x_i	μ_1	μ_2	$distance_1$	$distance_2$	Nearest Cluster
(1, 2)	(1,2)	(−1, 5)	0	5	μ_1
(2, 2)	(1,2)	(−1, 5)	1	6	μ_1
(2, 1)	(1,2)	(−1, 5)	2	7	μ_1
(−1, 5)	(1,2)	(−1, 5)	5	0	μ_2
(−2, −1)	(1,2)	(−1, 5)	6	7	μ_1
(−1, −1)	(1,2)	(−1, 5)	4	6	μ_1

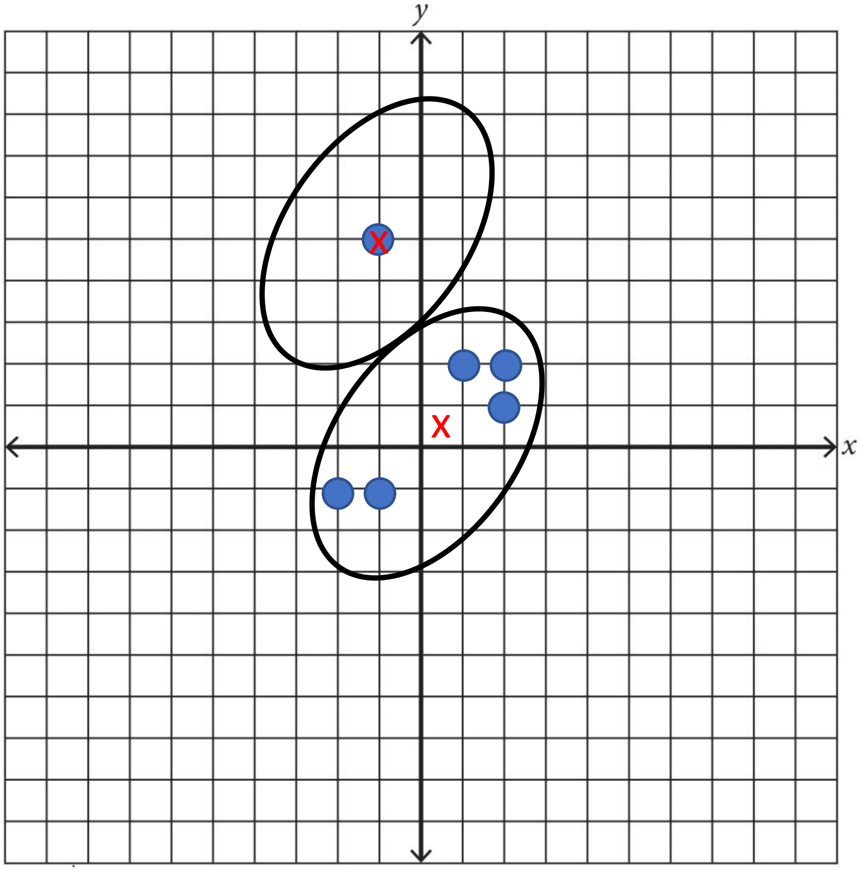
$\mu_1 = (2/5,\ 3/5)$

$\mu_2 = (-1{,}5)$

x_i	μ_1	μ_2	$distance_1$	$distance_2$	Nearest Cluster
(1, 2)	$(2/5,\ 3/5)$	(−1, 5)	2	5	μ_1
(2, 2)	$(2/5,\ 3/5)$	(−1, 5)	3	6	μ_1
(2, 1)	$(2/5,\ 3/5)$	(−1, 5)	2	7	μ_1
(−1, 5)	$(2/5,\ 3/5)$	(−1, 5)	6.8	0	μ_2
(−2, −1)	$(2/5,\ 3/5)$	(−1, 5)	5	7	μ_1
(−1, −1)	$(2/5,\ 3/5)$	(−1, 5)	4	6	μ_1

$\mu_1 = (2/5,\ 3/5)$

$\mu_2 = (-1{,}5)$



x_i	μ_1	μ_2	$distance_1$	$distance_2$	Nearest Cluster
(1, 2)	(1,2)	(−2, -1)	0	6	μ_1
(2, 2)	(1,2)	(−2, -1)	1	7	μ_1
(2, 1)	(1,2)	(−2, -1)	2	5	μ_1
(−1, 5)	(1,2)	(−2, -1)	5	7	μ_1
(−2, −1)	(1,2)	(−2, -1)	6	0	μ_2
(−1, −1)	(1,2)	(−2, -1)	4	1	μ_2

