Partha Sai - pua528

## Debiasing word embeddings

Word embedding are word vectors that have meaning, word vectors similar to each other will be close to each other in a vector space.

**After completing this lab you will be to:**

- Use and load pre-trained word vectors
- Measure similarity of word vectors using cosine similarity
- Solve word analogy probelms such as Man is to Woman as Boy is to ___ using word embeddings
- Reduce gender bias in word embeddings by modifying word embeddings to remove gender stereotypes, such as the association between the words *receptionist* and *female*

## Operations on word embeddings

### Task 1 - Cosine similarity

Similarity between two words represented as word vectors $u$ and $v$ can be measured by their cosine similarity:

$$\text{CosineSimilarity(u, v)} = \frac{u \cdot v}{||u||_2 ||v||_2} = cos(\theta) \tag{1}$$

Where:

$u \cdot v$ is the dot (inner) product of the two vectors

$||u||_2$ is the length of the vector $u$. The length also called Euclidean length or Euclidean norm defines a distance function defined as $||u||_2 = \sqrt{u_1^2 + \ldots + u_n^2}$

The normalized similarity between $u$ and $v$ is the cosine of the angle between the two vectors denoted as $\theta$. The cosine similarity of $u$ and $v$ will be close to 1 if the two vectors are similar, otherwise, the cosine similarity will be small.

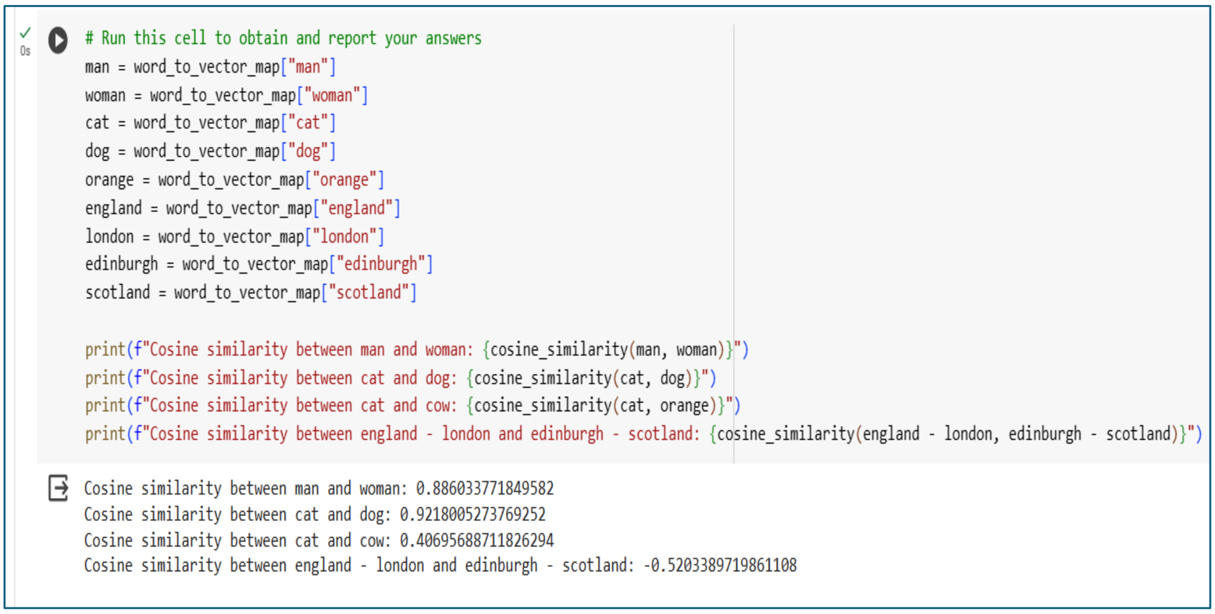
**Note**: We will be refering to the embedding of a word i.e the word vector and the word interchangeably in this lab.

**Task 1a:** Implement equation 1 in the `cosine_similarity()` function below.
Hint: check out the numpy documentation on np.dot, np.sum, and np.sqrt. Depending on how you choose to implement it, you can check out np.linalg.norm.

0s

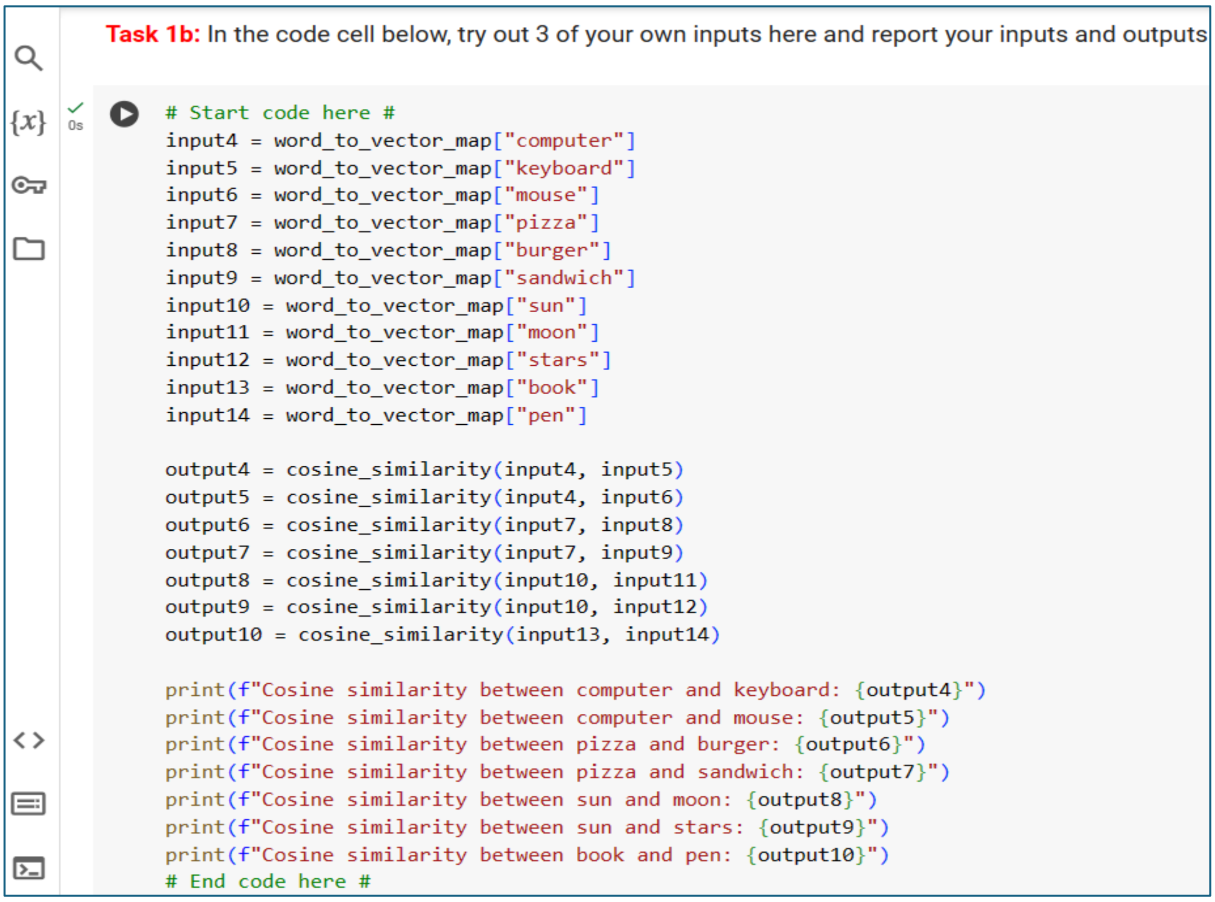
```python
def cosine_similarity(vector1, vector2):
    """
    Calculates the cosine similarity of two word vectors - vector1 and vector2
    Arguments:
        vector1 (ndarray): A word vector having shape (n,)
        vector2 (ndarray): A word vector having shape (n,)
    Returns:
        cosine_similarity (float): The cosine similarity between vector1 and vector2
    """

    # Start Code Here #
    # Compute the dot product between vector1 and vector2 (~ 1 line)
    dot = np.dot(vector1, vector2)

    # Compute the Euclidean norm or length of vector1 (~ 1 line)
    norm_vector1 = np.linalg.norm(vector1)

    # Compute the Euclidean norm or length of vector2 (~ 1 line)
    norm_vector2 = np.linalg.norm(vector2)

    # Compute the cosine similarity as defined in equation 1 (~ 1 line)
    cosine_similarity =  dot / (norm_vector1 * norm_vector2)
    # End Code Here #

    return cosine_similarity
```

Partha Sai - pua528



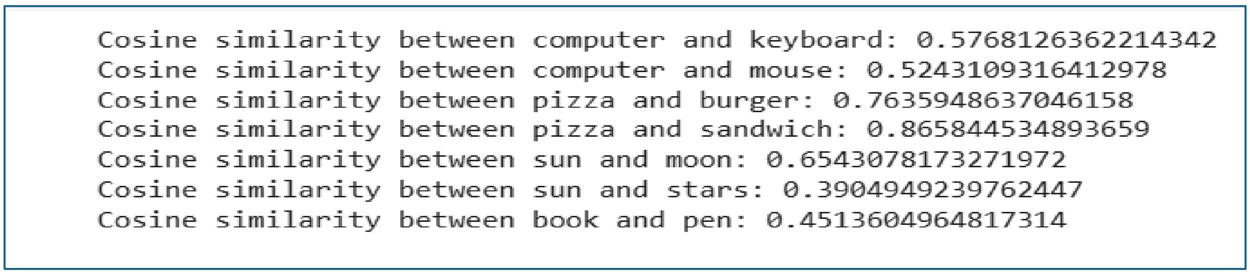
```
# Run this cell to obtain and report your answers
man = word_to_vector_map["man"]
woman = word_to_vector_map["woman"]
cat = word_to_vector_map["cat"]
dog = word_to_vector_map["dog"]
orange = word_to_vector_map["orange"]
england = word_to_vector_map["england"]
london = word_to_vector_map["london"]
edinburgh = word_to_vector_map["edinburgh"]
scotland = word_to_vector_map["scotland"]

print(f"Cosine similarity between man and woman: {cosine_similarity(man, woman)}")
print(f"Cosine similarity between cat and dog: {cosine_similarity(cat, dog)}")
print(f"Cosine similarity between cat and cow: {cosine_similarity(cat, orange)}")
print(f"Cosine similarity between england - london and edinburgh - scotland: {cosine_similarity(england - london, edinburgh - scotland)}")
```

```
Cosine similarity between man and woman: 0.886033771849582
Cosine similarity between cat and dog: 0.9218005273769252
Cosine similarity between cat and cow: 0.40695688711826294
Cosine similarity between england - london and edinburgh - scotland: -0.5203389719861108
```



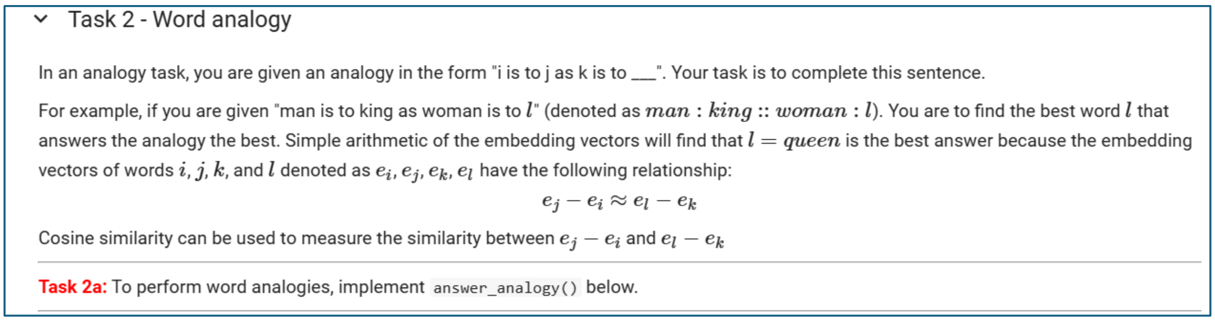
**Task 1b:** In the code cell below, try out 3 of your own inputs here and report your inputs and outputs

```
# Start code here #
input4 = word_to_vector_map["computer"]
input5 = word_to_vector_map["keyboard"]
input6 = word_to_vector_map["mouse"]
input7 = word_to_vector_map["pizza"]
input8 = word_to_vector_map["burger"]
input9 = word_to_vector_map["sandwich"]
input10 = word_to_vector_map["sun"]
input11 = word_to_vector_map["moon"]
input12 = word_to_vector_map["stars"]
input13 = word_to_vector_map["book"]
input14 = word_to_vector_map["pen"]

output4 = cosine_similarity(input4, input5)
output5 = cosine_similarity(input4, input6)
output6 = cosine_similarity(input7, input8)
output7 = cosine_similarity(input7, input9)
output8 = cosine_similarity(input10, input11)
output9 = cosine_similarity(input10, input12)
output10 = cosine_similarity(input13, input14)

print(f"Cosine similarity between computer and keyboard: {output4}")
print(f"Cosine similarity between computer and mouse: {output5}")
print(f"Cosine similarity between pizza and burger: {output6}")
print(f"Cosine similarity between pizza and sandwich: {output7}")
print(f"Cosine similarity between sun and moon: {output8}")
print(f"Cosine similarity between sun and stars: {output9}")
print(f"Cosine similarity between book and pen: {output10}")
# End code here #
```



```
Cosine similarity between computer and keyboard: 0.5768126362214342
Cosine similarity between computer and mouse: 0.5243109316412978
Cosine similarity between pizza and burger: 0.7635948637046158
Cosine similarity between pizza and sandwich: 0.865844534893659
Cosine similarity between sun and moon: 0.6543078173271972
Cosine similarity between sun and stars: 0.3904949239762447
Cosine similarity between book and pen: 0.4513604964817314
```

