere API Key: ········
```



```
[34]: # Step 6: Initialize Cohere Model with LangChain
cohere_llm = Cohere(cohere_api_key=COHERE_API_KEY, model="command")
```



```
[35]: #  Step 7: Create a Prompt Template
template = """
    You are an AI assistant helping to summarize and analyze a text document.
    Here is the document content:
    {text}
    Summary:
    - Provide a concise summary of the document.
    Key Takeaways:
    - List 3 important points from the text.
    Sentiment Analysis:
    - Determine if the sentiment of the document is Positive, Negative, or Neutral.
    """
prompt_template = PromptTemplate(input_variables=["text"], template=template)
```



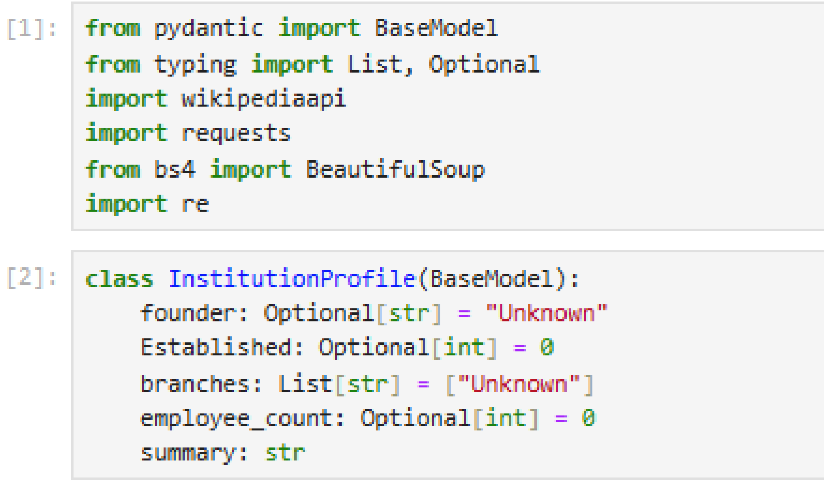
```
[36]: # Step 8: Format the Prompt and Generate Output
      formatted_prompt = prompt_template.format(text=text_content)
      response = cohere_llm.predict(formatted_prompt)
```

```
[37]: # Step 9: Display the Generated Output
      print("\n **Formatted Output** ")
      print(response)
```

```
 **Formatted Output**
Document Summary: This summary focuses on the key concepts and principles found in "The Courage to T
importance of a teacher's inward journey of self-reflection, creating a safe learning environment bu
eaching as a nurturing process rather than simply an authoritative role.

Key Takeaways:

1. Effective teaching starts with the teacher's own self-knowledge and reflection, empowering them t
2. Creating a learning environment where teachers and students engage in mutual learning fosters dee
3. Teaching should focus on nurturing and fostering curiosity, rather than simply transmitting knowl

