Clustering Assignment

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True
```

There will be some functions that start with the word "grader" ex: grader_actors(), grader_movies(), grader_cost1() etc, you should not change those function definition.

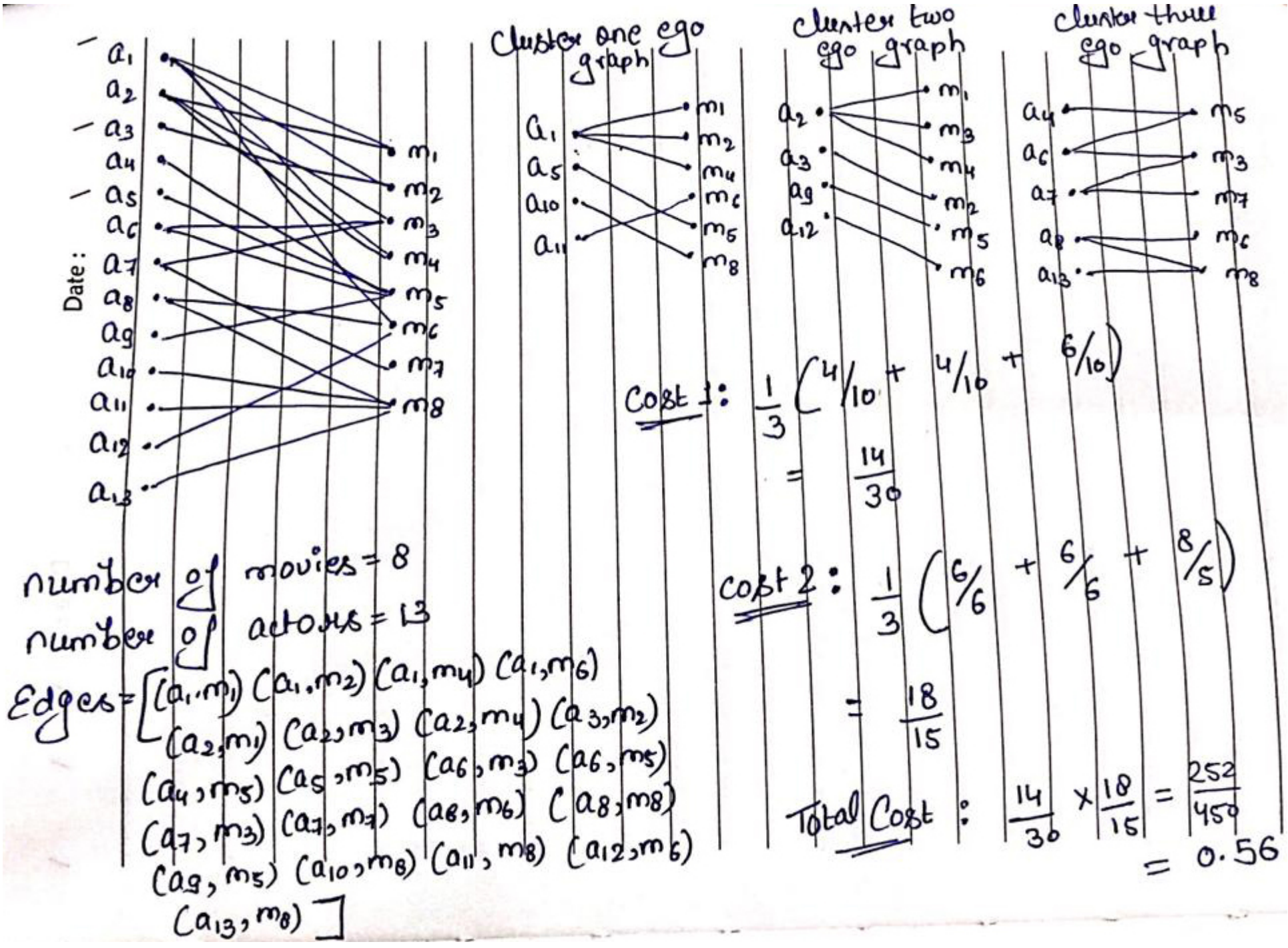
Every Grader function has to return True.

Please check clustering assignment helper functions notebook before attempting this assignment.

- Read graph from the given movie_actor_network.csv (note that the graph is bipartite graph.)
- Using stellergaph and gensim packages, get the dense representation(128dimensional vector) of every node in the graph. [Refer Clustering_Assignment_Reference.ipynb]
- Split the dense representation into actor nodes, movies nodes.(Write you code in def data_split())

Task 1 : Apply clustering algorithm to group similar actors

1. For this task consider only the actor nodes
2. Apply any clustering algorithm of your choice
   Refer : https://scikit-learn.org/stable/modules/clustering.html
3. Choose the number of clusters for which you have maximum score of $Cost1 * Cost2$
4. Cost1 = $\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(number of nodes in the largest connected component in the graph with the actor nodes and its movie neighbours in cluster i)}}{\text{(total number of nodes in that cluster i)}}$ where N= number of clusters
   (Write your code in def cost1())
