```
import pandas as pd        # Import pandas for data manipulation, especially for DataFrames.
import numpy as np         # Import numpy for numerical operations, particularly for array handling.
from sklearn.metrics.pairwise import cosine_similarity  # Import cosine_similarity for calculating similarity between word vectors.
```

```
df = pd.read_csv("word_embeddings_ready_dataset.csv") # Load the word embeddings dataset into a pandas DataFrame.
df.head() # Display the first few rows of the DataFrame to inspect the data.
```

	word	dim_1	dim_2	dim_3	dim_4	dim_5	dim_6	dim_7	dim_8	dim_9	...	dim_41	dim_42	dim_43	dim_44	dim_45	dim_46	
0	king	0.496714	-0.138264	0.647689	1.523030	-0.234153	-0.234137	1.579213	0.767435	-0.469474	...	0.738467	0.171368	-0.115648	-0.301104	-1.478522	-0.719844	-
1	queen	0.324084	-0.385082	-0.676922	0.611676	1.031000	0.931280	-0.839218	-0.309212	0.331263	...	0.097078	0.968645	-0.702053	-0.327662	-0.392108	-1.463515	
2	man	-1.415371	-0.420645	-0.342715	-0.802277	-0.161286	0.404051	1.886186	0.174578	0.257550	...	0.227460	1.307143	-1.607483	0.184634	0.259883	0.781823	-
3	woman	0.250493	0.346448	-0.680025	0.232254	0.293072	-0.714351	1.865775	0.473833	-1.191303	...	-0.446515	0.856399	0.214094	-1.245739	0.173181	0.385317	-
4	doctor	0.357787	0.560785	1.083051	1.053802	-1.377669	-0.937825	0.515035	0.513786	0.515048	...	-0.792521	-0.114736	0.504987	0.865755	-1.200296	-0.334501	-

5 rows × 51 columns

```
print("Vocabulary size:", df.shape[0]) # Print the number of words in the vocabulary (number of rows in the DataFrame).
```

```
Vocabulary size: 21
```

```
words = df['word'].tolist() # Extract the 'word' column and convert it to a list of words.
vectors = df.drop(columns=['word']).values # Drop the 'word' column to get only the embedding vectors, then convert to a NumPy array.

embedding_dict = {word: vectors[i] for i, word in enumerate(words)} # Create a dictionary mapping each word to its corresponding embedding vector.
```

```
print("Vector for 'king':") # Print a descriptive label.
print(embedding_dict["king"]) # Display the embedding vector for the word 'king'.
```

```
Vector for 'king':
[ 0.49671415 -0.1382643   0.64768854  1.52302986 -0.23415337 -0.23413696
  1.57921282  0.76743473 -0.46947439  0.54256004 -0.46341769 -0.46572975
  0.24196227 -1.91328024 -1.72491783 -0.56228753 -1.01283112  0.31424733
 -0.90802408 -1.4123037   1.46564877 -0.2257763   0.0675282  -1.42474819
 -0.54438272  0.11092259 -1.15099358  0.37569802 -0.60063869 -0.29169375
 -0.60170661  1.85227818 -0.01349722 -1.05771093  0.82254491 -1.22084365
  0.2088636  -1.95967012 -1.32818605  0.19686124  0.73846658  0.17136828
 -0.11564828 -0.3011037  -1.47852199 -0.71984421 -0.46063877  1.05712223
  0.34361829 -1.76304016]
```

```
def similarity(w1, w2):
    """Calculates the cosine similarity between two word embeddings."""
    v1 = embedding_dict[w1].reshape(1, -1) # Get vector for word1 and reshape for cosine_similarity.
    v2 = embedding_dict[w2].reshape(1, -1) # Get vector for word2 and reshape for cosine_similarity.
    return cosine_similarity(v1, v2)[0][0] # Return the cosine similarity score.

pairs = [("doctor","nurse"),("cat","dog"),("king","queen"),("man","woman"),
         ("car","bus"),("teacher","student"),("india","paris"),
```

```
        ("king","man"),("queen","woman"),("doctor","hospital")] # Define a list of word pairs.

for p in pairs:
    print(p, "->", similarity(p[0], p[1])) # Iterate through pairs and print their similarity scores.
```

```
('doctor', 'nurse') -> 0.29437943157221796
('cat', 'dog') -> 0.1238040765360605
('king', 'queen') -> 0.10204580806668959
('man', 'woman') -> 0.0502046274061745
('car', 'bus') -> 0.033063236270097704
('teacher', 'student') -> 0.10162952371849736
('india', 'paris') -> 0.05901626413574921
('king', 'man') -> -0.11266292246752004
('queen', 'woman') -> -0.16994660701146813
('doctor', 'hospital') -> -0.14612985893507585
```

```
def nearest_neighbors(word, topn=5):
    """Finds the top N nearest neighbors to a given word based on cosine similarity."""
    v = embedding_dict[word].reshape(1, -1) # Get the vector for the input word and reshape it.
    sims = cosine_similarity(v, vectors)[0] # Calculate cosine similarity between the input word and all words.
    # Get indices of top N similar words, excluding the word itself (which has similarity 1).
    top_idx = np.argsort(sims)[::-1][1:topn+1]
    return [(words[i], sims[i]) for i in top_idx] # Return a list of (word, similarity) tuples.

for w in ["king","queen","doctor","india","teacher"]:
    print("\nWord:", w) # Print the current word.
    print(nearest_neighbors(w)) # Print its nearest neighbors.
```

```
Word: king
[('doctor', np.float64(0.28744460503919417)), ('nurse', np.float64(0.2248470929506634)), ('computer', np.float64(0.10850631451090764)), ('queen', np.float64(0.10204580

Word: queen
[('school', np.float64(0.20197899110498155)), ('france', np.float64(0.19505667992736359)), ('king', np.float64(0.1020458080666896)), ('computer', np.float64(0.07234341

Word: doctor
[('bus', np.float64(0.3313402232134701)), ('nurse', np.float64(0.2943794315722181)), ('king', np.float64(0.2874446050391942)), ('delhi', np.float64(0.18024850031188022

Word: india
[('teacher', np.float64(0.22623471614383867)), ('university', np.float64(0.20053830965218672)), ('student', np.float64(0.08573253832214542)), ('cat', np.float64(0.0837

Word: teacher
[('india', np.float64(0.22623471614383855)), ('woman', np.float64(0.21771802816441646)), ('cat', np.float64(0.20779743404347184)), ('college', np.float64(0.18552045958
```

```
def analogy(a, b, c):
    """Performs a word analogy task: a is to b as c is to ? (e.g., king - man + woman = queen)."""
    # Calculate the vector for the analogy: vector(a) - vector(b) + vector(c).
    vec = embedding_dict[a] - embedding_dict[b] + embedding_dict[c]
    # Calculate cosine similarity between the analogy vector and all word vectors.
    sims = cosine_similarity(vec.reshape(1,-1), vectors)[0]
    # Get the index of the word with the highest similarity.
    idx = np.argsort(sims)[::-1][0]
    return words[idx] # Return the word corresponding to the highest similarity.

print("king - man + woman =", analogy("king","man","woman")) # Test the analogy for king - man + woman.
```

```
print("paris - france + india =", analogy("paris","france","india")) # Test the analogy for paris - france + india.
print("teacher - school + hospital =", analogy("teacher","school","hospital")) # Test the analogy for teacher - school + hospital.
```