$\mu_1 = (1,\ 5/2)$

$\mu_2 = (-3/2, -1)$

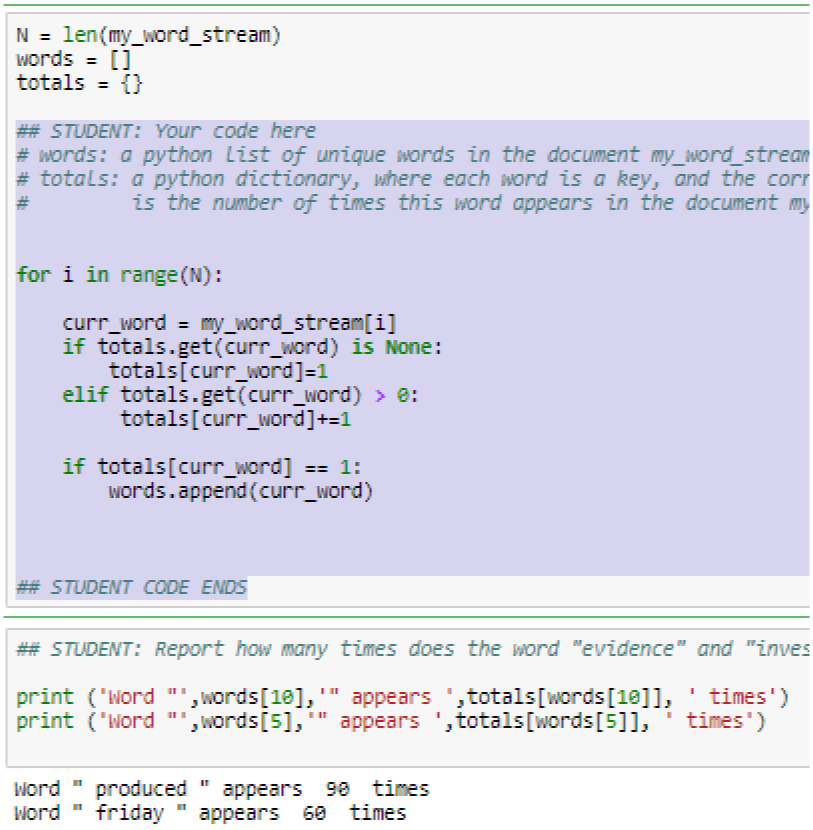
x_i	μ_1	μ_2	$distance_1$	$distance_2$	Nearest Cluster
(1, 2)	$(1, 5/2)$	$(-3/2, -1)$	1/2	11/2	μ_1
(2, 2)	$(1, 5/2)$	$(-3/2, -1)$	3/2	13/2	μ_1
(2, 1)	$(1, 5/2)$	$(-3/2, -1)$	5/2	11/2	μ_1
(−1, 5)	$(1, 5/2)$	$(-3/2, -1)$	9/2	13/2	μ_1
(−2, −1)	$(1, 5/2)$	$(-3/2, -1)$	13/2	1/2	μ_2
(−1, −1)	$(1, 5/2)$	$(-3/2, -1)$	11/2	1/2	μ_2

$\mu_1 = (1,\ 5/2)$

$\mu_2 = (-3/2, -1)$

P1.

```
## STUDENT: Your code here

# words: a python list of unique words in the document my_word_stream as the vocabulary

# totals: a python dictionary, where each word is a key, and the corresponding value

#       is the number of times this word appears in the document my_word_stream


for i in range(N):


  curr_word = my_word_stream[i]

  if totals.get(curr_word) is None:

    totals[curr_word]=1

  elif totals.get(curr_word) > 0:

     totals[curr_word]+=1


  if totals[curr_word] == 1:

    words.append(curr_word)


## STUDENT CODE ENDS
```



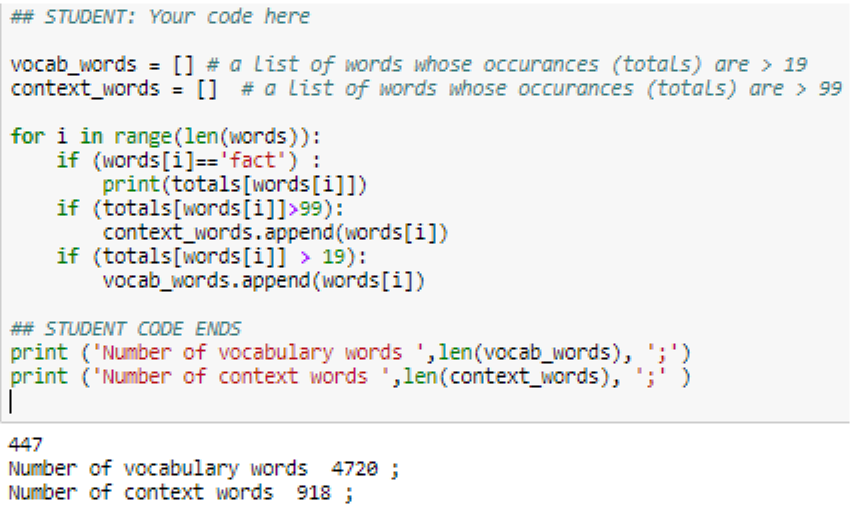
```
N = len(my_word_stream)
words = []
totals = {}

## STUDENT: Your code here
# words: a python list of unique words in the document my_word_stream
# totals: a python dictionary, where each word is a key, and the corr
#          is the number of times this word appears in the document my


for i in range(N):

    curr_word = my_word_stream[i]
    if totals.get(curr_word) is None:
        totals[curr_word]=1
    elif totals.get(curr_word) > 0:
         totals[curr_word]+=1

    if totals[curr_word] == 1:
        words.append(curr_word)



## STUDENT CODE ENDS
```

```
## STUDENT: Report how many times does the word "evidence" and "inves

print ('Word "',words[10],'" appears ',totals[words[10]], ' times')
print ('Word "',words[5],'" appears ',totals[words[5]], ' times')
```

```
Word " produced " appears  90  times
Word " friday " appears  60  times
```