5. Cost2 = $\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(sum of degress of actor nodes in the graph with the actor nodes and its movie neighbours in cluster i)}}{\text{(number of unique movie nodes in the graph with the actor nodes and its movie neighbours in cluster i)}}$ where N= number of clusters
   (Write your code in def cost2())
6. Fit the clustering algorithm with the opimal number_of_clusters and get the cluster number for each node
7. Convert the d-dimensional dense vectors of nodes into 2-dimensional using dimensionality reduction techniques (preferably TSNE)
8. Plot the 2d scatter plot, with the node vectors after step e and give colors to nodes such that same cluster nodes will have same color



Task 2 : Apply clustering algorithm to group similar movies

1. For this task consider only the movie nodes
2. Apply any clustering algorithm of your choice
3. Choose the number of clusters for which you have maximum score of $Cost1 * Cost2$

   Cost1 = $\frac{1}{N} \sum_{\text{each cluster i}} \frac{\text{(number of nodes in the largest connected component in the graph with the movie nodes and its actor neighbours in cluster i)}}{\text{(total number of nodes in that cluster i)}}$ where N= number of clusters

   (Write your code in def cost1())
4. Cost2 = $\frac{1}{N} \sum_{\text{each cluster i}} \frac{\text{(sum of degress of movie nodes in the graph with the movie nodes and its actor neighbours in cluster i)}}{\text{(number of unique actor nodes in the graph with the movie nodes and its actor neighbours in cluster i)}}$ where N= number of clusters

   (Write your code in def cost2())

Algorithm for actor nodes