Sentiment Analysis: Overall, the document's sentiment is Positive. The summary emphasizes the import
ve learning environment. Palmer's principles encourage a progressive and collaborative approach to e
```

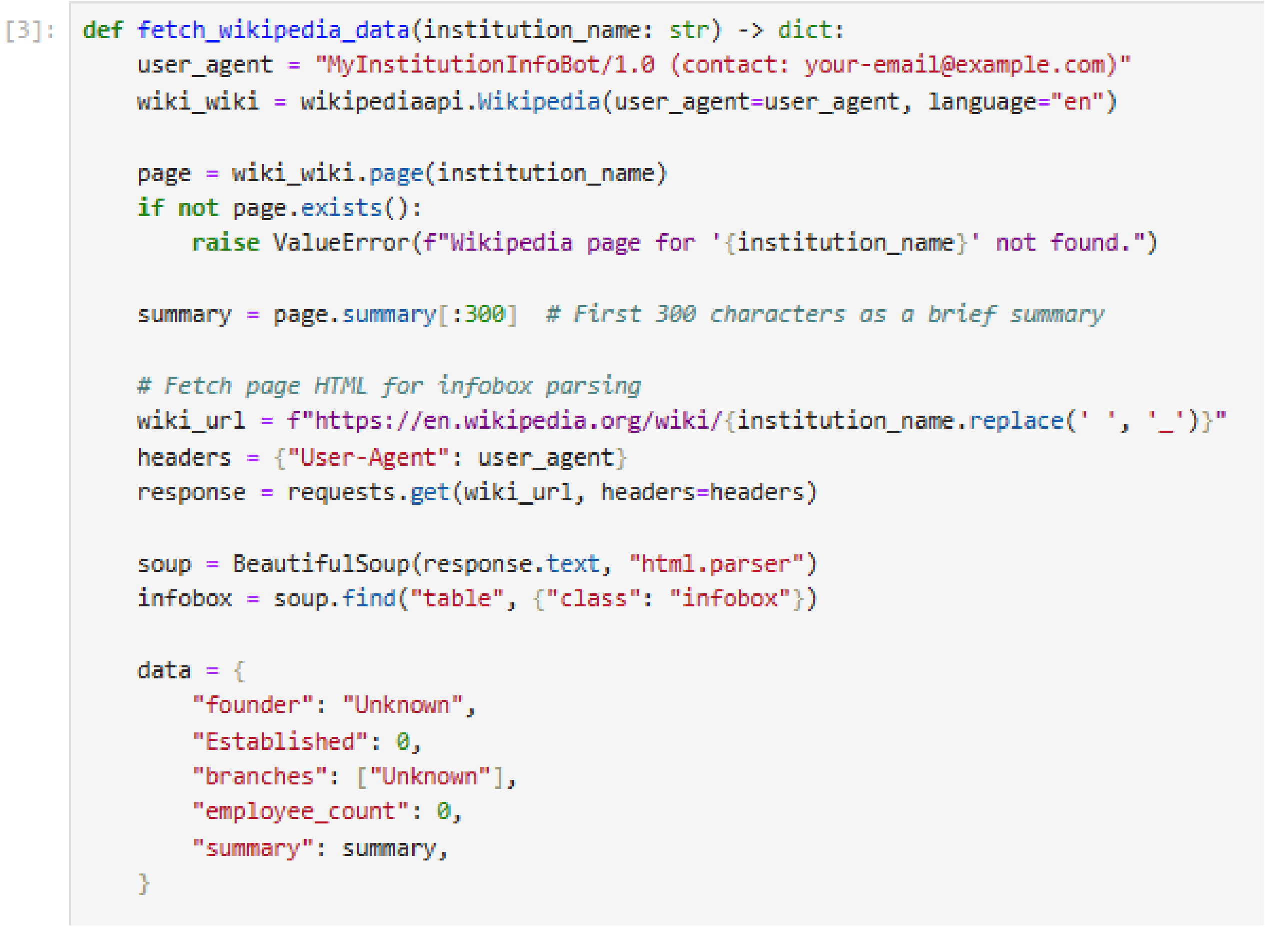
Experiment No.9:

Take the Institution name as input. Use Pydantic to define the schema for the desired output and create a custom output parser. Invoke the Chain and Fetch Results. Extract the below Institution related details from Wikipedia: The founder of the Institution. When it was founded. The current branches in the institution. How many employees are working in it? A brief 4-line summary of the institution.

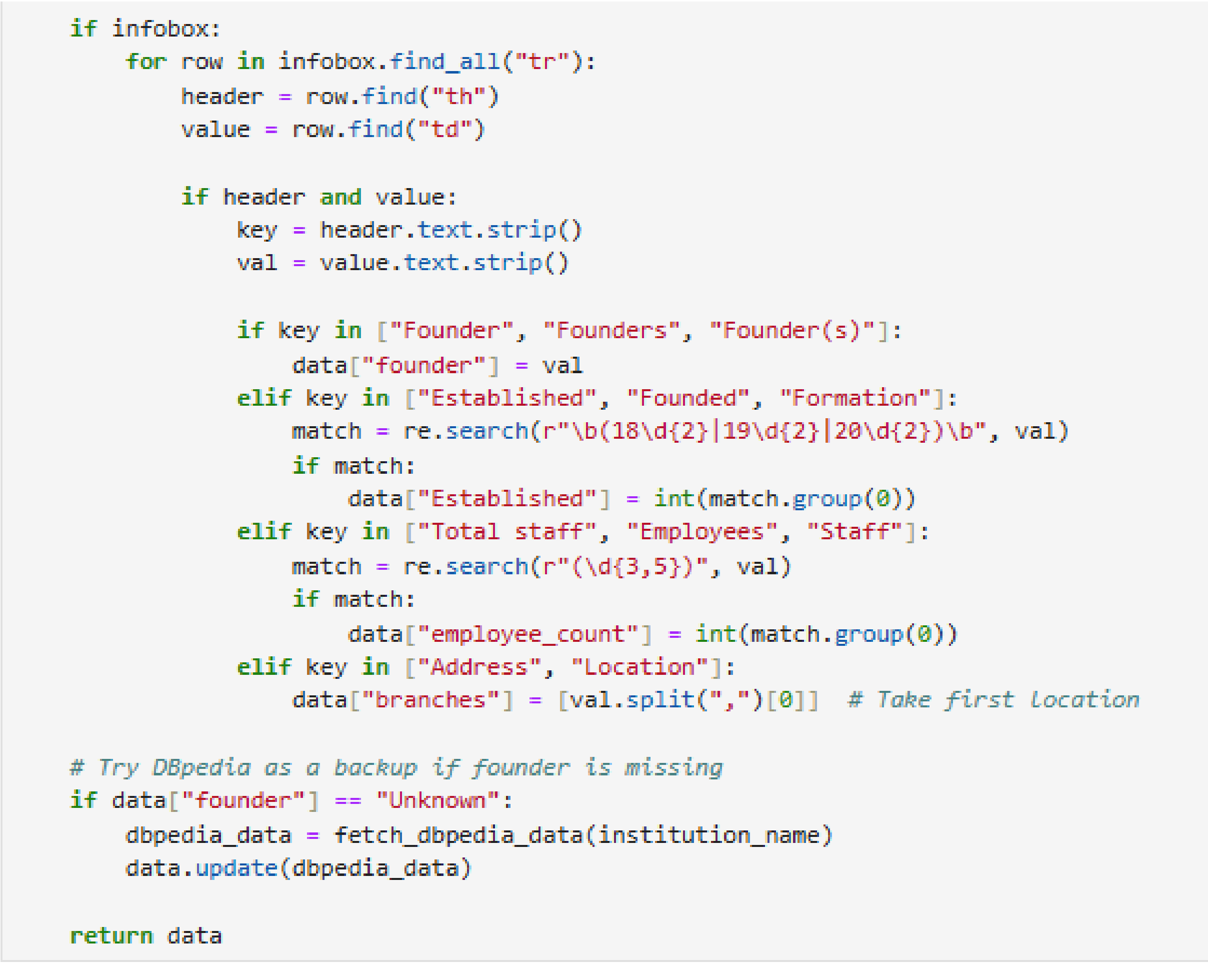


