GEN AI LAB PROGRAMS

1. CODE:

```
!pip install gensim scipy

# Import required libraries
import gensim.downloader as api  # For downloading pre-trained word vectors
from scipy.spatial.distance import cosine  # For calculating cosine similarity

# Load pre-trained Word2Vec model (Google News, 300 dimensions)
print("Loading Word2Vec model...")
model = api.load("word2vec-google-news-300")
print("Model loaded successfully.\n")

# Get and print the first 10 dimensions of the word vector for 'king'
vector = model['king']
print("First 10 dimensions of 'king' vector:")
print(vector[:10], "\n")

# Print top 10 most similar words to 'king'
print("Top 10 words most similar to 'king':")
for word, similarity in model.most_similar('king'):
    print(f"{word}: {similarity:.4f}")
print()

# Perform word analogy: king - man + woman ≈ queen
result = model.most_similar(positive=['king', 'woman'], negative=['man'], topn=1)
print("Analogy - 'king' - 'man' + 'woman' ≈ ?")
print(f"Result: {result[0][0]} (Similarity: {result[0][1]:.4f})\n")

# Analogy: paris + italy - france ≈ rome
print("Analogy - 'paris' + 'italy' - 'france' ≈ ?")
for word, similarity in model.most_similar(positive=['paris', 'italy'],
negative=['france']):
    print(f"{word}: {similarity:.4f}")
print()

# Analogy: walking + swimming - walk ≈ swim
```

```
print("Analogy - 'walking' + 'swimming' - 'walk' ≈ ?")
for word, similarity in model.most_similar(positive=['walking', 'swimming'],
negative=['walk']):
   print(f"{word}: {similarity:.4f}")
print()

# Calculate cosine similarity between 'king' and 'queen'
similarity = 1 - cosine(model['king'], model['queen'])
print(f"Cosine similarity between 'king' and 'queen': {similarity:.4f}")
```

OUTPUT:

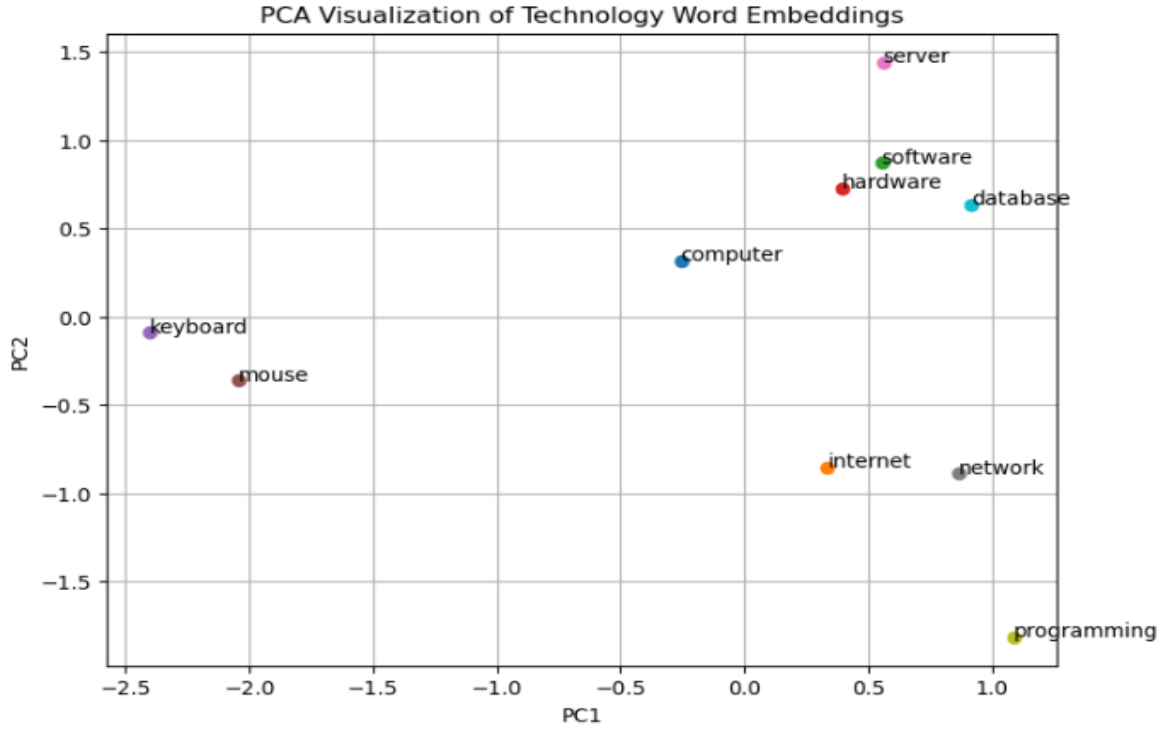
```
Requirement already satisfied: gensim in c:\users\sudarshan\anaconda3\lib\site-packages (4.3.3)
Requirement already satisfied: scipy in c:\users\sudarshan\anaconda3\lib\site-packages (1.13.1)
Requirement already satisfied: numpy<2.0,>=1.18.5 in c:\users\sudarshan\anaconda3\lib\site-packages (from gensim) (1.26.4)
Requirement already satisfied: smart-open>=1.8.1 in c:\users\sudarshan\anaconda3\lib\site-packages (from gensim) (5.2.1)
Loading Word2Vec model...
Model loaded successfully.

First 10 dimensions of 'king' vector:
[ 0.12597656  0.02978516  0.00860596  0.13964844 -0.02563477 -0.03613281
  0.11181641 -0.19824219  0.05126953  0.36328125]

Top 10 words most similar to 'king':
kings: 0.7138
queen: 0.6511
monarch: 0.6413
crown_prince: 0.6204
prince: 0.6160
sultan: 0.5865
ruler: 0.5798
princes: 0.5647
Prince_Paras: 0.5433
throne: 0.5422

Analogy - 'king' - 'man' + 'woman' ≈ ?
Result: queen (Similarity: 0.7118)

Analogy - 'paris' + 'italy' - 'france' ≈ ?
lohan: 0.5070
madrid: 0.4818
heidi: 0.4800
real_madrid: 0.4753
florence: 0.4682
diego: 0.4673
ronnie: 0.4672
juventus: 0.4672
joel: 0.4654
huntelaar: 0.4636

Analogy - 'walking' + 'swimming' - 'walk' ≈ ?
Swimming: 0.6006
swim: 0.5949
swimmers: 0.5835
swimmer: 0.5819
paddling: 0.5744
kayaking: 0.5662
swam: 0.5506
rowing: 0.5436
swims: 0.5371
canoeing: 0.5140

Cosine similarity between 'king' and 'queen': 0.6511
```