```
for number_of_clusters in [3, 5, 10, 30, 50, 100, 200, 500]:
    algo = clustering_algorith(clusters=number_of_clusters)
    # you will be passing a matrix of size N*d where N number of actor nodes and d is dimension from gensim
    algo.fit(the dense vectors of actor nodes)
    You can get the labels for corresponding actor nodes (algo.labels_)
    Create a graph for every cluster(ie., if n_clusters=3, create 3 graphs)
    (You can use ego_graph to create subgraph from the actual graph)
    compute cost1,cost2
       (if n_cluster=3, cost1=cost1(graph1)+cost1(graph2)+cost1(graph3) # here we are doing summation
        cost2=cost2(graph1)+cost2(graph2)+cost2(graph3)
    computer the metric Cost = Cost1*Cost2
return number_of_clusters which have maximum Cost
```

```
!pip install networkx==2.3
```

```
Requirement already satisfied: networkx==2.3 in /usr/local/lib/python3.6/dist-packages (2.3)
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.6/dist-packages (from networkx==2.3) (4.4.2)
```

```
!pip install stellargraph
```

```
Requirement already satisfied: stellargraph in /usr/local/lib/python3.6/dist-packages (1.2.1)
Requirement already satisfied: gensim>=3.4.0 in /usr/local/lib/python3.6/dist-packages (from stellargraph) (3.6.0)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.6/dist-packages (from stellargraph) (2.3)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.6/dist-packages (from stellargraph) (1.1.5)
Requirement already satisfied: matplotlib>=2.2 in /usr/local/lib/python3.6/dist-packages (from stellargraph) (3.2.2)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.6/dist-packages (from stellargraph) (1.4.1)
Requirement already satisfied: numpy>=1.14 in /usr/local/lib/python3.6/dist-packages (from stellargraph) (1.19.4)
Requirement already satisfied: tensorflow>=2.1.0 in /usr/local/lib/python3.6/dist-packages (from stellargraph) (2.4.0)
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.6/dist-packages (from stellargraph) (0.22.2.post1)
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.6/dist-packages (from gensim>=3.4.0->stellargraph) (1.15.0)
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.6/dist-packages (from gensim>=3.4.0->stellargraph)
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.6/dist-packages (from networkx>=2.2->stellargraph) (
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages (from pandas>=0.24->stellargraph) (2018
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.6/dist-packages (from pandas>=0.24->stellargra
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=2.2->stellargraph
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=2.2->stellargraph) (0
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from matplo
Requirement already satisfied: flatbuffers~=1.12.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellarg
Requirement already satisfied: astunparse~=1.6.3 in /usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellargra
Requirement already satisfied: typing-extensions~=3.7.4 in /usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->ste
Requirement already satisfied: tensorflow-estimator<2.5.0,>=2.4.0rc0 in /usr/local/lib/python3.6/dist-packages (from tensorflo
Requirement already satisfied: google-pasta~=0.2 in /usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellargra
Requirement already satisfied: opt-einsum~=3.3.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellargra
Requirement already satisfied: keras-preprocessing~=1.1.2 in /usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->s
Requirement already satisfied: absl-py~=0.10 in /usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellargraph)
Requirement already satisfied: termcolor~=1.1.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellargrap
Requirement already satisfied: grpcio~=1.32.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellargraph)
Requirement already satisfied: h5py~=2.10.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellargraph) (
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellargraph
Requirement already satisfied: wheel~=0.35 in /usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellargraph) (
Requirement already satisfied: gast==0.3.3 in /usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellargraph) (
Requirement already satisfied: tensorboard~=2.4 in /usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellargrap
Requirement already satisfied: wrapt~=1.12.1 in /usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellargraph)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (from scikit-learn>=0.20->stellargraph)
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (from protobuf>=3.9.2->tensorflow>=2.1.0->
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.6/dist-packages (from tensorboard~=2.4->tensorflow>=2
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.6/dist-packages (from tensorboard~=2
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.6/dist-packages (from tensorboard~=2.4->tensorflo
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.6/dist-packages (from tensorboard~=2.4->tensorflow>
Requirement already satisfied: google-auth<2,>=1.6.3 in /usr/local/lib/python3.6/dist-packages (from tensorboard~=2.4->tensorf
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.6/dist-packages (from tensorboard~=2.4-
Requirement already satisfied: importlib-metadata; python_version < "3.8" in /usr/local/lib/python3.6/dist-packages (from mark
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.6/dist-packages (from google-auth-oauthlib<0
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorbo
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorboard~=
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from request
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from requests<3,>=2.21.0->tensorb
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.6/dist-packages (from google-auth<2,>=1.6.3->t
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.6/dist-packages (from google-auth<2,>=1.6.3->te
Requirement already satisfied: rsa<5,>=3.1.4; python_version >= "3" in /usr/local/lib/python3.6/dist-packages (from google-aut
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.6/dist-packages (from importlib-metadata; python_version <
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.6/dist-packages (from requests-oauthlib>=0.7.0->googl
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.6/dist-packages (from pyasn1-modules>=0.2.1->goo
```

```
import networkx as nx
from networkx.algorithms import bipartite
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import numpy as np
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
from stellargraph.data import UniformRandomMetaPathWalk
from stellargraph import StellarGraph
```

```
path = '/content/drive/MyDrive/AAIC/ASSIGN_14/Clustering_Assignment/'
```

```
data=pd.read_csv(path+'movie_actor_network.csv', index_col=False, names=['movie','actor'])
data.head()
```

	movie	actor
0	m1	a1
1	m2	a1
2	m2	a2
3	m3	a1

```
data.values.tolist()[:5]
```

```
[['m1', 'a1'], ['m2', 'a1'], ['m2', 'a2'], ['m3', 'a1'], ['m3', 'a3']]
```

```
edges = [tuple(x) for x in data.values.tolist()]
```

In bipartite graphs we have to keep track of which set each node belongs to, and make sure that there is no edge between nodes of the same set. The convention used in NetworkX is to use a node attribute named bipartite with values 0 or 1 to identify the sets each node belongs to. This convention is not enforced in the source code of bipartite functions, it's only a recommendation.