Partha Sai - pua528



## Task 2 - Word analogy

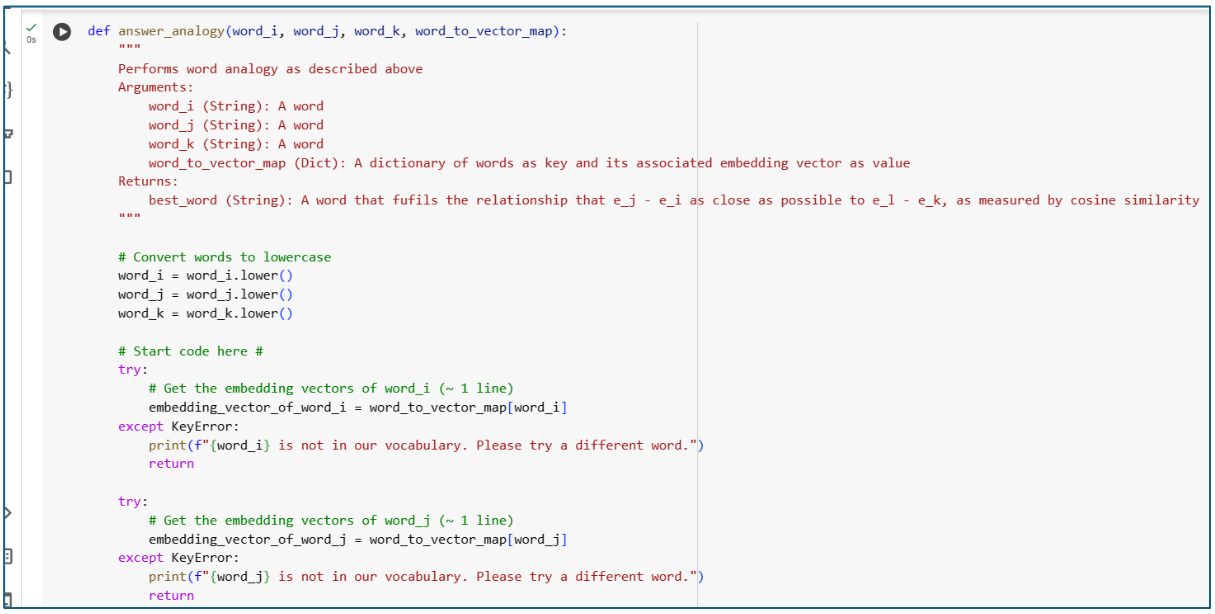
In an analogy task, you are given an analogy in the form "i is to j as k is to ___". Your task is to complete this sentence.

For example, if you are given "man is to king as woman is to $l$" (denoted as $man : king :: woman : l$). You are to find the best word $l$ that answers the analogy the best. Simple arithmetic of the embedding vectors will find that $l = queen$ is the best answer because the embedding vectors of words $i$, $j$, $k$, and $l$ denoted as $e_i, e_j, e_k, e_l$ have the following relationship:

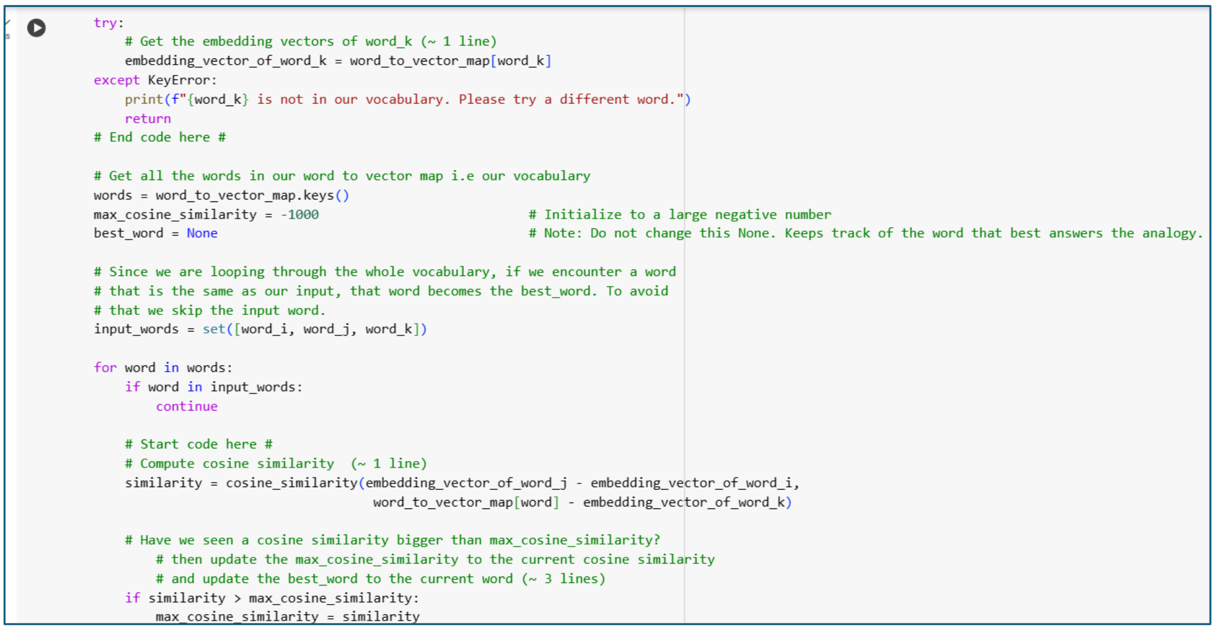
$$e_j - e_i \approx e_l - e_k$$

Cosine similarity can be used to measure the similarity between $e_j - e_i$ and $e_l - e_k$

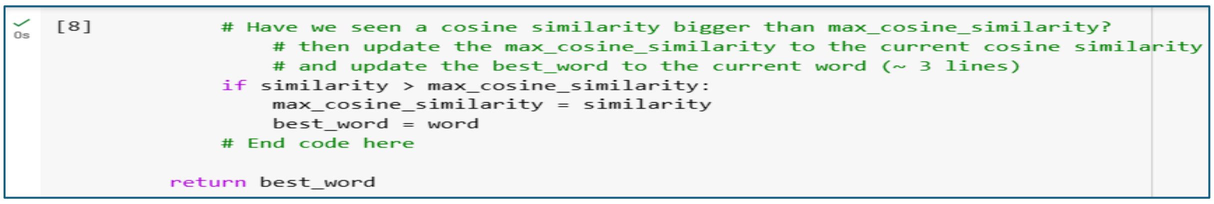
**Task 2a:** To perform word analogies, implement `answer_analogy()` below.



```python
def answer_analogy(word_i, word_j, word_k, word_to_vector_map):
    """
    Performs word analogy as described above
    Arguments:
        word_i (String): A word
        word_j (String): A word
        word_k (String): A word
        word_to_vector_map (Dict): A dictionary of words as key and its associated embedding vector as value
    Returns:
        best_word (String): A word that fufils the relationship that e_j - e_i as close as possible to e_l - e_k, as measured by cosine similarity
    """

    # Convert words to lowercase
    word_i = word_i.lower()
    word_j = word_j.lower()
    word_k = word_k.lower()

    # Start code here #
    try:
        # Get the embedding vectors of word_i (~ 1 line)
        embedding_vector_of_word_i = word_to_vector_map[word_i]
    except KeyError:
        print(f"{word_i} is not in our vocabulary. Please try a different word.")
        return

    try:
        # Get the embedding vectors of word_j (~ 1 line)
        embedding_vector_of_word_j = word_to_vector_map[word_j]
    except KeyError:
        print(f"{word_j} is not in our vocabulary. Please try a different word.")
        return
```

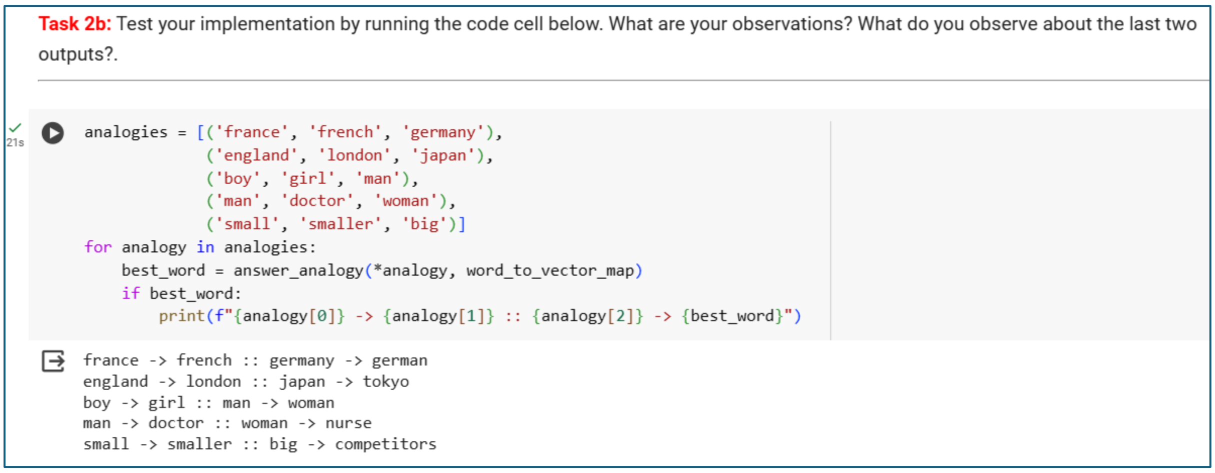


```python
    try:
        # Get the embedding vectors of word_k (~ 1 line)
        embedding_vector_of_word_k = word_to_vector_map[word_k]
    except KeyError:
        print(f"{word_k} is not in our vocabulary. Please try a different word.")
        return
    # End code here #

    # Get all the words in our word to vector map i.e our vocabulary
    words = word_to_vector_map.keys()
    max_cosine_similarity = -1000                          # Initialize to a large negative number
    best_word = None                                       # Note: Do not change this None. Keeps track of the word that best answers the analogy.

    # Since we are looping through the whole vocabulary, if we encounter a word
    # that is the same as our input, that word becomes the best_word. To avoid
    # that we skip the input word.
    input_words = set([word_i, word_j, word_k])

    for word in words:
        if word in input_words:
            continue

        # Start code here #
        # Compute cosine similarity  (~ 1 line)
        similarity = cosine_similarity(embedding_vector_of_word_j - embedding_vector_of_word_i,
                                       word_to_vector_map[word] - embedding_vector_of_word_k)

        # Have we seen a cosine similarity bigger than max_cosine_similarity?
            # then update the max_cosine_similarity to the current cosine similarity
            # and update the best_word to the current word (~ 3 lines)
        if similarity > max_cosine_similarity:
            max_cosine_similarity = similarity
```



```python
[8]
        # Have we seen a cosine similarity bigger than max_cosine_similarity?
            # then update the max_cosine_similarity to the current cosine similarity
            # and update the best_word to the current word (~ 3 lines)
        if similarity > max_cosine_similarity:
            max_cosine_similarity = similarity
            best_word = word
        # End code here

    return best_word
```

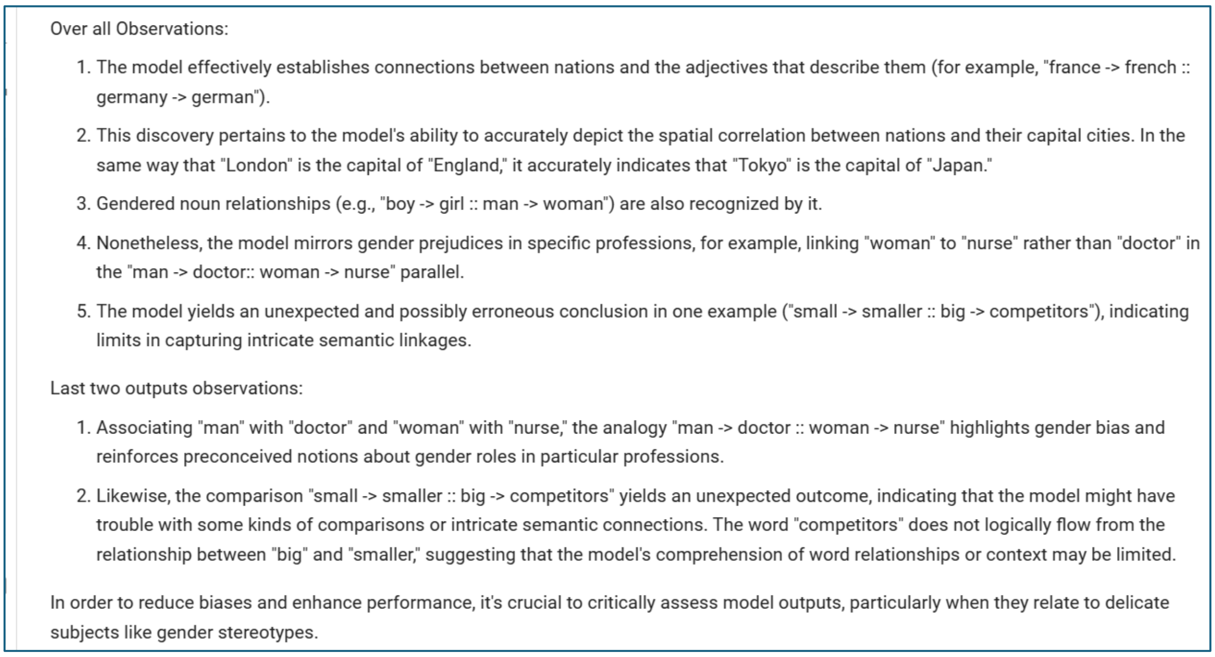
Partha Sai - pua528



**Task 2b:** Test your implementation by running the code cell below. What are your observations? What do you observe about the last two outputs?.

```
analogies = [('france', 'french', 'germany'),
             ('england', 'london', 'japan'),
             ('boy', 'girl', 'man'),
             ('man', 'doctor', 'woman'),
             ('small', 'smaller', 'big')]
for analogy in analogies:
    best_word = answer_analogy(*analogy, word_to_vector_map)
    if best_word:
        print(f"{analogy[0]} -> {analogy[1]} :: {analogy[2]} -> {best_word}")
```

```
france -> french :: germany -> german
england -> london :: japan -> tokyo
boy -> girl :: man -> woman
man -> doctor :: woman -> nurse
small -> smaller :: big -> competitors
```