P2.

```
## STUDENT: Your code here


vocab_words = [] # a list of words whose occurances (totals) are > 19

context_words = []  # a list of words whose occurances (totals) are > 99


for i in range(len(words)):

  if (words[i]=='fact') :

    print(totals[words[i]])

  if (totals[words[i]]>99):
        context_words.append(words[i])

    if (totals[words[i]] > 19):

        vocab_words.append(words[i])


## STUDENT CODE ENDS
```

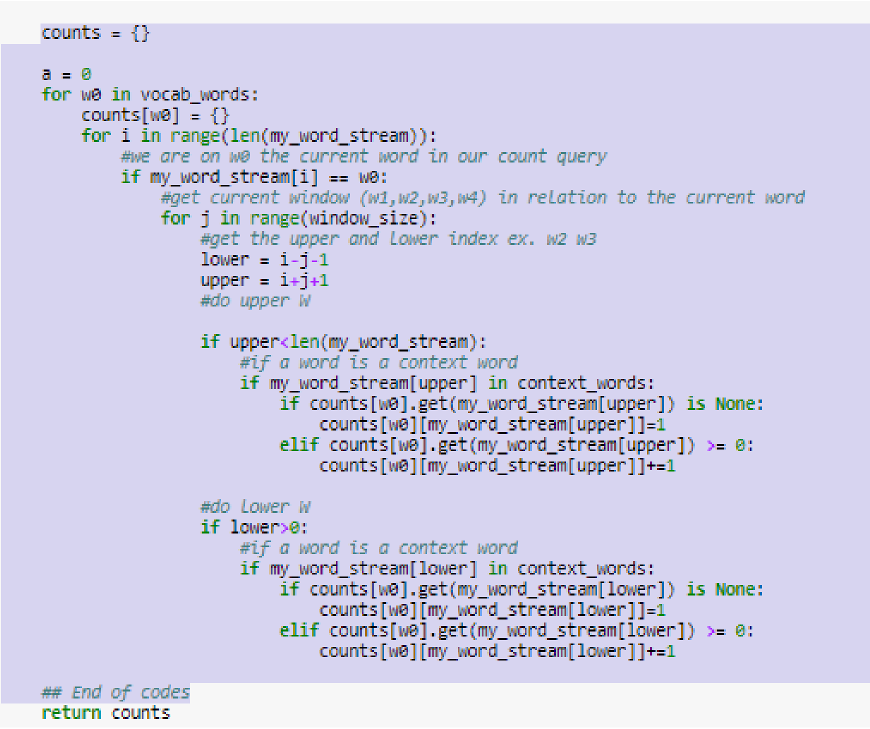


```
## STUDENT: Your code here

vocab_words = [] # a list of words whose occurances (totals) are > 19
context_words = []  # a list of words whose occurances (totals) are > 99

for i in range(len(words)):
    if (words[i]=='fact') :
        print(totals[words[i]])
    if (totals[words[i]]>99):
        context_words.append(words[i])
    if (totals[words[i]] > 19):
        vocab_words.append(words[i])

## STUDENT CODE ENDS
print ('Number of vocabulary words ',len(vocab_words), ';')
print ('Number of context words ',len(context_words), ';' )
```

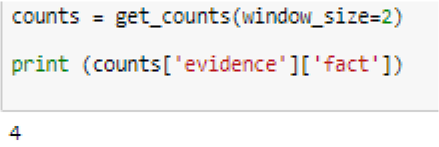
```
447
Number of vocabulary words  4720 ;
Number of context words  918 ;
```

P3.

```
counts = {}


    a = 0

    for w0 in vocab_words:

        counts[w0] = {}

        for i in range(len(my_word_stream)):

            #we are on w0 the current word in our count query

            if my_word_stream[i] == w0:

                #get current window (w1,w2,w3,w4) in relation to the current word

                for j in range(window_size):

                    #get the upper and lower index ex. w2 w3

                    lower = i-j-1

                    upper = i+j+1

                    #do upper W


                    if upper<len(my_word_stream):
```



```
counts = {}

a = 0
for w0 in vocab_words:
    counts[w0] = {}
    for i in range(len(my_word_stream)):
        #we are on w0 the current word in our count query
        if my_word_stream[i] == w0:
            #get current window (w1,w2,w3,w4) in relation to the current word
            for j in range(window_size):
                #get the upper and lower index ex. w2 w3
                lower = i-j-1
                upper = i+j+1
                #do upper W

                if upper<len(my_word_stream):
                    #if a word is a context word
                    if my_word_stream[upper] in context_words:
                        if counts[w0].get(my_word_stream[upper]) is None:
                            counts[w0][my_word_stream[upper]]=1
                        elif counts[w0].get(my_word_stream[upper]) >= 0:
                            counts[w0][my_word_stream[upper]]+=1

                #do lower W
                if lower>0:
                    #if a word is a context word
                    if my_word_stream[lower] in context_words:
                        if counts[w0].get(my_word_stream[lower]) is None:
                            counts[w0][my_word_stream[lower]]=1
                        elif counts[w0].get(my_word_stream[lower]) >= 0:
                            counts[w0][my_word_stream[lower]]+=1

## End of codes
return counts
```



