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

```
print("\nExploring Word Analogies:")

analogy_queries = [
    {'positive': ['woman', 'king'], 'negative': ['man'], 'name': 'king - man + woman'},
    {'positive': ['paris', 'india'], 'negative': ['france'], 'name': 'paris - france + india'},
    {'positive': ['doctor', 'school'], 'negative': ['teacher'], 'name': 'teacher - school + doctor
    {'positive': ['brother', 'sister'], 'negative': ['man'], 'name': 'brother - man + sister'},
    {'positive': ['japan', 'sushi'], 'negative': ['germany'], 'name': 'japan - germany + sushi'}
]

for query in analogy_queries:
    pos = query['positive']
    neg = query['negative']
    query_name = query['name']

    try:
        # The most_similar method takes positive and negative words to solve analogies
        result = word_vectors.most_similar(positive=pos, negative=neg, topn=1)
        print(f"\nAnalogy '{query_name} = ?':")
        for word, similarity in result:
            print(f"  - {word}: {similarity:.4f}")
    except KeyError as e:
        print(f"\nError for '{query_name}': One or more words not found in vocabulary: {e}")
```

```
Exploring Word Analogies:

Analogy 'king - man + woman = ?':
  - queen: 0.7118

Analogy 'paris - france + india = ?':
  - chennai: 0.5443

Analogy 'teacher - school + doctor = ?':
  - doctors: 0.6076

Analogy 'brother - man + sister = ?':
  - siblings: 0.6377

Analogy 'japan - germany + sushi = ?':
  - Sushi: 0.6540
```

```
print("\nExploring Nearest Neighbors (Most Similar Words):")

chosen_words = ['king', 'university', 'cat', 'running', 'beautiful', 'doctor']

for word in chosen_words:
    if word in word_vectors:
        print(f"\nTop 5 similar words to '{word}':")
        for similar_word, similarity in word_vectors.most_similar(word, topn=5):
            print(f"  - {similar_word}: {similarity:.4f}")
    else:
        print(f"\n'{word}' not found in vocabulary.")
```

```
Exploring Nearest Neighbors (Most Similar Words):
```

```
Top 5 similar words to 'king':
  - kings: 0.7138
  - queen: 0.6511
  - monarch: 0.6413
  - crown_prince: 0.6204
  - prince: 0.6160

Top 5 similar words to 'university':
  - universities: 0.7004
  - faculty: 0.6781
  - unversity: 0.6758
  - undergraduate: 0.6587
  - univeristy: 0.6585

Top 5 similar words to 'cat':
  - cats: 0.8099
  - dog: 0.7609
  - kitten: 0.7465
  - feline: 0.7326
  - beagle: 0.7151

Top 5 similar words to 'running':
  - Running: 0.6979
  - ran: 0.6085
  - run: 0.6063
  - runnning: 0.5533
  - runing: 0.5427

Top 5 similar words to 'beautiful':
  - gorgeous: 0.8353
  - lovely: 0.8107
  - stunningly_beautiful: 0.7329
  - breathtakingly_beautiful: 0.7231
  - wonderful: 0.6854

Top 5 similar words to 'doctor':
  - physician: 0.7806
  - doctors: 0.7477
  - gynecologist: 0.6948
  - surgeon: 0.6793
  - dentist: 0.6785
```

```
print("\nExploring Word Similarity:")

word_pairs = [
    ('doctor', 'nurse'),
    ('cat', 'dog'),
    ('car', 'bus'),
    ('king', 'queen'),
    ('man', 'woman'),
    ('good', 'bad'),
    ('happy', 'joyful'),
    ('apple', 'fruit'),
    ('tree', 'forest'),
    ('fast', 'slow'),
    ('sun', 'moon'),
    ('day', 'night')
]

for word1, word2 in word_pairs:
    if word1 in word_vectors and word2 in word_vectors:
        similarity = word_vectors.similarity(word1, word2)
        print(f"Similarity between '{word1}' and '{word2}': {similarity:.4f}")
```

```
    else:
        print(f"One or both words ('{word1}', '{word2}') not found in vocabulary.")
```

```
Exploring Word Similarity:
Similarity between 'doctor' and 'nurse': 0.6320
Similarity between 'cat' and 'dog': 0.7609
Similarity between 'car' and 'bus': 0.4693
Similarity between 'king' and 'queen': 0.6511
Similarity between 'man' and 'woman': 0.7664
Similarity between 'good' and 'bad': 0.7190
Similarity between 'happy' and 'joyful': 0.4238
Similarity between 'apple' and 'fruit': 0.6410
Similarity between 'tree' and 'forest': 0.4218
Similarity between 'fast' and 'slow': 0.5314
Similarity between 'sun' and 'moon': 0.4263
Similarity between 'day' and 'night': 0.5070
```

```
print("Loading GoogleNews-vectors-negative300.bin.gz model... This may take a few minutes.")
word_vectors = api.load('word2vec-google-news-300')

print(f"Model loaded successfully. Vocabulary size: {len(word_vectors.index_to_key)} words")

# Display example word vectors
example_words = ['king', 'queen', 'man', 'woman', 'apple', 'orange']

print("\nExample Word Vectors:")
for word in example_words:
    if word in word_vectors:
        print(f"{word}: {word_vectors[word][:5]}...") # Displaying first 5 dimensions for brevity
    else:
        print(f"{word}: Not found in vocabulary")
```

```
Loading GoogleNews-vectors-negative300.bin.gz model... This may take a few minutes.
[==================================================] 100.0% 1662.8/1662.8MB downloaded
Model loaded successfully. Vocabulary size: 3000000 words

Example Word Vectors:
king: [ 0.12597656  0.02978516  0.00860596  0.13964844 -0.02563477]...
queen: [ 0.00524902 -0.14355469 -0.06933594  0.12353516  0.13183594]...
man: [ 0.32617188  0.13085938  0.03466797 -0.08300781  0.08984375]...
woman: [ 0.24316406 -0.07714844 -0.10302734 -0.10742188  0.11816406]...
apple: [-0.06445312 -0.16015625 -0.01208496  0.13476562 -0.22949219]...
orange: [-0.10498047 -0.18261719  0.09912109  0.26367188 -0.19628906]...
```

```
!pip install gensim
print("gensim installed successfully.")
```

```
Collecting gensim
  Downloading gensim-4.4.0-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl.metadata (8
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.12/dist-packages (from gens
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python3.12/dist-packages (from gensi
Requirement already satisfied: smart_open>=1.8.1 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/dist-packages (from smart_open>=
Downloading gensim-4.4.0-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl (27.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 27.9/27.9 MB 33.6 MB/s eta 0:00:00
Installing collected packages: gensim
Successfully installed gensim-4.4.0
gensim installed successfully.
```

```
import gensim.downloader as api # Library for loading pre-trained word embedding models like Word
import numpy as np # Fundamental package for numerical computation, especially with arrays and ma
```

```
import pandas as pd # Library for data manipulation and analysis, particularly with DataFrames.
import nltk # Natural Language Toolkit for various NLP tasks like tokenization and stemming.
import matplotlib.pyplot as plt # Comprehensive library for creating static, animated, and intera

print("All required libraries imported successfully.")
```

```
All required libraries imported successfully.
```

```
words_to_visualize = [
    'king', 'queen', 'man', 'woman', 'prince', 'princess', 'monarch', 'throne',
    'cat', 'dog', 'tiger', 'lion', 'animal', 'pet', 'feline', 'canine',
    'paris', 'london', 'tokyo', 'new_york', 'city', 'capital', 'country',
    'happy', 'sad', 'angry', 'joy', 'emotion', 'feeling',
    'apple', 'banana', 'fruit', 'vegetable', 'food',
    'computer', 'software', 'technology', 'internet',
    'love', 'hate', 'peace', 'war'
]

print(f"Selected {len(words_to_visualize)} words for visualization: {words_to_visualize}")
```

```
'paris', 'london', 'tokyo', 'new_york', 'city', 'capital', 'country', 'happy', 'sad', 'angry', 'jo
```

```
vectors = []
labels = [] # To store the actual words that were found in the model

for word in words_to_visualize:
    if word in word_vectors:
        vectors.append(word_vectors[word])
        labels.append(word)
    else:
        print(f"'{word}' not found in vocabulary, skipping.")

# Convert the list of vectors to a NumPy array
word_vectors_array = np.array(vectors)

print(f"Extracted vectors for {len(labels)} out of {len(words_to_visualize)} words.")
print(f"Shape of word_vectors_array: {word_vectors_array.shape}")
```

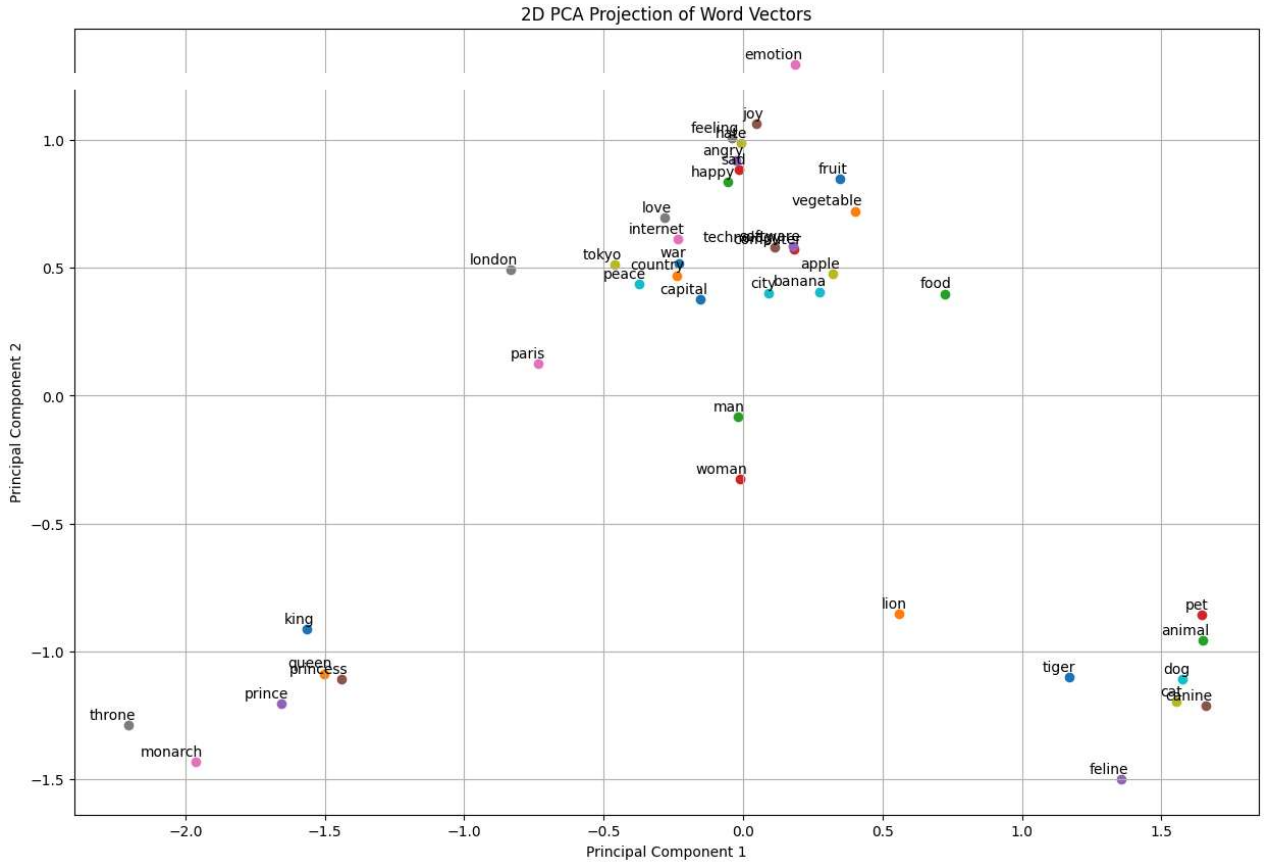
```
'new_york' not found in vocabulary, skipping.
Extracted vectors for 41 out of 42 words.
Shape of word_vectors_array: (41, 300)
```

```
from sklearn.decomposition import PCA

# Instantiate PCA with 2 components
pca = PCA(n_components=2)

# Fit PCA to the word vectors and transform them to 2D
word_vectors_2d = pca.fit_transform(word_vectors_array)

print(f"Original word vector array shape: {word_vectors_array.shape}")
print(f"Reduced word vector array shape (2D): {word_vectors_2d.shape}")
```

```
Original word vector array shape: (41, 300)
Reduced word vector array shape (2D): (41, 2)
```

```
plt.figure(figsize=(15, 10)) # Adjust figure size for better readability

for i, word_label in enumerate(labels):
```

```
    x, y = word_vectors_2d[i, 0], word_vectors_2d[i, 1]
    plt.scatter(x, y) # Plot the point
    plt.annotate(word_label, xy=(x, y), xytext=(5, 2), textcoords='offset points', ha='right', va

plt.title('2D PCA Projection of Word Vectors')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.show()
```




```
words_to_visualize = [
    'king', 'queen', 'man', 'woman', 'prince', 'princess',
    'cat', 'dog', 'lion', 'animal',
    'paris', 'london', 'tokyo', 'city',
    'happy', 'sad', 'angry', 'joy',
    'apple', 'banana', 'fruit',
    'computer', 'software', 'technology',
    'love', 'hate', 'peace'
]

print(f"Selected {len(words_to_visualize)} words for visualization: {words_to_visualize}")
```

```
Selected 27 words for visualization: ['king', 'queen', 'man', 'woman', 'prince', 'princess', 'cat'
```

```
vectors = []
labels = [] # To store the actual words that were found in the model

for word in words_to_visualize:
```

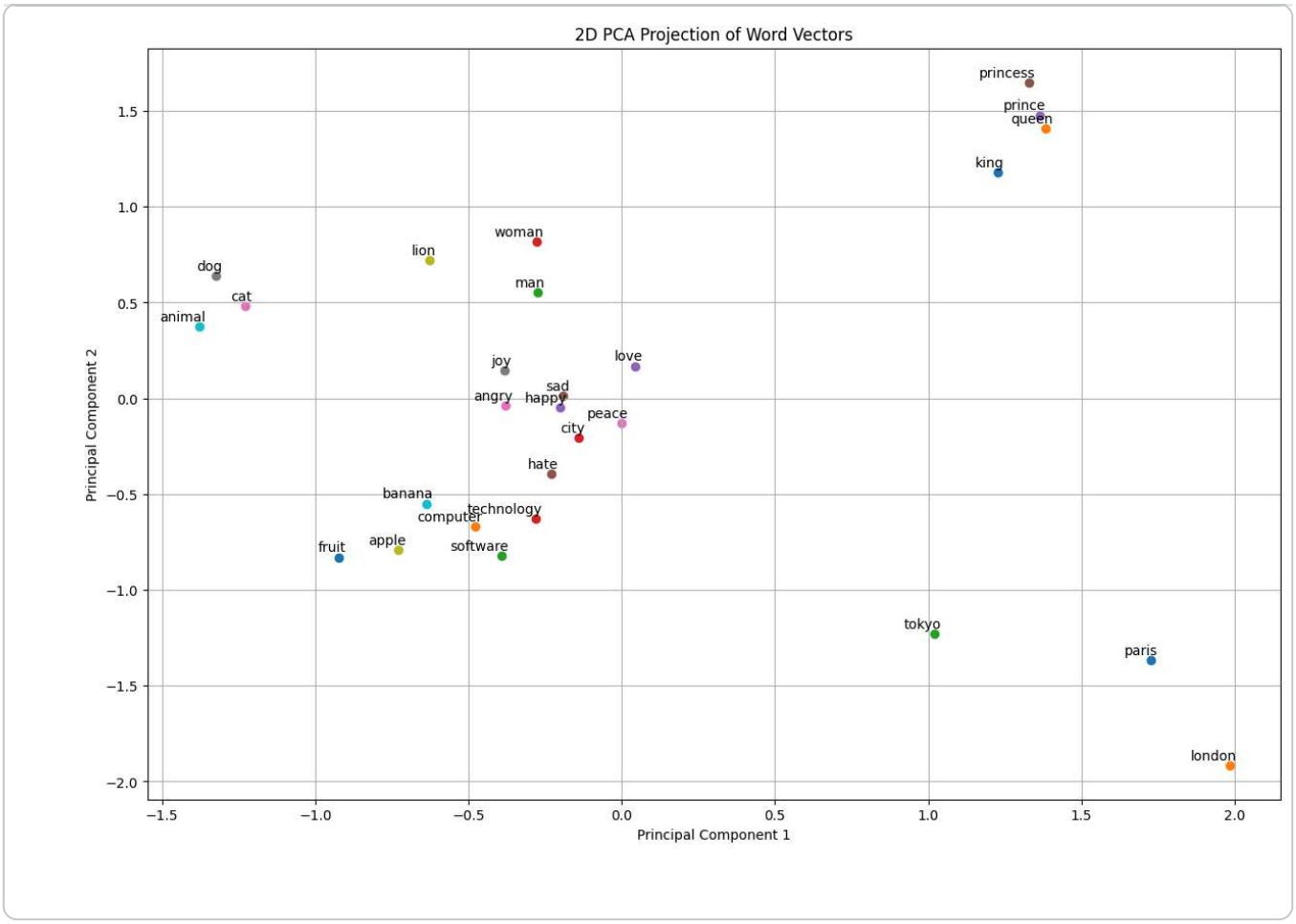
```
        if word in word_vectors:
            vectors.append(word_vectors[word])
            labels.append(word)
        else:
            print(f"'{word}' not found in vocabulary, skipping.")

    # Convert the list of vectors to a NumPy array
    word_vectors_array = np.array(vectors)

    print(f"Extracted vectors for {len(labels)} out of {len(words_to_visualize)} words.")
    print(f"Shape of word_vectors_array: {word_vectors_array.shape}")
```

```
Extracted vectors for 27 out of 27 words.
Shape of word_vectors_array: (27, 300)
```

```
from sklearn.decomposition import PCA

# Instantiate PCA with 2 components
pca = PCA(n_components=2)

# Fit PCA to the word vectors and transform them to 2D
word_vectors_2d = pca.fit_transform(word_vectors_array)

print(f"Original word vector array shape: {word_vectors_array.shape}")
print(f"Reduced word vector array shape (2D): {word_vectors_2d.shape}")
```

```
Original word vector array shape: (27, 300)
Reduced word vector array shape (2D): (27, 2)
```

```
plt.figure(figsize=(15, 10)) # Adjust figure size for better readability

for i, word_label in enumerate(labels):
    x, y = word_vectors_2d[i, 0], word_vectors_2d[i, 1]
    plt.scatter(x, y) # Plot the point
    plt.annotate(word_label, xy=(x, y), xytext=(5, 2), textcoords='offset points', ha='right', va

plt.title('2D PCA Projection of Word Vectors')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.show()
```


2D PCA Projection of Word Vectors
princess
prince
queen
king
woman
lion
dog
man
cat
animal
joy
love
sad
angry
happy
peace
city
hate
banana
technology
computer
fruit
apple
software
tokyo
paris
london
Principal Component 2
Principal Component 1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0