```
[1]: from pydantic import BaseModel
     from typing import List, Optional
     import wikipediaapi
     import requests
     from bs4 import BeautifulSoup
     import re
```

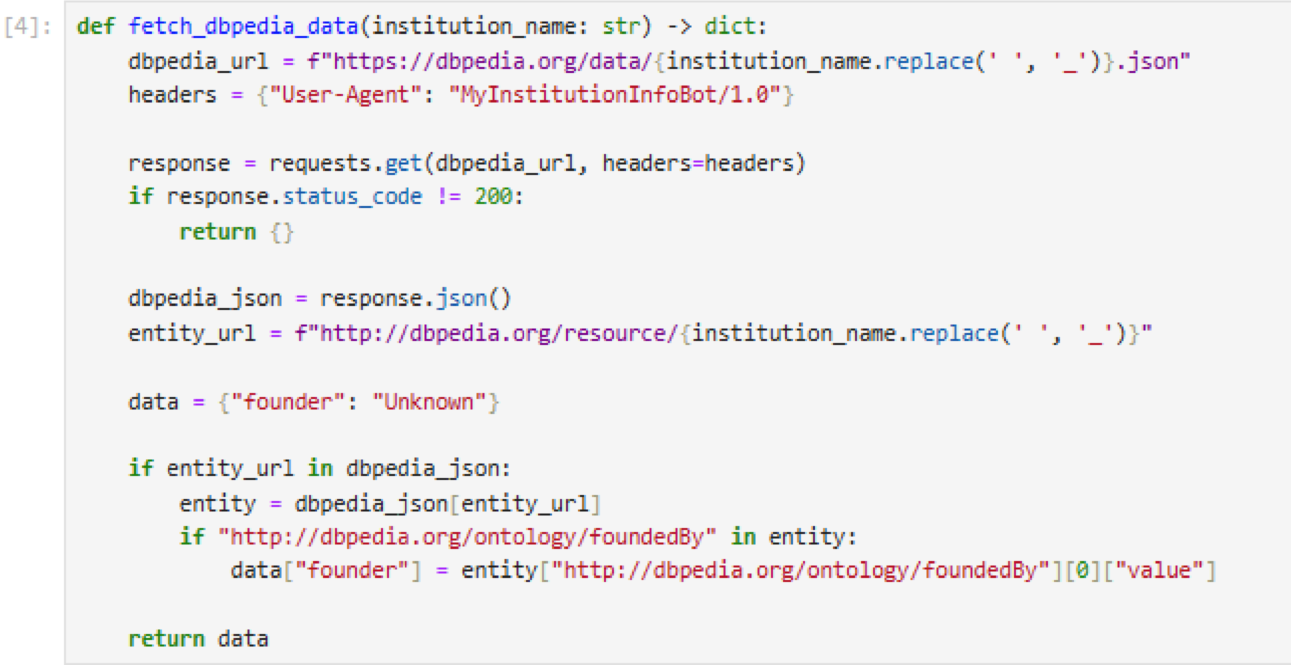
```
[2]: class InstitutionProfile(BaseModel):
         founder: Optional[str] = "Unknown"
         Established: Optional[int] = 0
         branches: List[str] = ["Unknown"]
         employee_count: Optional[int] = 0
         summary: str
```



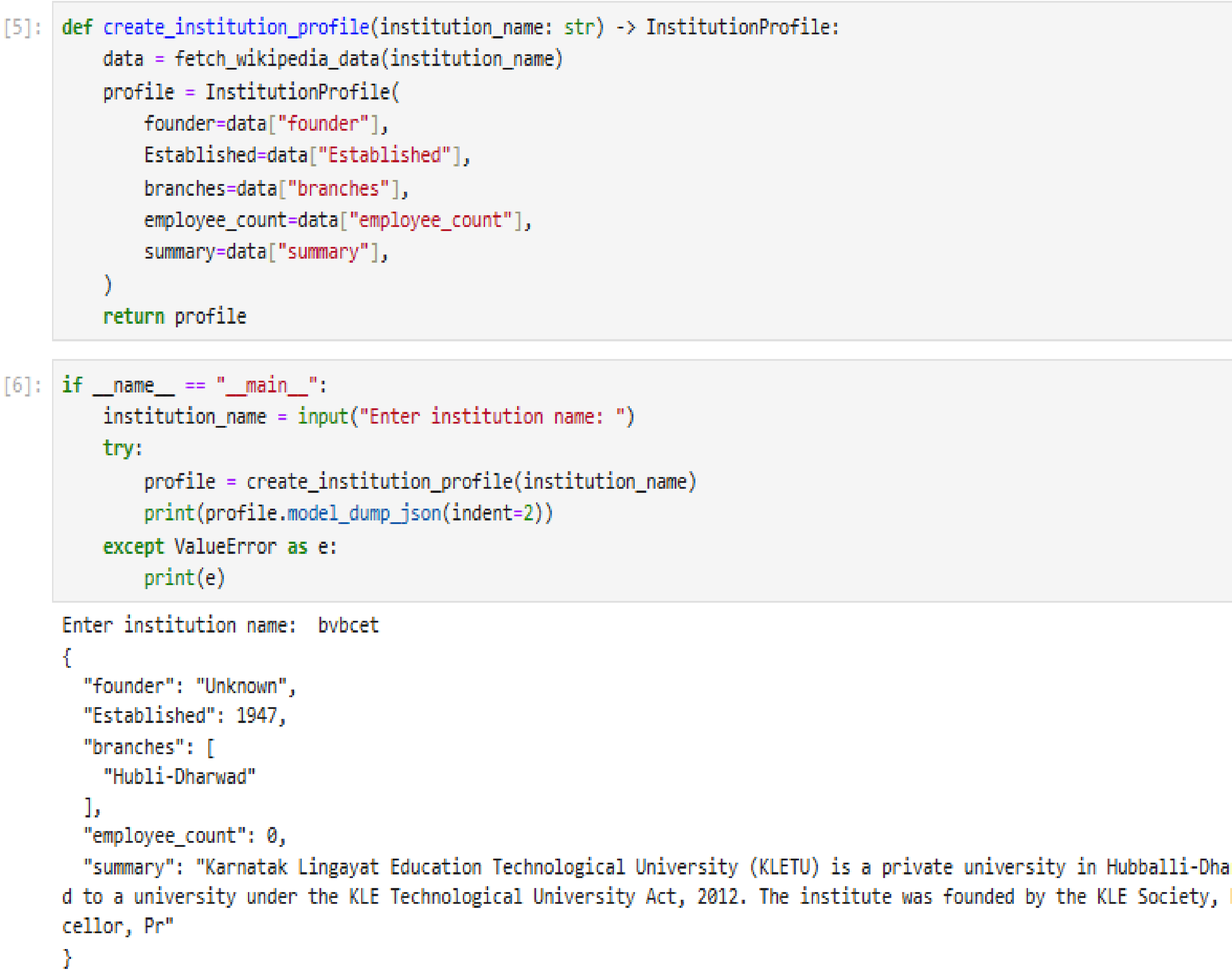
```
[3]: def fetch_wikipedia_data(institution_name: str) -> dict:
         user_agent = "MyInstitutionInfoBot/1.0 (contact: your-email@example.com)"
         wiki_wiki = wikipediaapi.Wikipedia(user_agent=user_agent, language="en")

