STEP 1 — Import Required Libraries

```
# Step 1: Import necessary libraries

!pip install gensim
import numpy as np                      # For handling numerical arrays and vectors
import gensim.downloader as api         # To download and load pre-trained word embedd
import matplotlib.pyplot as plt         # For plotting the visualization
from sklearn.manifold import TSNE       # t-SNE algorithm for dimensionality reduction
```

```
Collecting gensim
  Downloading gensim-4.4.0-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.12/dist-package
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-pac
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/dist-packages (from
Downloading gensim-4.4.0-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl (
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 27.9/27.9 MB 56.7 MB/s eta 0:00:00
Installing collected packages: gensim
Successfully installed gensim-4.4.0
```

STEP 2 — Load Pre-trained Word Embedding Model

```
# Step 2: Load the pre-trained GloVe model (100 dimensions)

model = api.load("glove-wiki-gigaword-100")   # Download and load the model

# Print vocabulary size of the model
print("Vocabulary size:", len(model))

# Display one example word vector
print("\nExample vector for word 'king':")
print(model['king'])   # Shows 100-dimensional vector
```

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

Example vector for word '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  ]
```

STEP 3 — Select Meaningful Word Groups (40–50 Words)

```
# Step 3: Create word groups

animals = ["dog", "cat", "lion", "tiger", "elephant", "wolf", "horse", "monkey"]

countries = ["india", "china", "france", "germany", "brazil", "japan", "canada"]

cities = ["delhi", "mumbai", "paris", "berlin", "tokyo", "toronto"]

technology = ["computer", "laptop", "keyboard", "internet", "software", "hardware", "

fruits = ["apple", "banana", "mango", "orange", "grapes", "pineapple"]

royalty = ["king", "queen", "prince", "princess", "emperor"]

vehicles = ["car", "bus", "train", "truck", "airplane"]

# Combine all groups into one list
word_list = animals + countries + cities + technology + fruits + royalty + vehicles

print("Total selected words:", len(word_list))
```

```
Total selected words: 44
```

STEP 4 — Extract Word Vectors

```
# Step 4: Extract vectors for selected words

word_vectors = []     # To store vectors
valid_words = []      # To store words that exist in vocabulary

for word in word_list:
    if word in model:                 # Check if word exists in embedding vocabulary
        word_vectors.append(model[word])   # Get the word vector
        valid_words.append(word)           # Store valid word

# Convert list into NumPy array
word_vectors = np.array(word_vectors)

print("Shape of word vector matrix:", word_vectors.shape)
```

```
Shape of word vector matrix: (44, 100)
```

STEP 5 — Apply t-SNE (Dimensionality Reduction)

```
# Step 5: Apply t-SNE

tsne = TSNE(
    n_components=2,     # Reduce to 2 dimensions
    random_state=42,    # For reproducibility
    perplexity=10       # Controls clustering (good for small datasets)
)

# Perform dimensionality reduction
tsne_result = tsne.fit_transform(word_vectors)
```

```
print("Shape after t-SNE reduction:", tsne_result.shape)
```

```
Shape after t-SNE reduction: (44, 2)
```

STEP 6 — Plot the t-SNE Visualization

```
# Step 6: Create scatter plot

plt.figure(figsize=(12, 8))   # Set figure size

# Extract X and Y coordinates
x = tsne_result[:, 0]
y = tsne_result[:, 1]

# Plot scatter points
plt.scatter(x, y)

# Annotate each point with corresponding word
for i, word in enumerate(valid_words):
    plt.annotate(word, (x[i], y[i]))

# Add title and labels
plt.title("t-SNE Visualization of Word Embeddings")
plt.xlabel("Dimension 1")
plt.ylabel("Dimension 2")

plt.grid(True)
plt.show()
```