```
# Install gensim if not already installed
!pip install gensim

"""
STEP 2: Import Libraries

We import:
- gensim to load the Word2Vec model
- numpy for vector math
- pandas for result tables
- matplotlib and PCA for visualization
"""

# to load the pre-trained GoogleNews Word2Vec model
import gensim
from gensim.models import KeyedVectors

# numerical computations
import numpy as np

# table formatting for output
import pandas as pd

# visualization
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# computing cosine similarity manually
from sklearn.metrics.pairwise import cosine_similarity
```

```
Collecting gensim
  Downloading gensim-4.4.0-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl.metadata (8.4 kB)
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)
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 63.0 MB/s eta 0:00:00
Installing collected packages: gensim
Successfully installed gensim-4.4.0
```

```
# Download the pre-trained GoogleNews-vectors-negative300 model if it doesn't exist
import os
import gensim.downloader as api

MODEL_NAME = "word2vec-google-news-300"

print(f"Downloading and loading {MODEL_NAME} model...")
# Download the GoogleNews-vectors-negative300 model using gensim's API
model = api.load(MODEL_NAME)

print("Model loaded successfully!")

# print model stats
print("Vocabulary Size:", len(model.key_to_index))
print("Vector Size (dimensions):", model.vector_size)

# display a vector for a sample word
sample_word = "king"
print(f"\nVector for '{sample_word}':\n", model[sample_word])
```

```
Downloading and loading word2vec-google-news-300 model...
[==================================================] 100.0% 1662.8/1662.8MB downloaded
Model loaded successfully!
Vocabulary Size: 3000000
Vector Size (dimensions): 300

Vector for 'king':
 [ 1.25976562e-01  2.97851562e-02  8.60595703e-03  1.39648438e-01
 -2.56347656e-02 -3.61328125e-02  1.11816406e-01 -1.98242188e-01
  5.12695312e-02  3.63281250e-01 -2.42187500e-01 -3.02734375e-01
 -1.77734375e-01 -2.49023438e-02 -1.67968750e-01 -1.69921875e-01
  3.46679688e-02  5.21850586e-03  4.63867188e-02  1.28906250e-01
  1.36718750e-01  1.12792969e-01  5.95703125e-02  1.36718750e-01
```

```
  1.01074219e-01 -1.76757812e-01 -2.51953125e-01  5.98144531e-02
  3.41796875e-01 -3.11279297e-02  1.04492188e-01  6.17675781e-02
  1.24511719e-01  4.00390625e-01 -3.22265625e-01  8.39843750e-02
  3.90625000e-02  5.85937500e-03  7.03125000e-02  1.72851562e-01
  1.38671875e-01 -2.31445312e-01  2.83203125e-01  1.42578125e-01
  3.41796875e-01 -2.39257812e-02 -1.09863281e-01  3.32031250e-02
 -5.46875000e-02  1.53198242e-02 -1.62109375e-01  1.58203125e-01
 -2.59765625e-01  2.01416016e-02 -1.63085938e-01  1.35803223e-03
 -1.44531250e-01 -5.68847656e-02  4.29687500e-02 -2.46582031e-02
  1.85546875e-01  4.47265625e-01  9.58251953e-03  1.31835938e-01
  9.86328125e-02 -1.85546875e-01 -1.00097656e-01 -1.33789062e-01
 -1.25000000e-01  2.83203125e-01  1.23046875e-01  5.32226562e-02
 -1.77734375e-01  8.59375000e-02 -2.18505859e-02  2.05078125e-02