         page = wiki_wiki.page(institution_name)
         if not page.exists():
             raise ValueError(f"Wikipedia page for '{institution_name}' not found.")

         summary = page.summary[:300]  # First 300 characters as a brief summary

         # Fetch page HTML for infobox parsing
         wiki_url = f"https://en.wikipedia.org/wiki/{institution_name.replace(' ', '_')}"
         headers = {"User-Agent": user_agent}
         response = requests.get(wiki_url, headers=headers)

         soup = BeautifulSoup(response.text, "html.parser")
         infobox = soup.find("table", {"class": "infobox"})

         data = {
             "founder": "Unknown",
             "Established": 0,
             "branches": ["Unknown"],
             "employee_count": 0,
             "summary": summary,
         }
```



```python
    if infobox:
        for row in infobox.find_all("tr"):
            header = row.find("th")
            value = row.find("td")

            if header and value:
                key = header.text.strip()
                val = value.text.strip()

                if key in ["Founder", "Founders", "Founder(s)"]:
                    data["founder"] = val
                elif key in ["Established", "Founded", "Formation"]:
                    match = re.search(r"\b(18\d{2}|19\d{2}|20\d{2})\b", val)
                    if match:
                        data["Established"] = int(match.group(0))
                elif key in ["Total staff", "Employees", "Staff"]:
                    match = re.search(r"(\d{3,5})", val)
                    if match:
                        data["employee_count"] = int(match.group(0))
                elif key in ["Address", "Location"]:
                    data["branches"] = [val.split(",")[0]]  # Take first location