2. CODE:

```
!pip install gensim matplotlib scikit-learn
import gensim.downloader as api
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Load model
model = api.load("word2vec-google-news-300")

# Select 10 domain-specific words (technology domain)
words = ['computer', 'internet', 'software', 'hardware', 'keyboard', 'mouse', 'server',
'network', 'programming', 'database']
vectors = [model[word] for word in words]

# Dimensionality reduction using PCA
pca = PCA(n_components=2)
reduced = pca.fit_transform(vectors)

# Generate 5 semantically similar words for a given input
input_word = 'computer'
similar_words = model.most_similar(input_word, topn=5)

# Print the similar words to terminal
print(f"Top 5 words similar to '{input_word}':")
for word, score in similar_words:
    print(f"{word}: {score:.4f}")

# Plot the word embeddings
plt.figure(figsize=(8, 6))
for i, word in enumerate(words):
    plt.scatter(reduced[i, 0], reduced[i, 1])
    plt.annotate(word, (reduced[i, 0], reduced[i, 1]))
plt.title("PCA Visualization of Technology Word Embeddings")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.grid(True)

# Show the plot
plt.show()
```

OUTPUT:

```
Top 5 words similar to 'computer':
computers: 0.7979
laptop: 0.6640
laptop_computer: 0.6549
Computer: 0.6473
com_puter: 0.6082
```