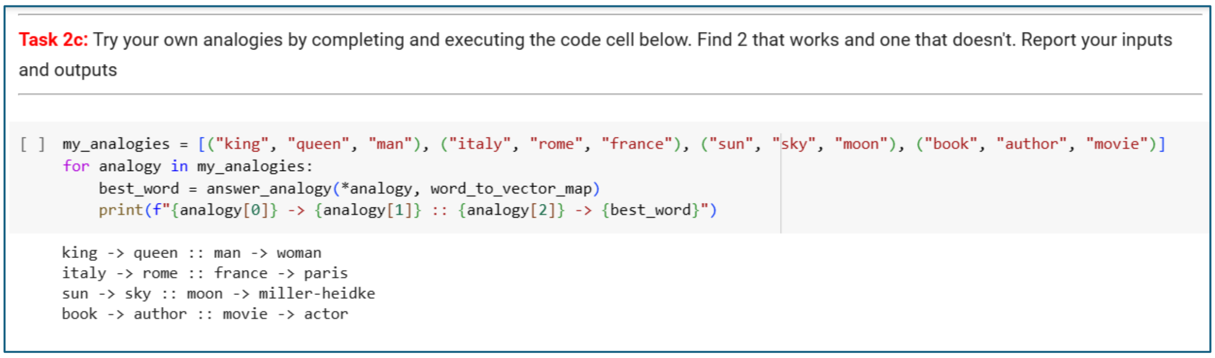
Over all Observations:

1. The model effectively establishes connections between nations and the adjectives that describe them (for example, "france -> french :: germany -> german").
2. This discovery pertains to the model's ability to accurately depict the spatial correlation between nations and their capital cities. In the same way that "London" is the capital of "England," it accurately indicates that "Tokyo" is the capital of "Japan."
3. Gendered noun relationships (e.g., "boy -> girl :: man -> woman") are also recognized by it.
4. Nonetheless, the model mirrors gender prejudices in specific professions, for example, linking "woman" to "nurse" rather than "doctor" in the "man -> doctor:: woman -> nurse" parallel.
5. The model yields an unexpected and possibly erroneous conclusion in one example ("small -> smaller :: big -> competitors"), indicating limits in capturing intricate semantic linkages.

Last two outputs observations:

1. Associating "man" with "doctor" and "woman" with "nurse," the analogy "man -> doctor :: woman -> nurse" highlights gender bias and reinforces preconceived notions about gender roles in particular professions.
2. Likewise, the comparison "small -> smaller :: big -> competitors" yields an unexpected outcome, indicating that the model might have trouble with some kinds of comparisons or intricate semantic connections. The word "competitors" does not logically flow from the relationship between "big" and "smaller," suggesting that the model's comprehension of word relationships or context may be limited.

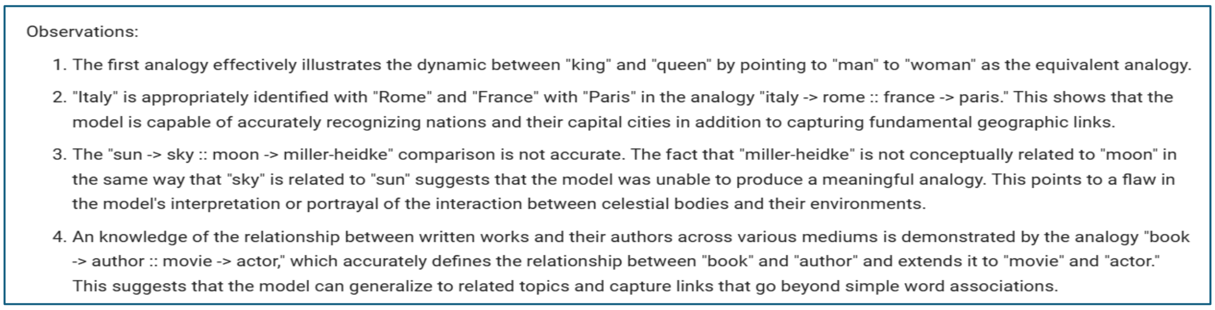
In order to reduce biases and enhance performance, it's crucial to critically assess model outputs, particularly when they relate to delicate subjects like gender stereotypes.



**Task 2c:** Try your own analogies by completing and executing the code cell below. Find 2 that works and one that doesn't. Report your inputs and outputs

```
[ ] my_analogies = [("king", "queen", "man"), ("italy", "rome", "france"), ("sun", "sky", "moon"), ("book", "author", "movie")]
    for analogy in my_analogies:
        best_word = answer_analogy(*analogy, word_to_vector_map)
        print(f"{analogy[0]} -> {analogy[1]} :: {analogy[2]} -> {best_word}")
```

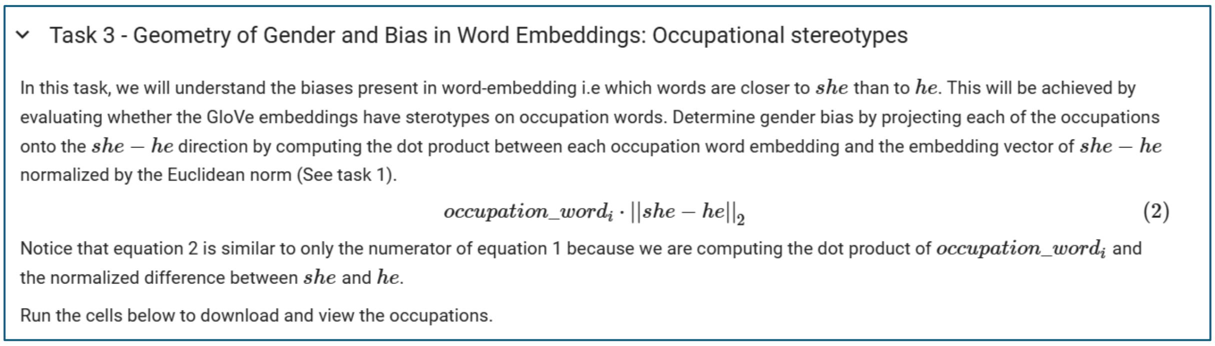
```
king -> queen :: man -> woman
italy -> rome :: france -> paris
sun -> sky :: moon -> miller-heidke
book -> author :: movie -> actor
```



Observations:

1. The first analogy effectively illustrates the dynamic between "king" and "queen" by pointing to "man" to "woman" as the equivalent analogy.
2. "Italy" is appropriately identified with "Rome" and "France" with "Paris" in the analogy "italy -> rome :: france -> paris." This shows that the model is capable of accurately recognizing nations and their capital cities in addition to capturing fundamental geographic links.
3. The "sun -> sky :: moon -> miller-heidke" comparison is not accurate. The fact that "miller-heidke" is not conceptually related to "moon" in the same way that "sky" is related to "sun" suggests that the model was unable to produce a meaningful analogy. This points to a flaw in the model's interpretation or portrayal of the interaction between celestial bodies and their environments.
4. An knowledge of the relationship between written works and their authors across various mediums is demonstrated by the analogy "book -> author :: movie -> actor," which accurately defines the relationship between "book" and "author" and extends it to "movie" and "actor." This suggests that the model can generalize to related topics and capture links that go beyond simple word associations.

Partha Sai - pua528



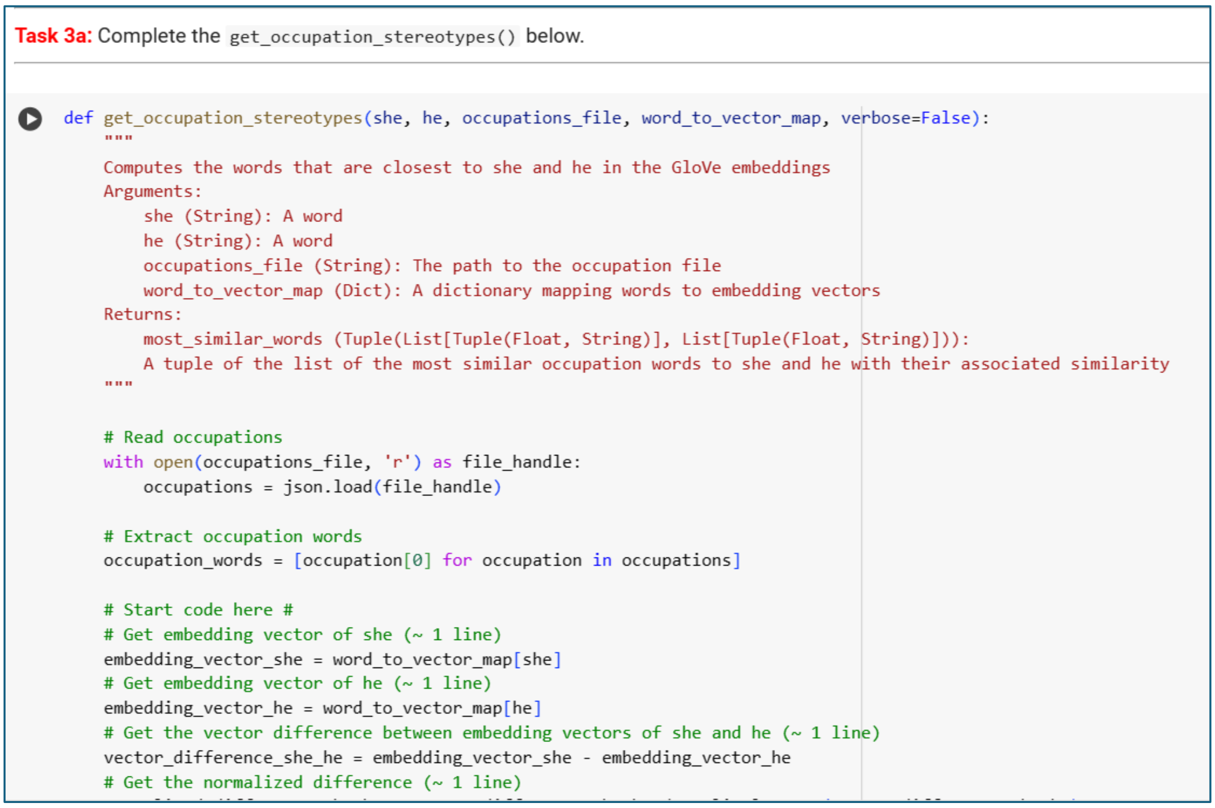
## Task 3 - Geometry of Gender and Bias in Word Embeddings: Occupational stereotypes

In this task, we will understand the biases present in word-embedding i.e which words are closer to $she$ than to $he$. This will be achieved by evaluating whether the GloVe embeddings have sterotypes on occupation words. Determine gender bias by projecting each of the occupations onto the $she - he$ direction by computing the dot product between each occupation word embedding and the embedding vector of $she - he$ normalized by the Euclidean norm (See task 1).

$$occupation\_word_i \cdot ||she - he||_2 \tag{2}$$

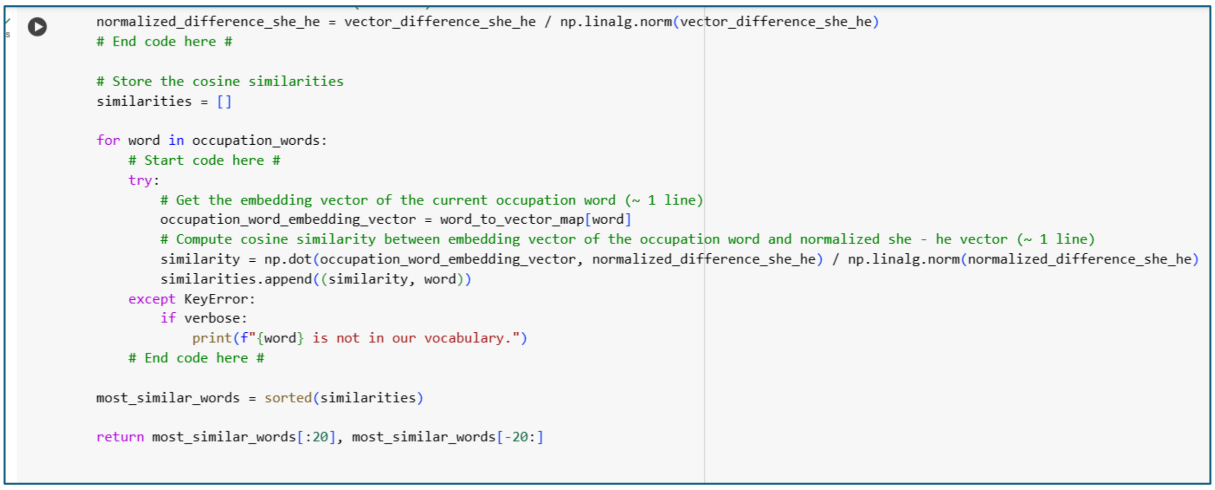
Notice that equation 2 is similar to only the numerator of equation 1 because we are computing the dot product of $occupation\_word_i$ and the normalized difference between $she$ and $he$.