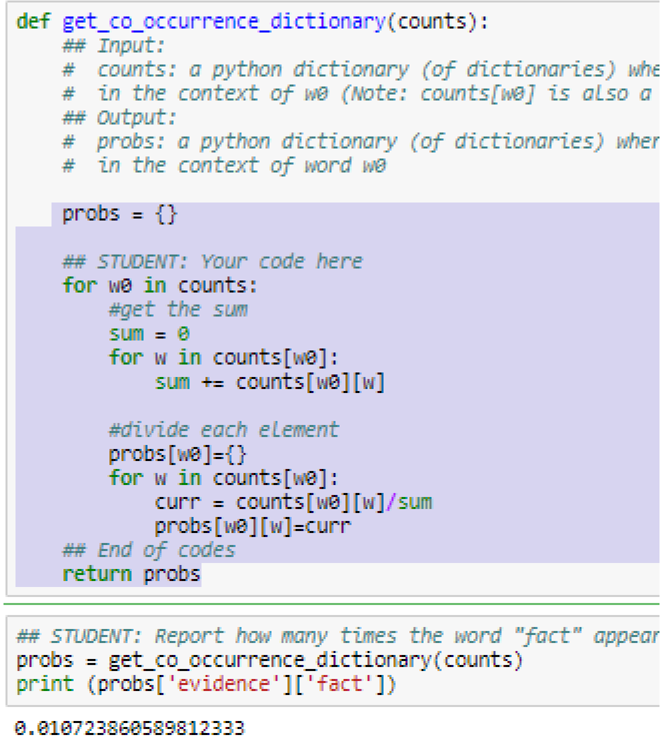
```
counts = get_counts(window_size=2)

print (counts['evidence']['fact'])
```

```
4
```

```
            #if a word is a context word
            if my_word_stream[upper] in context_words:
              if counts[w0].get(my_word_stream[upper]) is None:
                counts[w0][my_word_stream[upper]]=1
              elif counts[w0].get(my_word_stream[upper]) >= 0:
                counts[w0][my_word_stream[upper]]+=1

          #do lower W
          if lower>0:
            #if a word is a context word
            if my_word_stream[lower] in context_words:
              if counts[w0].get(my_word_stream[lower]) is None:
                counts[w0][my_word_stream[lower]]=1
              elif counts[w0].get(my_word_stream[lower]) >= 0:
                counts[w0][my_word_stream[lower]]+=1

  ## End of codes
```

P4.

```
probs = {}

  ## STUDENT: Your code here
  for w0 in counts:
    #get the sum
    sum = 0
    for w in counts[w0]:
      sum += counts[w0][w]

    #divide each element
```



```
def get_co_occurrence_dictionary(counts):
    ## Input:
    #  counts: a python dictionary (of dictionaries) whe
    #  in the context of w0 (Note: counts[w0] is also a
    ## Output:
    #  probs: a python dictionary (of dictionaries) wher
    #  in the context of word w0

    probs = {}

    ## STUDENT: Your code here
    for w0 in counts:
        #get the sum
        sum = 0
        for w in counts[w0]:
            sum += counts[w0][w]

        #divide each element
        probs[w0]={}
        for w in counts[w0]:
            curr = counts[w0][w]/sum
            probs[w0][w]=curr
    ## End of codes
    return probs
```

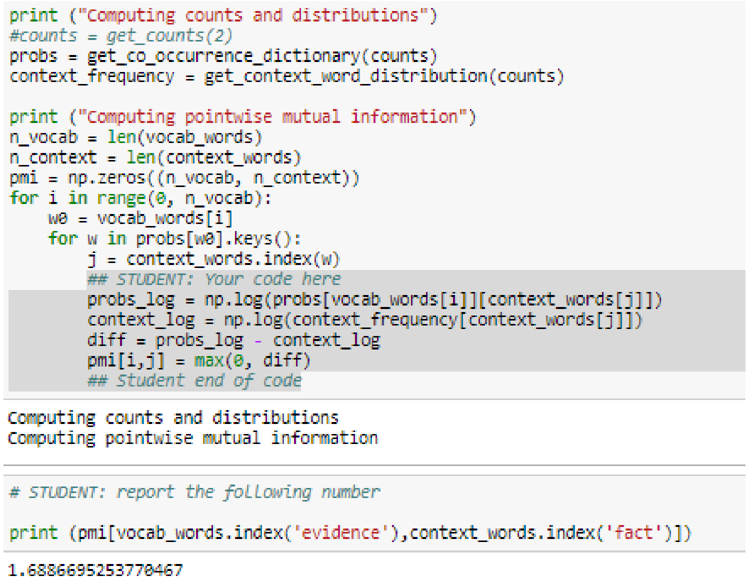
```
## STUDENT: Report how many times the word "fact" appear
probs = get_co_occurrence_dictionary(counts)
print (probs['evidence']['fact'])
```

```
0.010723860589812333
```

```
        probs[w0]={}
        for w in counts[w0]:
            curr = counts[w0][w]/sum
            probs[w0][w]=curr
    ## End of codes
    return probs
```