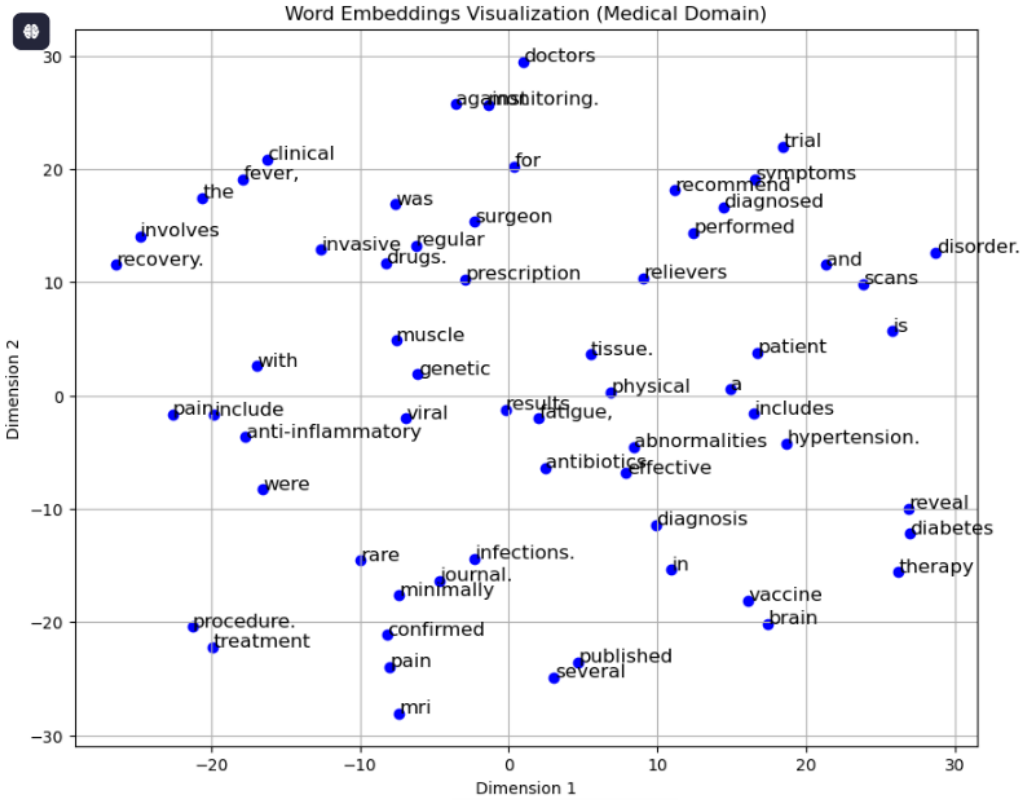


3. CODE

```
!pip install gensim matplotlib scikit-learn
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
import numpy as np
from gensim.models import Word2Vec

# Sample domain-specific corpus (medical domain)
medical_corpus = [
    "The patient was diagnosed with diabetes and hypertension.",
    "MRI scans reveal abnormalities in the brain tissue.",
    "The treatment involves antibiotics and regular monitoring.",
    "Symptoms include fever, fatigue, and muscle pain.",
    "The vaccine is effective against several viral infections.",
    "Doctors recommend physical therapy for recovery.",
    "The clinical trial results were published in the journal.",
    "The surgeon performed a minimally invasive procedure.",
    "The prescription includes pain relievers and anti-inflammatory drugs.",
    "The diagnosis confirmed a rare genetic disorder."
]

# Preprocess corpus (tokenize sentences and convert to lowercase)
processed_corpus = [sentence.lower().split() for sentence in medical_corpus]

# Train a Word2Vec model
model = Word2Vec(sentences=processed_corpus, vector_size=100, window=5,
min_count=1, workers=4, epochs=50)

# Extract embeddings for visualization
words = list(model.wv.index_to_key)  # List of words in the vocabulary
embeddings = np.array([model.wv[word] for word in words])  # Word embeddings
for each word

# Dimensionality reduction using t-SNE
tsne = TSNE(n_components=2, random_state=42, perplexity=5)
tsne_result = tsne.fit_transform(embeddings)

# Visualization of word embeddings
plt.figure(figsize=(10, 8))
```

```
plt.scatter(tsne_result[:, 0], tsne_result[:, 1], color="blue")

# Annotating each point with the corresponding word
for i, word in enumerate(words):
    plt.text(tsne_result[i, 0] + 0.02, tsne_result[i, 1] + 0.02, word, fontsize=12)

plt.title("Word Embeddings Visualization (Medical Domain)")
plt.xlabel("Dimension 1")
plt.ylabel("Dimension 2")
plt.grid(True)

plt.show()

# Analyze domain-specific semantics
def find_similar_words(input_word, top_n=5):
    try:
        similar_words = model.wv.most_similar(input_word, topn=top_n)
        print(f"Words similar to '{input_word}':")
        for word, similarity in similar_words:
            print(f"  {word} ({similarity:.2f})")
    except KeyError:
        print(f"'{input_word}' not found in vocabulary.")

# Generate semantically similar words
find_similar_words("treatment")
find_similar_words("vaccine")
```

OUTPUT:




```
Words similar to 'treatment':
  procedure. (0.27)
  confirmed (0.15)
  muscle (0.13)
  monitoring. (0.12)
  fatigue, (0.12)
Words similar to 'vaccine':
  brain (0.26)
  recommend (0.21)
  procedure. (0.19)
  therapy (0.19)
  in (0.18)
```

Copy Improve

5.

