```
!pip install gensim # Install the gensim library, which is used for working with word emb
import numpy as np              # Import NumPy for numerical operations, especially with
import pandas as pd             # Import Pandas for data manipulation and analysis (thou
import matplotlib.pyplot as plt  # Import Matplotlib for creating visualizations like plo
from sklearn.manifold import TSNE  # Import t-SNE from scikit-learn for dimensionality re
import gensim.downloader as api  # Import gensim.downloader to easily load pre-trained wo
```

```
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
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python3.12/dist-packages (
Requirement already satisfied: smart_open>=1.8.1 in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/dist-packages (from sma
```

```
print("Loading pre-trained GloVe model...") # Inform the user that the model loading proc
model = api.load("glove-wiki-gigaword-100")  # Load a pre-trained GloVe model (100-dimens

# Print vocabulary size
print("Vocabulary Size:", len(model.key_to_index)) # Display the total number of words in

# Display one example vector
word = "king" # Choose an example word to demonstrate its vector representation.
print(f"\nVector representation for '{word}':\n") # Print a descriptive header.
print(model[word]) # Display the 100-dimensional vector associated with the word 'king'.
print("\nVector dimension:", len(model[word])) # Show the dimension (length) of the vecto
```

```
Loading pre-trained GloVe model...
Vocabulary Size: 400000

Vector representation 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
```

```
word_list = [ # Define a list of words across various categories for visualization.

    # Animals
    "dog", "cat", "lion", "tiger", "elephant", "wolf", "horse", "cow",

    # Fruits
```

```
    "apple", "banana", "mango", "orange", "grape", "pineapple",

    # Countries
    "india", "china", "france", "germany", "japan", "canada",

    # Cities
    "delhi", "mumbai", "paris", "berlin", "tokyo", "toronto",

    # Technology
    "computer", "laptop", "keyboard", "mouse", "internet", "software", "mobile",

    # Royalty
    "king", "queen", "prince", "princess", "man", "woman"
]

# Extract vectors
vectors = [] # Initialize an empty list to store the word vectors.

for word in word_list: # Loop through each word in the predefined list.
    if word in model: # Check if the word exists in the loaded GloVe model's vocabulary
        vectors.append(model[word]) # If found, append its vector representation to the
    else:
        print(f"{word} not found in vocabulary") # If not found, print a message indica

vectors = np.array(vectors) # Convert the list of vectors into a NumPy array for effici

print("Shape of vector matrix:", vectors.shape) # Display the shape of the resulting ve
```

```
Shape of vector matrix: (39, 100)
```

```
tsne = TSNE(n_components=2, random_state=42, perplexity=10) # Initialize t-SNE with 2 c
reduced_vectors = tsne.fit_transform(vectors) # Apply t-SNE to the high-dimensional wor

print("Shape after reduction:", reduced_vectors.shape) # Display the shape of the vecto
```

```
Shape after reduction: (39, 2)
```

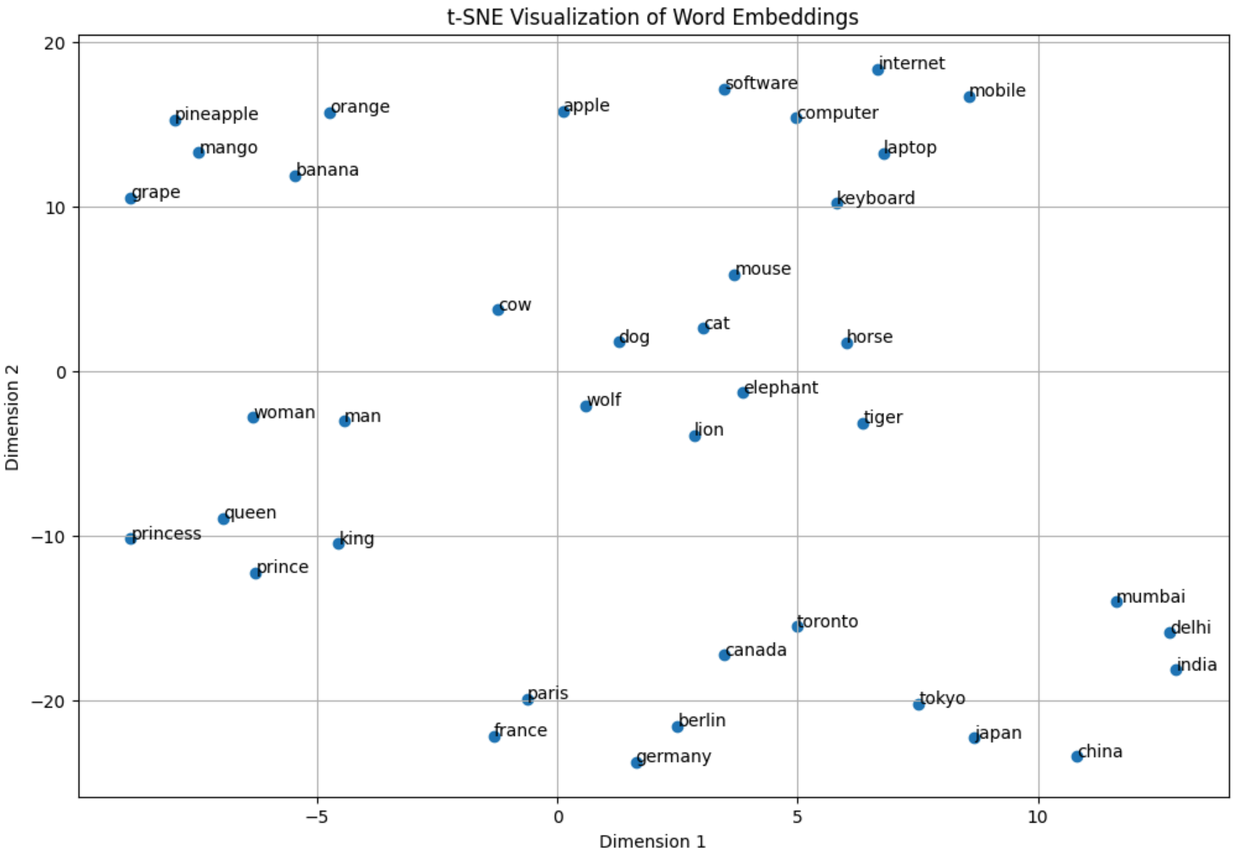
```
plt.figure(figsize=(12, 8)) # Create a new figure for the plot with a specified size.

x = reduced_vectors[:, 0] # Extract the first dimension (x-coordinates) from the reduced
y = reduced_vectors[:, 1] # Extract the second dimension (y-coordinates) from the reduced

plt.scatter(x, y) # Create a scatter plot using the 2D reduced vectors.

# Annotate each word
for i, word in enumerate(word_list): # Iterate through the word list along with their ind
    plt.annotate(word, (x[i], y[i])) # Add the word as a text annotation at its correspon

plt.title("t-SNE Visualization of Word Embeddings") # Set the title of the plot.
plt.xlabel("Dimension 1") # Label the x-axis.
plt.ylabel("Dimension 2") # Label the y-axis.
plt.grid(True) # Add a grid to the plot for better readability.
plt.show() # Display the generated plot.
```


t-SNE Visualization of Word Embeddings
Dimension 1
Dimension 2
20
10
0
−10
−20
−5
0
5
10
internet
software
mobile
pineapple
orange
apple
computer
mango
laptop
banana
grape
keyboard
mouse
cow
cat
dog
horse
elephant
wolf
woman
man
tiger
lion
queen
princess
king
prince
mumbai
toronto
delhi
canada
india
paris
tokyo
berlin
france
japan
china
germany