 -1.39648438e-01  2.51464844e-02  1.38671875e-01 -1.05468750e-01
  1.38671875e-01  8.88671875e-02 -7.51953125e-02 -2.13623047e-02
  1.72851562e-01  4.63867188e-02 -2.65625000e-01  8.91113281e-03
  1.49414062e-01  3.78417969e-02  2.38281250e-01 -1.24511719e-01
 -2.17773438e-01 -1.81640625e-01  2.97851562e-02  5.71289062e-02
 -2.89306641e-02  1.24511719e-02  9.66796875e-02 -2.31445312e-01
  5.81054688e-02  6.68945312e-02  7.08007812e-02 -3.08593750e-01
 -2.14843750e-01  1.45507812e-01 -4.27734375e-01 -9.39941406e-03
  1.54296875e-01 -7.66601562e-02  2.89062500e-01  2.77343750e-01
 -4.86373901e-04 -1.36718750e-01  3.24218750e-01 -2.46093750e-01
 -3.03649902e-03 -2.11914062e-01  1.25000000e-01  2.69531250e-01
  2.04101562e-01  8.25195312e-02 -2.01171875e-01 -1.60156250e-01
 -3.78417969e-02 -1.20117188e-01  1.15234375e-01 -4.10156250e-02
 -3.95507812e-02 -8.98437500e-02  6.34765625e-03  2.03125000e-01
  1.86523438e-01  2.73437500e-01  6.29882812e-02  1.41601562e-01
 -9.81445312e-02  1.38671875e-01  1.82617188e-01  1.73828125e-01
  1.73828125e-01 -2.37304688e-01  1.78710938e-01  6.34765625e-02
  2.36328125e-01 -2.08984375e-01  8.74023438e-02 -1.66015625e-01
 -7.91015625e-02  2.43164062e-01 -8.88671875e-02  1.26953125e-01
 -2.16796875e-01 -1.73828125e-01 -3.59375000e-01 -8.25195312e-02
 -6.49414062e-02  5.07812500e-02  1.35742188e-01 -7.47070312e-02
 -1.64062500e-01  1.15356445e-02  4.45312500e-01 -2.15820312e-01
 -1.11328125e-01 -1.92382812e-01  1.70898438e-01 -1.25000000e-01
  2.65502930e-03  1.92382812e-01 -1.74804688e-01  1.39648438e-01
  2.92968750e-01  1.13281250e-01  5.95703125e-02 -6.39648438e-02
  9.96093750e-02 -2.72216797e-02  1.96533203e-02  4.27246094e-02
 -2.46093750e-01  6.39648438e-02 -2.25585938e-01 -1.68945312e-01
  2.89916992e-03  8.20312500e-02  3.41796875e-01  4.32128906e-02
  1.32812500e-01  1.42578125e-01  7.61718750e-02  5.98144531e-02
 -1.19140625e-01  2.74658203e-03 -6.29882812e-02 -2.72216797e-02
 -4.82177734e-03 -8.20312500e-02 -2.49023438e-02 -4.00390625e-01
```

```
"""
STEP 4: Compute Similarity for Word Pairs

We compute cosine similarity: values range from -1 (opposite) to +1 (very similar).
"""

word_pairs = [
    ("doctor", "nurse"),
    ("cat", "dog"),
    ("car", "bus"),
    ("king", "queen"),
    ("apple", "car"),
    ("teacher", "student"),
    ("india", "china"),
    ("hospital", "doctor"),
    ("computer", "laptop"),
    ("river", "ocean")
]

results = []

for w1, w2 in word_pairs:
    score = model.similarity(w1, w2)
    results.append([w1, w2, score])
    print(f"{w1} — {w2}: {score:.4f}")

# convert to table
df = pd.DataFrame(results, columns=["Word1", "Word2", "CosineSimilarity"])
df
```

```
doctor – nurse: 0.6320
cat – dog: 0.7609
car – bus: 0.4693
king – queen: 0.6511
apple – car: 0.1283
teacher – student: 0.6301
india – china: 0.3533
hospital – doctor: 0.5143
computer – laptop: 0.6640
river – ocean: 0.4772
```

	Word1	Word2	CosineSimilarity
0	doctor	nurse	0.631952
1	cat	dog	0.760946
2	car	bus	0.469337
3	king	queen	0.651096
