```
import sys
# Check if running in Google Colab and install gensim if necessary
if 'google.colab' in sys.modules:
  !pip install gensim
# Load pre-trained word embeddings library
import gensim.downloader as api

# Import numpy for numerical operations
import numpy as np
# Import pandas for data manipulation (though not directly used in this cell)
import pandas as pd

# Import matplotlib for plotting visualizations
import matplotlib.pyplot as plt

# Import TSNE for dimensionality reduction
from sklearn.manifold import TSNE
```

```
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)
```

```
# Load a pre-trained Word2Vec model (GloVe common crawl 100-dimensional vectors)
model = api.load("glove-wiki-gigaword-100")

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

# Display the vector representation for the word 'king'
print("Vector for word 'king':\n", model['king'])
# Print the dimension of the word vector for 'king'
print("Vector dimension:", model['king'].shape)
```

```
Vocabulary size: 400000
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 ]
Vector dimension: (100,)
```

```
# Define a list of words categorized for clarity, which will be used for embedding visualization
word_list = [
    # Animals
    "dog", "cat", "lion", "tiger", "elephant", "wolf", "fox", "cow",

    # Fruits
    "apple", "banana", "orange", "mango", "grape", "pineapple",

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

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

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

    # Vehicles
    "car", "bus", "train", "bicycle", "airplane", "truck"
]
```

```
vectors = []

# Iterate through each word in the defined word_list
for word in word_list:
    # Check if the word exists in the pre-trained model's vocabulary
    if word in model:
        # If found, append its vector representation to the 'vectors' list
        vectors.append(model[word])
    else:
        # If not found, print a message indicating it's missing from the vocabulary
        print(f"{word} not found in vocabulary")

# Convert the list of word vectors into a NumPy array for further processing
vectors = np.array(vectors)
# Print the shape of the resulting vector matrix (number of words x vector dimension)
print("Shape of vector matrix:", vectors.shape)
```

```
Shape of vector matrix: (38, 100)
```

```
# Initialize a t-SNE model for dimensionality reduction
# n_components=2: Reduce the vectors to 2 dimensions for 2D visualization
# random_state=42: Set a random seed for reproducibility
# perplexity=10: A hyperparameter for t-SNE, influencing the balance between local and global aspects of the data
tsne = TSNE(n_components=2, random_state=42, perplexity=10)
# Apply t-SNE to the word vectors to reduce their dimensionality
reduced_vectors = tsne.fit_transform(vectors)

# Print the shape of the vectors after dimensionality reduction
print("Shape after reduction:", reduced_vectors.shape)
```

```
Shape after reduction: (38, 2)
```

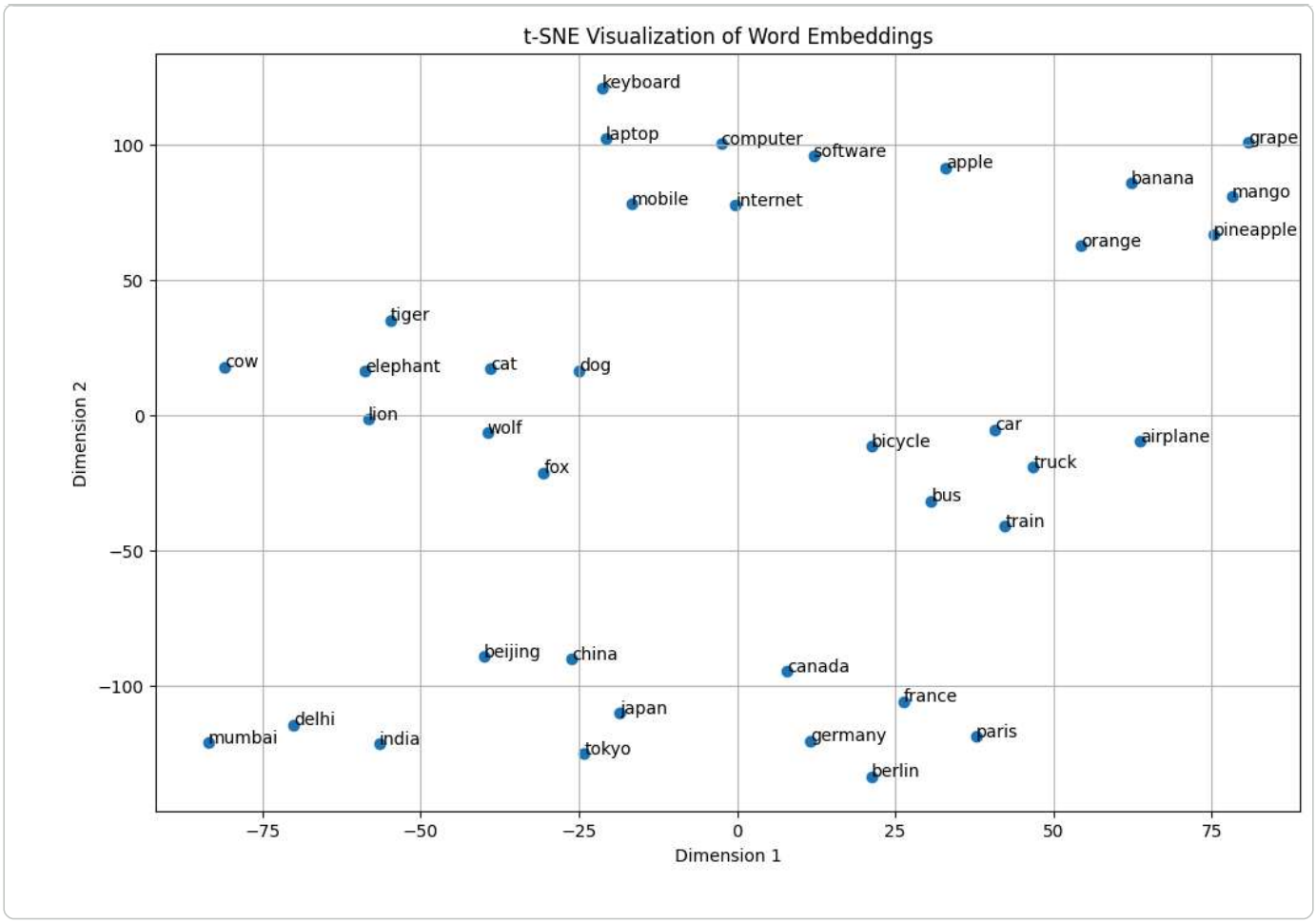
```
# Create a new figure for the plot with a specified size for better visibility
plt.figure(figsize=(12, 8))

# Extract the first dimension of the reduced vectors for x-coordinates
x = reduced_vectors[:, 0]
# Extract the second dimension of the reduced vectors for y-coordinates
y = reduced_vectors[:, 1]

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

# Annotate each point in the scatter plot with its corresponding word
for i, word in enumerate(word_list):
    plt.annotate(word, (x[i], y[i]))

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


t-SNE Visualization of Word Embeddings
keyboard
laptop
computer
software
apple
grape
banana
mango
mobile
internet
pineapple
orange
tiger
cow
elephant
cat
dog
lion
wolf
car
bicycle
airplane
truck
fox
bus
train
beijing
china
canada
france
japan
delhi
mumbai
india
tokyo
germany
paris
berlin
Dimension 1
Dimension 2
100
50
0
−50
−100
−75
−50
−25
0
25
50
75