```
Start coding or generate with AI.
```

```
# Define a list of words for which to find similar words
chosen_words = ["king", "doctor", "cat", "computer", "happy", "water"]

print("\nTop similar words for chosen words:")
# Iterate through each chosen word
for word in chosen_words:
    try:
        # Get the top 5 most similar words from the pre-trained model
        similar_words = model.most_similar(word, topn=5)
        print(f"\nWords similar to '{word}':")
        # Print each similar word and its similarity score
        for s_word, score in similar_words:
            print(f"  {s_word:<15}: {score:.4f}")
    except KeyError as e:
        # Handle cases where a word is not found in the model's vocabulary
        print(f"  Could not find embedding for '{word}': {e}")
```

```
Top similar words for chosen words:

Words similar to 'king':
  prince         : 0.8236
  queen          : 0.7839
  ii             : 0.7746
  emperor        : 0.7736
  son            : 0.7667

Words similar to 'doctor':
  nurse          : 0.7977
  physician      : 0.7965
  patient        : 0.7612
  child          : 0.7559
  teacher        : 0.7538

Words similar to 'cat':
```

```
  dog            : 0.9218
  rabbit         : 0.8488
  monkey         : 0.8041
  rat            : 0.7892
  cats           : 0.7865

Words similar to 'computer':
  computers      : 0.9165
  software       : 0.8815
  technology     : 0.8526
  electronic     : 0.8126
  internet       : 0.8060

Words similar to 'happy':
  'm             : 0.9142
  everyone       : 0.8976
  everybody      : 0.8965
  really         : 0.8840
  me             : 0.8785

Words similar to 'water':
  dry            : 0.8274
  natural        : 0.7858
  sand           : 0.7737
  waste          : 0.7724
  drinking       : 0.7562
```

```
import sys

# Check if gensim is installed, if not, install it
try:
    import gensim.downloader as api
except ImportError:
    print("gensim not found. Installing...")
    %pip install gensim # Install gensim using pip magic command
    import gensim.downloader as api # Import gensim after installation

# Load a pre-trained Word2Vec model
# Using 'glove-wiki-gigaword-50' as it's relatively small and fast to download for demonstration.
# You can choose other models like 'word2vec-google-news-300' for better quality if needed (larger do
print("Loading word embeddings model...")
```

```
model = api.load("glove-wiki-gigaword-50") # Load the specified GloVe model
print("Model loaded successfully.")

# Define a list of word pairs for similarity calculation
word_pairs = [
    ("doctor", "nurse"),
    ("cat", "dog"),
    ("car", "bus"),
    ("king", "queen"),
    ("man", "woman"),
    ("apple", "orange"),
    ("good", "bad"),
    ("happy", "sad"),
    ("sun", "moon"),
    ("tree", "flower"),
    ("computer", "keyboard"),
    ("ocean", "sea")
]

print("\nComputing similarity for word pairs:")
results = [] # Initialize a list to store results
for word1, word2 in word_pairs:
    try:
        # Compute cosine similarity between the two words using the loaded model
        similarity = model.similarity(word1, word2)
        results.append((word1, word2, similarity)) # Store the words and their similarity
        print(f"  {word1:<10} - {word2:<10}: {similarity:.4f}") # Print formatted similarity score
    except KeyError as e:
        # Handle cases where one or both words are not found in the model's vocabulary
        print(f"  Could not find embedding for one of the words in pair ({word1}, {word2}): {e}")

print("\nInterpretation: A higher similarity value indicates that the words are semantically closer."
```

```
Loading word embeddings model...
Model loaded successfully.

Computing similarity for word pairs:
  doctor     - nurse     : 0.7977
  cat        - dog       : 0.9218
```

```
  car        - bus      : 0.8211
  king       - queen    : 0.7839
  man        - woman    : 0.8860
  apple      - orange   : 0.5388
  good       - bad      : 0.7965
  happy      - sad      : 0.6891
  sun        - moon     : 0.6543
  tree       - flower   : 0.7542
  computer   - keyboard : 0.5768
  ocean      - sea      : 0.8812

Interpretation: A higher similarity value indicates that the words are semantically closer.