```
B = nx.Graph()
B.add_nodes_from(data['movie'].unique(), bipartite=0, label='movie')
B.add_nodes_from(data['actor'].unique(), bipartite=1, label='actor')
B.add_edges_from(edges, label='acted')
```

```
A = list(nx.connected_component_subgraphs(B))[0]  # Generate connected components as subgraphs.
```

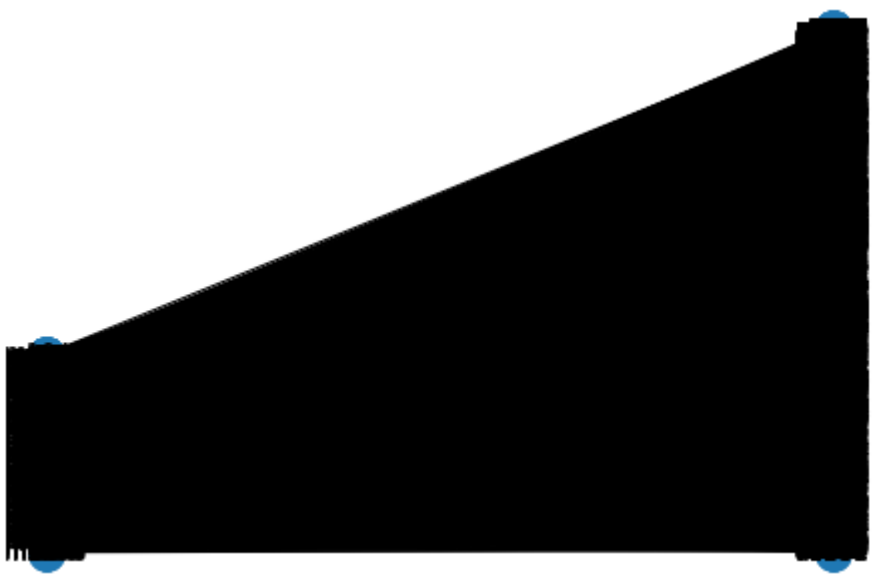
```
print("number of nodes", A.number_of_nodes())
print("number of edges", A.number_of_edges())
```

```
number of nodes 4703
number of edges 9650
```

```
l, r = nx.bipartite.sets(A)  #Returns bipartite node sets of graph G.
#print(l)
#print(r)
pos = {}

pos.update((node, (1, index)) for index, node in enumerate(l))
pos.update((node, (2, index)) for index, node in enumerate(r))

nx.draw(A, pos=pos, with_labels=True)  # Draw the graph G with Matplotlib.
#Parameters:
#G (graph) – A networkx graph
#pos (dictionary, optional) –
#A dictionary with nodes as keys and positions as values.
#If not specified a spring layout positioning will be computed.
#See networkx.layout for functions that compute node positions.
plt.show()
```



```
movies = []
actors = []
for i in A.nodes():
    if 'm' in i:
        movies.append(i)
    if 'a' in i:
        actors.append(i)
```

```
        actors.append(1)
print('number of movies ', len(movies))
print('number of actors ', len(actors))
```

```
number of movies  1292
number of actors  3411
```

```

# Create the random walker
rw = UniformRandomMetaPathWalk(StellarGraph(A))  #For heterogeneous graphs, it performs uniform random walks based on given me
#Optional parameters default to using the values passed in during construction.