Run the cells below to download and view the occupations.

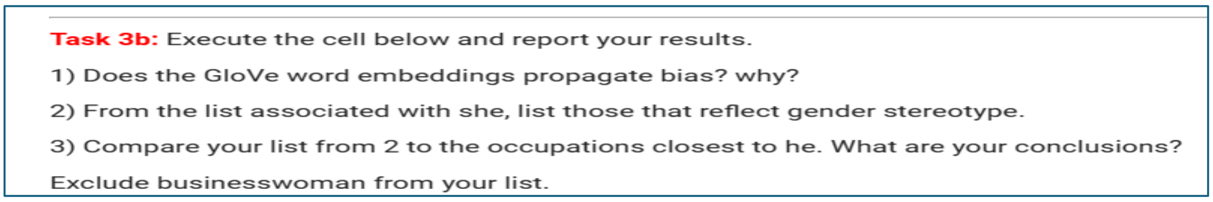


**Task 3a:** Complete the `get_occupation_stereotypes()` below.

```python
def get_occupation_stereotypes(she, he, occupations_file, word_to_vector_map, verbose=False):
    """
    Computes the words that are closest to she and he in the GloVe embeddings
    Arguments:
        she (String): A word
        he (String): A word
        occupations_file (String): The path to the occupation file
        word_to_vector_map (Dict): A dictionary mapping words to embedding vectors
    Returns:
        most_similar_words (Tuple(List[Tuple(Float, String)], List[Tuple(Float, String)])):
        A tuple of the list of the most similar occupation words to she and he with their associated similarity
    """

    # Read occupations
    with open(occupations_file, 'r') as file_handle:
        occupations = json.load(file_handle)

    # Extract occupation words
    occupation_words = [occupation[0] for occupation in occupations]

    # Start code here #
    # Get embedding vector of she (~ 1 line)
    embedding_vector_she = word_to_vector_map[she]
    # Get embedding vector of he (~ 1 line)
    embedding_vector_he = word_to_vector_map[he]
    # Get the vector difference between embedding vectors of she and he (~ 1 line)
    vector_difference_she_he = embedding_vector_she - embedding_vector_he
    # Get the normalized difference (~ 1 line)
```



```python
    normalized_difference_she_he = vector_difference_she_he / np.linalg.norm(vector_difference_she_he)
    # End code here #

    # Store the cosine similarities
    similarities = []

    for word in occupation_words:
        # Start code here #
        try:
            # Get the embedding vector of the current occupation word (~ 1 line)
            occupation_word_embedding_vector = word_to_vector_map[word]
            # Compute cosine similarity between embedding vector of the occupation word and normalized she - he vector (~ 1 line)
            similarity = np.dot(occupation_word_embedding_vector, normalized_difference_she_he) / np.linalg.norm(normalized_difference_she_he)
            similarities.append((similarity, word))
        except KeyError:
            if verbose:
                print(f"{word} is not in our vocabulary.")
        # End code here #

    most_similar_words = sorted(similarities)

    return most_similar_words[:20], most_similar_words[-20:]
```



**Task 3b:** Execute the cell below and report your results.

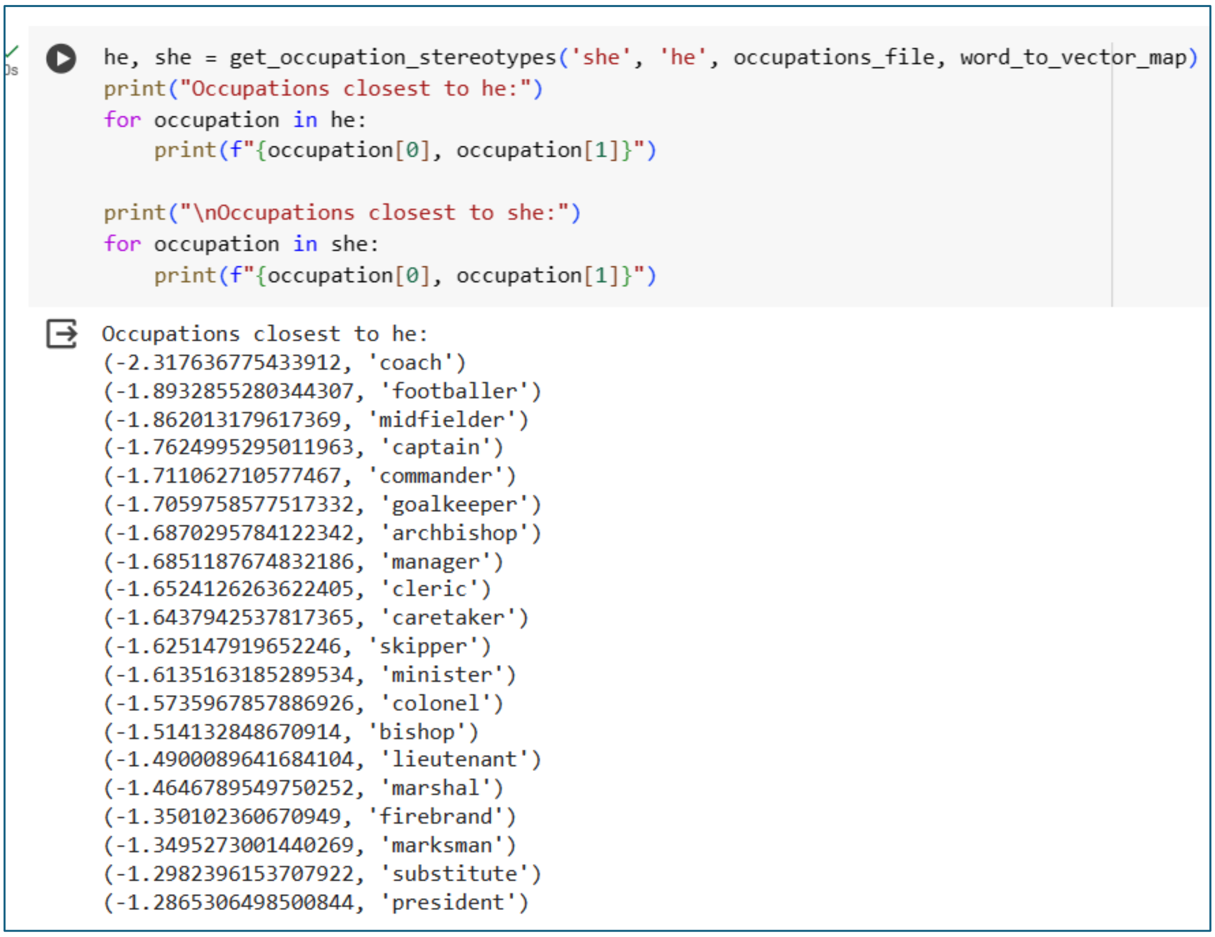
1) Does the GloVe word embeddings propagate bias? why?

2) From the list associated with she, list those that reflect gender stereotype.

3) Compare your list from 2 to the occupations closest to he. What are your conclusions?

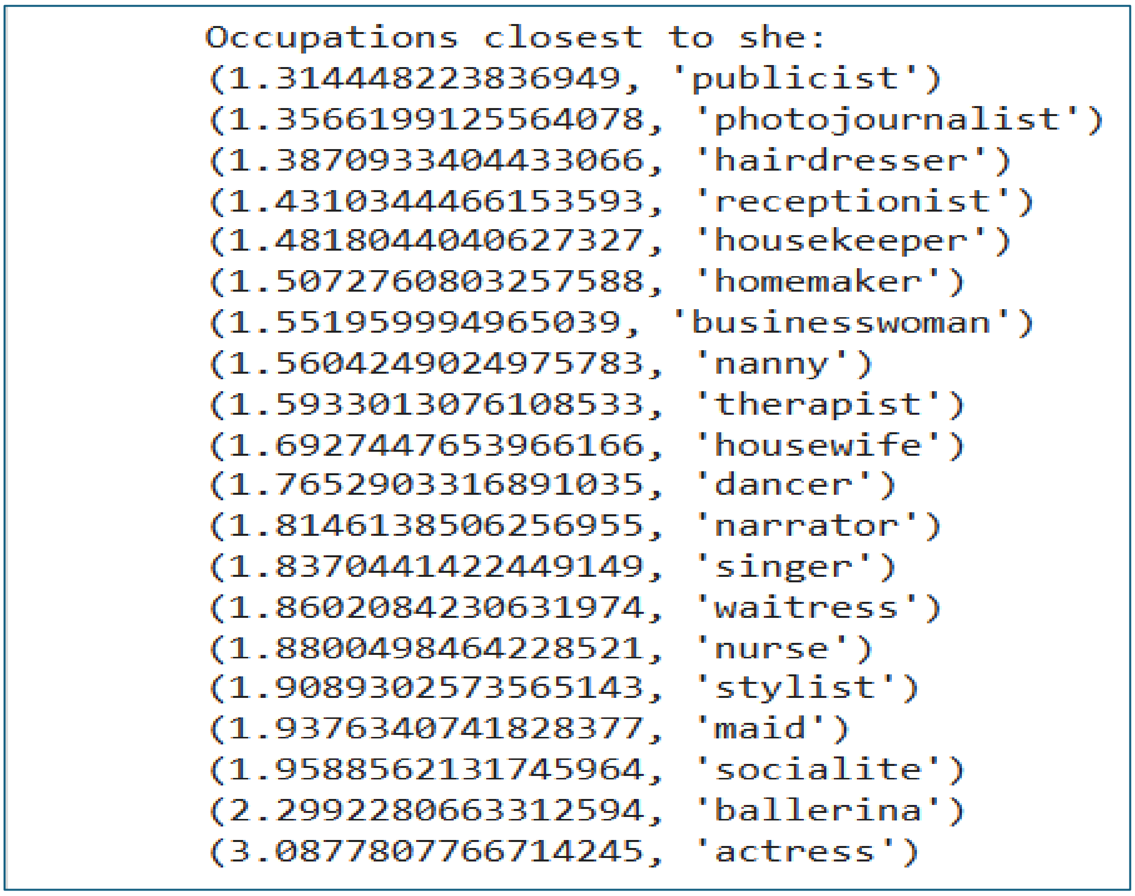
Exclude businesswoman from your list.

Partha Sai - pua528



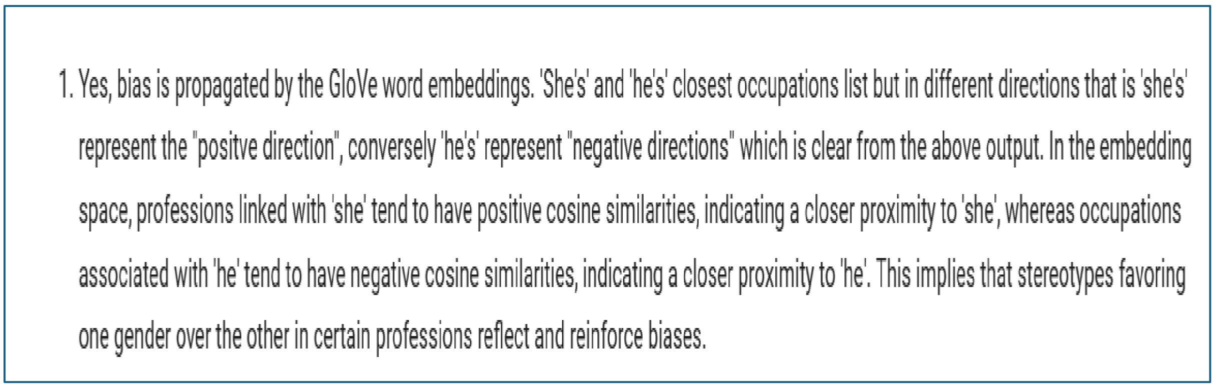
```python
he, she = get_occupation_stereotypes('she', 'he', occupations_file, word_to_vector_map)
print("Occupations closest to he:")
for occupation in he:
    print(f"{occupation[0]}, occupation[1]}")

print("\nOccupations closest to she:")
for occupation in she:
    print(f"{occupation[0]}, occupation[1]}")
```

