PG2: Use dimensionality reduction (e.g., PCA or t-SNE) to visualize word embeddings for PG 1. Select 10 words from a specific domain (e.g., sports, technology) and visualize their embeddings. Analyze clusters and relationships.

Generate contextually rich outputs using embeddings. Write a program to generate 5 semantically similar words for a given input

Soln:

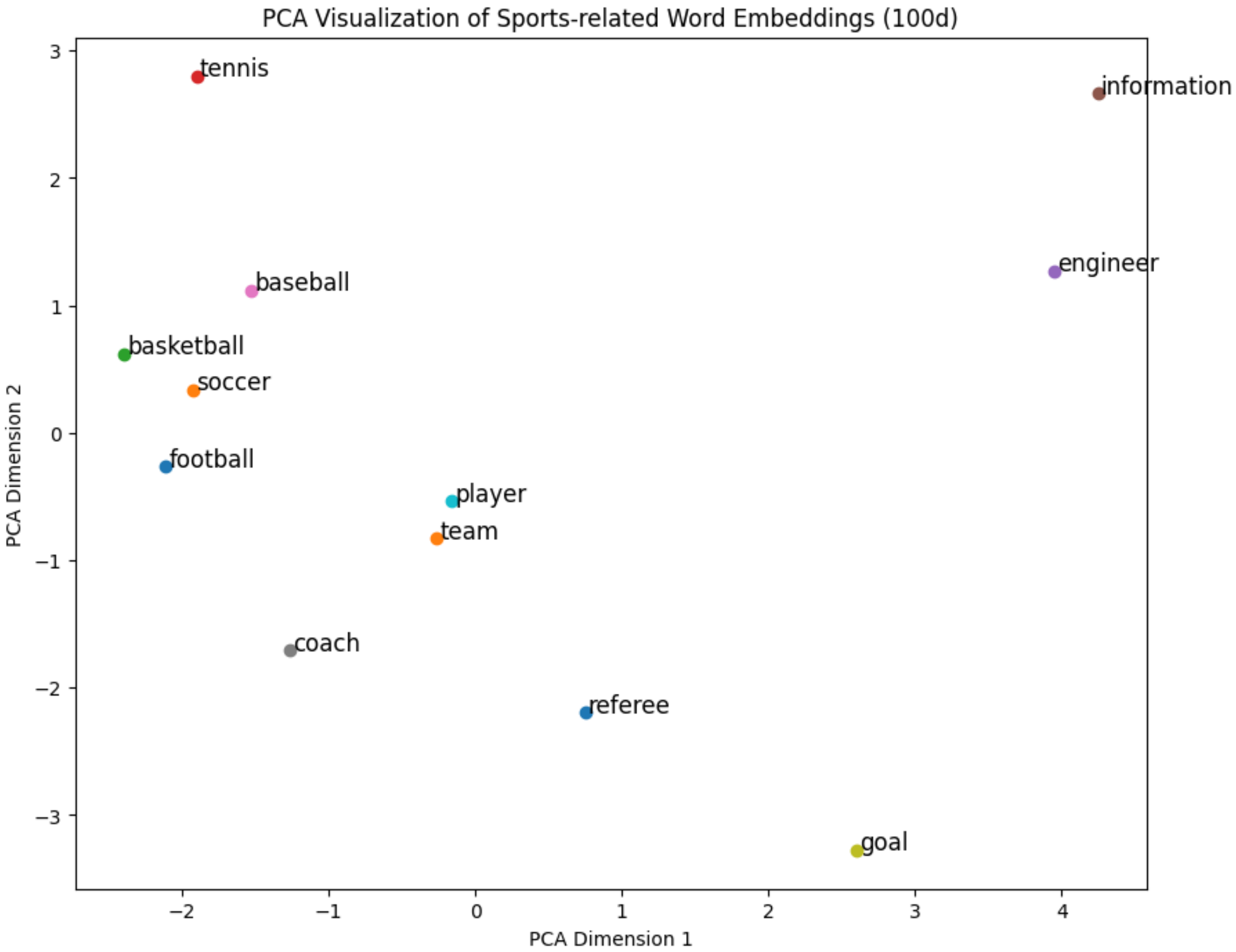
```
!pip install gensim
```

#Gensim: A Python library for NLP and word embeddings.

Use dimensionality reduction (e.g., PCA or t-SNE) to visualize word embeddings for PG1. Select 10 words from a specific domain (e.g., sports, technology) and visualize their embeddings. Analyze clusters and relationships. Generate contextually rich outputs using embeddings. Write aprogram to generate 5 semantically similar words for a given input.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from gensim.models import KeyedVectors

# Load pre-trained GloVe embeddings (100d model)
model_100d =
KeyedVectors.load_word2vec_format("/content/glove.6B.100d.word2vec.txt"
, binary=False,limit=500000)

# Select 10 words from a specific domain (sports) # Included other
words to show how embeddings are different
words = ['football', 'soccer', 'basketball',
'tennis','engineer','information', 'baseball', 'coach', 'goal',
'player', 'referee', 'team']
word_vectors = np.array([model_100d[word] for word in words])