P5.

```
## STUDENT: Your code here
        probs_log = np.log(probs[vocab_words[i]][context_words[j]])
        context_log = np.log(context_frequency[context_words[j]])
        diff = probs_log - context_log
        pmi[i,j] = max(0, diff)
        ## Student end of code
```



```
print ("Computing counts and distributions")
#counts = get_counts(2)
probs = get_co_occurrence_dictionary(counts)
context_frequency = get_context_word_distribution(counts)

print ("Computing pointwise mutual information")
n_vocab = len(vocab_words)
n_context = len(context_words)
pmi = np.zeros((n_vocab, n_context))
for i in range(0, n_vocab):
    w0 = vocab_words[i]
    for w in probs[w0].keys():
        j = context_words.index(w)
        ## STUDENT: Your code here
        probs_log = np.log(probs[vocab_words[i]][context_words[j]])
        context_log = np.log(context_frequency[context_words[j]])
        diff = probs_log - context_log
        pmi[i,j] = max(0, diff)
        ## Student end of code
```

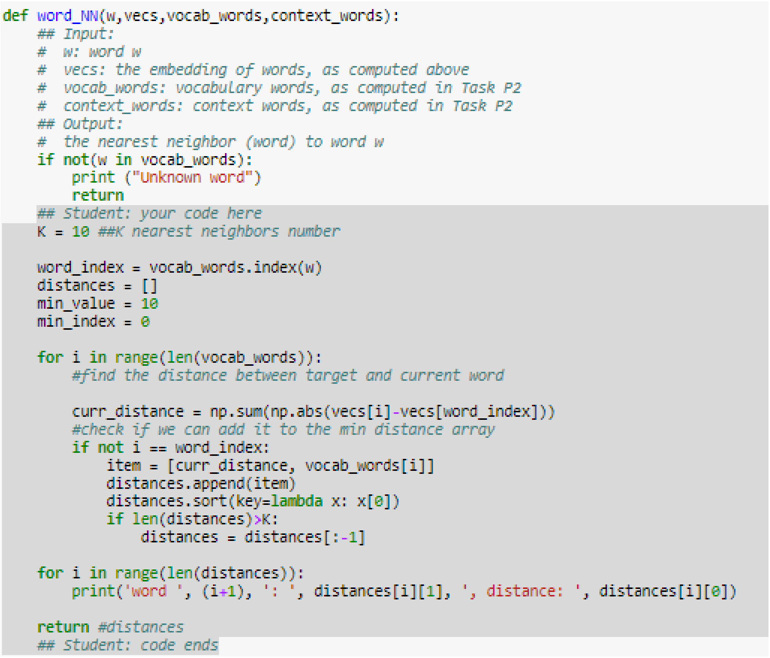
```
Computing counts and distributions
Computing pointwise mutual information
```

```
# STUDENT: report the following number

print (pmi[vocab_words.index('evidence'),context_words.index('fact')])
```

```
1.6886695253770467
```

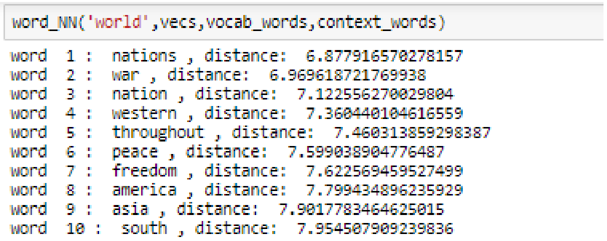
P6. ## Student: your code here

```
    K = 10 ##K nearest neighbors number


    word_index = vocab_words.index(w)
    distances = []
    min_value = 10
    min_index = 0


    for i in range(len(vocab_words)):
        #find the distance between target and current word


        curr_distance = np.sum(np.abs(vecs[i]-vecs[word_index]))
```

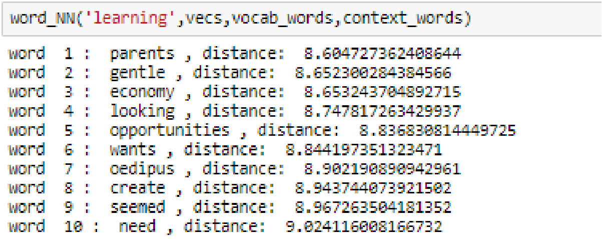


```
def word_NN(w,vecs,vocab_words,context_words):
    ## Input:
    #  w: word w
    #  vecs: the embedding of words, as computed above
    #  vocab_words: vocabulary words, as computed in Task P2
    #  context_words: context words, as computed in Task P2
    ## Output:
    #  the nearest neighbor (word) to word w
    if not(w in vocab_words):
        print ("Unknown word")
        return
    ## Student: your code here
    K = 10 ##K nearest neighbors number

    word_index = vocab_words.index(w)
    distances = []
    min_value = 10
    min_index = 0

    for i in range(len(vocab_words)):
        #find the distance between target and current word

        curr_distance = np.sum(np.abs(vecs[i]-vecs[word_index]))
        #check if we can add it to the min distance array
        if not i == word_index:
            item = [curr_distance, vocab_words[i]]
            distances.append(item)
            distances.sort(key=lambda x: x[0])
            if len(distances)>K:
                distances = distances[:-1]

    for i in range(len(distances)):
        print('word ', (i+1), ': ', distances[i][1], ', distance: ', distances[i][0])

    return #distances
    ## Student: code ends
```