```

```
# numpy is used for numerical operations, including handling embeddings and similarity calculations.
import numpy as np
# matplotlib.pyplot is used for creating static, interactive, and animated visualizations.
import matplotlib.pyplot as plt
# seaborn is built on matplotlib and provides a high-level interface for drawing attractive and informative st
import seaborn as sns
```

```
# Define a list of analogy queries, each with positive (words to add) and negative (words to subtract) compone
analogy_queries = [
    {
        "positive": ["woman", "king"],
        "negative": ["man"]
    },
    {
        "positive": ["queen", "man"],
        "negative": ["woman"]
    },
    {
        "positive": ["Berlin", "France"],
        "negative": ["Paris"]
    },
    {
        "positive": ["Japan", "Tokyo"],
        "negative": ["China"]
    },
    {
```

```
        "positive": ["big", "small"],
        "negative": ["large"]
    }
]

print("Defined analogy queries:")
# Print each analogy query for verification
for query in analogy_queries:
    print(f"  Positive: {query['positive']}, Negative: {query['negative']}")
```

```
Defined analogy queries:
  Positive: ['woman', 'king'], Negative: ['man']
  Positive: ['queen', 'man'], Negative: ['woman']
  Positive: ['Berlin', 'France'], Negative: ['Paris']
  Positive: ['Japan', 'Tokyo'], Negative: ['China']
  Positive: ['big', 'small'], Negative: ['large']
```

```
print("\nPerforming word analogy queries:")
# Iterate through each defined analogy query
for i, query in enumerate(analogy_queries):
    try:
        # The most_similar function handles the vector arithmetic: positive words are added, negative words ar
        # topn=1 provides the single best answer for the analogy.
        result = model.most_similar(positive=query['positive'], negative=query['negative'], topn=1)
        # Construct a readable string for the analogy question
        print(f"\nAnalogy {i+1}: {query['positive'][1]} - {query['negative'][0]} + {query['positive'][0]} = ?"
        # Print the result (the analogous word) and its similarity score
        print(f"  Result: {result[0][0]:<15}: {result[0][1]:.4f}")
    except KeyError as e:
        # Handle cases where one or more words in the query are not found in the model's vocabulary
        print(f"  Could not find embedding for one of the words in query (Positive: {query['positive']}, Negat
```

```

Performing word analogy queries:

Analogy 1: king - man + woman = ?
  Result: queen          : 0.8524

Analogy 2: man - woman + queen = ?
```

```
  Result: king           : 0.8612
  Could not find embedding for one of the words in query (Positive: ['Berlin', 'France'], Negative: ['Paris']):
  Could not find embedding for one of the words in query (Positive: ['Japan', 'Tokyo'], Negative: ['China']): "

Analogy 5: small - large + big = ?
  Result: like           : 0.8611
```

```
print("Revising analogy queries to ensure all words are in the model's vocabulary and to improve analogy relev
# Revise the list of analogy queries with words more likely to be in the model's vocabulary
analogy_queries = [
    {
        "positive": ["woman", "king"],
        "negative": ["man"]
    }, # king - man + woman = queen (gender analogy)
    {
        "positive": ["queen", "man"],
        "negative": ["woman"]
    }, # queen - woman + man = king (gender analogy)
    {
        "positive": ["Paris", "Germany"],
        "negative": ["France"]
    }, # Germany - France + Paris = Berlin (capital-country relation) - often problematic with smaller models
    {
        "positive": ["Tokyo", "China"],
        "negative": ["Japan"]
    }, # China - Japan + Tokyo = Beijing (capital-country relation) - often problematic with smaller models
    {
        "positive": ["walking", "swim"],
        "negative": ["walk"]
    }  # swim - walk + walking = swimming (verb tense analogy)
]

print("Updated analogy queries:")
# Print the updated analogy queries for verification
for query in analogy_queries:
    print(f"  Positive: {query['positive']}, Negative: {query['negative']}")
```

```
Revising analogy queries to ensure all words are in the model's vocabulary and to improve analogy relevance.
Updated analogy queries:
  Positive: ['woman', 'king'], Negative: ['man']
  Positive: ['queen', 'man'], Negative: ['woman']
  Positive: ['Paris', 'Germany'], Negative: ['France']
  Positive: ['Tokyo', 'China'], Negative: ['Japan']
  Positive: ['walking', 'swim'], Negative: ['walk']
```

```
print("\nPerforming word analogy queries with revised list:")
for i, query in enumerate(analogy_queries):
    try:
        # The most_similar function handles the vector arithmetic: positive words are added, negative words ar
        # topn=1 provides the single best answer for the analogy.
        result = model.most_similar(positive=query['positive'], negative=query['negative'], topn=1)
        # Constructing the analogy string based on the 'positive' and 'negative' words for better readability
        analogy_str = f"{query['positive'][1]} - {query['negative'][0]} + {query['positive'][0]} = ?"
        print(f"\nAnalogy {i+1}: {analogy_str}")
        print(f"  Result: {result[0][0]:<15}: {result[0][1]:.4f}")
    except KeyError as e:
        print(f"  Could not find embedding for one of the words in query (Positive: {query['positive']}, Negat
```

```
Performing word analogy queries with revised list:

Analogy 1: king - man + woman = ?
