In [37]:

```
# Import Pandas
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer

# Import linear_kernel
from sklearn.metrics.pairwise import linear_kernel
from sklearn.metrics.pairwise import cosine_similarity

corpus=[ 'Julie loves me more than Linda loves me',
         'Jane likes me more than Julie loves me',
         'harry likes kiwi fruit']

# Creating coun vectorizer object. Note: we have removed the stop words
vectorizer = CountVectorizer(stop_words='english')
vectors = vectorizer.fit_transform(corpus)
print('\n ---Corpus converted to term-frequency vector:--\n', vectors.toarray())

#Array mapping from feature integer indices to feature name or we can say the unique te
rms present in the corpus
print('\n ---Unique terms in the corpus:--- \n', vectorizer.get_feature_names())

#Compute similarity score
cosine_sim = cosine_similarity(vectors, vectors)
cosine_sim2 = linear_kernel(vectors, vectors)
print('\n--Cosine similarity using cosine similarity function:--\n',cosine_sim)
print('\n--Cosine similarity using Linear Kernel function:--\n',cosine_sim2)



# You can write your own function as well for computing cosine similarity like followin
g:
"""
doc1=vectors.toarray()[0, :]
doc2=vectors.toarray()[1,:]
doc3=vectors.toarray()[2,:]
cos_sim_doc1_doc2 = dot(doc1, doc2)/(norm(doc1)*norm(doc2))

cos_sim_doc1_doc3 = dot(doc1, doc3)/(norm(doc1)*norm(doc3))

cos_sim_doc2_doc3 = dot(doc2, doc3)/(norm(doc2)*norm(doc3))

print(cos_sim_doc1_doc2)
print(cos_sim_doc1_doc3)
print(cos_sim_doc2_doc3)
"""
```

```
 ---Corpus converted to term-frequency vector:--
 [[0 0 0 1 0 0 1 2]
 [0 0 1 1 0 1 0 1]
 [1 1 0 0 1 1 0 0]]

 ---Unique terms in the corpus:---
 ['fruit', 'harry', 'jane', 'julie', 'kiwi', 'likes', 'linda', 'loves']

--Cosine similarity using cosine similarity function:--
 [[1.         0.61237244 0.        ]
 [0.61237244 1.         0.25      ]
 [0.         0.25       1.        ]]

--Cosine similarity using Linear Kernel function:--
 [[6. 3. 0.]
 [3. 4. 1.]
 [0. 1. 4.]]
```

Out[37]:

```
'\ndoc1=vectors.toarray()[0, :]\ndoc2=vectors.toarray()[1,:]\ndoc3=vector
s.toarray()[2,:]\ncos_sim_doc1_doc2 = dot(doc1, doc2)/(norm(doc1)*norm(doc
2))\n\ncos_sim_doc1_doc3 = dot(doc1, doc3)/(norm(doc1)*norm(doc3))\n\ncos_
sim_doc2_doc3 = dot(doc2, doc3)/(norm(doc2)*norm(doc3))\n\nprint(cos_sim_d
oc1_doc2)\nprint(cos_sim_doc1_doc3)\nprint(cos_sim_doc2_doc3)\n'
```

You can see that document 1 and 2 are more similar. You will also notice that their is differnce between the similarity score given by the cosine_similarity() and linear_kernel(). This happens becuase byy default 'CountVectorizer' fit.transform function does not length normalizes the data.

Q2. Repeat the above excercise but this time use tf-idf weights to convert the corpus?

In [2]:

```
# Import Pandas
import pandas as pd
#Import TfIdfVectorizer from scikit-learn
from sklearn.feature_extraction.text import TfidfVectorizer

# Import linear_kernel
from sklearn.metrics.pairwise import linear_kernel
from sklearn.metrics.pairwise import cosine_similarity

corpus=[ 'Julie loves me more than Linda loves me',
         'Jane likes me more than Julie loves me',
         'harry likes kiwi fruit']

#Define a TF-IDF Vectorizer Object. Remove all english stop words such as 'the', 'a'
tfidf = TfidfVectorizer(stop_words='english')
#tfidf = TfidfVectorizer()



#Construct the required TF-IDF matrix by fitting and transforming the data
tfidf_matrix = tfidf.fit_transform(corpus)

#Output the shape of tfidf_matrix

print(tfidf_matrix.shape)
print(tfidf_matrix.toarray())
#Array mapping from feature integer indices to feature name.
#print(tfidf.get_feature_names())
# Compute the cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
cosine_sim2 = cosine_similarity(tfidf_matrix, tfidf_matrix)

print('\n--Cosine similarity using cosine similarity function:--\n',cosine_sim)
print('\n--Cosine similarity using Linear Kernel function:--\n',cosine_sim2)
```

```
(3, 8)
[[0.         0.         0.         0.38550292 0.         0.
  0.50689001 0.77100584]
 [0.         0.         0.60465213 0.45985353 0.         0.45985353
  0.         0.45985353]
 [0.52863461 0.52863461 0.         0.         0.52863461 0.40204024
  0.         0.        ]]

--Cosine similarity using cosine similarity function:--
 [[1.         0.53182464 0.        ]
 [0.53182464 1.         0.18487962]
 [0.         0.18487962 1.        ]]

--Cosine similarity using Linear Kernel function:--
 [[1.         0.53182464 0.        ]
 [0.53182464 1.         0.18487962]
 [0.         0.18487962 1.        ]]
```

You can see that document 1 and 2 are more similar. You will also notice that their is differnce between the similarity score given by the cosine_similarity() and linear_kernel(). This happens becuase byy default 'TfidfVectorizer()' fit.transform function length normalizes the data.