# Dimensionality reduction using PCA
# Using PCA to reduce to 2D for visualization
pca = PCA(n_components=2)
pca_result = pca.fit_transform(word_vectors)

# Plotting the words in 2D space
plt.figure(figsize=(10, 8))
```

```
for i, word in enumerate(words):
    plt.scatter(pca_result[i, 0], pca_result[i, 1])
    plt.text(pca_result[i, 0] + 0.02, pca_result[i, 1], word,
fontsize=12)
plt.title("PCA Visualization of Sports-related Word Embeddings (100d)")
plt.xlabel("PCA Dimension 1")
plt.ylabel("PCA Dimension 2")
plt.show()

# 5 Semantically Similar Words Generator Function
def get_similar_words(word, model, topn=5):
    similar_words = model.similar_by_word(word, topn=topn)
    return similar_words

# Example: Get 5 words similar to "football"
similar_words_football = get_similar_words('football', model_100d,
topn=5)
print(f"Words similar to 'football': {similar_words_football}")


Output:
```



Output: `Words similar to 'football': [('soccer', 0.8732221722602844), ('basketball', 0.8555637001991272), ('league', 0.815336287021637), ('rugby', 0.8007532954216003), ('hockey', 0.7833694815635681)]`

```python
# Select the words you want to print embeddings for
words_to_print = ['football', 'soccer']

# Print their embeddings
for word in words_to_print:
    if word in model_100d:
        print(f"Vector embedding for '{word}':\n{model_100d[word]}\n")
    else:
        print(f"Word '{word}' not found in the embeddings model.")
```

Output:

```
Vector embedding for 'football':
[ 0.43865    0.10537    0.45972   -1.0724    -1.2471     0.76351
  0.47528    0.083857  -0.9127    -0.27328   -0.018591  -1.184
  0.22748    0.16847   -0.52158    0.11339    1.3757     0.11892
 -0.37683    0.51149   -0.8833     0.96259    0.18143   -0.407
  0.036181  -0.74432   -0.0027401 -0.70068    0.53103    0.45114
 -0.72884    1.0631    -0.28008   -0.63848    0.15645   -0.46927
 -1.0071     1.033     -1.4354    -0.27485    0.048984   0.13951
  0.43072   -0.78791    0.41097    0.58509    1.0155    -0.1839
  0.27487   -0.90866   -0.30441   -0.17396    0.020941   0.62813
  0.10978   -2.3885    -0.56364   -0.27193    0.98728    0.70608
 -0.512      0.52636   -0.78503   -0.68714    0.38121    0.097582
 -0.20237    0.43208   -0.30527    0.57925    0.62619   -0.47415
  0.33834   -0.28421   -0.097465   0.19597    0.54849    0.59918
 -0.41576    0.1021     0.6766     0.0042009 -0.12354   -0.76613
 -0.27436   -0.68248   -1.0789    -0.16708    0.81671    0.026999
 -0.38707    0.40448   -1.0995     0.64718   -0.12802   -0.26084
 -0.96701    0.88078    1.012     -0.022223 ]
```

Vector embedding for 'soccer':

[8.3777e-01 5.1890e-01 6.4015e-01 -6.2606e-01 -9.7474e-01 1.0127e+00

 6.2729e-02 4.4316e-01 -8.3299e-01 7.9888e-02 -1.1815e-02 -1.1265e+00

 1.2554e-01 -3.4206e-01 -5.1422e-01 3.8526e-01 1.0032e+00 -1.5172e-03

 -2.2684e-01 3.5658e-01 -6.2449e-01 8.7271e-01 3.6670e-01 4.6462e-01

 -1.0046e-01 -4.4798e-01 -2.1813e-01 -5.6423e-01 5.6665e-01 5.1601e-01

 -5.6511e-01 7.1919e-01 -6.5347e-01 -9.5952e-02 5.6028e-01 -4.9956e-01

 -7.4757e-01 6.8516e-01 -1.4518e+00 -1.1207e-01 1.0241e-01 3.0537e-02

1.1326e-02 -8.6873e-01 6.3622e-01 4.9539e-01 3.0538e-01 7.7133e-02

7.4048e-02 -7.1163e-01 -1.9159e-01 -3.4168e-01 -4.7185e-01 5.6794e-01

3.7454e-01 -1.9207e+00 -8.6040e-01 5.7058e-01 1.0700e+00 9.2101e-01

-6.4825e-01 5.3516e-01 -1.5556e-01 -9.0021e-01 -1.7459e-01 3.3146e-02

-5.7512e-01 2.9963e-01 -4.0008e-01 -1.0765e-01 4.1384e-01 -7.2178e-01

1.1442e-01 -2.1291e-01 5.4949e-02 1.3213e-01 7.8766e-01 8.9291e-02

-6.6689e-01 3.3998e-01 9.7163e-01 -8.4871e-02 1.7542e-01 -4.6039e-01

-8.5885e-02 -7.5960e-01 -1.5071e+00 2.1545e-01 2.1209e-01 -4.4837e-01

-2.5882e-01 3.3814e-01 -4.7979e-01 2.1059e-01 2.3621e-01 -3.6699e-01

-8.1440e-01 5.4515e-01 9.7946e-01 2.3367e-01]