Homework Assignment 2: Numpy Array and DataFrame

1. Define a function to analyze a numpy array

- Assume we have an array X which contains term frequency of each document. In this array, each row presents a document, each column denotes a word, and each value, say $x_{i,j}$, denotes the frequency of the word j in document i. Therefore, if there are m documents, n words, X has a shape of (m, n).

  Define a function named `analyze_tf` which:

  - take X as an input.
  - calculate the document frequency df_j for word j, e.g. how many documents contain word j. Save the result to array df (df has shape of $(n,)$).
  - divides word frequency $x_{i,j}$ by the total number of words in document i. Save the result as an array named tf (tf has shape of (m, n)).
  - for each $x_{i,j}$, calculates $tf\_idf_{i,j} = \frac{tf_{i,j}}{df_j}$. The reason is, if a word appears in most documents, it does not have the discriminative power and often is called a `stop` word. The inverse of df can downgrade the weight of such words. tf_idf has shape of (m, n)
  - Now, please print the following:
    - print the index of the longest document
    - print the indexes of words with the top 3 largest df values
    - for the longest document, print the indexes of words with top 3 largest values in the tf_idf array.
  - return the tf_idf array.
- Note, for all the steps, **do not use any loop**. Just use array functions and broadcasting for high performance computation.

In [3]:

```
import numpy as np
import pandas as pd
```

In [50]:

```
def analyze_tf(X):

    tf_idf = None

    # Add your code here

    #print index of the longest document

    #print indexes of words with the top 3 largest df values

    #return index of top_3 words in the longest document

    return tf_idf
```

In [51]:

```
# dtm.csv is a csv file for test.
# It contains word counts in a few documents

dtm = pd.read_csv("dtm.csv")
print(dtm.head())
analyze_tf(dtm.values)
```

```
   texas  freeze  leaves  millions  north  mexico  without  power  frozen  \
0      1     1.0     1.0       1.0    1.0     1.0      1.0      1     0.0   
1      1     0.0     0.0       0.0    0.0     0.0      0.0      1     1.0   
2      1     0.0     0.0       0.0    0.0     0.0      1.0      1     0.0   

   wind  turbines  contribute  rolling  blackouts  across  2.7  million  \
0   0.0       0.0         0.0      0.0        0.0     0.0  0.0      0.0   
1   1.0       1.0         1.0      1.0        1.0     1.0  0.0      0.0   
2   0.0       0.0         0.0      0.0        0.0     0.0  1.0      1.0   

   people  winter  storm  
0     0.0     0.0    0.0  
1     0.0     0.0    0.0  
2     1.0     1.0    1.0  
Indexes of the longest documents: 1
Indexes of words with the top 3 largest df values: [0 7 6]
Indexes of words with top 3 largest tf_idf values in the longest document: [ 9 11  8]
```

Out[51]:

```
array([[0.04166667, 0.125     , 0.125     , 0.125     , 0.125     ,
        0.125     , 0.0625    , 0.04166667, 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ],
       [0.03703704, 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.03703704, 0.11111111, 0.11111111,
        0.11111111, 0.11111111, 0.11111111, 0.11111111, 0.11111111,
        0.        , 0.        , 0.        , 0.        , 0.        ],
       [0.04166667, 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.0625    , 0.04166667, 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.125     , 0.125     , 0.125     , 0.125     , 0.125     ]])
```

2. Define a function to analyze car dataset using pandas

- Define a function named `emotion_analysis` to do the follows:
  - Read "emotion.csv" as a dataframe with the first row in the csv file as column names
  - Count the number of samples labeled for each emotion (i.e. each value in the column "emotion). Print the counts.
  - Create a new column called `length` to store the number of words in the text column. (hint: "apply" function to split the text by space and then count elements in the resulting list)

- Show the min, max, and mean values of sadness, happiness, and text length for each emotion. Print the results
- Create a cross tabulation to show the average anxiety score of each emotion and each worry value. Use "emotion" as row index and "worry" as column index. Print the table.
    - This function does not have any return. Just print out the result of each calculation step.

In [52]:

```python
def emotion_analysis():

    # add your code
```

In [20]:

```python
emotion_analysis()
```

```
===The number of samples labeled for each emotion===
Anxiety       1381
Sadness        357
Relaxation     333
Fear           230
Anger          107
Happiness       39
Desire          27
Disgust         17
Name: emotion, dtype: int64


=== min, max, and mean values of sadness, happiness, and text length for each emotion===
             sadness           happiness                length          
                mean amin amax      mean amin amax        mean amin  amax
emotion                                                                  
Anger       5.672897    1    9  3.177570    1    8  122.084112   88   374
Anxiety     5.719768    1    9  3.333816    1    9  118.066618   59   541
Desire      4.148148    1    8  4.925926    2    8  150.259259   89  1018
Disgust     4.764706    1    8  3.764706    1    6  108.411765   58   158
Fear        6.565217    1    9  3.056522    1    9  118.852174   80   322
Happiness   2.666667    1    9  7.230769    4    9  122.461538   92   272
Relaxation  2.858859    1    9  5.369369    1    9  119.696697    1   292
Sadness     7.436975    2    9  3.112045    1    9  122.117647   85   544


=== Cross tabulation of anxiety score by emotion and worry ===
worry          1         2         3         4         5         6         7  \
emotion                                                                         
Anger       5.50  1.250000  3.416667  4.222222  5.125000  5.833333  6.040000   
Anxiety     3.00  7.000000  5.222222  5.941176  6.149425  6.938931  7.597166   
Desire       NaN  1.000000  3.142857  3.250000  3.666667  6.375000  6.000000   
Disgust      NaN  5.000000       NaN  4.500000  5.000000  4.800000  5.750000   
Fear         NaN       NaN  6.000000  5.000000  5.714286  5.454545  6.918033   
Happiness   1.00  3.000000  2.400000  3.000000  1.000000  2.625000  4.000000   
Relaxation  1.25  1.777778  2.301887  3.029851  3.090909  4.137931  4.219512   
Sadness     2.50  3.857143  3.368421  4.606061  4.882353  5.325301  6.156863   

worry              8         9  
emotion                         
Anger       7.000000  7.437500  
Anxiety     8.220126  8.770186  
Desire           NaN       NaN  
Disgust          NaN       NaN  
Fear        7.643836  8.256757  
Happiness   2.000000  6.500000  
Relaxation  3.562500  4.200000  
Sadness     6.565217  7.258065  
```