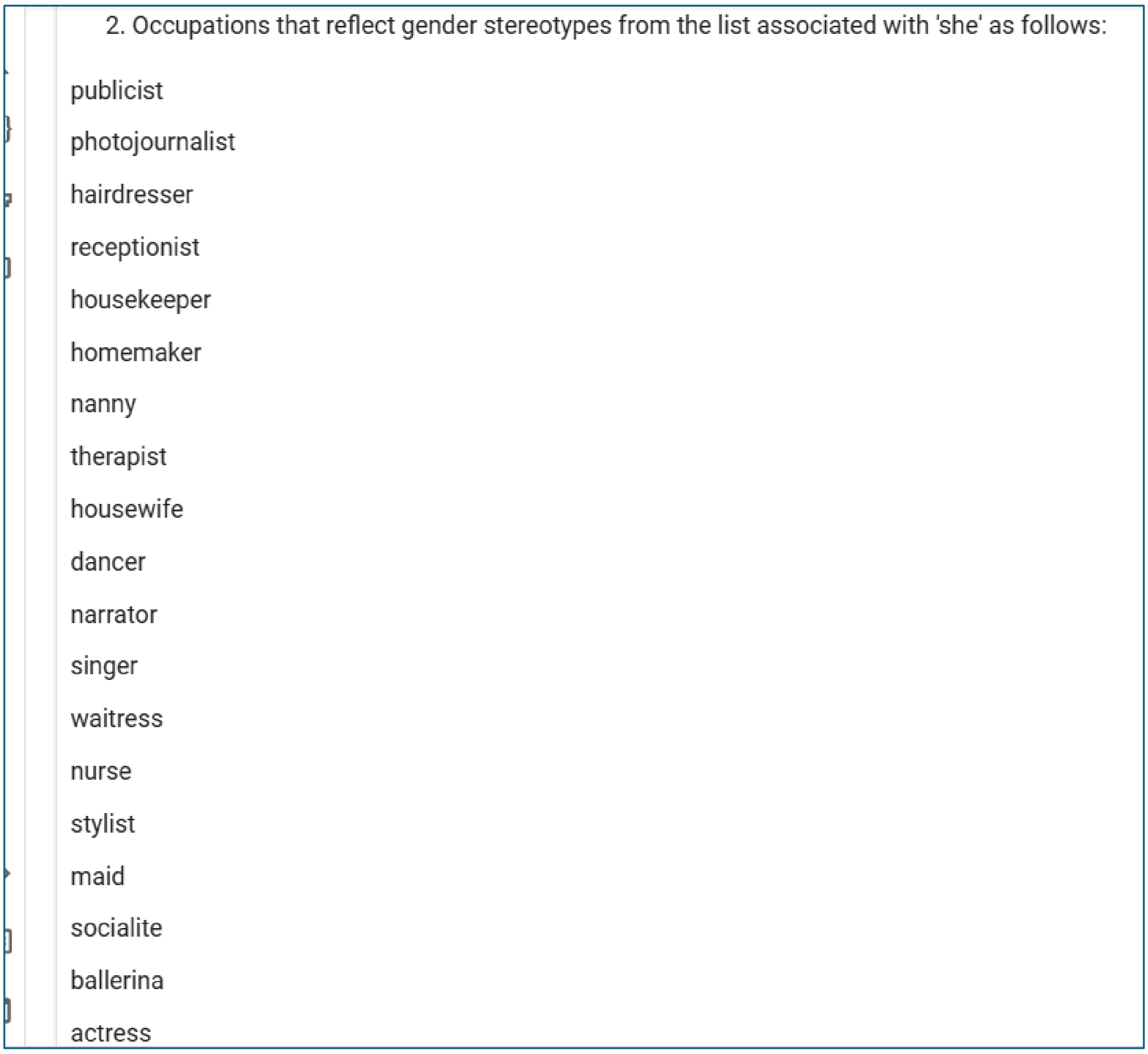
```
Occupations closest to he:
(-2.317636775433912, 'coach')
(-1.8932855280344307, 'footballer')
(-1.862013179617369, 'midfielder')
(-1.7624995295011963, 'captain')
(-1.711062710577467, 'commander')
(-1.7059758577517332, 'goalkeeper')
(-1.6870295784122342, 'archbishop')
(-1.6851187674832186, 'manager')
(-1.6524126263622405, 'cleric')
(-1.6437942537817365, 'caretaker')
(-1.625147919652246, 'skipper')
(-1.6135163185289534, 'minister')
(-1.5735967857886926, 'colonel')
(-1.514132848670914, 'bishop')
(-1.4900089641684104, 'lieutenant')
(-1.4646789549750252, 'marshal')
(-1.350102360670949, 'firebrand')
(-1.3495273001440269, 'marksman')
(-1.2982396153707922, 'substitute')
(-1.2865306498500844, 'president')
```



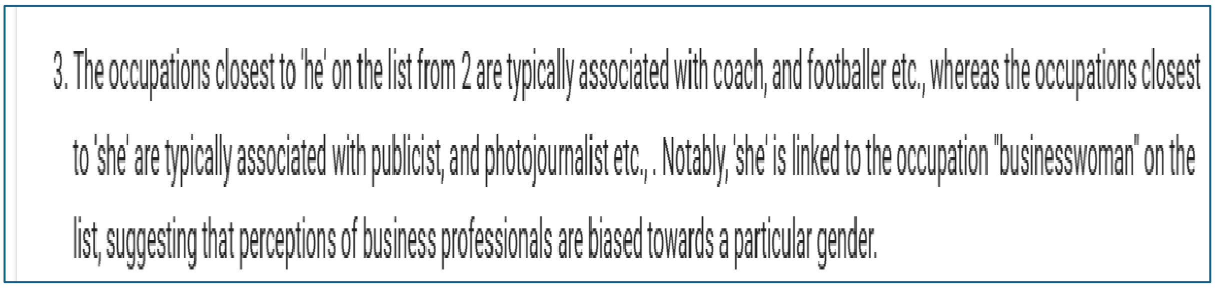
```
Occupations closest to she:
(1.314448223836949, 'publicist')
(1.3566199125564078, 'photojournalist')
(1.3870933404433066, 'hairdresser')
(1.4310344466153593, 'receptionist')
(1.4818044040627327, 'housekeeper')
(1.5072760803257588, 'homemaker')
(1.551959994965039, 'businesswoman')
(1.5604249024975783, 'nanny')
(1.5933013076108533, 'therapist')
(1.6927447653966166, 'housewife')
(1.7652903316891035, 'dancer')
(1.8146138506256955, 'narrator')
(1.8370441422449149, 'singer')
(1.8602084230631974, 'waitress')
(1.8800498464228521, 'nurse')
(1.9089302573565143, 'stylist')
(1.9376340741828377, 'maid')
(1.9588562131745964, 'socialite')
(2.2992280663312594, 'ballerina')
(3.0877807766714245, 'actress')
```

Partha Sai - pua528

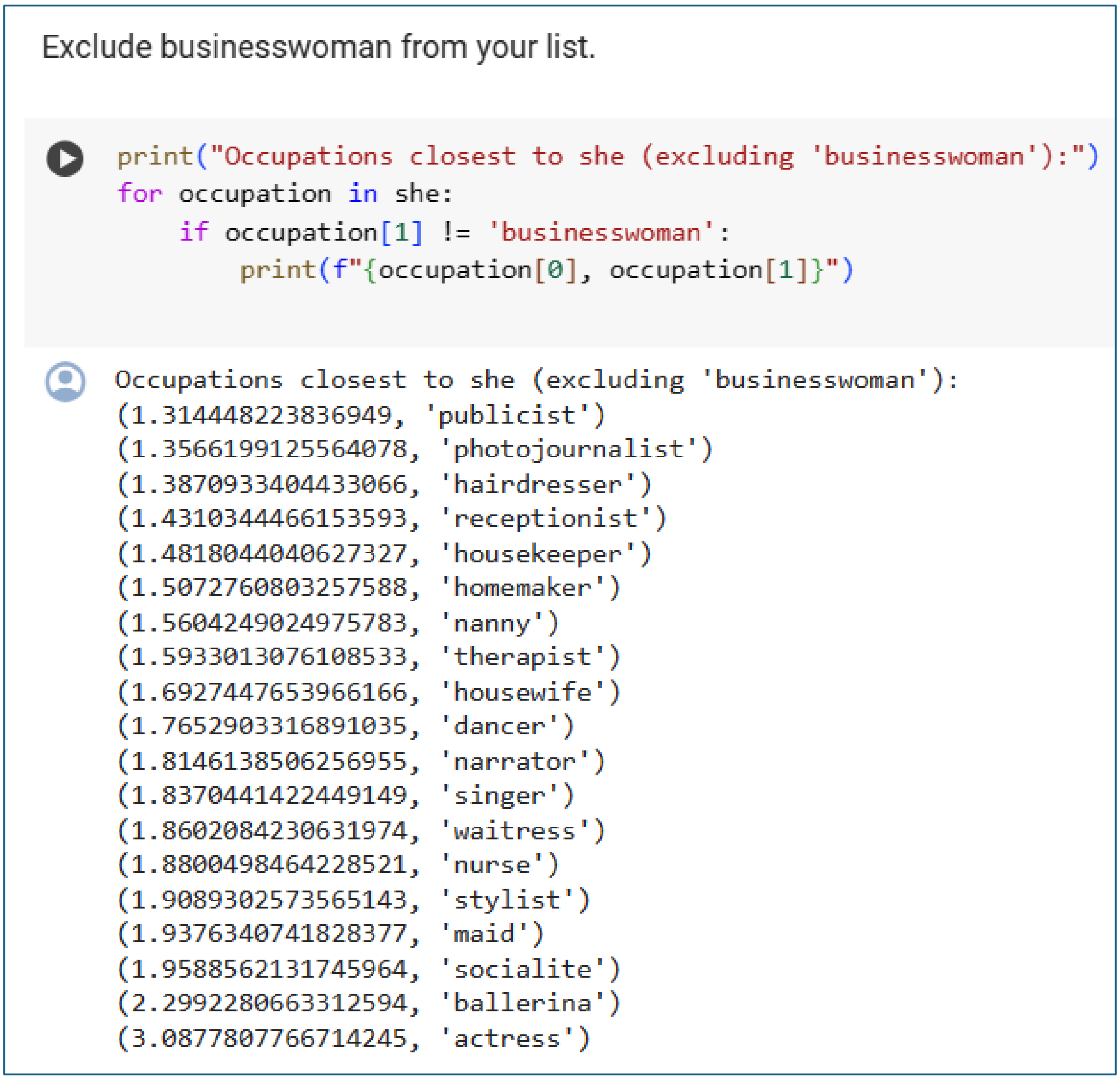

1. Yes, bias is propagated by the GloVe word embeddings. 'She's' and 'he's' closest occupations list but in different directions that is 'she's' represent the "positve direction", conversely 'he's' represent "negative directions" which is clear from the above output. In the embedding space, professions linked with 'she' tend to have positive cosine similarities, indicating a closer proximity to 'she', whereas occupations associated with 'he' tend to have negative cosine similarities, indicating a closer proximity to 'he'. This implies that stereotypes favoring one gender over the other in certain professions reflect and reinforce biases.


2. Occupations that reflect gender stereotypes from the list associated with 'she' as follows:

publicist
photojournalist
hairdresser
receptionist
housekeeper
homemaker
nanny
therapist
housewife
dancer
narrator
singer
waitress
nurse
stylist
maid
socialite
ballerina
actress


3. The occupations closest to 'he' on the list from 2 are typically associated with coach, and footballer etc., whereas the occupations closest to 'she' are typically associated with publicist, and photojournalist etc., . Notably, 'she' is linked to the occupation "businesswoman" on the list, suggesting that perceptions of business professionals are biased towards a particular gender.

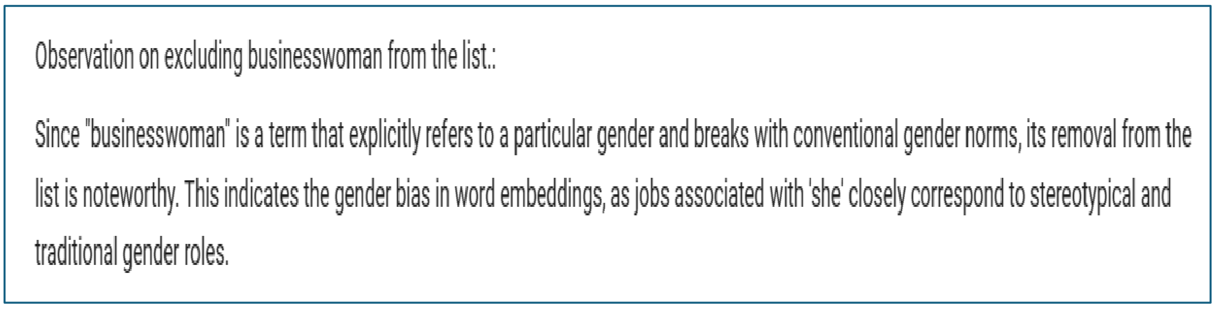
Partha Sai - pua528



Exclude businesswoman from your list.

```python
print("Occupations closest to she (excluding 'businesswoman'):")
for occupation in she:
    if occupation[1] != 'businesswoman':
        print(f"{occupation[0], occupation[1]}")
```

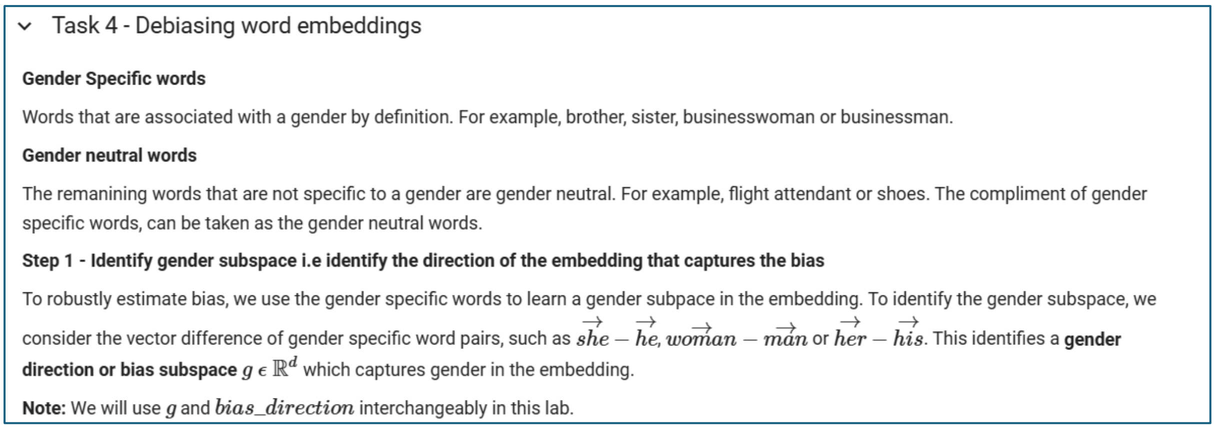
```
Occupations closest to she (excluding 'businesswoman'):
(1.314448223836949, 'publicist')
(1.3566199125564078, 'photojournalist')
(1.3870933404433066, 'hairdresser')
(1.4310344466153593, 'receptionist')
(1.4818044040627327, 'housekeeper')
(1.5072760803257588, 'homemaker')
(1.5604249024975783, 'nanny')
(1.5933013076108533, 'therapist')
(1.6927447653966166, 'housewife')
(1.7652903316891035, 'dancer')
(1.8146138506256955, 'narrator')
(1.8370441422449149, 'singer')
(1.8602084230631974, 'waitress')
(1.8800498464228521, 'nurse')
(1.9089302573565143, 'stylist')
(1.9376340741828377, 'maid')
(1.9588562131745964, 'socialite')
(2.2992280663312594, 'ballerina')
(3.0877807766714245, 'actress')
```



Observation on excluding businesswoman from the list.:

Since "businesswoman" is a term that explicitly refers to a particular gender and breaks with conventional gender norms, its removal from the list is noteworthy. This indicates the gender bias in word embeddings, as jobs associated with 'she' closely correspond to stereotypical and traditional gender roles.

Partha Sai - pua528



## Task 4 - Debiasing word embeddings

**Gender Specific words**

Words that are associated with a gender by definition. For example, brother, sister, businesswoman or businessman.

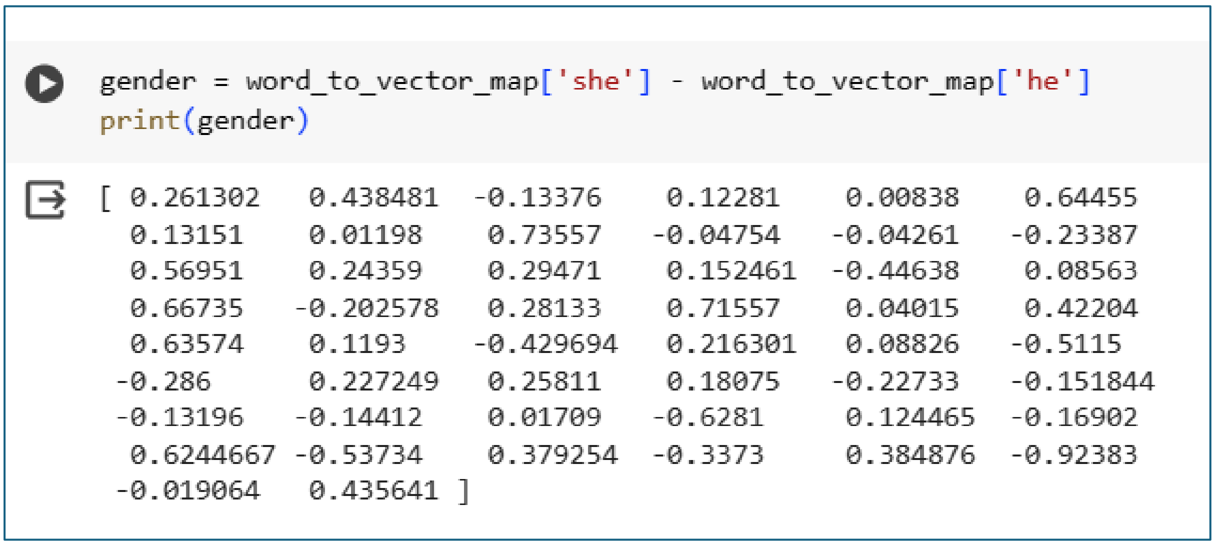
**Gender neutral words**

The remanining words that are not specific to a gender are gender neutral. For example, flight attendant or shoes. The compliment of gender specific words, can be taken as the gender neutral words.

**Step 1 - Identify gender subspace i.e identify the direction of the embedding that captures the bias**

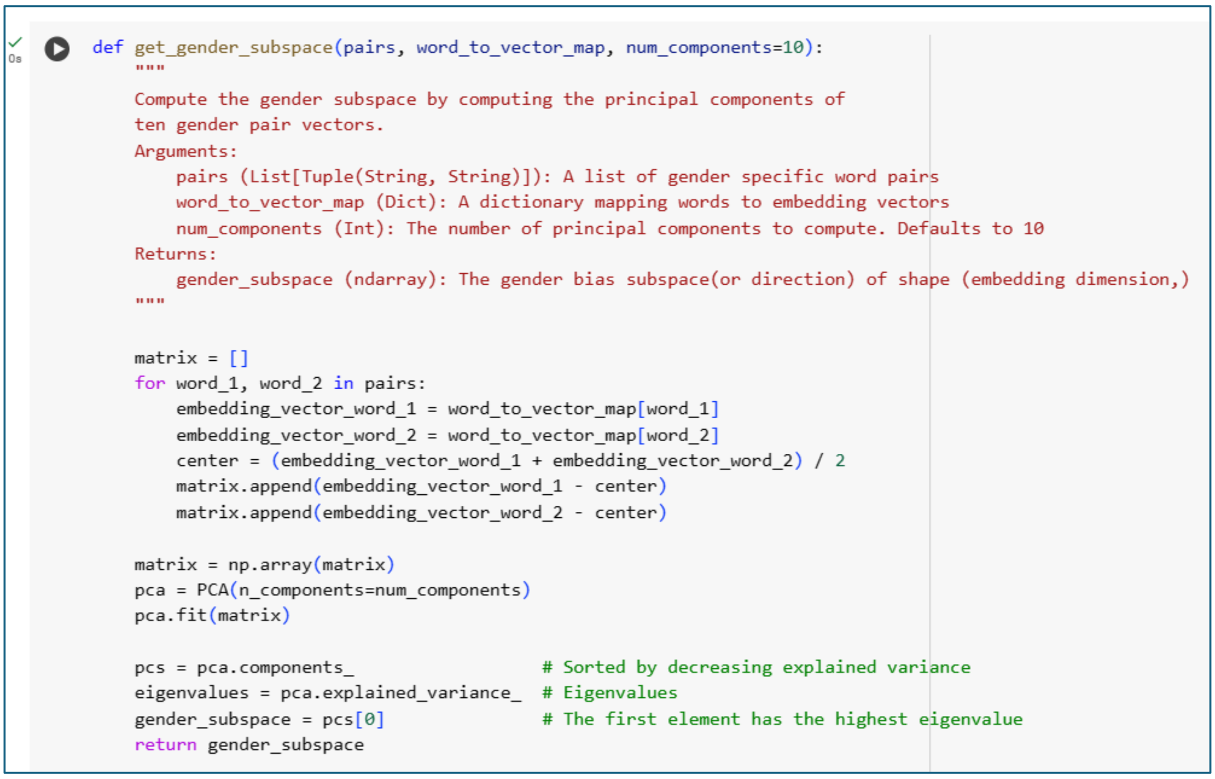
To robustly estimate bias, we use the gender specific words to learn a gender subpace in the embedding. To identify the gender subspace, we consider the vector difference of gender specific word pairs, such as $\vec{she} - \vec{he}$, $\vec{woman} - \vec{man}$ or $\vec{her} - \vec{his}$. This identifies a **gender direction or bias subspace** $g \epsilon \mathbb{R}^d$ which captures gender in the embedding.

**Note:** We will use $g$ and $bias\_direction$ interchangeably in this lab.



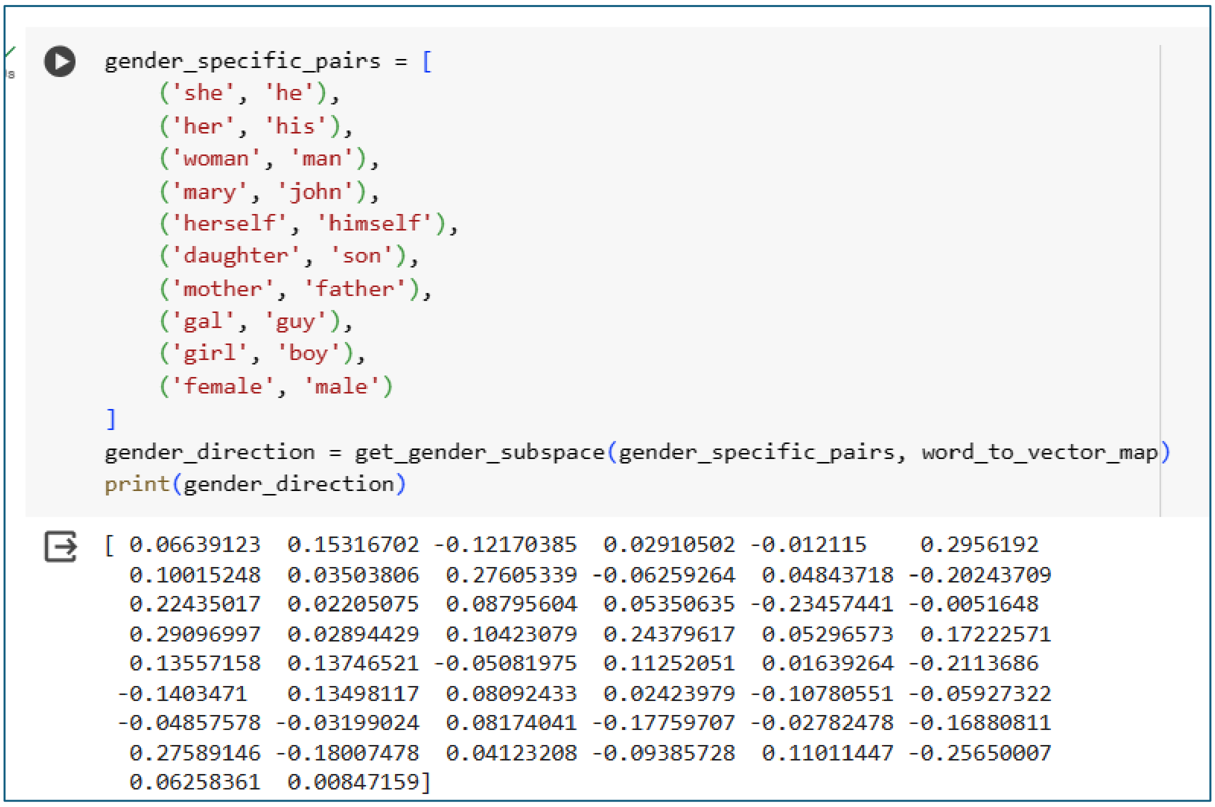
```
gender = word_to_vector_map['she'] - word_to_vector_map['he']
print(gender)
```

```
[ 0.261302   0.438481  -0.13376    0.12281    0.00838    0.64455
  0.13151    0.01198    0.73557   -0.04754   -0.04261   -0.23387
  0.56951    0.24359    0.29471    0.152461  -0.44638    0.08563
  0.66735   -0.202578   0.28133    0.71557    0.04015    0.42204
  0.63574    0.1193    -0.429694   0.216301   0.08826   -0.5115
 -0.286      0.227249   0.25811    0.18075   -0.22733   -0.151844
 -0.13196   -0.14412    0.01709   -0.6281     0.124465  -0.16902
  0.6244667 -0.53734    0.379254  -0.3373     0.384876  -0.92383
 -0.019064   0.435641 ]
```



```
def get_gender_subspace(pairs, word_to_vector_map, num_components=10):
    """
    Compute the gender subspace by computing the principal components of
    ten gender pair vectors.
    Arguments:
        pairs (List[Tuple(String, String)]): A list of gender specific word pairs
        word_to_vector_map (Dict): A dictionary mapping words to embedding vectors
        num_components (Int): The number of principal components to compute. Defaults to 10
    Returns:
        gender_subspace (ndarray): The gender bias subspace(or direction) of shape (embedding dimension,)
    """

    matrix = []
    for word_1, word_2 in pairs:
        embedding_vector_word_1 = word_to_vector_map[word_1]
        embedding_vector_word_2 = word_to_vector_map[word_2]
        center = (embedding_vector_word_1 + embedding_vector_word_2) / 2
        matrix.append(embedding_vector_word_1 - center)
        matrix.append(embedding_vector_word_2 - center)

    matrix = np.array(matrix)
    pca = PCA(n_components=num_components)
    pca.fit(matrix)

    pcs = pca.components_                   # Sorted by decreasing explained variance
    eigenvalues = pca.explained_variance_   # Eigenvalues
    gender_subspace = pcs[0]                # The first element has the highest eigenvalue
    return gender_subspace
```

Partha Sai - pua528



```python
gender_specific_pairs = [
    ('she', 'he'),
    ('her', 'his'),
    ('woman', 'man'),
    ('mary', 'john'),
    ('herself', 'himself'),
    ('daughter', 'son'),
    ('mother', 'father'),
    ('gal', 'guy'),
    ('girl', 'boy'),
    ('female', 'male')
]
gender_direction = get_gender_subspace(gender_specific_pairs, word_to_vector_map)
print(gender_direction)
```

```
[ 0.06639123  0.15316702 -0.12170385  0.02910502 -0.012115    0.2956192
  0.10015248  0.03503806  0.27605339 -0.06259264  0.04843718 -0.20243709
  0.22435017  0.02205075  0.08795604  0.05350635 -0.23457441 -0.0051648
  0.29096997  0.02894429  0.10423079  0.24379617  0.05296573  0.17222571
  0.13557158  0.13746521 -0.05081975  0.11252051  0.01639264 -0.2113686
 -0.1403471   0.13498117  0.08092433  0.02423979 -0.10780551 -0.05927322
 -0.04857578 -0.03199024  0.08174041 -0.17759707 -0.02782478 -0.16880811
  0.27589146 -0.18007478  0.04123208 -0.09385728  0.11011447 -0.25650007
  0.06258361  0.00847159]
```

Partha Sai - pua528

**Task 4a:** Run the cell below to computes the similarity between the gender embedding and the embedding vectors of male and female names. What can you observe?

```python
print('Names and their similarities with simple gender subspace')
names = ["mary", "john", "sweta", "david", "kazim", "angela"]
for name in names:
    print(name, cosine_similarity(word_to_vector_map[name], gender))

print()
print('Names and their similarities with PCA based gender subspace')
names = ["mary", "john", "sweta", "david", "kazim", "angela"]
for name in names:
    print(name, cosine_similarity(word_to_vector_map[name], gender_direction))
```

```
Names and their similarities with simple gender subspace
mary 0.3457399102816379
john -0.17879783833420468
sweta 0.17016456601128147
david -0.133226156007866 7
kazim -0.32658964009764835
angela 0.2600799146632235

Names and their similarities with PCA based gender subspace
mary 0.2637091204419718
john -0.3816839789078354
sweta 0.1773704777691709
david -0.3165647635266187
kazim -0.3249838182709315
angela 0.18623308926276097
```

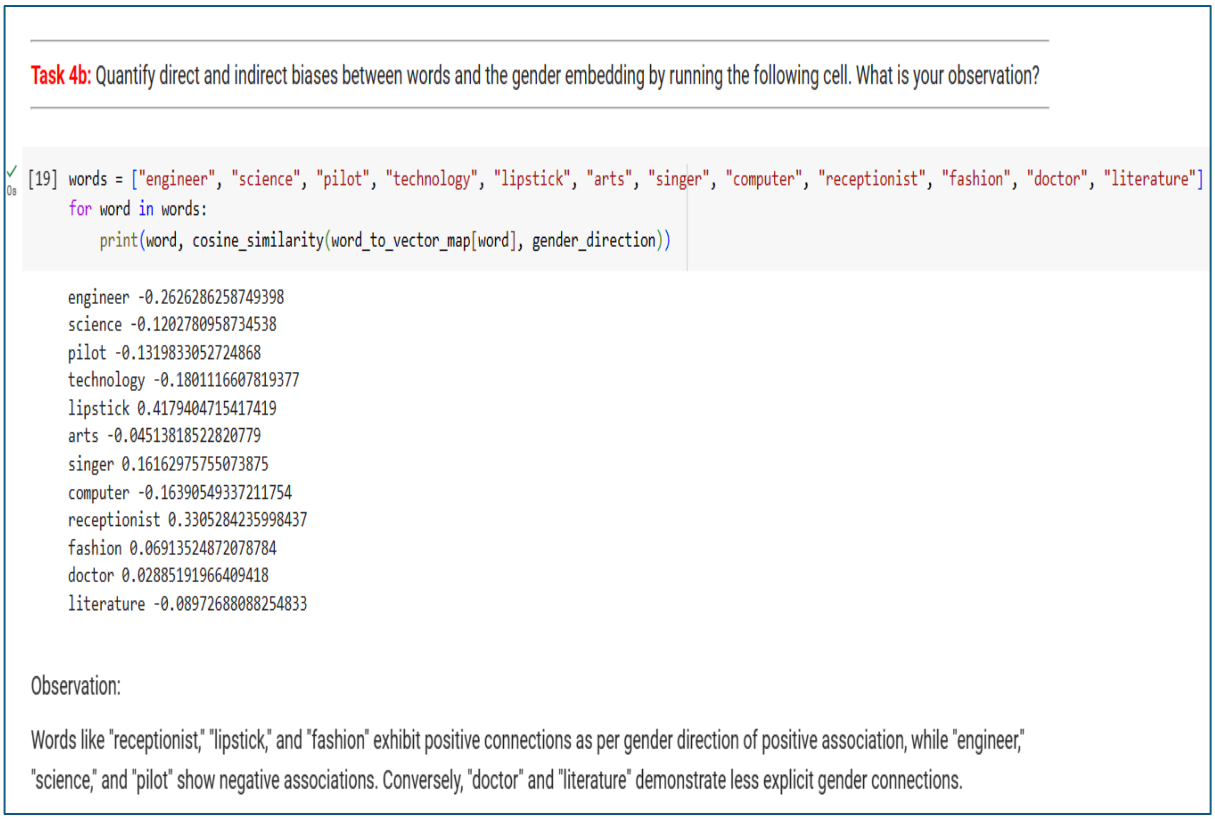
Observation:

1. Names commonly associated with women, such "mary" ,"sweta", and "angela," exhibit positive cosine similarity in the simple gender subspace, suggesting that they are closer to the gender direction vector. On the other hand, names that are typically associated with men, such as "john" , and "david," have negative cosine similarity, indicating that they are further away from the gender direction vector.

2. Similar patterns can be seen in the PCA-based gender subdomain, where names associated with women have positive cosine similarities while those associated with men have negative cosine similarities. The degree of similarity varies, though, suggesting that the PCA-based subspace and the simple gender subspace reflect gender bias in distinct ways.

3. It's interesting to note that some names, like "sweta" and "kazim," show positive similarities with the gender direction vector in both situations.

Partha Sai - pua528



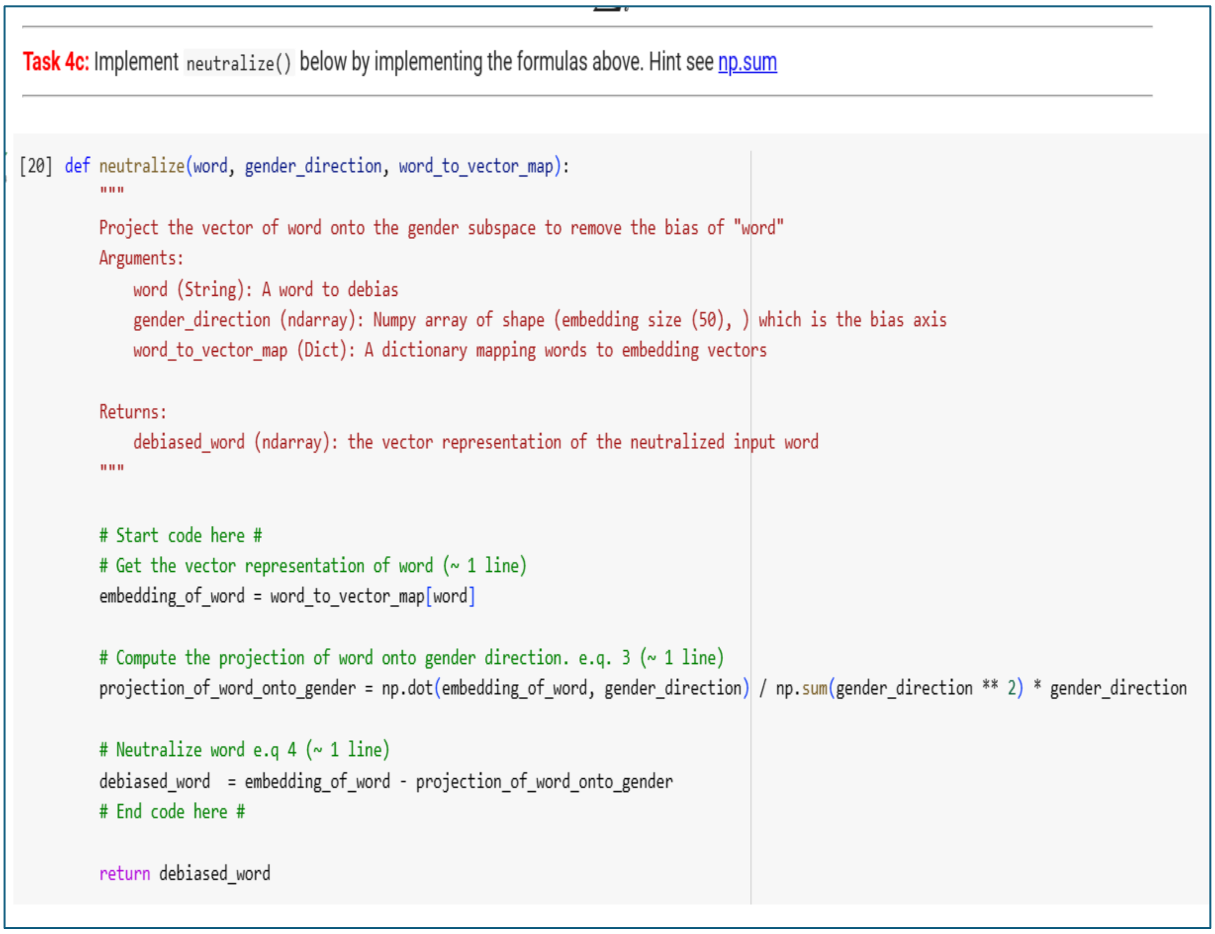
**Task 4b:** Quantify direct and indirect biases between words and the gender embedding by running the following cell. What is your observation?

```
[19] words = ["engineer", "science", "pilot", "technology", "lipstick", "arts", "singer", "computer", "receptionist", "fashion", "doctor", "literature"]
     for word in words:
         print(word, cosine_similarity(word_to_vector_map[word], gender_direction))
```

```
engineer -0.2626286258749398
science -0.1202780958734538
pilot -0.1319833052724868
technology -0.1801116607819377
lipstick 0.4179404715417419
arts -0.04513818522820779
singer 0.16162975755073875
computer -0.16390549337211754
receptionist 0.3305284235998437
fashion 0.06913524872078784
doctor 0.02885191966409418
literature -0.08972688088254833
```

Observation:

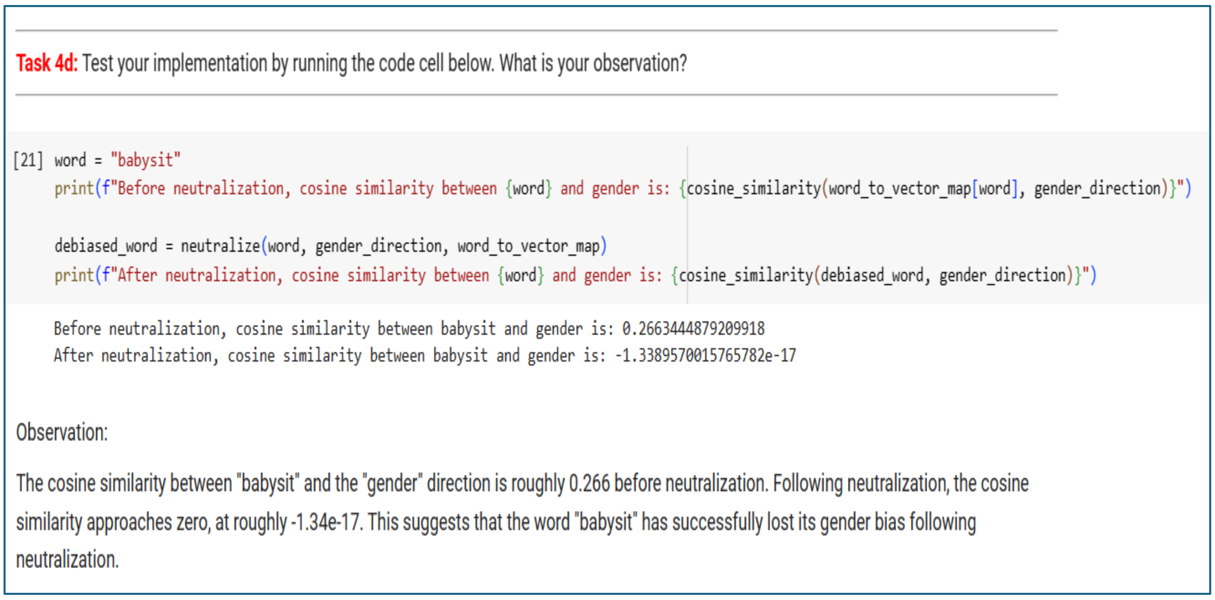
Words like "receptionist," "lipstick," and "fashion" exhibit positive connections as per gender direction of positive association, while "engineer," "science," and "pilot" show negative associations. Conversely, "doctor" and "literature" demonstrate less explicit gender connections.