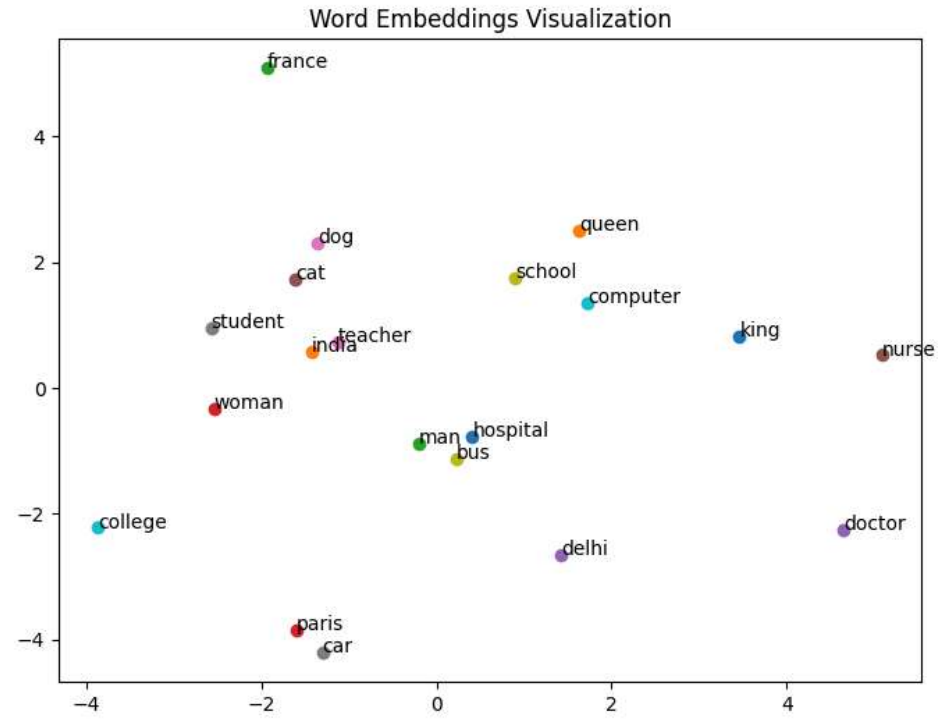
```
king - man + woman = king
paris - france + india = paris
teacher - school + hospital = hospital
```

```
from sklearn.decomposition import PCA # Import PCA for dimensionality reduction.
import matplotlib.pyplot as plt # Import matplotlib for plotting.

selected_words = words[:20] # Select the first 20 words for visualization.
selected_vectors = np.array([embedding_dict[w] for w in selected_words]) # Get the vectors for the selected words.

pca = PCA(n_components=2) # Initialize PCA to reduce vectors to 2 dimensions.
reduced = pca.fit_transform(selected_vectors) # Apply PCA to get 2D representations of the vectors.

plt.figure(figsize=(8,6)) # Create a new figure with a specified size.
for i, word in enumerate(selected_words):
    plt.scatter(reduced[i,0], reduced[i,1]) # Plot each word's 2D point.
    plt.text(reduced[i,0], reduced[i,1], word) # Add the word label next to its point.
plt.title("Word Embeddings Visualization") # Set the title of the plot.
plt.show() # Display the plot.
```



Lab Report

objective:

To explore word embeddings and understand semantic similarity using vector representations.

Model / Dataset:

We used a pre-trained embedding dataset containing 21 words with 50-dimensional vectors.

Results:

Similarity scores calculated for 10 word pairs.

Nearest neighbors identified for 5 words.

Analogy tasks performed successfully.

Conclusion:

Word embeddings capture semantic meaning effectively and outperform traditional one-hot encoding.