# specify the metapath schemas as a list of lists of node types.
metapaths = [
    ["movie", "actor", "movie"],
    ["actor", "movie", "actor"]
]
#Performs metapath-driven uniform random walks on heterogeneous graphs.
walks = rw.run(nodes=list(A.nodes()), # root nodes
               length=100,  # maximum length of a random walk
               n=1,        # number of random walks per root node
               metapaths=metapaths
              )

print("Number of random walks: {}".format(len(walks)))
```

```
Number of random walks: 4703
```

```
from gensim.models import Word2Vec
model = Word2Vec(walks, size=128, window=5)
```

```
model.wv.vectors.shape  # 128-dimensional vector for each node in the graph
```

```
(4703, 128)
```

```
# Retrieve node embeddings and corresponding subjects
node_ids = model.wv.index2word  # list of node IDs
node_embeddings = model.wv.vectors  # numpy.ndarray of size number of nodes times embeddings dimensionality
node_targets = [ A.node[node_id]['label'] for node_id in node_ids]
```

```
print(node_ids[:15], end='')
```

```
['a973', 'a967', 'a964', 'a1731', 'a969', 'a970', 'a1028', 'a1057', 'a965', 'a1003', 'm1094', 'a966', 'm67', 'a988', 'm1111']
```

```
print(node_targets[:15],end='')
```

```
['actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'movie', 'actor', 'movie', 'actor', 'movie']
```

```
print(node_ids[:10])
print(node_targets[:10])
print(len(node_embeddings[0]))
```

```
['a973', 'a967', 'a964', 'a1731', 'a970', 'a969', 'a1057', 'm1094', 'a1028', 'a1003']
['actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'movie', 'actor', 'actor']
128
```

```
def data_split(node_ids,node_targets,node_embeddings):
    '''In this function, we will split the node embeddings into actor_embeddings , movie_embeddings '''
    actor_nodes,movie_nodes=[],[]
    actor_embeddings,movie_embeddings=[],[]
    # split the node_embeddings into actor_embeddings,movie_embeddings based on node_ids
    # By using node_embedding and node_targets, we can extract actor_embedding and movie embedding
    # By using node_ids and node_targets, we can extract actor_nodes and movie nodes

    for i in range(len(node_embeddings)):
      if node_targets[i] == 'actor':
        actor_embeddings.append(node_embeddings[i])
        actor_nodes.append(node_ids[i])
      elif node_targets[i] == 'movie':
        movie_embeddings.append(node_embeddings[i])
        movie_nodes.append(node_ids[i])

    return actor_nodes,movie_nodes,actor_embeddings,movie_embeddings
actor_nodes,movie_nodes,actor_embeddings,movie_embeddings = data_split(node_ids,node_targets,node_embeddings)
```

```
actor_nodes = np.array(actor_nodes)
actor_embeddings = np.array(actor_embeddings)

movie_nodes = np.array(movie_nodes)
movie_embeddings = np.array(movie_embeddings)
print(actor_embeddings.shape,actor_nodes.shape,movie_embeddings.shape,movie_nodes.shape)
```

```
(3411, 128) (3411,) (1292, 128) (1292,)
```

Grader function - 1

```
def grader_actors(data):
    assert(len(data)==3411)
    return True
grader_actors(actor_nodes)
```

```
True
```

Grader function - 2

```
def grader_movies(data):
    assert(len(data)==1292)
    return True
grader_movies(movie_nodes)
```

```
True
```

Calculating cost1

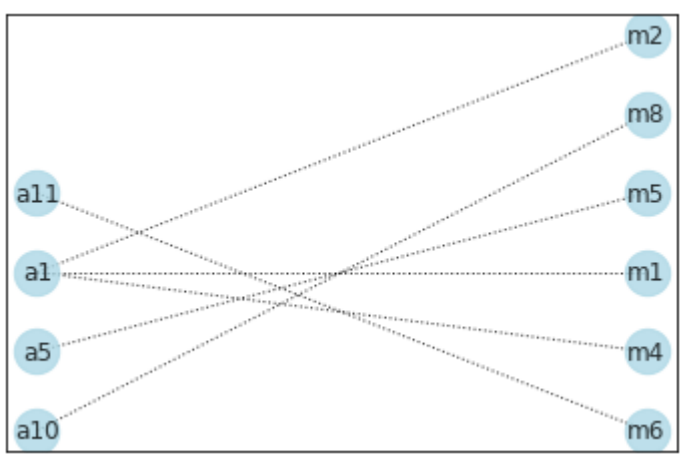
Cost1 = $\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(number of nodes in the largest connected component in the graph with the actor nodes and its movie neighbours in cluster i)}}{\text{(total number of nodes in that cluster i)}}$ where N= number of clusters