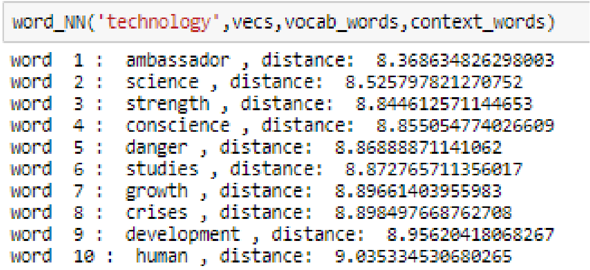
```
        #check if we can add it to the min distance array

        if not i == word_index:

            item = [curr_distance, vocab_words[i]]

            distances.append(item)

            distances.sort(key=lambda x: x[0])

            if len(distances)>K:

                distances = distances[:-1]


    for i in range(len(distances)):

        print('word ', (i+1), ': ', distances[i][1], ', distance: ',

distances[i][0])


    return #distances

    ## Student: code ends
```



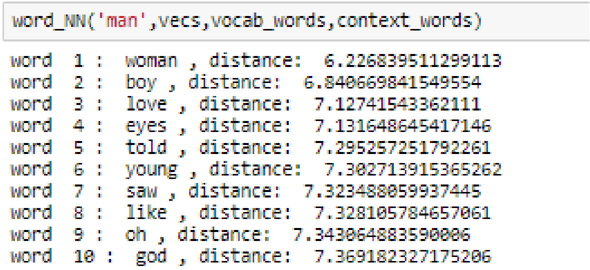
```
word_NN('world',vecs,vocab_words,context_words)

word  1 :  nations , distance:  6.877916570278157
word  2 :  war , distance:  6.969618721769938
word  3 :  nation , distance:  7.122556270029804
word  4 :  western , distance:  7.360440104616559
word  5 :  throughout , distance:  7.460313859298387
word  6 :  peace , distance:  7.599038904776487
word  7 :  freedom , distance:  7.622569459527499
word  8 :  america , distance:  7.799434896235929
word  9 :  asia , distance:  7.9017783464625015
word  10 :  south , distance:  7.954507909239836
```



```
word_NN('learning',vecs,vocab_words,context_words)

word  1 :  parents , distance:  8.604727362408644
word  2 :  gentle , distance:  8.652300284384566
word  3 :  economy , distance:  8.653243704892715
word  4 :  looking , distance:  8.747817263429937
word  5 :  opportunities , distance:  8.836830814449725
word  6 :  wants , distance:  8.844197351323471
word  7 :  oedipus , distance:  8.902190890942961
word  8 :  create , distance:  8.943744073921502
word  9 :  seemed , distance:  8.967263504181352
word  10 :  need , distance:  9.024116008166732
```



```
word_NN('technology',vecs,vocab_words,context_words)

word  1 :  ambassador , distance:  8.368634826298003
word  2 :  science , distance:  8.525797821270752
word  3 :  strength , distance:  8.844612571144653
word  4 :  conscience , distance:  8.855054774026609
word  5 :  danger , distance:  8.86888871141062
word  6 :  studies , distance:  8.872765711356017
word  7 :  growth , distance:  8.89661403955983
word  8 :  crises , distance:  8.898497668762708
word  9 :  development , distance:  8.95620418068267
word  10 :  human , distance:  9.035334530680265
```



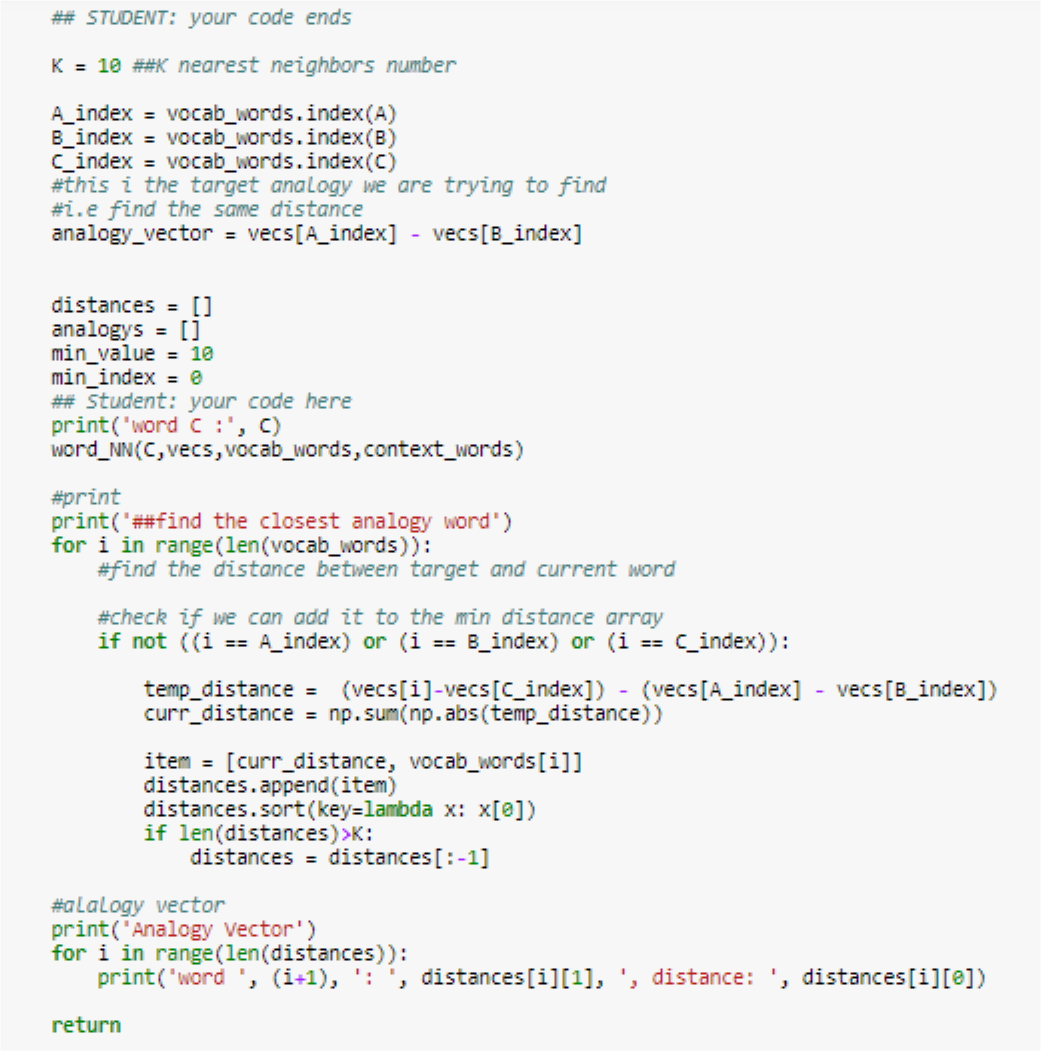
```
word_NN('man',vecs,vocab_words,context_words)

word  1 :  woman , distance:  6.226839511299113
word  2 :  boy , distance:  6.840669841549554
word  3 :  love , distance:  7.12741543362111
word  4 :  eyes , distance:  7.131648645417146
word  5 :  told , distance:  7.295257251792261
word  6 :  young , distance:  7.302713915365262
word  7 :  saw , distance:  7.323488059937445
word  8 :  like , distance:  7.328105784657061
word  9 :  oh , distance:  7.343064883590006
word  10 :  god , distance:  7.369182327175206
```