Q3. Now build a complete recommeder system for a real world datatset. Read the data it the file 'movies_metadata.csv'. Use 'overview' attribute to compute similarity between the movies. Finally list top 10 similar movies to 'Father of the Bride Part II'. Note: You can use some basic pre-processing techniques for example, removing rows with missing values etc.

In [39]:

```
# Import Pandas
import pandas as pd


#Import TfIdfVectorizer from scikit-learn
from sklearn.feature_extraction.text import TfidfVectorizer

# Import linear_kernel
from sklearn.metrics.pairwise import linear_kernel
```

In [40]:

```
# Load Movies Metadata
metadata = pd.read_csv('dataset\movies_metadata.csv', low_memory=False)

# Print the first three rows
print(metadata.head(3))


#Print plot overviews of the first 5 movies.
print(metadata['overview'].head())
```

```
   adult                              belongs_to_collection    budget  \
0  False  {'id': 10194, 'name': 'Toy Story Collection', ...  30000000
1  False                                                NaN  65000000
2  False  {'id': 119050, 'name': 'Grumpy Old Men Collect...         0

                                              genres  \
0  [{'id': 16, 'name': 'Animation'}, {'id': 35, '...
1  [{'id': 12, 'name': 'Adventure'}, {'id': 14, '...
2  [{'id': 10749, 'name': 'Romance'}, {'id': 35, ...

                               homepage     id    imdb_id original_languag
e  \
0  http://toystory.disney.com/toy-story    862  tt0114709                e
n
1                                   NaN   8844  tt0113497                e
n
2                                   NaN  15602  tt0113228                e
n

     original_title                                           overview
...  \
0         Toy Story  Led by Woody, Andy's toys live happily in his ...
...
1           Jumanji  When siblings Judy and Peter discover an encha...
...
2  Grumpier Old Men  A family wedding reignites the ancient feud be...
...

  release_date      revenue runtime  \
0   1995-10-30  373554033.0    81.0
1   1995-12-15  262797249.0   104.0
2   1995-12-22          0.0   101.0

                                    spoken_languages    status  \
0           [{'iso_639_1': 'en', 'name': 'English'}]  Released
1  [{'iso_639_1': 'en', 'name': 'English'}, {'iso...  Released
2           [{'iso_639_1': 'en', 'name': 'English'}]  Released

                                           tagline             title  vi
deo  \
0                                              NaN         Toy Story  Fa
lse
1          Roll the dice and unleash the excitement!           Jumanji  Fa
lse
2  Still Yelling. Still Fighting. Still Ready for...  Grumpier Old Men  Fa
lse

  vote_average vote_count
0          7.7     5415.0
1          6.9     2413.0
2          6.5       92.0

[3 rows x 24 columns]
0    Led by Woody, Andy's toys live happily in his ...
1    When siblings Judy and Peter discover an encha...
2    A family wedding reignites the ancient feud be...
3    Cheated on, mistreated and stepped on, the wom...
4    Just when George Banks has recovered from his ...
Name: overview, dtype: object
```

In [41]:

```
#Replace NaN with an empty string
metadata['overview'] = metadata['overview'].fillna('')
```

In [42]:

```
#Define a TF-IDF Vectorizer Object. Remove all english stop words such as 'the', 'a'
tfidf = TfidfVectorizer(stop_words='english')



#Construct the required TF-IDF matrix by fitting and transforming the data
tfidf_matrix = tfidf.fit_transform(metadata['overview'])

#Output the shape of tfidf_matrix
print(tfidf_matrix.shape)

#Array mapping from feature integer indices to feature name.
print(tfidf.get_feature_names()[5000:5010])
```

```
(45466, 75827)
['avails', 'avaks', 'avalanche', 'avalanches', 'avallone', 'avalon', 'avan
t', 'avanthika', 'avanti', 'avaracious']
```

In [43]:

```
# Compute the cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)

print('cosine similiarity matrix shape:', cosine_sim.shape)

#Construct a reverse map of indices and movie titles
indices = pd.Series(metadata.index, index=metadata['title']).drop_duplicates()


print(indices[:10])
```

```
cosine similiarity matrix shape: (45466, 45466)
title
Toy Story                      0
Jumanji                        1
Grumpier Old Men               2
Waiting to Exhale              3
Father of the Bride Part II    4
Heat                           5
Sabrina                        6
Tom and Huck                   7
Sudden Death                   8
GoldenEye                      9
dtype: int64
```

In [44]:

```
# Function that takes in movie title as input and outputs most similar movies
def get_recommendations(title, cosine_sim=cosine_sim):
    # Get the index of the movie that matches the title
    idx = indices[title]

    # Get the pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return metadata['title'].iloc[movie_indices]
get_recommendations('Father of the Bride Part II')
```

Out[44]:

```
6793         Father of the Bride
6571                       Kuffs
6306             North to Alaska
19801                    Babbitt
34466          You're Killing Me
13611        The Magic of Méliès
5005                     Wendigo
27974           I Start Counting
43887     George of the Jungle 2
7097           The Out of Towners
Name: title, dtype: object
```

Code credit

https://www.datacamp.com/community/tutorials/recomr systems-python (https://www.datacamp.com/community/tutorials/recom systems-python)

More on Text similiarity

https://medium.com/@adriensieg/text-similarities-da019229c894 (https://medium.com/@adriensieg/text-similarities-da019229c894)

More on recommender system

https://goodboychan.github.io/chans_jupyter/python/datacamp/natural_language_processing/2020/07/17/04-TF-IDF-and-similarity-scores.html#Cosine-similarity (https://goodboychan.github.io/chans_jupyter/python/datacamp/natural_language_processing/2020/07/17/04-TF-IDF-and-similarity-scores.html#Cosine-similarity)

In [ ]: