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

```
!pip install gensim
```

```
Collecting gensim
  Downloading gensim-4.4.0-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: smart_open>=1.8.1 in /usr/local/lib/python3.12/di
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/dist-packages
Downloading gensim-4.4.0-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 27.9/27.9 MB 67.2 MB/s eta 0:00:00
Installing collected packages: gensim
Successfully installed gensim-4.4.0
```

```python
import gensim
import numpy as np
from scipy.spatial.distance import cosine
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

print("Libraries imported successfully.")
```

```
Libraries imported successfully.
```

```python
import gensim.downloader as api

print("Downloading and loading pre-trained GloVe model...")
word_vectors = api.load("glove-wiki-gigaword-50")
print("Model loaded successfully.")

# 2. Print the total number of words in the vocabulary
vocabulary_size = len(word_vectors.key_to_index)
print(f"Vocabulary size: {vocabulary_size}")

# 3. Choose 3-5 diverse words
example_words = ['king', 'woman', 'computer', 'apple', 'river']

# 4. For each chosen word, retrieve its embedding vector and print it
print("\nExample word vectors:")
for word in example_words:
    if word in word_vectors.key_to_index:
        vector = word_vectors[word]
        print(f"Word: '{word}'\nVector: {vector[:5]}...{vector[-5:]} (first anc
    else:
        print(f"Word: '{word}' not found in vocabulary.\n")
```

```
Downloading and loading pre-trained GloVe model...
[==================================================] 100.0% 66.0/66.0MB download
Model loaded successfully.
Vocabulary size: 400000

Example word vectors:
Word: 'king'
Vector: [ 0.50451   0.68607  -0.59517  -0.022801  0.60046 ]...[ 0.78321 -0.91241

Word: 'woman'
Vector: [-0.18153  0.64827 -0.5821  -0.49451  1.5415 ]...[-0.85223 -0.23073 -1.3

Word: 'computer'
Vector: [ 0.079084 -0.81504   1.7901    0.91653   0.10797 ]...[ 0.50697  1.2357

Word: 'apple'
Vector: [ 0.52042 -0.8314   0.49961  1.2893   0.1151 ]...[ 0.97391  0.40027 -0.2

Word: 'river'
Vector: [ 0.73109  1.0242  -0.26714 -0.33449 -1.4873 ]...[-0.62487  0.26281 -1.4
```

```
print("\nComputing cosine similarity for word pairs...")

# 1. Create a list of at least 10 diverse word pairs
word_pairs = [
    ('king', 'queen'),
    ('man', 'woman'),
    ('good', 'bad'),
    ('cat', 'dog'),
    ('apple', 'fruit'),
    ('car', 'automobile'),
    ('run', 'walk'),
    ('computer', 'keyboard'),
    ('happy', 'joyful'),
    ('tree', 'car'), # Less obvious semantic relationship
    ('water', 'ocean')
]

# 2-5. Iterate through each word pair, check existence, calculate similarity, a
print("\nWord Pair Similarities:")
for word1, word2 in word_pairs:
    if word1 in word_vectors.key_to_index and word2 in word_vectors.key_to_inde
        similarity = word_vectors.similarity(word1, word2)
        print(f"  ('{word1}', '{word2}'): {similarity:.4f}")
    else:
        missing_words = []
        if word1 not in word_vectors.key_to_index:
            missing_words.append(word1)
        if word2 not in word_vectors.key_to_index:
            missing_words.append(word2)
            print(f"  Skipping ('{word1}', '{word2}'): Word(s) '{', '.join(missing_

    print("Similarity computation complete.")
```

```
Computing cosine similarity for word pairs...

Word Pair Similarities:
  ('king', 'queen'): 0.7839
  ('man', 'woman'): 0.8860
  ('good', 'bad'): 0.7965
  ('cat', 'dog'): 0.9218
  ('apple', 'fruit'): 0.5918
  ('car', 'automobile'): 0.6956
  ('run', 'walk'): 0.6831
  ('computer', 'keyboard'): 0.5768
  ('happy', 'joyful'): 0.5550
  ('tree', 'car'): 0.3753
  ('water', 'ocean'): 0.7453
Similarity computation complete.
```

```
print("\nExploring nearest neighbors for chosen words...")

# 1. Create a list of at least 5 diverse words
target_words = ['king', 'computer', 'fast', 'love', 'city']

# 2-4. Iterate through each word, find most similar, and print
for word in target_words:
    print(f"\nNearest neighbors for '{word}':")
    if word in word_vectors.key_to_index:
        try:
            similar_words = word_vectors.most_similar(word, topn=10)
            for similar_word, score in similar_words:
                print(f"  - {similar_word}: {score:.4f}")
        except KeyError:
            print(f"  Could not find similar words for '{word}'.")
    else:
        print(f"  '{word}' not found in vocabulary.")

print("Nearest neighbor exploration complete.")
```

```
  - queen: 0.7839
  - ii: 0.7746
  - emperor: 0.7736
  - son: 0.7667
  - uncle: 0.7627
  - kingdom: 0.7542
  - throne: 0.7540
```

```
    - software: 0.8815
    - technology: 0.8526
    - electronic: 0.8126
    - internet: 0.8060
    - computing: 0.8026
    - devices: 0.8016
    - digital: 0.7992
    - applications: 0.7913
    - pc: 0.7883

  Nearest neighbors for 'fast':
    - slow: 0.8744
    - faster: 0.8049
    - pace: 0.8023
    - turning: 0.7816
    - better: 0.7752
    - easy: 0.7750
    - turn: 0.7707
    - way: 0.7650
    - catch: 0.7645
    - coming: 0.7642

  Nearest neighbors for 'love':
    - dream: 0.8430
    - life: 0.8403
    - dreams: 0.8399
    - loves: 0.8361
    - me: 0.8352
    - my: 0.8227
    - mind: 0.8218
    - loving: 0.8108
    - wonder: 0.8073
    - soul: 0.8015

  Nearest neighbors for 'city':
    - town: 0.8688
    - downtown: 0.8534
    - where: 0.8525
    - cities: 0.8505
    - area: 0.8322
    - in: 0.8228
    - outside: 0.8224
    - near: 0.8144
    - central: 0.8133
    - nearby: 0.7948
  Nearest neighbor exploration complete.
```

```
print("\nPerforming word analogy tasks...")

# 1. Define a list of at least 3-5 analogy examples
analogy_examples = [
    {'positive': ['king', 'woman'], 'negative': ['man'], 'query_str': 'king - m
    {'positive': ['paris', 'india'], 'negative': ['france'], 'query_str': 'pari
    {'positive': ['walk', 'ran'], 'negative': ['walked'], 'query_str': 'walk - 
    {'positive': ['spain', 'madrid'], 'negative': ['france'], 'query_str': 'spa
    {'positive': ['tall', 'taller'], 'negative': ['short'], 'query_str': 'tall 
```

```
]

# 2-3. Iterate through each analogy, solve it, and print the results
for analogy in analogy_examples:
    positive_words = analogy['positive']
    negative_words = analogy['negative']
    query_str = analogy['query_str']

    print(f"\nAnalogy: {query_str} = ?")
    try:
        # Check if all words are in the vocabulary first
        all_words_present = True
        for word in positive_words + negative_words:
            if word not in word_vectors.key_to_index:
                print(f"  Skipping analogy: word '{word}' not found in vocabula
                all_words_present = False
                break

        if all_words_present:
            result = word_vectors.most_similar(positive=positive_words, negativ
            if result:
                analogy_word, score = result[0]
                print(f"  Result: '{analogy_word}' (similarity: {score:.4f})")
            else:
                print("  No result found for this analogy.")
    except KeyError as e:
        print(f"  An error occurred: {e}")

print("Word analogy tasks complete.")
```

```
Performing word analogy tasks...

Analogy: king - man + woman = ?
  Result: 'queen' (similarity: 0.8524)

Analogy: paris - france + india = ?
  Result: 'delhi' (similarity: 0.8889)

Analogy: walk - walked + ran = ?
  Result: 'running' (similarity: 0.8324)

Analogy: spain - france + madrid = ?
  Result: 'valencia' (similarity: 0.8636)

Analogy: tall - short + taller = ?
  Result: '7-feet' (similarity: 0.6891)
Word analogy tasks complete.
```

```
print("\nPreparing data for visualization...")

# 1. Choose a diverse set of words for visualization
# Include some related words and some unrelated words to see clustering
words_to_visualize = [
    'king', 'queen', 'man', 'woman', 'prince', 'princess',
    'paris', 'france', 'london', 'england', 'rome', 'italy',
    'apple', 'orange', 'banana', 'fruit', 'vegetable',
    'dog', 'cat', 'animal', 'pet', 'wolf',
    'run', 'walk', 'jump', 'move', 'fast',
    'computer', 'software', 'technology', 'keyboard', 'mouse',
    'happy', 'sad', 'joyful', 'angry', 'emotion'
]

# 2. Extract vectors for these words, handling words not in vocabulary
vectors_for_visualization = []
labels_for_visualization = []

for word in words_to_visualize:
    if word in word_vectors.key_to_index:
        vectors_for_visualization.append(word_vectors[word])
        labels_for_visualization.append(word)
    else:
        print(f"Warning: Word '{word}' not found in vocabulary, skipping.")

# Convert list of vectors to a numpy array
X = np.array(vectors_for_visualization)

print(f"Prepared {len(labels_for_visualization)} words for visualization.")
print(f"Shape of vectors array (X): {X.shape}")
```

```
Preparing data for visualization...
Prepared 37 words for visualization.
Shape of vectors array (X): (37, 50)
```

```
print("\nApplying PCA for dimensionality reduction...")

pca = PCA(n_components=2) # Reduce to 2 dimensions for 2D plotting
X_pca = pca.fit_transform(X)

print("PCA dimensionality reduction complete.")
print(f"Shape of PCA reduced vectors (X_pca): {X_pca.shape}")
```

```
Applying PCA for dimensionality reduction...
PCA dimensionality reduction complete.
Shape of PCA reduced vectors (X_pca): (37, 2)
```

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

tsne = TSNE(n_components=2, random_state=42) # Reduce to 2 dimensions for 2D pl
X_tsne = tsne.fit_transform(X)
```

```
print("t-SNE dimensionality reduction complete.")
print(f"Shape of t-SNE reduced vectors (X_tsne): {X_tsne.shape}")
```

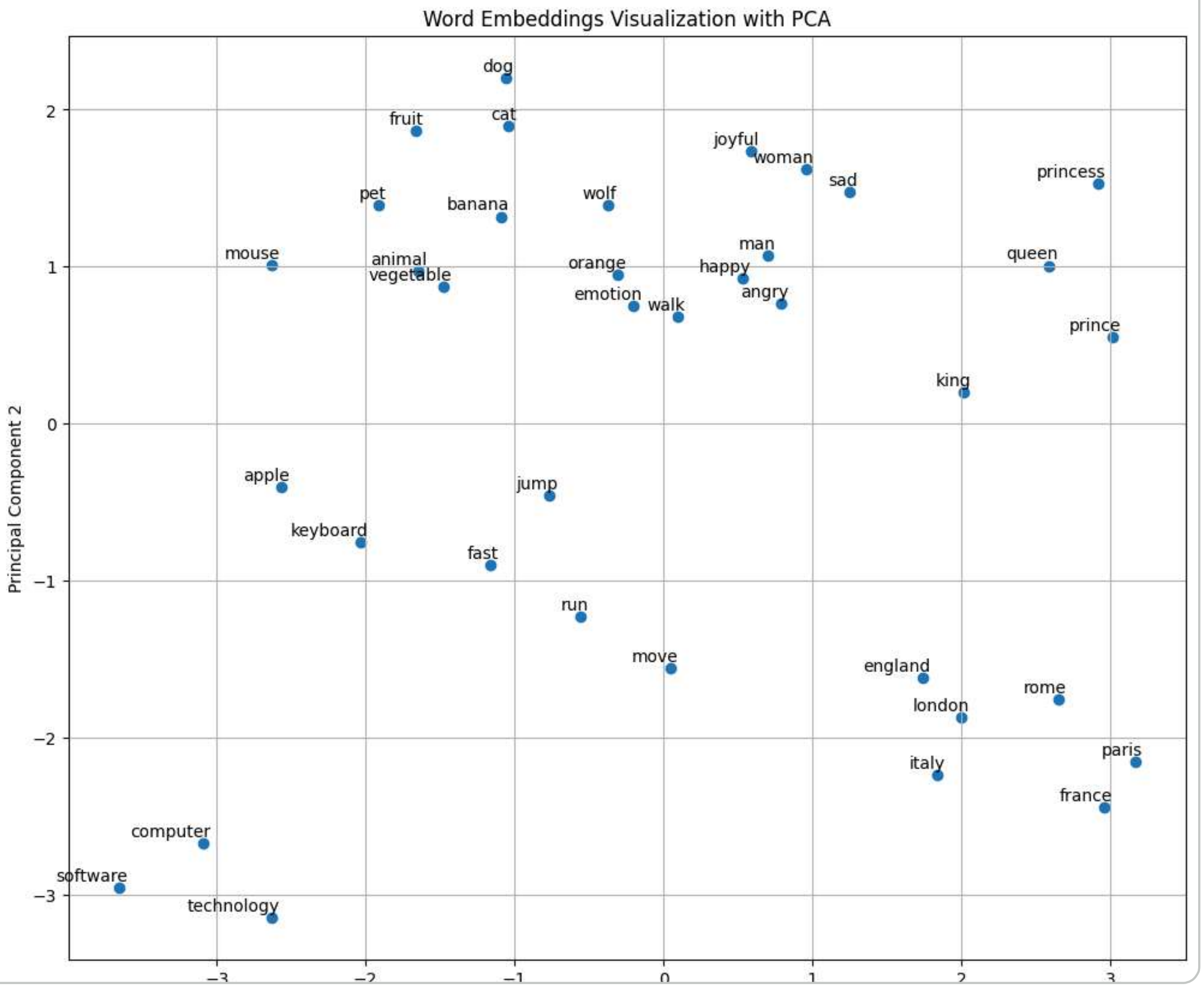
```
Applying t-SNE for dimensionality reduction...
t-SNE dimensionality reduction complete.
Shape of t-SNE reduced vectors (X_tsne): (37, 2)
```

```
print("\nVisualizing word embeddings with PCA...")

plt.figure(figsize=(12, 10))
plt.scatter(X_pca[:, 0], X_pca[:, 1])
for i, word in enumerate(labels_for_visualization):
    plt.annotate(word, xy=(X_pca[i, 0], X_pca[i, 1]), xytext=(5, 2), textcoords
plt.title('Word Embeddings Visualization with PCA')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.show()

print("PCA visualization complete.")
```

```
Visualizing word embeddings with PCA...
```

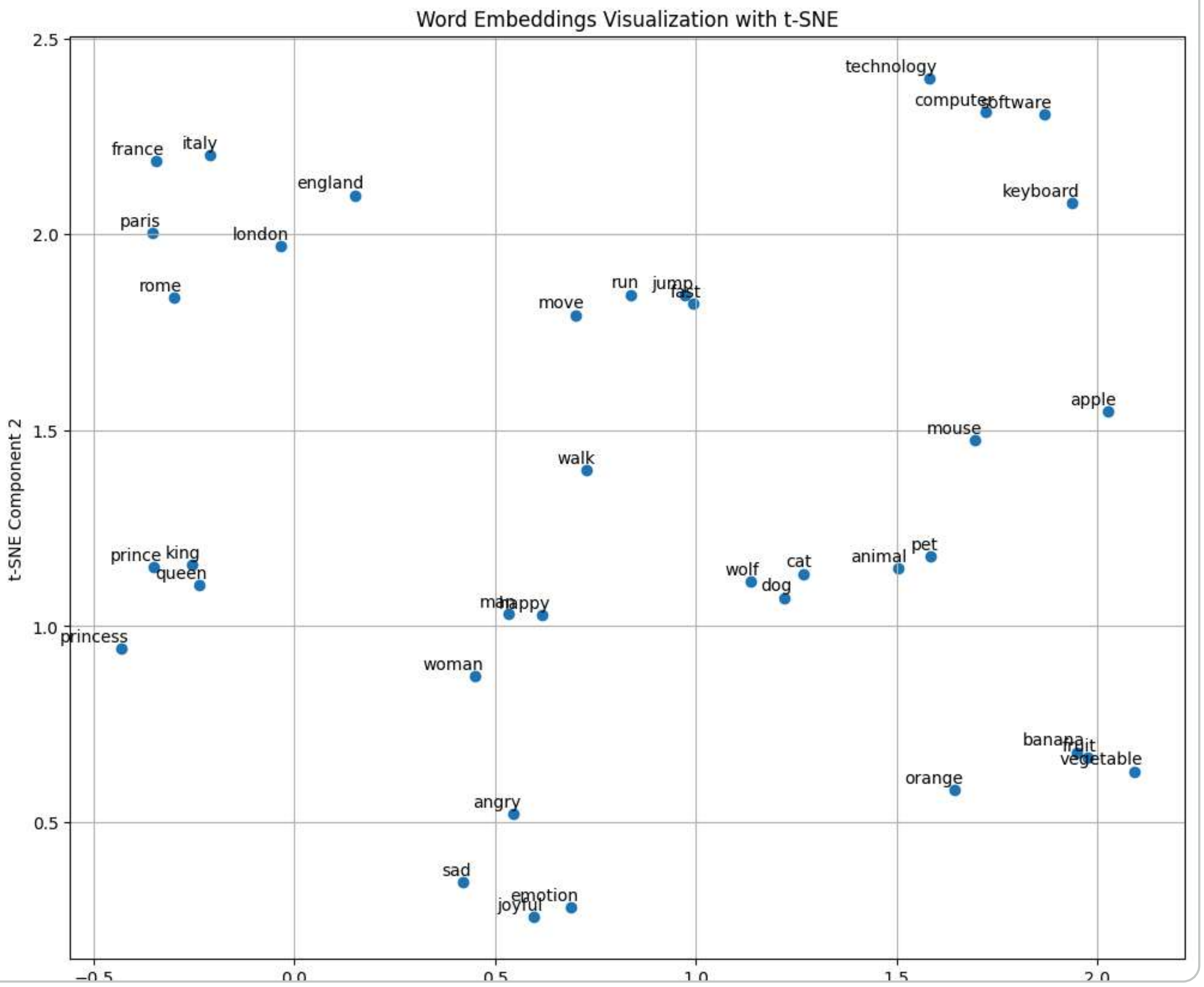


```
print("\nVisualizing word embeddings with t-SNE...")

plt.figure(figsize=(12, 10))
plt.scatter(X_tsne[:, 0], X_tsne[:, 1])
for i, word in enumerate(labels_for_visualization):
    plt.annotate(word, xy=(X_tsne[i, 0], X_tsne[i, 1]), xytext=(5, 2), textcoor
plt.title('Word Embeddings Visualization with t-SNE')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.grid(True)
plt.show()

print("t-SNE visualization complete.")
```

```
Visualizing word embeddings with t-SNE...
```


Word Embeddings Visualization with t-SNE
t-SNE Component 2
technology
computer
software
france
italy
england
keyboard
paris
london
rome
run
jump
fast
move
apple
mouse
walk
prince
king
queen
wolf
cat
dog
animal
pet
man
happy
princess
woman
banana
fruit
vegetable
orange
angry
sad
emotion
joyful