P7.

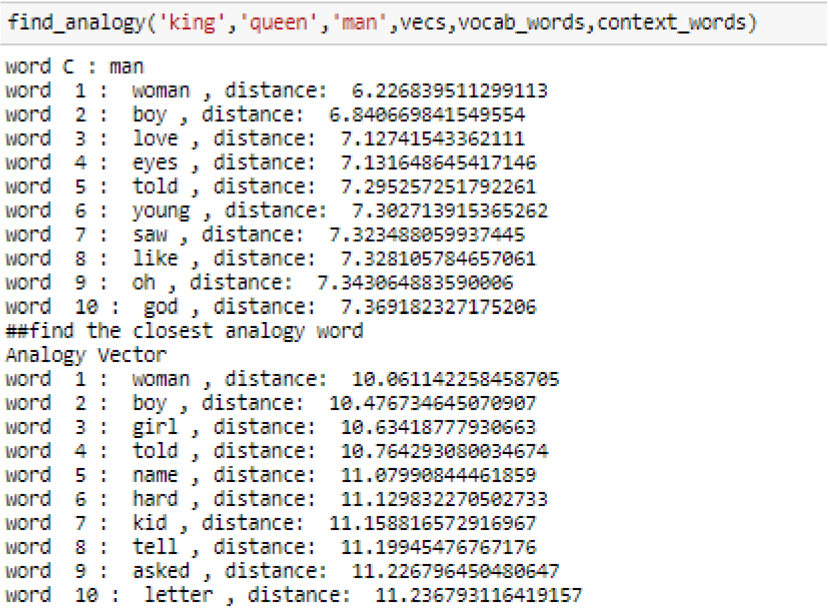
```
def find_analogy(A,B,C,vecs,vocab_words,context_words):

    ## Input:

    #  A, B, C: words A, B, C

    #  vecs: the embedding of words, as computed above

    #  vocab_words: vocabulary words, as computed in Task P2

    #  context_words: context words, as computed in Task P2

    ## Output:
```

```
#  the word that solves the analogy problem
## STUDENT: Your code here


## STUDENT: your code ends


K = 10 ##K nearest neighbors number


A_index = vocab_words.index(A)
B_index = vocab_words.index(B)
C_index = vocab_words.index(C)
#this i the target analogy we are trying to find
#i.e find the same distance
analogy_vector = vecs[A_index] - vecs[B_index]



distances = []
analogys = []
min_value = 10
min_index = 0
## Student: your code here
print('word C :', C)
word_NN(C,vecs,vocab_words,context_words)


#print
print('##find the closest analogy word')
for i in range(len(vocab_words)):
  #find the distance between target and current word


  #check if we can add it to the min distance array
```

```
    if not ((i == A_index) or (i == B_index) or (i == C_index)):

        temp_distance =  (vecs[i]-vecs[C_index]) - (vecs[A_index] - vecs[B_index])
        curr_distance = np.sum(np.abs(temp_distance))

        item = [curr_distance, vocab_words[i]]
        distances.append(item)
        distances.sort(key=lambda x: x[0])
        if len(distances)>K:
            distances = distances[:-1]

#alalogy vector
print('Analogy Vector')
for i in range(len(distances)):
    print('word ', (i+1), ': ', distances[i][1], ', distance: ', distances[i][0])

return
```