**Task 4c:** Implement `neutralize()` below by implementing the formulas above. Hint see np.sum

```
[20] def neutralize(word, gender_direction, word_to_vector_map):
         """
         Project the vector of word onto the gender subspace to remove the bias of "word"
         Arguments:
             word (String): A word to debias
             gender_direction (ndarray): Numpy array of shape (embedding size (50), ) which is the bias axis
             word_to_vector_map (Dict): A dictionary mapping words to embedding vectors

         Returns:
             debiased_word (ndarray): the vector representation of the neutralized input word
         """

         # Start code here #
         # Get the vector representation of word (~ 1 line)
         embedding_of_word = word_to_vector_map[word]

         # Compute the projection of word onto gender direction. e.q. 3 (~ 1 line)
         projection_of_word_onto_gender = np.dot(embedding_of_word, gender_direction) / np.sum(gender_direction ** 2) * gender_direction

         # Neutralize word e.q 4 (~ 1 line)
         debiased_word  = embedding_of_word - projection_of_word_onto_gender
         # End code here #

         return debiased_word
```

Partha Sai - pua528



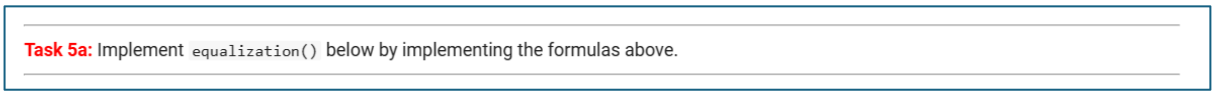
**Task 4d:** Test your implementation by running the code cell below. What is your observation?

```
[21] word = "babysit"
     print(f"Before neutralization, cosine similarity between {word} and gender is: {cosine_similarity(word_to_vector_map[word], gender_direction)}")

     debiased_word = neutralize(word, gender_direction, word_to_vector_map)
     print(f"After neutralization, cosine similarity between {word} and gender is: {cosine_similarity(debiased_word, gender_direction)}")
```

```
     Before neutralization, cosine similarity between babysit and gender is: 0.2663444879209918
     After neutralization, cosine similarity between babysit and gender is: -1.3389570015765782e-17
```

Observation:

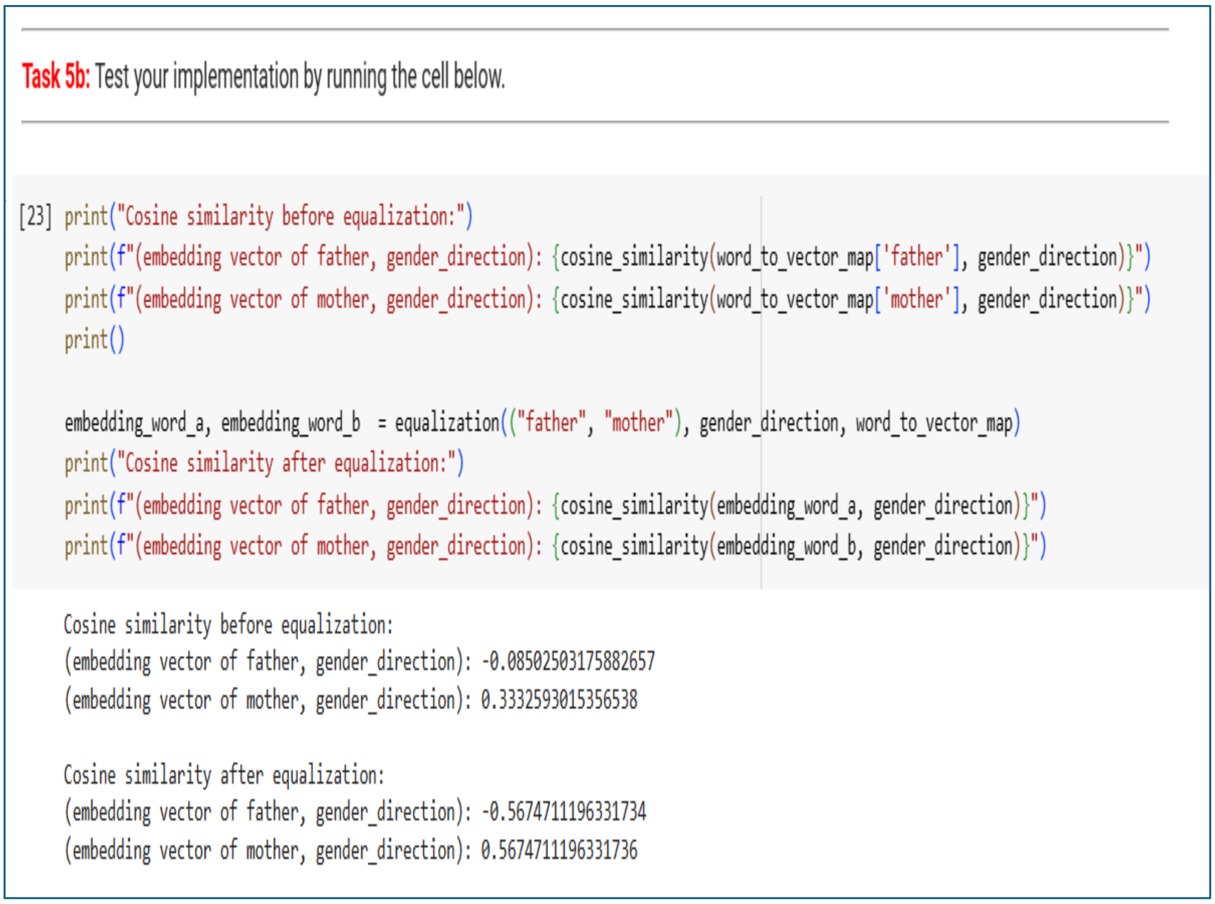
The cosine similarity between "babysit" and the "gender" direction is roughly 0.266 before neutralization. Following neutralization, the cosine similarity approaches zero, at roughly -1.34e-17. This suggests that the word "babysit" has successfully lost its gender bias following neutralization.



**Task 5a:** Implement `equalization()` below by implementing the formulas above.

```
def equalization(equality_set, bias_direction, word_to_vector_map):
    """
    Equalize the pair of gender specific words in the equality set
ensuring that
    any neutral word is equidistant to all words in the equality set.
    Arguments:
        equality_set (Tuple(String, String)): a tuple of strings of
gender specific
        words to debias e.g ("grandmother", "grandfather")
        bias_direction (ndarray): numpy array of shape (embedding
dimension,). The
        embedding vector representing the bias direction
        word_to_vector_map (Dict):  A dictionary mapping words to
embedding vectors
    Returns:
        embedding_word_a (ndarray): numpy array of shape (embedding
dimension,). The
        embedding vector representing the first word
        embedding_word_b (ndarray): numpy array of shape (embedding
dimension,). The
        embedding vector representing the second word
    """

    # Start code here #
    # Get the vector representation of word pair by unpacking
equality_set  (~ 3 line)
    word_a, word_b = equality_set
    embedding_word_a = word_to_vector_map[word_a]
    embedding_word_b = word_to_vector_map[word_b]
    # Compute the mean (eq. 5) of embedding_word_a and embedding_word_a
(~ 1 line)
    mean = (embedding_word_a + embedding_word_b) / 2

    # Compute the projection of mean representation onto the bias
direction (eq. 6) (~ 1 line)
    mean_B = np.dot(mean, bias_direction) /
np.linalg.norm(bias_direction)**2 * bias_direction

    # Compute the projection onto the orthogonal subspace (eq. 7) (~ 1
line)
    mean_othorgonal = mean - mean_B

    # Compute the projection of th embedding of word a onto the bias
direction (eq. 8) (~ 1 line)
    embedding_word_a_on_bias_direction = np.dot(embedding_word_a,
bias_direction) / np.linalg.norm(bias_direction)**2 * bias_direction

    # Compute the projection of th embedding of word b onto the bias
direction (eq. 9) (~ 1 line)
    embedding_word_b_on_bias_direction = np.dot(embedding_word_b,
bias_direction) / np.linalg.norm(bias_direction)**2 * bias_direction

    # Re-embed embedding of word a using eq. 10 (~ 1 long line)
    new_embedding_word_a_on_bias_direction = (np.abs(1 -
np.linalg.norm(mean_othorgonal)**2)**0.5 /
np.linalg.norm(embedding_word_a - mean_B - mean_othorgonal)**2) *
(embedding_word_a_on_bias_direction - mean_B)

    # Re-embed embedding of word b using eq. 11 (~ 1 long line)
    new_embedding_word_b_on_bias_direction = (np.abs(1 -
np.linalg.norm(mean_othorgonal)**2)**0.5 /
np.abs(np.linalg.norm(embedding_word_b - mean_B - mean_othorgonal))**2)
* (embedding_word_b_on_bias_direction - mean_B)

    # Equalize embedding of word a using eq. 12 (~ 1 line)
    embedding_word_a =  mean_othorgonal +
new_embedding_word_a_on_bias_direction

    # Equalize embedding of word b using eq. 13 (~ 1 line)
    embedding_word_b = mean_othorgonal +
new_embedding_word_b_on_bias_direction

    # End code here #

    return embedding_word_a, embedding_word_b
```

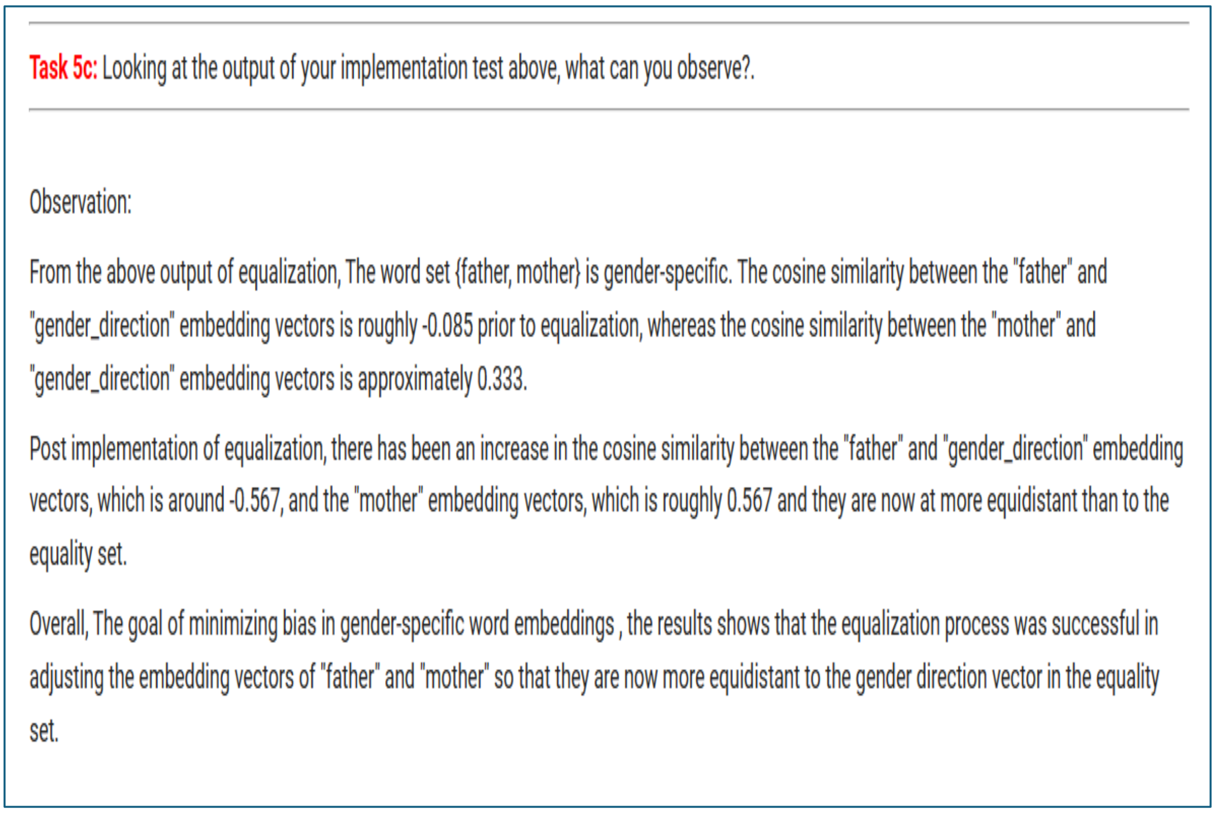
Partha Sai - pua528



**Task 5b:** Test your implementation by running the cell below.

```
[23] print("Cosine similarity before equalization:")
     print(f"(embedding vector of father, gender_direction): {cosine_similarity(word_to_vector_map['father'], gender_direction)}")
     print(f"(embedding vector of mother, gender_direction): {cosine_similarity(word_to_vector_map['mother'], gender_direction)}")
     print()

     embedding_word_a, embedding_word_b  = equalization(("father", "mother"), gender_direction, word_to_vector_map)
     print("Cosine similarity after equalization:")
     print(f"(embedding vector of father, gender_direction): {cosine_similarity(embedding_word_a, gender_direction)}")
     print(f"(embedding vector of mother, gender_direction): {cosine_similarity(embedding_word_b, gender_direction)}")
```

```
Cosine similarity before equalization:
(embedding vector of father, gender_direction): -0.08502503175882657
(embedding vector of mother, gender_direction): 0.3332593015356538

Cosine similarity after equalization:
(embedding vector of father, gender_direction): -0.5674711196331734
(embedding vector of mother, gender_direction): 0.5674711196331736
```



**Task 5c:** Looking at the output of your implementation test above, what can you observe?.

Observation:

From the above output of equalization, The word set {father, mother} is gender-specific. The cosine similarity between the "father" and "gender_direction" embedding vectors is roughly -0.085 prior to equalization, whereas the cosine similarity between the "mother" and "gender_direction" embedding vectors is approximately 0.333.

Post implementation of equalization, there has been an increase in the cosine similarity between the "father" and "gender_direction" embedding vectors, which is around -0.567, and the "mother" embedding vectors, which is roughly 0.567 and they are now at more equidistant than to the equality set.

Overall, The goal of minimizing bias in gender-specific word embeddings , the results shows that the equalization process was successful in adjusting the embedding vectors of "father" and "mother" so that they are now more equidistant to the gender direction vector in the equality set.