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

# Import libraries
import gensim.downloader as api
import numpy as np
from numpy.linalg import norm

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

# 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.")

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

# 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.")

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

```
analyze_similarity("music", "art")


# 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.")


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

Output:

```
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

Similarity between 'cat' and 'dog': 0.7609

Similarity between 'computer' and 'keyboard': 0.3964

Similarity between 'music' and 'art': 0.4010

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: 0.6574Word 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
```

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

Similarity between 'computer' and 'keyboard': 0.3964

Similarity between 'music' and 'art': 0.4010

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
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
```

Program 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.

2a. Use dimensionality reduction (e.g., PCA or t-SNE) to visualize word embeddings for Q 1.

```
# Install required libraries
!pip install gensim numpy matplotlib sklearn

# 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

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

# 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 []

# 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=min(len(words) - 1, 30))  # Adjust 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()

# Example word relationships to explore
words_to_explore = ["king", "man", "woman", "queen", "prince", "princess", "royal", "throne"]

# Get related words using vector arithmetic
filtered_words = explore_word_relationships("king", "man", "woman")

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

# Get vectors for the words to visualize
```

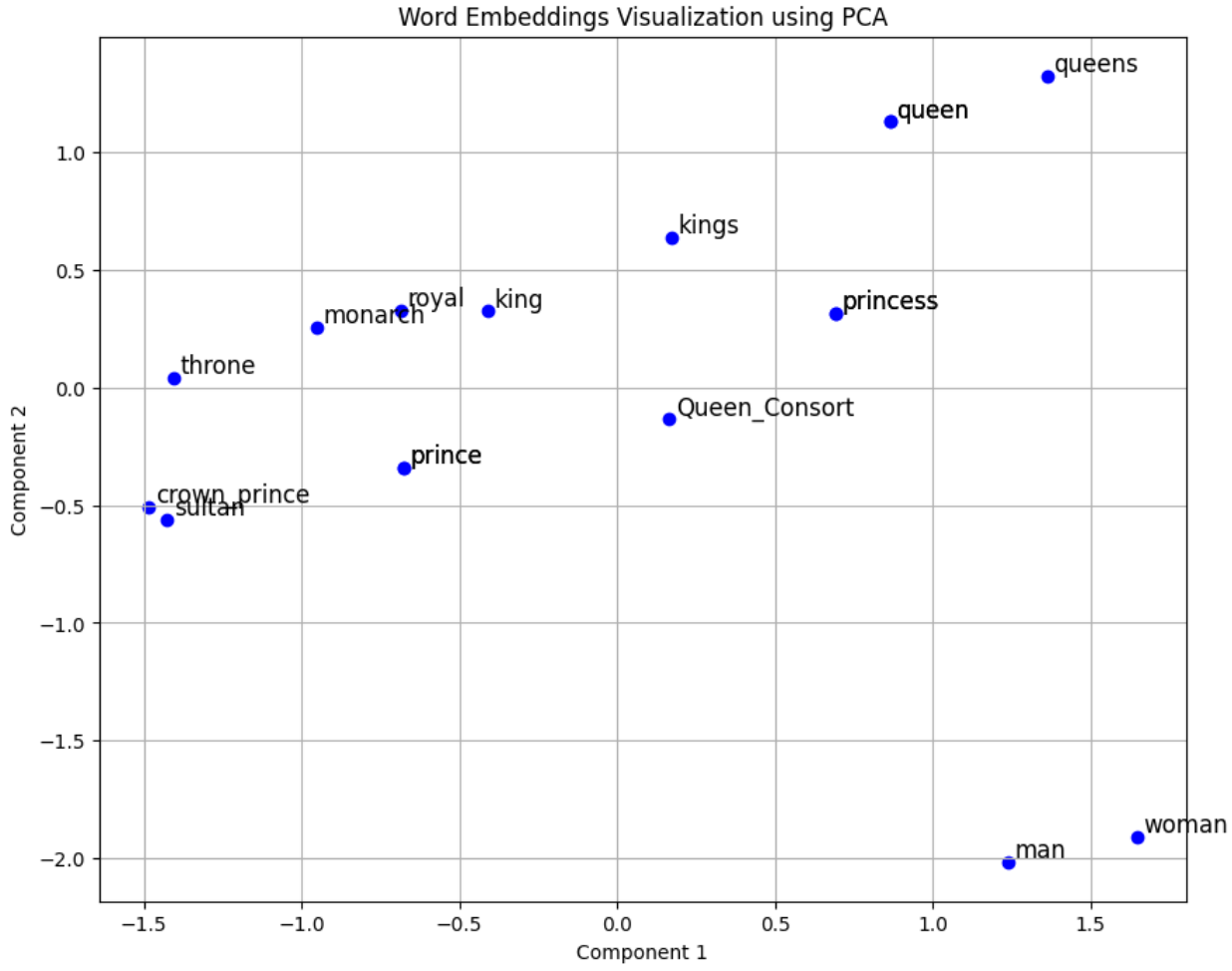
vectors_to_visualize = np.array([word_vectors[word] for word in words_to_visualize if word in word_vectors])

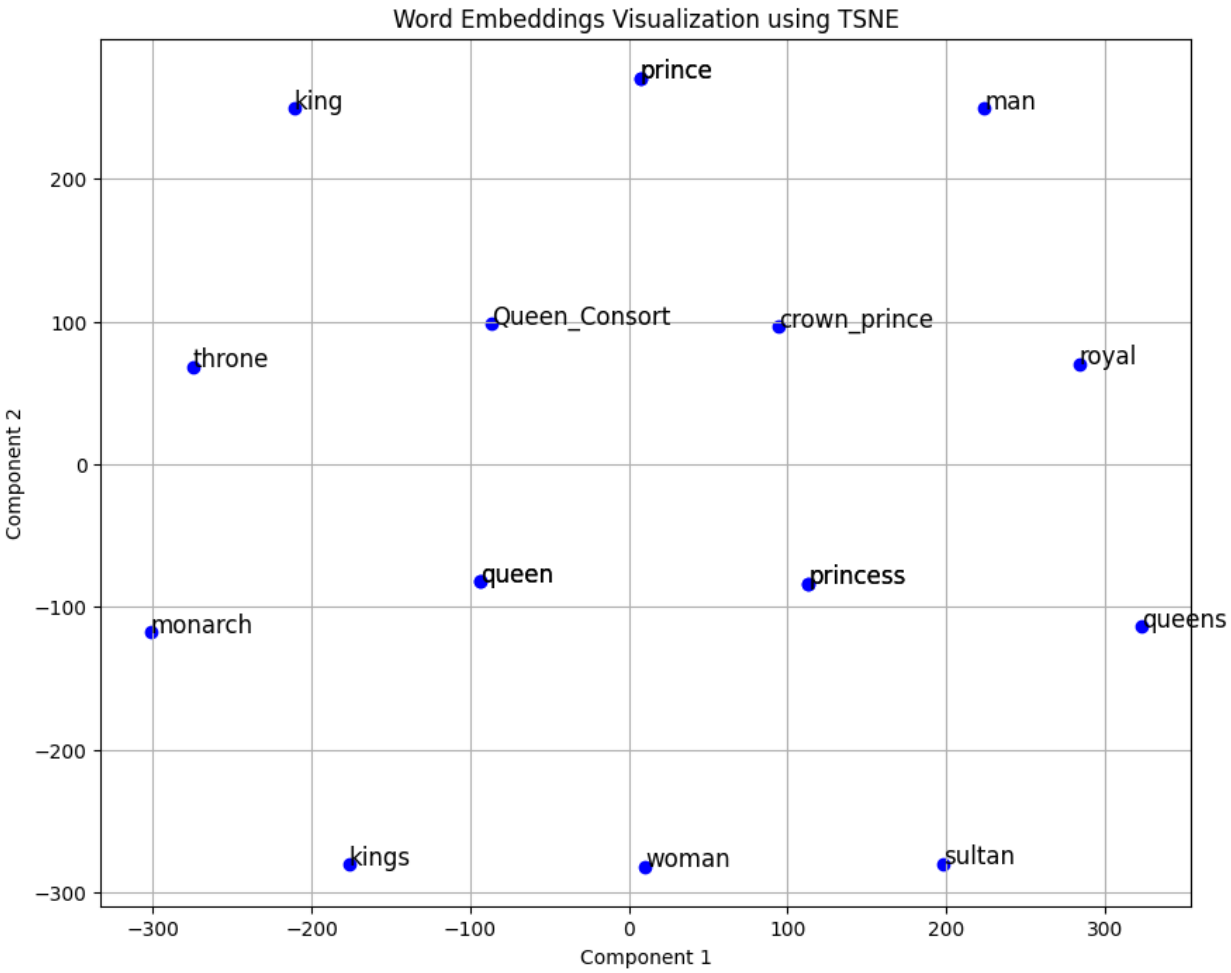
Visualize using PCA

visualize_word_embeddings(words_to_visualize, vectors_to_visualize, method='pca')

Visualize using t-SNE

visualize_word_embeddings(words_to_visualize, vectors_to_visualize, method='tsne')




Word Embeddings Visualization using TSNE
prince
king
man
Queen_Consort
crown_prince
throne
royal
queen
princess
monarch
queens
kings
woman
sultan
Component 2
Component 1
200
100
0
−100
−200
−300
−300
−200
−100
0
100
200
300

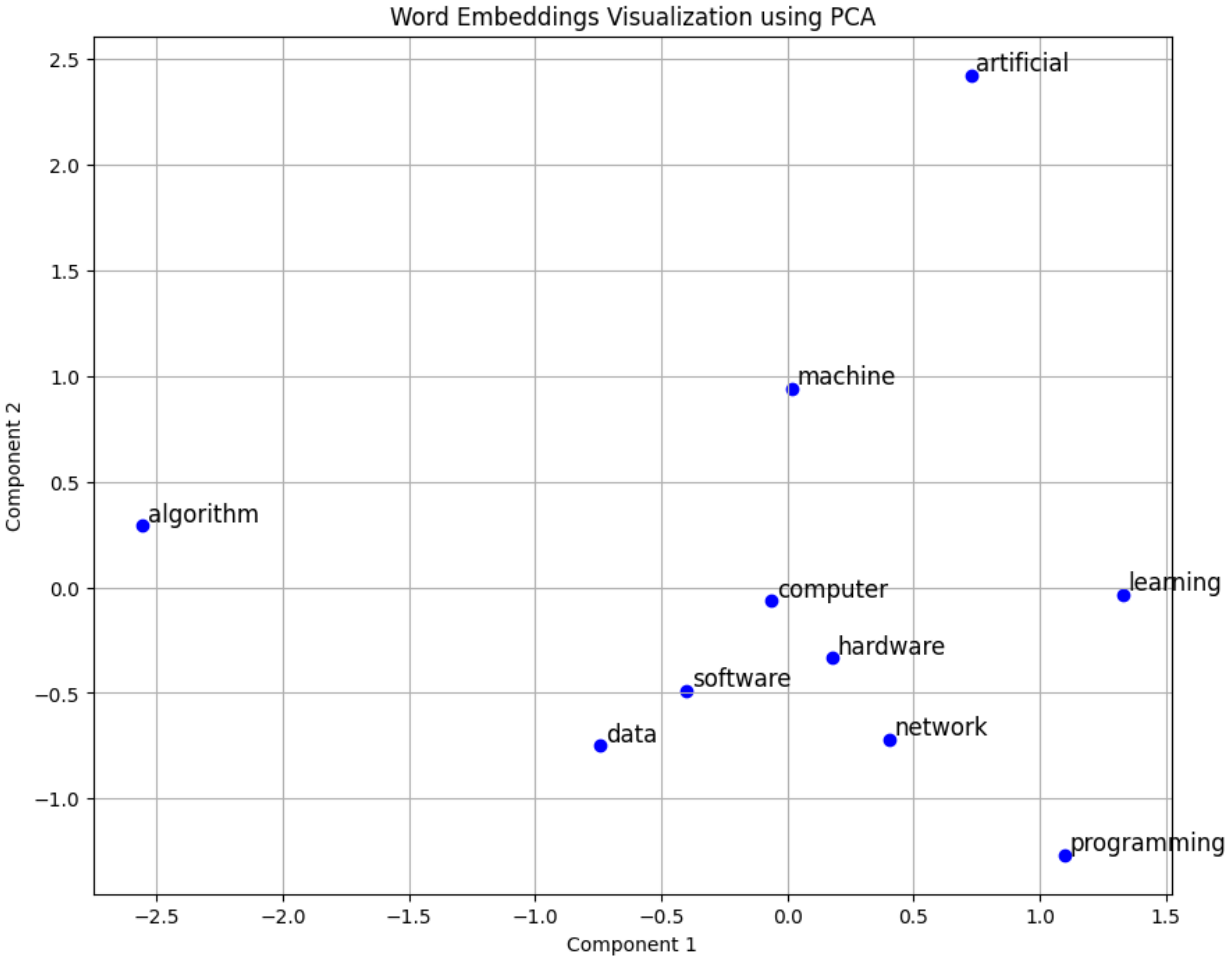
****
**

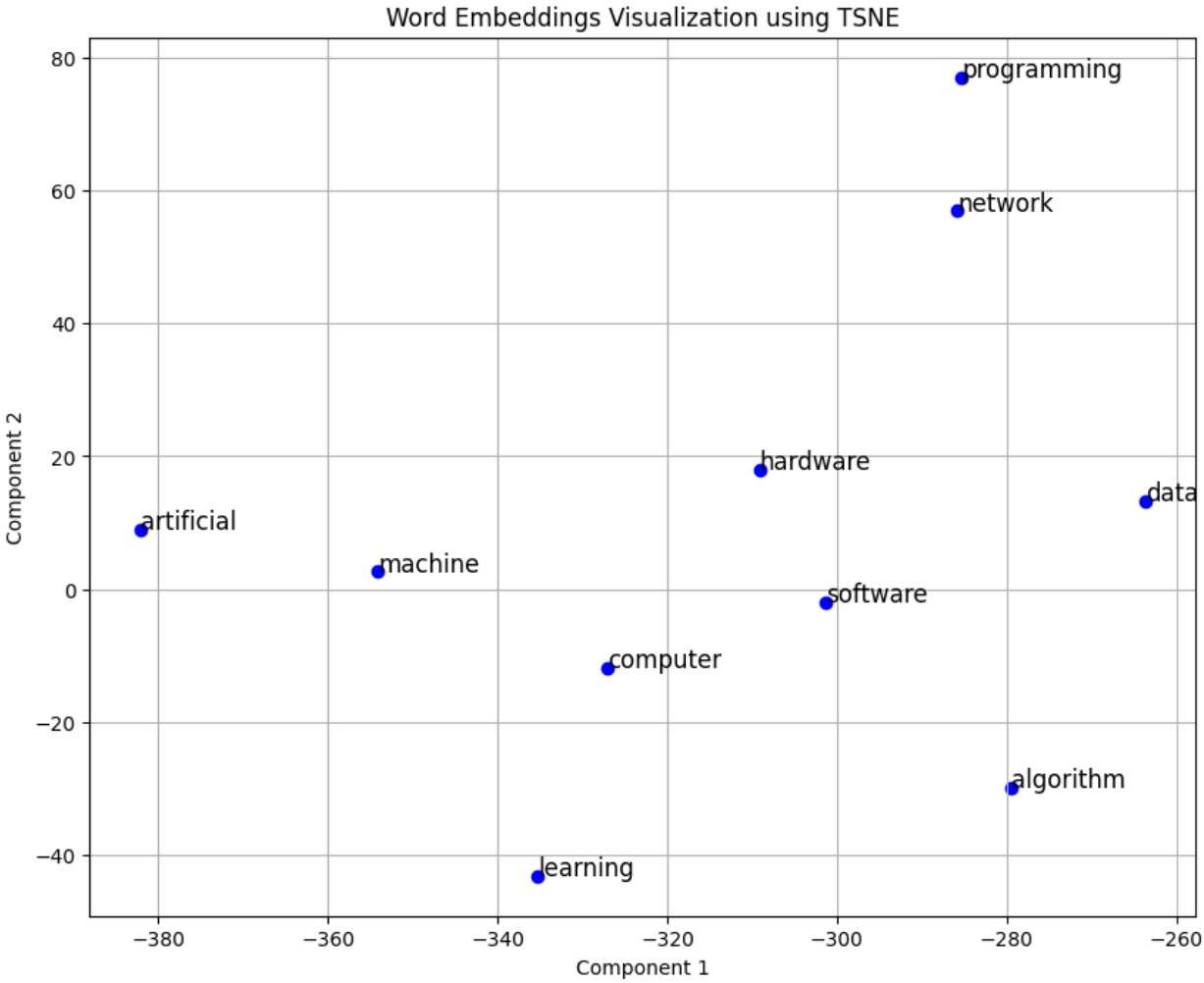
2b. Select 10 words from a specific domain (e.g., sports, technology) and visualize their embeddings. Analyze clusters and relationships. Generate contextually rich outputs using embeddings. Write a program to generate 5 semantically similar words for a given input.

```
# Install required libraries
!pip install gensim scikit-learn matplotlib

# 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

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

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

# Get vectors for the selected words (ensure they exist in the vocabulary)
domain_vectors = np.array([word_vectors[word] for word in domain_words if word in
word_vectors])

# 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()

# Visualize using PCA
visualize_word_embeddings(domain_words, domain_vectors, method='pca')

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

```
# Function to generate 5 semantically similar words
def generate_similar_words(word):
    try:
        if word not in word_vectors:
            raise KeyError(word)

        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.")

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


Word Embeddings Visualization using PCA
artificial
machine
algorithm
computer
learning
hardware
software
data
network
programming
Component 1
Component 2
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Word Embeddings Visualization using TSNE
programming
network
hardware
data
artificial
machine
software
computer
algorithm
learning
Component 1
Component 2
80
60
40
20
0
−20
−40
−380
−360
−340
−320
−300
−280
−260