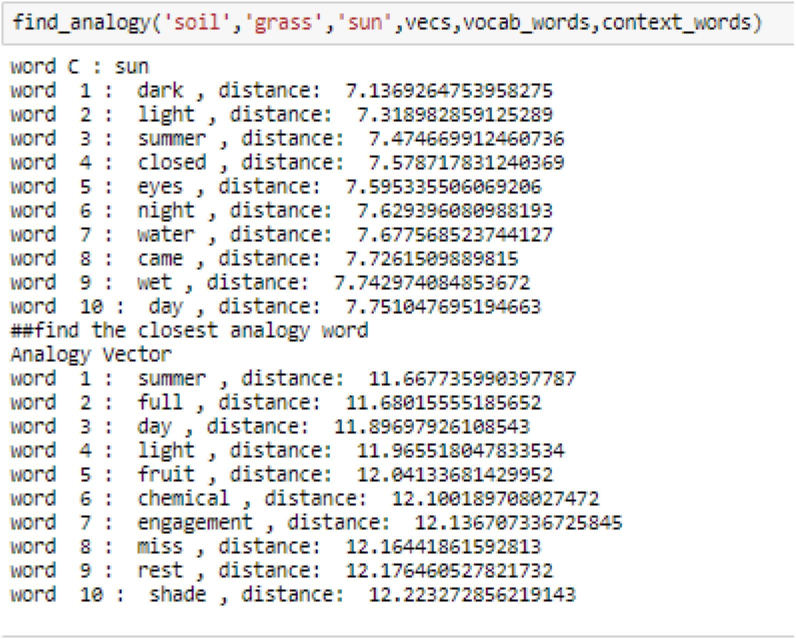


```
    ## STUDENT: your code ends

    K = 10 ##K nearest neighbors number

    A_index = vocab_words.index(A)
    B_index = vocab_words.index(B)
    C_index = vocab_words.index(C)
    #this i the target analogy we are trying to find
    #i.e find the same distance
    analogy_vector = vecs[A_index] - vecs[B_index]


    distances = []
    analogys = []
    min_value = 10
    min_index = 0
    ## Student: your code here
    print('word C :', C)
    word_NN(C,vecs,vocab_words,context_words)

    #print
    print('##find the closest analogy word')
    for i in range(len(vocab_words)):
        #find the distance between target and current word

        #check if we can add it to the min distance array
        if not ((i == A_index) or (i == B_index) or (i == C_index)):

            temp_distance =  (vecs[i]-vecs[C_index]) - (vecs[A_index] - vecs[B_index])
            curr_distance = np.sum(np.abs(temp_distance))

            item = [curr_distance, vocab_words[i]]
            distances.append(item)
            distances.sort(key=lambda x: x[0])
            if len(distances)>K:
                distances = distances[:-1]

    #alalogy vector
    print('Analogy Vector')
    for i in range(len(distances)):
        print('word ', (i+1), ': ', distances[i][1], ', distance: ', distances[i][0])

    return
```



```
find_analogy('king','queen','man',vecs,vocab_words,context_words)
```

```
word C : man
word  1 :  woman , distance:  6.226839511299113
word  2 :  boy , distance:  6.840669841549554
word  3 :  love , distance:  7.12741543362111
word  4 :  eyes , distance:  7.131648645417146
word  5 :  told , distance:  7.295257251792261
word  6 :  young , distance:  7.302713915365262
word  7 :  saw , distance:  7.323488059937445
word  8 :  like , distance:  7.328105784657061
word  9 :  oh , distance:  7.343064883590006
word  10 :  god , distance:  7.369182327175206
##find the closest analogy word
Analogy Vector
word  1 :  woman , distance:  10.061142258458705
word  2 :  boy , distance:  10.476734645070907
word  3 :  girl , distance:  10.63418777930663
word  4 :  told , distance:  10.764293080034674
word  5 :  name , distance:  11.07990844461859
word  6 :  hard , distance:  11.129832270502733
word  7 :  kid , distance:  11.158816572916967
word  8 :  tell , distance:  11.19945476767176
word  9 :  asked , distance:  11.226796450480647
word  10 :  letter , distance:  11.236793116419157
```



```
find_analogy('soil','grass','sun',vecs,vocab_words,context_words)
```

```
word C : sun
word  1 :  dark , distance:  7.1369264753958275
word  2 :  light , distance:  7.318982859125289
word  3 :  summer , distance:  7.474669912460736
word  4 :  closed , distance:  7.578717831240369
word  5 :  eyes , distance:  7.595335506069206
word  6 :  night , distance:  7.629396080988193
word  7 :  water , distance:  7.677568523744127
word  8 :  came , distance:  7.7261509889815
word  9 :  wet , distance:  7.742974084853672
word  10 :  day , distance:  7.751047695194663
##find the closest analogy word
Analogy Vector
word  1 :  summer , distance:  11.667735990397787
word  2 :  full , distance:  11.68015555185652
word  3 :  day , distance:  11.89697926108543
word  4 :  light , distance:  11.965518047833534
word  5 :  fruit , distance:  12.04133681429952
word  6 :  chemical , distance:  12.100189708027472
word  7 :  engagement , distance:  12.136707336725845
word  8 :  miss , distance:  12.16441861592813
word  9 :  rest , distance:  12.176460527821732
word  10 :  shade , distance:  12.223272856219143
```