    # Try DBpedia as a backup if founder is missing
    if data["founder"] == "Unknown":
        dbpedia_data = fetch_dbpedia_data(institution_name)
        data.update(dbpedia_data)

    return data
```



```python
[4]: def fetch_dbpedia_data(institution_name: str) -> dict:
         dbpedia_url = f"https://dbpedia.org/data/{institution_name.replace(' ', '_')}.json"
         headers = {"User-Agent": "MyInstitutionInfoBot/1.0"}

         response = requests.get(dbpedia_url, headers=headers)
         if response.status_code != 200:
             return {}

         dbpedia_json = response.json()
         entity_url = f"http://dbpedia.org/resource/{institution_name.replace(' ', '_')}"

         data = {"founder": "Unknown"}

         if entity_url in dbpedia_json:
             entity = dbpedia_json[entity_url]
             if "http://dbpedia.org/ontology/foundedBy" in entity:
                 data["founder"] = entity["http://dbpedia.org/ontology/foundedBy"][0]["value"]

         return data
```



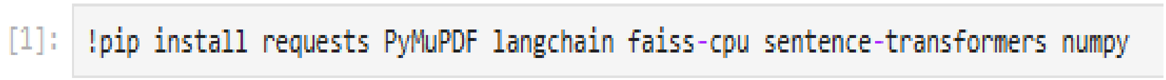
```
[5]: def create_institution_profile(institution_name: str) -> InstitutionProfile:
         data = fetch_wikipedia_data(institution_name)
         profile = InstitutionProfile(
             founder=data["founder"],
             Established=data["Established"],
             branches=data["branches"],
             employee_count=data["employee_count"],
             summary=data["summary"],
         )
         return profile
```