3 (Bonus). Calculate Word Cooccurrences

A word cooccurrence is defined as two words, say w_1, w_2, both appear in a document (but they are not necessarily next to each other). If two words frequently appear together in a list of documents, for example, "sport" and "game", these words can become topic words.

Define a function, `find_coocur` , as follows:

- take an array input similar to that of Q1, e.g. X of shape (m, n)
- calculate coocurrences between any pair of words, w_1, w_2 in the m documents, where $w_1 \neq w_2$. Save the cooccrrences as an array of shape (n, n)
- return the cooccurrence array

Again, **do not use any loop**. Just use array functions and broadcasting for high performance computation.

In [37]:

```python
def find_coocur(x):
    c = None

    # add your code

    return c
```

In [38]:

```python
# x is a test array
x = np.array([[1,0,2,0], [1,1,0,1], [2,0,1,1]])
print(x)

# For this toy example, w_0 and w_3 coocur 1 time in d_1,
# and 2 times d_2. Therefore, the total coocurrences is 3.

find_coocur(x)
```

```
[[1 0 2 0]
 [1 1 0 1]
 [2 0 1 1]]
```

Out[38]:

```
array([[0, 1, 4, 3],
       [1, 0, 0, 1],
       [4, 0, 0, 1],
       [3, 1, 1, 0]])
```

In [42]:

```python
# Structure of your solution to Assignment 1

import numpy as np
import pandas as pd


# best practice to test your class
# if your script is exported as a module,
# the following part is ignored
# this is equivalent to main() in Java

if __name__ == "__main__":

    # Test Question 1
    print("\n")
    print("=== Test Question 1 ===")

    dtm = pd.read_csv("dtm.csv")
    x = dtm.values
    analyze_tf(x)

    print("\n")
    print("=== Test Question 2 ===")
    emotion_analysis()

    print("\n")
    print("=== Test Question 3 ===")
    print(find_coocur(x))
```

```
=== Test Question 1 ===
Indexes of longest three documents: [4 3 2]
Indexes of words with the top 3 largest df values: [0 1 4]
Indexes of words with top 3 largest values: [[ 2  3  0]
 [ 6  7  8]
 [21 19 18]]


=== Test Question 2 ===
===The number of samples labeled for each emotion===
Anxiety       1381
Sadness        357
Relaxation     333
Fear           230
Anger          107
Happiness       39
Desire          27
Disgust         17
Name: emotion, dtype: int64


=== min, max, and mean values of sadness, happiness, and text length for each emotion===
             sadness            happiness                 length
                mean amin amax       mean amin amax         mean amin  amax
emotion
Anger       5.672897    1    9  3.177570    1    8  122.084112   88   374
Anxiety     5.719768    1    9  3.333816    1    9  118.066618   59   541
Desire      4.148148    1    8  4.925926    2    8  150.259259   89  1018
Disgust     4.764706    1    8  3.764706    1    6  108.411765   58   158
Fear        6.565217    1    9  3.056522    1    9  118.852174   80   322
Happiness   2.666667    1    9  7.230769    4    9  122.461538   92   272
Relaxation  2.858859    1    9  5.369369    1    9  119.696697    1   292
Sadness     7.436975    2    9  3.112045    1    9  122.117647   85   544


=== Cross tabulation of anxiety score by emotion and worry ===
worry          1         2         3         4         5         6         7  \
emotion
Anger       5.50  1.250000  3.416667  4.222222  5.125000  5.833333  6.040000
Anxiety     3.00  7.000000  5.222222  5.941176  6.149425  6.938931  7.597166
Desire       NaN  1.000000  3.142857  3.250000  3.666667  6.375000  6.000000
Disgust      NaN  5.000000       NaN  4.500000  5.000000  4.800000  5.750000
Fear         NaN       NaN  6.000000  5.000000  5.714286  5.454545  6.918033
Happiness   1.00  3.000000  2.400000  3.000000  1.000000  2.625000  4.000000
Relaxation  1.25  1.777778  2.301887  3.029851  3.090909  4.137931  4.219512
Sadness     2.50  3.857143  3.368421  4.606061  4.882353  5.325301  6.156863

worry              8         9
emotion
Anger       7.000000  7.437500
Anxiety     8.220126  8.770186
Desire           NaN       NaN
Disgust          NaN       NaN
Fear        7.643836  8.256757
Happiness   2.000000  6.500000
Relaxation  3.562500  4.200000
Sadness     6.565217  7.258065


=== Test Question 3 ===
[[0. 2. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0.]
 [2. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0.]
 [1. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1.]
 [1. 1. 0. 0. 0. 0. 2. 1. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 2. 0. 2. 2. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 2. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 2. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0.]
 [1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0.]
 [1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1. 1.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 1.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 1.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0.]]
```

```
In [ ]:
```