  Result: queen          : 0.8524

Analogy 2: man - woman + queen = ?
  Result: king           : 0.8612
  Could not find embedding for one of the words in query (Positive: ['Paris', 'Germany'], Negative: ['France'])
  Could not find embedding for one of the words in query (Positive: ['Tokyo', 'China'], Negative: ['Japan']): "

Analogy 5: swim - walk + walking = ?
  Result: swimming       : 0.8072
```

```
print("Further revising analogy queries with commonly found words to ensure successful execution.")
# Further revise analogy queries to use more common words and robust linguistic relationships
analogy_queries = [
```

```
    {
        "positive": ["woman", "king"],
        "negative": ["man"]
    }, # king - man + woman = queen (gender analogy)
    {
        "positive": ["queen", "man"],
        "negative": ["woman"]
    }, # queen - woman + man = king (gender analogy)
    {
        "positive": ["sister", "man"],
        "negative": ["woman"]
    }, # man - woman + sister = brother (familial relation, gender analogy)
    {
        "positive": ["smaller", "big"],
        "negative": ["small"]
    }, # big - small + smaller = bigger (comparative adjective analogy)
    {
        "positive": ["running", "go"],
        "negative": ["run"]
    }  # go - run + running = going (verb tense analogy)
]

print("Final updated analogy queries:")
# Print the final updated analogy queries for verification
for query in analogy_queries:
    print(f"  Positive: {query['positive']}, Negative: {query['negative']}")
```

```
Further revising analogy queries with commonly found words to ensure successful execution.
Final updated analogy queries:
  Positive: ['woman', 'king'], Negative: ['man']
  Positive: ['queen', 'man'], Negative: ['woman']
  Positive: ['sister', 'man'], Negative: ['woman']
  Positive: ['smaller', 'big'], Negative: ['small']
  Positive: ['running', 'go'], Negative: ['run']
```

```
print("\nPerforming word analogy queries with final revised list:")
# Iterate through each analogy query in the final revised list
for i, query in enumerate(analogy_queries):
    try:
```

```
        # The most_similar function handles the vector arithmetic: positive words are added, negative words ar
        # topn=1 provides the single best answer for the analogy.
        result = model.most_similar(positive=query['positive'], negative=query['negative'], topn=1)
        # Construct a readable string for the analogy question
        analogy_str = f"{query['positive'][1]} - {query['negative'][0]} + {query['positive'][0]} = ?"
        print(f"\nAnalogy {i+1}: {analogy_str}")
        # Print the result (the analogous word) and its similarity score
        print(f"  Result: {result[0][0]:<15}: {result[0][1]:.4f}")
    except KeyError as e:
        # Handle cases where one or more words in the query are not found in the model's vocabulary
        print(f"  Could not find embedding for one of the words in query (Positive: {query['positive']}, Negat
```

```
Performing word analogy queries with final revised list:

Analogy 1: king - man + woman = ?
  Result: queen          : 0.8524

Analogy 2: man - woman + queen = ?
  Result: king           : 0.8612

Analogy 3: man - woman + sister = ?
  Result: friend         : 0.8550

Analogy 4: big - small + smaller = ?
  Result: bigger         : 0.8704

Analogy 5: go - run + running = ?
  Result: get            : 0.8915
```

```
# Define a list of diverse words chosen for visualization purposes
# These words cover various semantic categories like royalty, professions, animals, emotions, objects, and act
visualization_words = [
    "king", "queen", "man", "woman", "prince", "princess",
    "doctor", "nurse", "teacher", "engineer", "artist", "student",
    "cat", "dog", "bird", "fish", "elephant", "lion",
    "happy", "sad", "angry", "joy", "love", "hate",
    "computer", "phone", "car", "house", "book", "chair",
    "run", "walk", "eat", "sleep", "read", "write",
```

```
    "Paris", "London", "NewYork", "Tokyo", "Rome", "Beijing"
]

# Print the number of selected words and the list itself
print(f"Selected words for visualization ({len(visualization_words)} words):")
print(visualization_words)
```

```
Selected words for visualization (42 words):
['king', 'queen', 'man', 'woman', 'prince', 'princess', 'doctor', 'nurse', 'teacher', 'engineer', 'artist', 'st
```

