```
# STEP 2 — Import Required Libraries

# Install gensim if not already installed
!pip install gensim

# gensim: Used to load pre-trained Word2Vec/GloVe embeddings
import gensim.downloader as api

# numpy: Used for numerical operations and vector calculations
import numpy as np

# sklearn: Used for similarity calculation and dimensionality reduction (PCA)
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.decomposition import PCA

# matplotlib: Used for visualization
import matplotlib.pyplot as plt
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.12/dist-packages (4.4.0)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.12/dist-packages (from gensim) (2.0.2)
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python3.12/dist-packages (from gensim) (1.16.3)
Requirement already satisfied: smart_open>=1.8.1 in /usr/local/lib/python3.12/dist-packages (from gensim) (7.5.0)
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/dist-packages (from smart_open>=1.8.1->gensim) (2.1.1)
```

```
# ================================
# STEP 3 — Load Pre-trained Embeddings
# ================================

# Load a small pre-trained Word2Vec model (fast for lab use)
model = api.load("glove-wiki-gigaword-100")  # 100-dimensional GloVe vectors

# Print vocabulary size
print("Vocabulary Size:", len(model.key_to_index))

# Display example word vector
word = "king"
vector = model[word]

print(f"\nVector for '{word}':\n")
print(vector)
print("\nVector Dimension:", len(vector))
```

```
[==================================================] 100.0% 128.1/128.1MB downloaded
Vocabulary Size: 400000

Vector for 'king':

[-0.32307  -0.87616   0.21977   0.25268   0.22976   0.7388   -0.37954
 -0.35307  -0.84369  -1.1113   -0.30266   0.33178  -0.25113   0.30448
 -0.077491 -0.89815   0.092496 -1.1407   -0.58324   0.66869  -0.23122
 -0.95855   0.28262  -0.078848  0.75315   0.26584   0.3422   -0.33949
  0.95608   0.065641  0.45747   0.39835   0.57965   0.39267  -0.21851
  0.58795  -0.55999   0.63368  -0.043983 -0.68731  -0.37841   0.38026
  0.61641  -0.88269  -0.12346  -0.37928  -0.38318   0.23868   0.6685
 -0.43321  -0.11065   0.081723  1.1569    0.78958  -0.21223  -2.3211
 -0.67806   0.44561   0.65707   0.1045    0.46217   0.19912   0.25802
  0.057194  0.53443  -0.43133  -0.34311   0.59789  -0.58417   0.068995
  0.23944  -0.85181   0.30379  -0.34177  -0.25746  -0.031101 -0.16285
  0.45169  -0.91627   0.64521   0.73281  -0.22752   0.30226   0.044801
 -0.83741   0.55006  -0.52506  -1.7357    0.4751   -0.70487   0.056939
 -0.7132    0.089623  0.41394  -1.3363   -0.61915  -0.33089  -0.52881
  0.16483  -0.98878 ]

Vector Dimension: 100
```

```
# ================================
# STEP 4 — Word Similarity
# ================================

word_pairs = [
    ("doctor", "nurse"),
    ("cat", "dog"),
    ("car", "bus"),
```

```
    ("king", "queen"),
    ("man", "woman"),
    ("school", "university"),
    ("india", "china"),
    ("apple", "banana"),
    ("teacher", "student"),
    ("sun", "moon")
]

print("Word Similarities:\n")

for w1, w2 in word_pairs:
    similarity = model.similarity(w1, w2)
    print(f"{w1} - {w2} : {similarity:.4f}")
```

```
Word Similarities:

doctor - nurse : 0.7522
cat - dog : 0.8798
car - bus : 0.7373
king - queen : 0.7508
man - woman : 0.8323
school - university : 0.7548
india - china : 0.5997
apple - banana : 0.5054
teacher - student : 0.8083
sun - moon : 0.6138
```

```
# ================================
# STEP 5 — Nearest Neighbors
# ================================

test_words = ["king", "university", "doctor", "india", "computer"]

for word in test_words:
    print(f"\nTop similar words to '{word}':")
    similar_words = model.most_similar(word, topn=5)
    for w, score in similar_words:
        print(f"{w} : {score:.4f}")
```

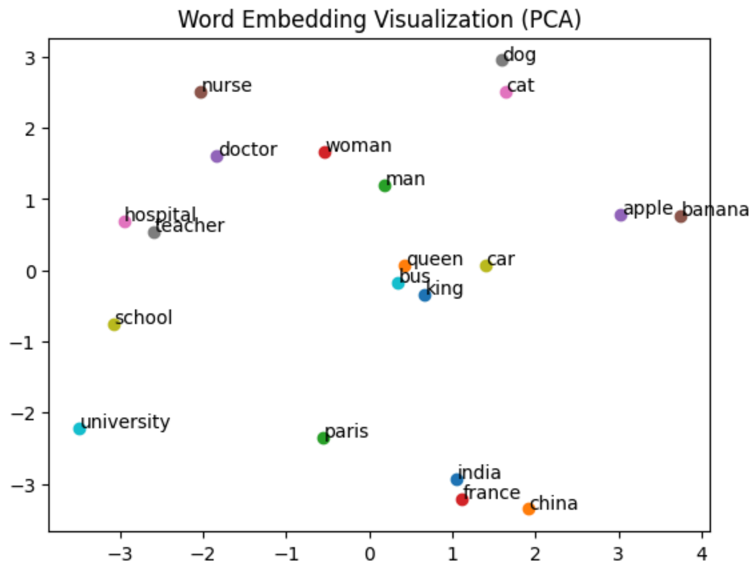
```
Top similar words to 'king':
prince : 0.7682
queen : 0.7508
son : 0.7021
brother : 0.6986
monarch : 0.6978

Top similar words to 'university':
college : 0.8294
harvard : 0.8156
yale : 0.8114
professor : 0.8104
graduate : 0.7993

Top similar words to 'doctor':
physician : 0.7673
nurse : 0.7522
dr. : 0.7175
doctors : 0.7081
patient : 0.7074

Top similar words to 'india':
pakistan : 0.8370
indian : 0.7802
delhi : 0.7712
bangladesh : 0.7662
lanka : 0.7639

Top similar words to 'computer':
computers : 0.8752
software : 0.8373
technology : 0.7642
pc : 0.7366
hardware : 0.7290
```

```
# ================================
# STEP 6 — Word Analogies
```

```
# ================================

analogies = [
    ("king", "man", "woman"),
    ("paris", "france", "india"),
    ("teacher", "school", "hospital")
]

for a, b, c in analogies:
    print(f"\n{a} - {b} + {c} = ?")
    result = model.most_similar(positive=[a, c], negative=[b], topn=3)
    for word, score in result:
        print(f"{word} : {score:.4f}")
```

```
king - man + woman = ?
queen : 0.7699
monarch : 0.6843
throne : 0.6756

paris - france + india = ?
delhi : 0.8655
mumbai : 0.7719
bombay : 0.7222

teacher - school + hospital = ?
nurse : 0.7799
doctor : 0.7613
patient : 0.6909
```

```
# ================================
# STEP 7 — Visualization using PCA
# ================================

words = [
    "king", "queen", "man", "woman",
    "doctor", "nurse", "hospital", "teacher",
    "school", "university", "india", "china",
    "paris", "france", "apple", "banana",
    "cat", "dog", "car", "bus"
]

# Get word vectors
word_vectors = np.array([model[word] for word in words])

# Reduce dimensions to 2D
pca = PCA(n_components=2)
reduced_vectors = pca.fit_transform(word_vectors)

# Plot
plt.figure()
for i, word in enumerate(words):
    x, y = reduced_vectors[i]
    plt.scatter(x, y)
    plt.text(x+0.01, y+0.01, word)

plt.title("Word Embedding Visualization (PCA)")
plt.show()
```


Word Embedding Visualization (PCA)
dog
cat
nurse
doctor
woman
man
hospital
teacher
apple
banana
queen
car
bus
king
school
university
paris
india
france
china
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
4