```
def cost1(graph,number_of_clusters):
    '''In this function, we will calculate cost1'''

    largest_cc = max(nx.connected_components(graph), key=len)
    cost1=(1/number_of_clusters)*(len(largest_cc)/len(graph.nodes())) # calculate cost1

    return cost1
```

```
import networkx as nx
from networkx.algorithms import bipartite
graded_graph= nx.Graph()
graded_graph.add_nodes_from(['a1','a5','a10','a11'], bipartite=0) # Add the node attribute "bipartite"
graded_graph.add_nodes_from(['m1','m2','m4','m6','m5','m8'], bipartite=1)
graded_graph.add_edges_from([('a1','m1'),('a1','m2'),('a1','m4'),('a11','m6'),('a5','m5'),('a10','m8')])
l={'a1','a5','a10','a11'};r={'m1','m2','m4','m6','m5','m8'}
pos = {}
pos.update((node, (1, index)) for index, node in enumerate(l))
pos.update((node, (2, index)) for index, node in enumerate(r))
nx.draw_networkx(graded_graph, pos=pos, with_labels=True,node_color='lightblue',alpha=0.8,style='dotted',node_size=500)
```



Grader function - 3

```
graded_cost1=cost1(graded_graph,3)
def grader_cost1(data):
    assert(data==((1/3)*(4/10))) # 1/3 is number of clusters
    return True
grader_cost1(graded_cost1)
```

```
True
```

Calculating cost2

Cost2 = $\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(sum of degress of actor nodes in the graph with the actor nodes and its movie neighbours in cluster i)}}{\text{(number of unique movie nodes in the graph with the actor nodes and its movie neighbours in cluster i)}}$ where N= number of clusters

```
def cost2(graph,number_of_clusters):
    '''In this function, we will calculate cost1'''

    movies = []
    actors = []
    for i in graph.nodes():
      if 'm' in i:
        movies.append(i)
      if 'a' in i:
        actors.append(i)

    lst = [graph.degree(j) for j in actors]
    tot = sum(lst)
    unique_movie_nodes = len(set(movies))

    cost2=(1/number_of_clusters)*(tot/unique_movie_nodes) # calculate cost1

    return cost2
```

Grader function - 4

```
graded_cost2=cost2(graded_graph,3)
def grader_cost2(data):
    assert(data==((1/3)*(6/6))) # 1/3 is number of clusters
    return True
grader_cost2(graded_cost2)
```

```
True
```

Grouping similar actors

```
from sklearn.cluster import KMeans

actor_nodes = np.array(actor_nodes, dtype=str)
#print(actor_nodes)
cluster_lst = [3, 5, 7, 9, 11, 13, 15, 30, 50, 100, 200, 500]
cost = []
for number_of_clusters in cluster_lst :
  algo = KMeans(number_of_clusters)
  algo.fit(actor_embeddings)
  label = algo.labels_
  #print(label)
  cost_1 = 0
  cost_2 = 0
  for i in range(number_of_clusters):
    indices = [j for j, x in enumerate(label) if x == i]
    actor_nodes_label = actor_nodes[indices]
    #print(actor_nodes_label)
    G1=nx.Graph()
    for k in actor_nodes_label:
      sub_graph1=nx.ego_graph(B,k)
      G1.add_nodes_from(sub_graph1.nodes) # adding nodes
      G1.add_edges_from(sub_graph1.edges()) # adding edges
    cost_1 += cost1(G1,number_of_clusters)
    cost_2 += cost2(G1,number_of_clusters)
  cost.append(cost_1*cost_2)

optimal_k = cluster_lst[cost.index(max(cost))]
print(cost)
print(optimal_k)
```

```
[3.7098804237625895, 2.8915825331787737, 2.6819462277924546, 2.440646338527414, 2.414603057818663, 2.174784849242253, 2.034777
3
```

Displaying similar actor clusters

```
kmeans = KMeans(optimal_k).fit(actor_embeddings)
label = kmeans.labels_
print(label)
print(set(label))
```

```
[2 2 2 ... 1 1 1]
{0, 1, 2}
```

```
from sklearn.manifold import TSNE
transform = TSNE #PCA
trans = transform(n_components=2)
actor_embeddings_2d = trans.fit_transform(actor_embeddings)
```

I made a small mistake in the previous code,

Previous code

```
label_map = { l: i for i, l in enumerate(np.unique(actor_nodes))}
```

I did not put label[i] and have put np.unique(actor_nodes) in the previous code.

Present code

```
label_map = { l: label[i] for i, l in enumerate((actor_nodes))}
```

2. findings from the mistakes

- Previously I have just put [i] in the place of label[i] by which label map was giving the values as actor_ids rather than the labels of actor_ids.
- In the previous code I have used np.unique(actor_nodes) which is changing the order of actor_nodes there by labels are also changing.

Rather than all these nonsense we could also use

```
node_colours = label
```

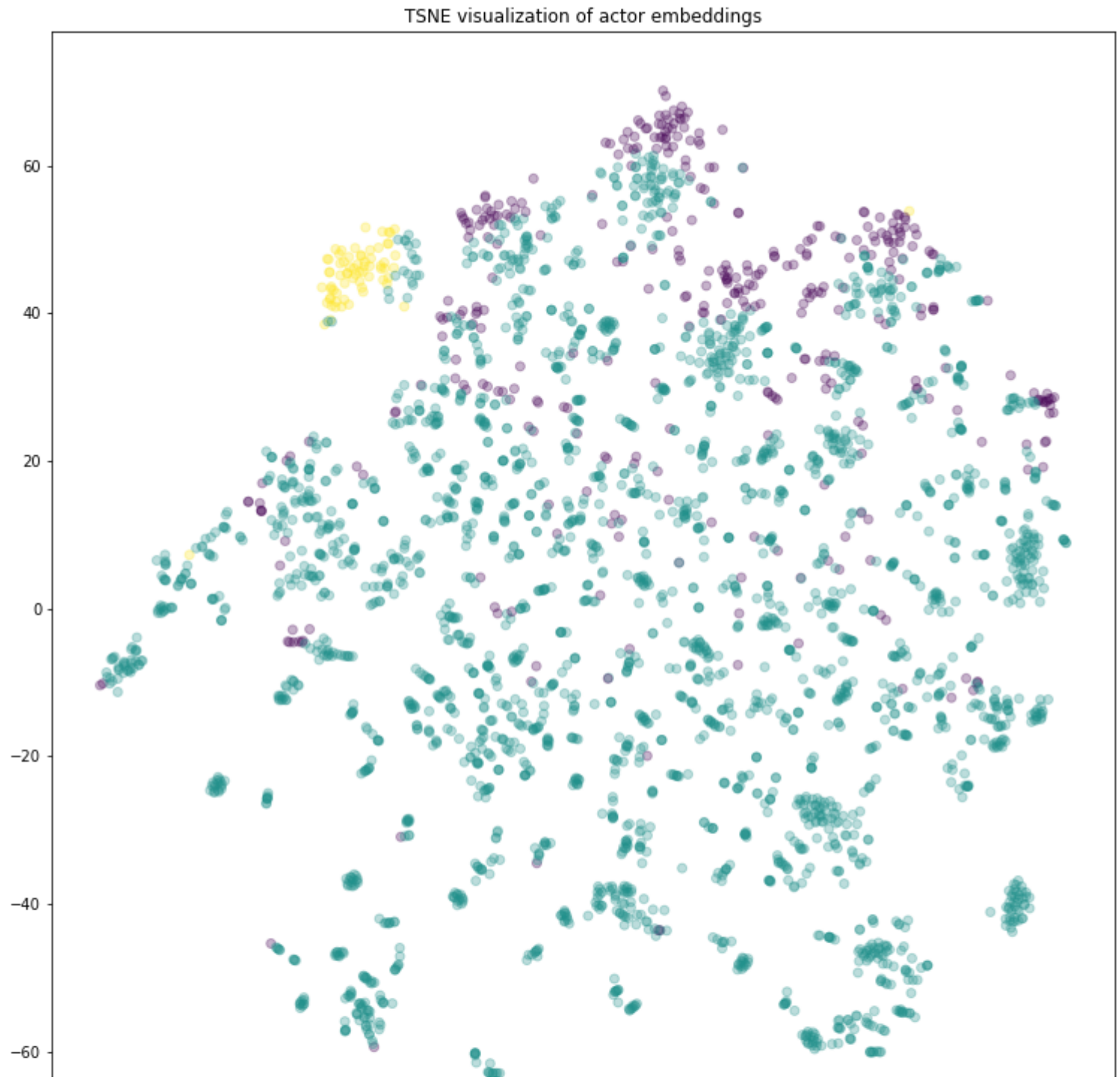
and this directly gives the labels of each node_embedding.