4	apple	car	0.128307
5	teacher	student	0.630137
6	india	china	0.353308
7	hospital	doctor	0.514324
8	computer	laptop	0.664049
9	river	ocean	0.477181

Next steps: Generate code with df New interactive sheet

```
"""
STEP 5: Find Nearest Neighbors

For each chosen word, print the top 5 most similar words.
"""

words_to_check = ["king", "university", "hospital", "computer", "india"]

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

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

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

Top 5 similar to 'hospital':
Hospital (0.7932)
hopsital (0.7784)
hosptial (0.7582)
hospitals (0.7213)
intensive_care (0.7206)

Top 5 similar to 'computer':
computers (0.7979)
laptop (0.6640)
laptop_computer (0.6549)
Computer (0.6473)
com_puter (0.6082)

Top 5 similar to 'india':
indian (0.6967)
usa (0.6836)
pakistan (0.6815)
chennai (0.6676)
```

```
america (0.6589)
```

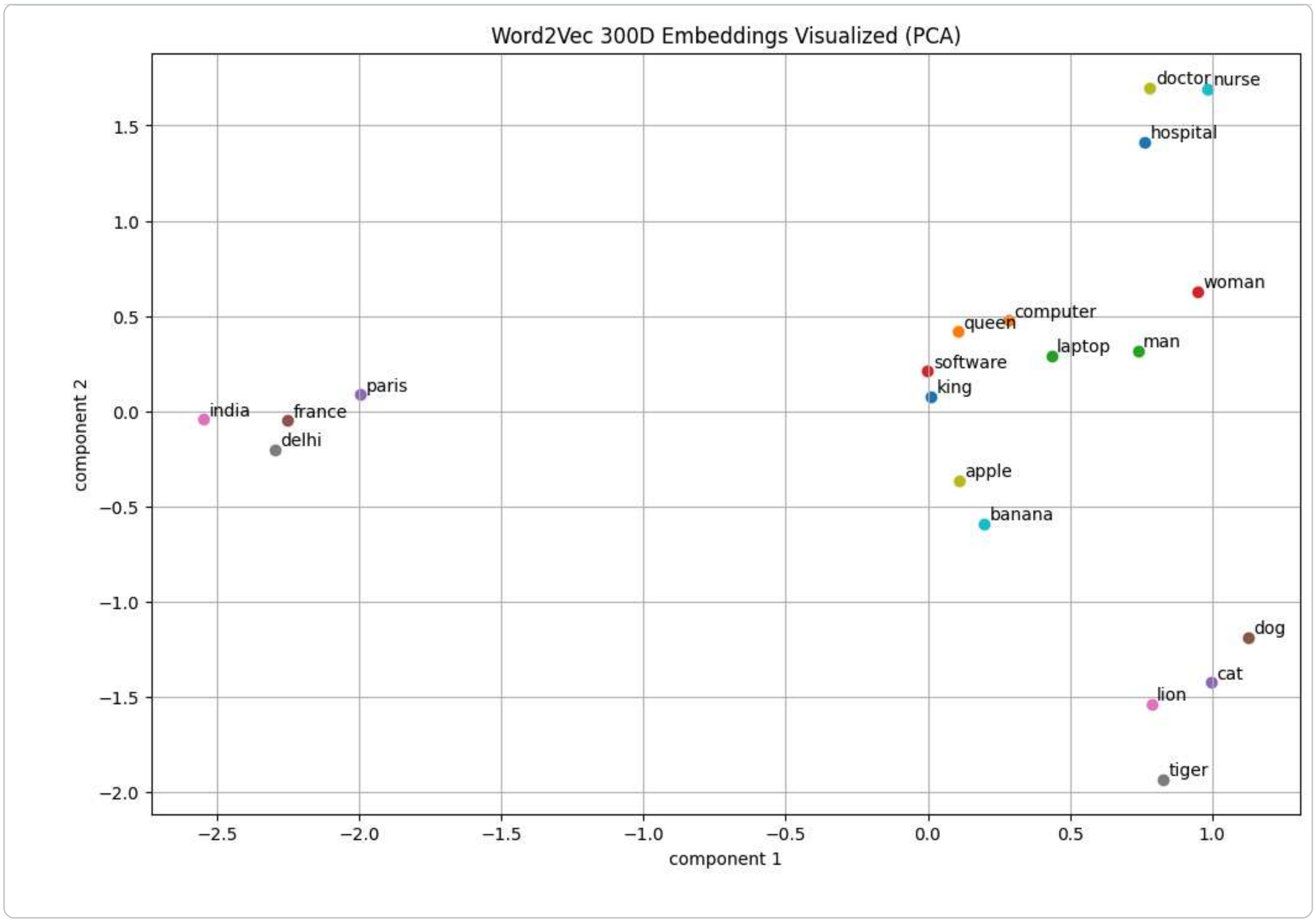
```
"""
STEP 6: Word Analogies

Word analogies use vector arithmetic: vector relationships encode meaning.
"""

# Example: king - man + woman ≈ queen
print("king - man + woman =", model.most_similar(positive=["king","woman"], negative=["man"], topn=1))

# Example: paris - france + india
print("paris - france + india =", model.most_similar(positive=["paris","india"], negative=["france"], topn=1))

# Example: doctor analogies
print("doctor - man + woman =", model.most_similar(positive=["doctor","woman"], negative=["man"], topn=1))
```

```
king - man + woman = [('queen', 0.7118193507194519)]
paris - france + india = [('chennai', 0.5442505478858948)]
doctor - man + woman = [('gynecologist', 0.7093892097473145)]
```

```
"""
STEP 7: Manual Cosine Similarity

Compute similarity between two vectors using sklearn.
"""

vec1 = model["doctor"].reshape(1, -1)
vec2 = model["nurse"].reshape(1, -1)

manual_score = cosine_similarity(vec1, vec2)[0][0]
print("Manual cosine similarity (doctor, nurse):", manual_score)
```

```
Manual cosine similarity (doctor, nurse): 0.6319524
```

```
"""
STEP 8: Visualize Embeddings with PCA

We will plot selected words in 2D space using PCA.
"""

words_for_plot = [
    "king", "queen", "man", "woman",
    "paris", "france", "india", "delhi",
    "doctor", "nurse", "hospital",
    "computer", "laptop", "software",
    "cat", "dog", "lion", "tiger",
    "apple", "banana"
]

vectors = np.array([model[w] for w in words_for_plot])

pca = PCA(n_components=2)
reduced = pca.fit_transform(vectors)

plt.figure(figsize=(12, 8))
for i, word in enumerate(words_for_plot):
    x, y = reduced[i]
    plt.scatter(x, y)
    plt.text(x + 0.02, y + 0.02, word)
plt.title("Word2Vec 300D Embeddings Visualized (PCA)")
plt.xlabel("component 1")
plt.ylabel("component 2")
plt.grid()
plt.show()
```


Word2Vec 300D Embeddings Visualized (PCA)
component 1
component 2
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
doctor
nurse
hospital
woman
computer
queen
laptop
man
software
king
paris
india
france
delhi
apple
banana
dog
cat
lion
tiger