PG1: Explore pre-trained word vectors. Explore word relationships using vector arithmetic. Perform arithmetic operations and analyze results.

Soln:

```
!pip install gensim
```

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

```
from gensim.scripts.glove2word2vec import glove2word2vec
from gensim.models import KeyedVectors

# Paths to the GloVe file and output Word2Vec file
glove_input_file = "/content/glove.6B.100d.txt"  # Path to GloVe file
word2vec_output_file = "/content/glove.6B.100d.word2vec.txt"  # Output
file in Word2Vec format

# Convert GloVe format to Word2Vec format
glove2word2vec(glove_input_file, word2vec_output_file)

# Load the converted Word2Vec model
model = KeyedVectors.load_word2vec_format(word2vec_output_file,
binary=False)

# Test the loaded model
print(model.most_similar("king"))
```

#GloVe embeddings are converted to Word2Vec format for compatibility with libraries like Gensim, which require the Word2Vec format for efficient vector operations and model functionality.

Output: `[('prince', 0.7682328820228577), ('queen', 0.7507690787315369), ('son', 0.7020888328552246), ('brother', 0.6985775232315063), ('monarch', 0.6977890729904175), ('throne', 0.6919989585876465), ('kingdom', 0.6811409592628479), ('father', 0.6802029013633728), ('emperor', 0.6712858080863953), ('ii', 0.6676074266433716)]`

Explore Word Relationships

Example 1: Find Similar Words

```
similar_to_mysore = model.similar_by_vector(model['mysore'], topn=5)
print(f"Words similar to 'mysore': {similar_to_mysore}")
```

```
Output: Words similar to 'mysore': [('mysore', 1.0), ('cochin', 0.6752076148986816), ('hyderabad', 0.6592637896537781), ('jaipur', 0.6591896414756775), ('perak', 0.6516631245613098)]
```

Example 2: Gender Analogy (king - man + woman = queen)

```
# Perform vector arithmetic
result_vector_1 = model['actor'] - model['man'] + model['woman']

# Find the most similar word
result_1 = model.similar_by_vector(result_vector_1, topn=1)
print(f"'actor - man + woman' = {result_1}")
Output: 'actor - man + woman' = [('actress', 0.9160683155059814)]
```

Example 3: Country-City Relationship (India - Delhi + Bangalore)

```
# Perform vector arithmetic
result_vector_2 = model['india'] - model['delhi'] + model['washington']

# Find the most similar word
result_2 = model.similar_by_vector(result_vector_2, topn=3)
print(f"'India - Delhi + Washington' = {result_2}")
Output: 'India - Delhi + Washington' = [('states', 0.8375228643417358), ('united', 0.8281229734420776), ('washington', 0.8155243396759033)]
```

Perform Arithmetic Operations

```
scaled_vector = model['hotel'] * 2  # Scales the 'king' vector by a factor of 2
result_2 = model.similar_by_vector(scaled_vector, topn=3)
result_2
```

[('hotel', 1.0),
('hotels', 0.7933705449104309),
('restaurant', 0.7762866020202637)]

Example 2: Normalizing Vectors

```
import numpy as np
normalized_vector = model['fish'] / np.linalg.norm(model['fish'])
result_2 = model.similar_by_vector(normalized_vector, topn=3)
result_2
```

[('fish', 1.0), ('shrimp', 0.7793381810188293), ('salmon', 0.760814368724823)]

Example 3: Averaging Vectors

```
average_vector = (model['king'] + model['woman'] + model['man']) / 3
result_2 = model.similar_by_vector(average_vector, topn=3)
result_2
```

[('man', 0.9197071194648743),
('woman', 0.8637868165969849),
('father', 0.8270207047462463)]

Model Comparision

```
# Paths to the GloVe file and output Word2Vec file
glove_input_file = "/content/glove.6B.50d.txt"  # Path to GloVe file
word2vec_output_file = "/content/glove.6B.50d.word2vec.txt"  # Output
file in Word2Vec format

# Convert GloVe format to Word2Vec format
glove2word2vec(glove_input_file, word2vec_output_file)

# Load the converted Word2Vec model
model_50d = KeyedVectors.load_word2vec_format(word2vec_output_file,
binary=False)

# Paths to the GloVe file and output Word2Vec file
glove_input_file = "/content/glove.6B.100d.txt"  # Path to GloVe file
word2vec_output_file = "/content/glove.6B.100d.word2vec.txt"  # Output
file in Word2Vec format

# Convert GloVe format to Word2Vec format
glove2word2vec(glove_input_file, word2vec_output_file)

# Load the converted Word2Vec model
model_100d = KeyedVectors.load_word2vec_format(word2vec_output_file,
binary=False)
```

Calculate similarity between two words

```
word1 = "hospital"
word2 = "doctor"

# Similarity in 50d
similarity_50d = model_50d.similarity(word1, word2)

# Similarity in 100d
similarity_100d = model_100d.similarity(word1, word2)

# Results
print(f"Similarity (50d) between '{word1}' and '{word2}':
{similarity_50d:.4f}")
print(f"Similarity (100d) between '{word1}' and '{word2}':
{similarity_100d:.4f}")
```

```
Output : Similarity (50d) between 'hospital' and 'doctor': 0.6724

Similarity (100d) between 'hospital' and 'doctor': 0.6901
```

Calculate distance between two words

```
# Calculate distance between two words
distance_50d = model_50d.distance(word1, word2)
distance_100d = model_100d.distance(word1, word2)

# Results
print(f"Distance (50d) between '{word1}' and '{word2}':
{distance_50d:.4f}")
print(f"Distance (100d) between '{word1}' and '{word2}':
{distance_100d:.4f}")
```

```
Distance (50d) between 'hospital' and 'doctor': 0.3276
Distance (100d) between 'hospital' and 'doctor': 0.3099
```

Analysis of Results

1. **'actor - man + woman' = actress (0.916)**
   - The result confirms that the model has captured gender analogies, where subtracting "man" and adding "woman" to "actor" produces the semantically related word "actress."
2. **'India - Delhi + Washington' = ['states', 0.838], ['united', 0.828], ['washington', 0.816]**
   - The arithmetic operation shows that "India - Delhi + Washington" produces words like "states" and "united," suggesting a shift from a city to broader political entities, such as countries or states.
3. **Scaling Vectors ('hotel' * 2) = [('hotel', 1.0), ('hotels', 0.793), ('restaurant', 0.776)]**
   - The scaled vector results in "hotel" being the most similar to itself, and its plural form "hotels" is the second most similar, followed by related terms like "restaurant."
4. **Normalizing Vectors ('fish') = [('fish', 1.0), ('shrimp', 0.779), ('salmon', 0.761)]**
   - Normalizing the vector for "fish" leads to very similar words like "shrimp" and "salmon," which are semantically related types of fish.
5. **Averaging Vectors ('king' + 'woman' + 'man') / 3 = [('man', 0.920), ('woman', 0.864), ('father', 0.827)]**
   - Averaging the vectors of "king," "woman," and "man" results in "man" and "woman" being the most similar words, indicating that the averaged vector represents a central concept of human relationships.
6. **Similarity and Distance Calculation for 'hospital' and 'doctor':**
   - **Similarity**: 0.6724 (50d) vs. 0.6901 (100d)
     - The similarity between "hospital" and "doctor" is higher in the 100d model, indicating that the higher-dimensional model captures the relationship between these words more accurately.
   - **Distance**: 0.3276 (50d) vs. 0.3099 (100d)

    - The distance between "hospital" and "doctor" is smaller in the 100d model, confirming that the 100d model finds them closer in the vector space, aligning with the similarity results.

Conclusion

- Higher-dimensional models (100d) generally provide more accurate and nuanced word relationships, both in terms of **similarity** and **distance**.
- Arithmetic operations like scaling, averaging, and vector shifts (analogies) allow deeper exploration of word meanings and relationships, and these can vary slightly with model dimensions.