```
[6]: if __name__ == "__main__":
         institution_name = input("Enter institution name: ")
         try:
             profile = create_institution_profile(institution_name)
             print(profile.model_dump_json(indent=2))
         except ValueError as e:
             print(e)
```

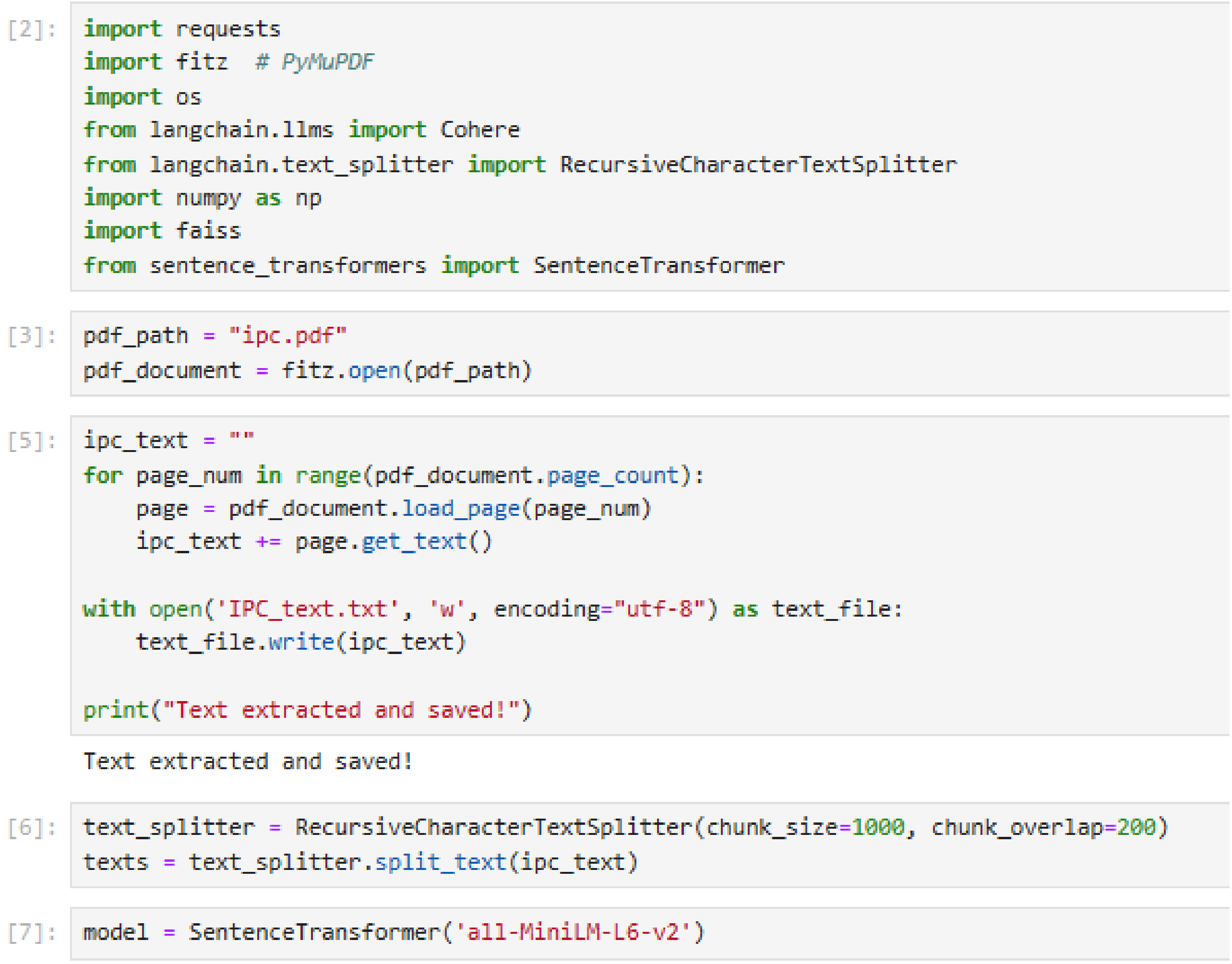
```
Enter institution name:  bvbcet
{
  "founder": "Unknown",
  "Established": 1947,
  "branches": [
    "Hubli-Dharwad"
  ],
  "employee_count": 0,
  "summary": "Karnatak Lingayat Education Technological University (KLETU) is a private university in Hubballi-Dha
d to a university under the KLE Technological University Act, 2012. The institute was founded by the KLE Society,
cellor, Pr"
}
```

Experiment No.10:

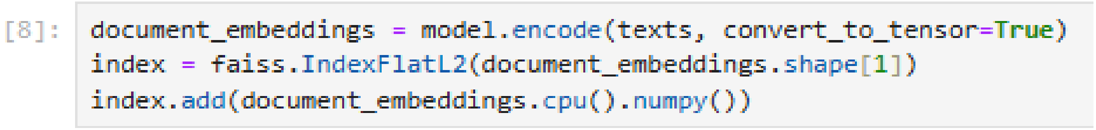
Build a chat bot for the Indian Penal Code. We'll start by downloading the official Indian Penal Code document, and then we'll create a chat bot that can interact with it. Users will be able to ask questions about the Indian Penal Code and have a conversation with it.



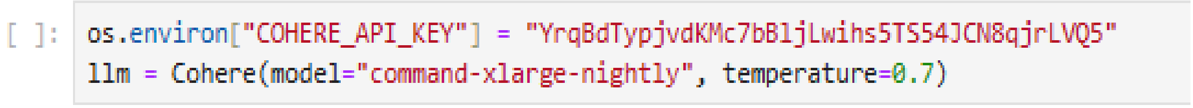
```
[1]: !pip install requests PyMuPDF langchain faiss-cpu sentence-transformers numpy
```



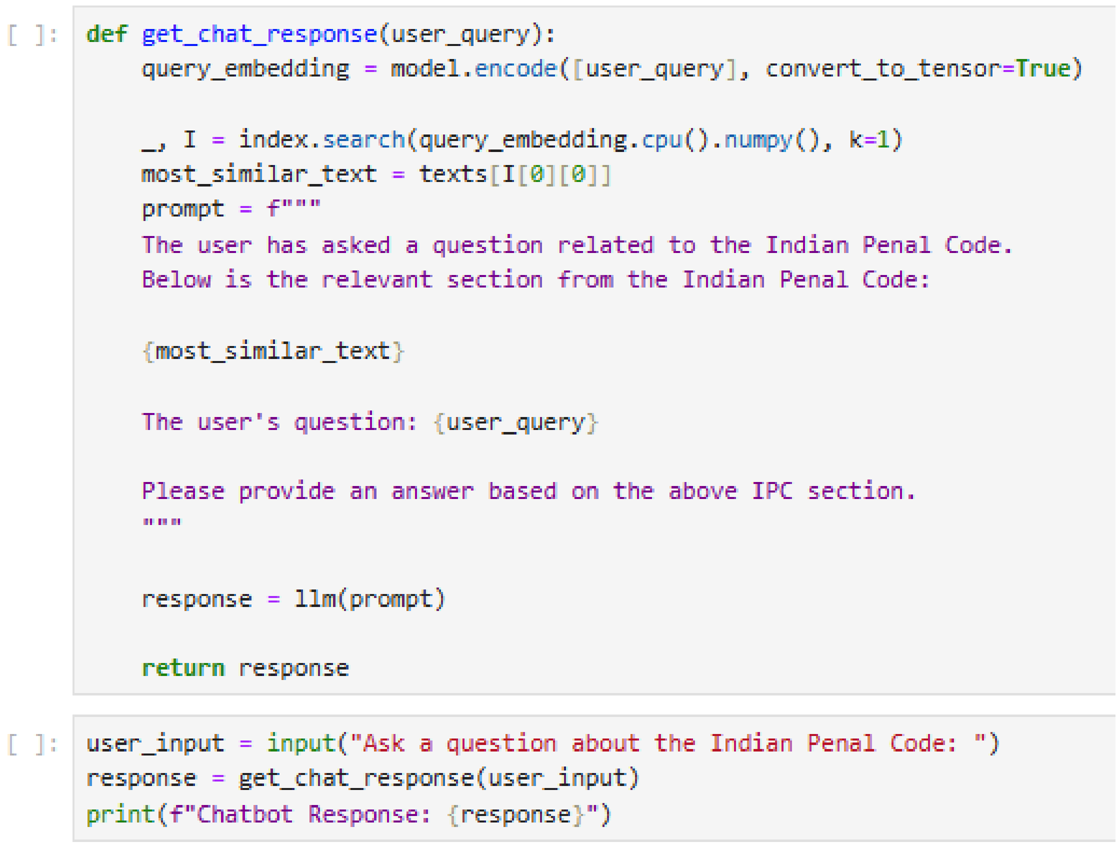
```
[2]: import requests
     import fitz  # PyMuPDF
     import os
     from langchain.llms import Cohere
     from langchain.text_splitter import RecursiveCharacterTextSplitter
     import numpy as np
     import faiss
     from sentence_transformers import SentenceTransformer