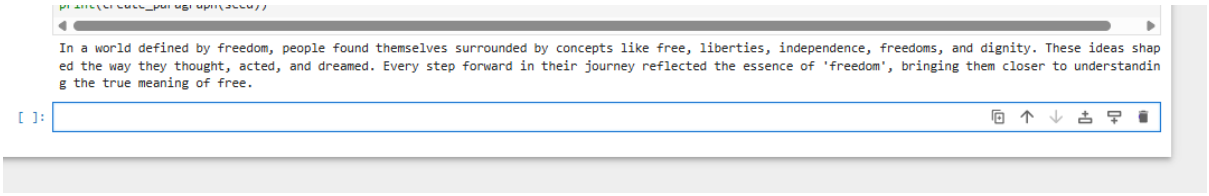
CODE:

```
# Required Libraries:
# Install gensim if not already installed:
# pip install gensim

import gensim.downloader as api  # For loading pre-trained word embeddings
from gensim.models import KeyedVectors  # For working with word vectors
import random  # For shuffling similar words

# Load pre-trained GloVe word vectors (100-dimensional, trained on Wikipedia + Gigaword)
model = api.load("glove-wiki-gigaword-100")

# Function to generate similar words for a given seed word
def generate_similar_words(seed_word, topn=10):
    # Check if the seed word exists in the model vocabulary
    if seed_word in model:
        # Return top 'n' similar words based on cosine similarity
        return [word for word, _ in model.most_similar(seed_word, topn=topn)]
    else:
        # Return empty list if word not in vocabulary
        return []

# Function to create a meaningful paragraph using the seed and its similar words
def create_paragraph(seed_word):
    similar_words = generate_similar_words(seed_word, topn=10)
    if not similar_words:
        return f"No similar words found for '{seed_word}'."

    # Randomly shuffle similar words and select 5
    random.shuffle(similar_words)
    selected_words = similar_words[:5]

    # Construct a short creative paragraph
    paragraph = f"In a world defined by {seed_word}, "
    paragraph += f"people found themselves surrounded by concepts like {', '.join(selected_words[:-1])}, and {selected_words[-1]}. "
```

```
    paragraph += f"These ideas shaped the way they thought, acted, and dreamed.
Every step forward in their journey reflected the essence of '{seed_word}', "
    paragraph += f"bringing them closer to understanding the true meaning of
{selected_words[0]}."

    return paragraph

# Example usage
seed = "freedom"  # You can change this to any word like 'love', 'innovation', etc.
print(create_paragraph(seed))
```

OUTPUT:

In a world defined by freedom, people found themselves surrounded by concepts like equality, democracy, rights, free, and independence. These ideas shaped the way they thought, acted, and dreamed. Every step forward in their journey reflected the essence of 'freedom', bringing them closer to understanding the true meaning of equality.



\

6.
CODE:

```
# Step 1: Install required libraries (only run once)
# pip install transformers torch

# Step 2: Import necessary library
from transformers import pipeline

# Step 3: Load the sentiment analysis pipeline using a pre-trained model
sentiment_pipeline = pipeline("sentiment-analysis")

# Step 4: Define input sentences (simulating real-world user reviews)
input_sentences = [
    "The new phone I bought is absolutely amazing!",
    "Worst customer service ever. I'm never coming back.",
    "The experience was average, nothing special.",
    "Fast delivery and the packaging was perfect.",
    "The product broke within two days. Very disappointed."
]

# Step 5: Perform sentiment analysis
results = sentiment_pipeline(input_sentences)

# Step 6: Display the results
print("Sentiment Analysis Results:\n")
for sentence, result in zip(input_sentences, results):
    print(f"Input Sentence: {sentence}")
    print(f"Predicted Sentiment: {result['label']}, Confidence Score:
{result['score']:.2f}\n")
```

OUTPUT:

Collapse Output

```
No model was supplied, defaulted to distilbert/distilbert-base-uncased-finetuned-sst-2-english and revision 714eb0f (https://huggingface.co/distilbert/di
stilbert-base-uncased-finetuned-sst-2-english).
Using a pipeline without specifying a model name and revision in production is not recommended.
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better performance, install t
he package with: `pip install huggingface_hub[hf_xet]` or `pip install hf_xet`
```

Loading widget...

```
C:\Users\sudarshan\anaconda3\Lib\site-packages\huggingface_hub\file_download.py:143: UserWarning: `huggingface_hub` cache-system uses symlinks by default
to efficiently store duplicated files but your machine does not support them in C:\Users\sudarshan\.cache\huggingface\hub\models--distilbert--distilbert-
base-uncased-finetuned-sst-2-english. Caching files will still work but in a degraded version that might require more space on your disk. This warning ca
n be disabled by setting the `HF_HUB_DISABLE_SYMLINKS_WARNING` environment variable. For more details, see https://huggingface.co/docs/huggingface_hub/ho
w-to-cache#limitations.
To support symlinks on Windows, you either need to activate Developer Mode or to run Python as an administrator. In order to activate developer mode, see
this article: https://docs.microsoft.com/en-us/windows/apps/get-started/enable-your-device-for-development
  warnings.warn(message)
```

Loading widget...

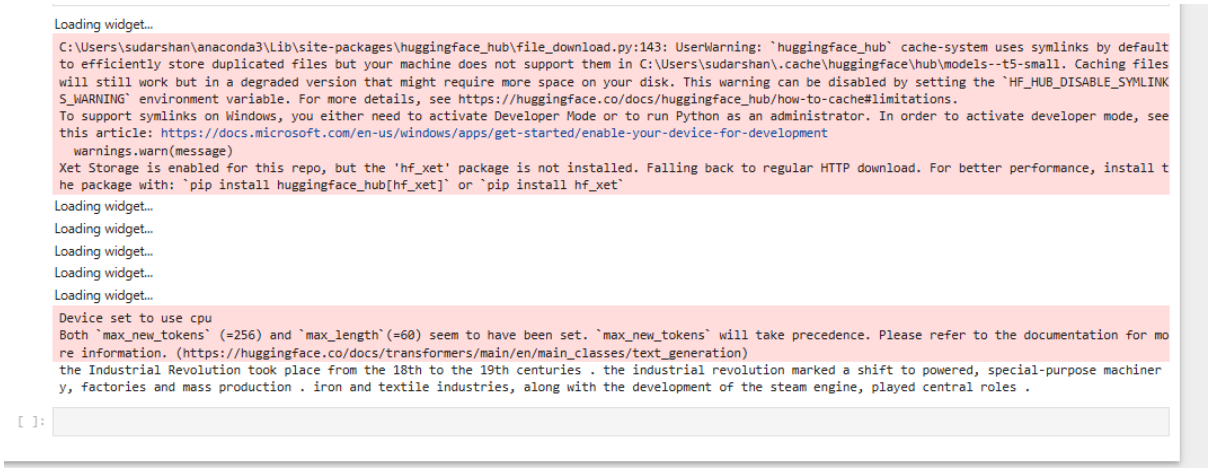
Loading widget...

```
Device set to use cpu
Sentiment Analysis Results:

Input Sentence: The new phone I bought is absolutely amazing!
Predicted Sentiment: POSITIVE, Confidence Score: 1.00

Input Sentence: Worst customer service ever. I'm never coming back.
Predicted Sentiment: NEGATIVE, Confidence Score: 1.00

Input Sentence: The experience was average, nothing special.
Predicted Sentiment: NEGATIVE, Confidence Score: 1.00

Input Sentence: Fast delivery and the packaging was perfect.
Predicted Sentiment: POSITIVE, Confidence Score: 1.00

Input Sentence: The product broke within two days. Very disappointed.
Predicted Sentiment: NEGATIVE, Confidence Score: 1.00
```

7. CODE

```
# Required libraries (install before running this script):

# pip install transformers torch


from transformers import pipeline  # Import the summarization pipeline from Hugging Face Transformers


# Load a smaller and faster pre-trained model for summarization

# 't5-small' is lightweight and quick, ideal for small/medium passages

summarizer = pipeline("summarization", model="t5-small")


# Input text to be summarized

text = """

The Industrial Revolution, which took place from the 18th to the 19th centuries, was a period during which predominantly agrarian, rural societies in Europe and America became industrial and urban. Prior to the Industrial Revolution, manufacturing was often done in people's homes, using hand tools or basic machines. Industrialization marked a shift to powered, special-purpose machinery, factories and mass production. The iron and textile industries, along with the development of the steam engine, played central roles in the Industrial Revolution, which also saw improved systems of transportation, communication and banking. While industrialization brought about an increased volume and variety of manufactured goods and an improved standard of living for some, it also resulted in often grim employment and living conditions for the poor and working classes.

"""


# Generate the summary of the input text

summary = summarizer(text, max_length=60, min_length=30, do_sample=False)


# Print the summarized output

print(summary[0]['summary_text'])
```

OUTPUT:



NOTE: INSTALL ALL THE MODULES MENTIONED IN COMMENTS