```
# Revise the list of words for visualization to fit within the 20-30 word range, while maintaining diversity.
visualization_words = [
    "king", "queen", "man", "woman",
    "doctor", "nurse", "teacher",
    "cat", "dog", "bird",
    "happy", "sad", "angry",
    "computer", "phone", "car",
    "run", "walk", "eat", "sleep",
    "Paris", "London", "Tokyo"
]

# Print the number of revised words and the updated list
print(f"Revised selected words for visualization ({len(visualization_words)} words):")
print(visualization_words)
```

```
Revised selected words for visualization (23 words):
['king', 'queen', 'man', 'woman', 'doctor', 'nurse', 'teacher', 'cat', 'dog', 'bird', 'happy', 'sad', 'angry',
```

```
word_vectors = [] # List to store the numerical vector embeddings
words_found = []    # List to store words successfully found in the model's vocabulary

print("Retrieving word embeddings for visualization words...")
# Iterate through each word selected for visualization
for word in visualization_words:
    try:
        # Retrieve the vector embedding for the word from the pre-trained model
        vec = model[word]
```

```
            word_vectors.append(vec) # Add the vector to the list
            words_found.append(word)   # Add the word to the list of found words
        except KeyError:
            # If a word is not found in the model's vocabulary, print a warning and skip it
            print(f"  Warning: Word '{word}' not found in model's vocabulary. Skipping.")

# Convert the list of word vectors into a NumPy array for efficient numerical operations
word_vectors_array = np.array(word_vectors)

# Print summary information about the retrieved embeddings
print(f"Successfully retrieved embeddings for {len(words_found)} out of {len(visualization_words)} words.")
print("Shape of word_vectors_array:", word_vectors_array.shape) # Show the dimensions of the resulting array
```

```
Retrieving word embeddings for visualization words...
  Warning: Word 'Paris' not found in model's vocabulary. Skipping.
  Warning: Word 'London' not found in model's vocabulary. Skipping.
  Warning: Word 'Tokyo' not found in model's vocabulary. Skipping.
Successfully retrieved embeddings for 20 out of 23 words.
Shape of word_vectors_array: (20, 50)
```

```
from sklearn.decomposition import PCA # Import Principal Component Analysis for linear dimensionality reductio
from sklearn.manifold import TSNE     # Import t-Distributed Stochastic Neighbor Embedding for non-linear dime

print("Imported PCA and TSNE for dimensionality reduction.")
```

```
Imported PCA and TSNE for dimensionality reduction.
```

```
print("Applying t-SNE for dimensionality reduction...")

# Instantiate t-SNE model
# n_components=2 for projecting data into a 2-dimensional space
# random_state is set for reproducibility of the results
# perplexity is a crucial parameter, typically between 5 and 50, and should be less than the number of samples
# Here, it's set to 5, which is suitable for 20 samples.
tsne = TSNE(n_components=2, random_state=42, perplexity=5)

# Fit the t-SNE model to the high-dimensional word vectors and transform them to 2D
word_vectors_2d = tsne.fit_transform(word_vectors_array)
```

```
# Print confirmation and the shape of the resulting 2D array
print("Dimensionality reduction complete. Shape of word_vectors_2d:", word_vectors_2d.shape)
```

```
Applying t-SNE for dimensionality reduction...
Dimensionality reduction complete. Shape of word_vectors_2d: (20, 2)
```

```
print("Plotting word embeddings...")

# Create a new figure with a specified size for better visualization
plt.figure(figsize=(12, 12))
# Create a scatter plot of the 2D word embeddings
sns.scatterplot(
    x=word_vectors_2d[:, 0], y=word_vectors_2d[:, 1], # X and Y coordinates from the 2D array
    hue=words_found,  # Color each point based on the corresponding word for easy identification
    palette='tab20', # Use a distinct color palette to differentiate words
    s=100, # Set the size of the scatter plot points
    legend='full' # Display the full legend showing all words
)

# Annotate each point in the scatter plot with its corresponding word
# The offset (0.1, 0.1) is added to prevent text from overlapping with the points
for i, word in enumerate(words_found):
    plt.annotate(word, (word_vectors_2d[i, 0] + 0.1, word_vectors_2d[i, 1] + 0.1), fontsize=9)

# Set the title and labels for the plot for clarity
plt.title('2D t-SNE Projection of Word Embeddings', fontsize=16)
plt.xlabel('t-SNE Dimension 1', fontsize=12)
plt.ylabel('t-SNE Dimension 2', fontsize=12)
plt.grid(True) # Add a grid to the plot for better readability
plt.show() # Display the generated plot

print("Word embeddings plot displayed.")
```

```
Plotting word embeddings...
```