[3]: pdf_path = "ipc.pdf"
     pdf_document = fitz.open(pdf_path)

[5]: ipc_text = ""
     for page_num in range(pdf_document.page_count):
         page = pdf_document.load_page(page_num)
         ipc_text += page.get_text()

     with open('IPC_text.txt', 'w', encoding="utf-8") as text_file:
         text_file.write(ipc_text)

     print("Text extracted and saved!")

     Text extracted and saved!

[6]: text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
     texts = text_splitter.split_text(ipc_text)

[7]: model = SentenceTransformer('all-MiniLM-L6-v2')
```



```
[8]: document_embeddings = model.encode(texts, convert_to_tensor=True)
     index = faiss.IndexFlatL2(document_embeddings.shape[1])
     index.add(document_embeddings.cpu().numpy())
```



```
[ ]: os.environ["COHERE_API_KEY"] = "YrqBdTypjvdKMc7bBljLwihs5TS54JCN8qjrLVQ5"
     llm = Cohere(model="command-xlarge-nightly", temperature=0.7)
```



[ ]:

```
def get_chat_response(user_query):
    query_embedding = model.encode([user_query], convert_to_tensor=True)

    _, I = index.search(query_embedding.cpu().numpy(), k=1)
    most_similar_text = texts[I[0][0]]
    prompt = f"""
    The user has asked a question related to the Indian Penal Code.
    Below is the relevant section from the Indian Penal Code:

    {most_similar_text}

    The user's question: {user_query}

    Please provide an answer based on the above IPC section.
    """

    response = llm(prompt)

    return response
```

[ ]:

```
user_input = input("Ask a question about the Indian Penal Code: ")
response = get_chat_response(user_input)
print(f"Chatbot Response: {response}")
```