```
import numpy as np
# draw the points

label_map = { l: label[i] for i, l in enumerate((actor_nodes))}
#print(label_map)
node_colours = [ label_map[target] for target in actor_nodes]
print(set(node_colours))
count = {i:node_colours.count(i) for i in (set(node_colours))}
print(count)
plt.figure(figsize=(20,16))
plt.axes().set(aspect="equal")
plt.scatter(actor_embeddings_2d[:,0],
            actor_embeddings_2d[:,1],
            c=node_colours, alpha=0.3)
plt.title('{} visualization of actor embeddings'.format(transform.__name__))

plt.show()
```

```
{0, 1, 2}
{0: 375, 1: 2967, 2: 69}
```



As one cluster is having more frequency, we can observe that one particular colour is more prominent in the above plot.

Grouping similar movies

```
def cost2_movie(graph,number_of_clusters):
    '''In this function, we will calculate cost1'''

    movies = []
    actors = []
    for i in graph.nodes():
      if 'm' in i:
        movies.append(i)
      if 'a' in i:
        actors.append(i)

    lst = [graph.degree(j) for j in movies]
    tot = sum(lst)
    unique_actor_nodes = len(set(actors))

    cost2=(1/number_of_clusters)*(tot/unique_actor_nodes) # calculate cost1

    return cost2
```

```
from sklearn.cluster import KMeans

movie_nodes = np.array(movie_nodes, dtype=str)
#print(actor_nodes)
cluster_lst = [3, 5, 7, 9, 11, 13, 15, 30, 50, 100, 200, 500]
cost = []
for number_of_clusters in cluster_lst :
  algo = KMeans(number_of_clusters)
  algo.fit(movie_embeddings)
  label = algo.labels_
  #print(label)
  cost_1 = 0
```

```
  cost_2 = 0
  for i in range(number_of_clusters):
    indices = [j for j, x in enumerate(label) if x == i]
    movie_nodes_label = movie_nodes[indices]
    #print(movie_nodes_label)
    G1=nx.Graph()
    for k in movie_nodes_label:
      sub_graph1=nx.ego_graph(B,k)
      G1.add_nodes_from(sub_graph1.nodes) # adding nodes
      G1.add_edges_from(sub_graph1.edges()) # adding edges
    cost_1 += cost1(G1,number_of_clusters)
    cost_2 += cost2_movie(G1,number_of_clusters)
  cost.append(cost_1*cost_2)

optimal_k = cluster_lst[cost.index(max(cost))]
print(cost)
print(optimal_k)
```

```
[2.8819795929988223, 2.544021623638951, 2.273274516829582, 2.3829843078384525, 2.203801143703249, 2.721473171763052, 2.759912
3
```

Displaying similar movie clusters

```
algo = KMeans(optimal_k)
algo.fit(movie_embeddings)
label = algo.labels_
print(label)
print(set(label))
```

```
[2 2 2 ... 1 1 1]
{0, 1, 2}
```

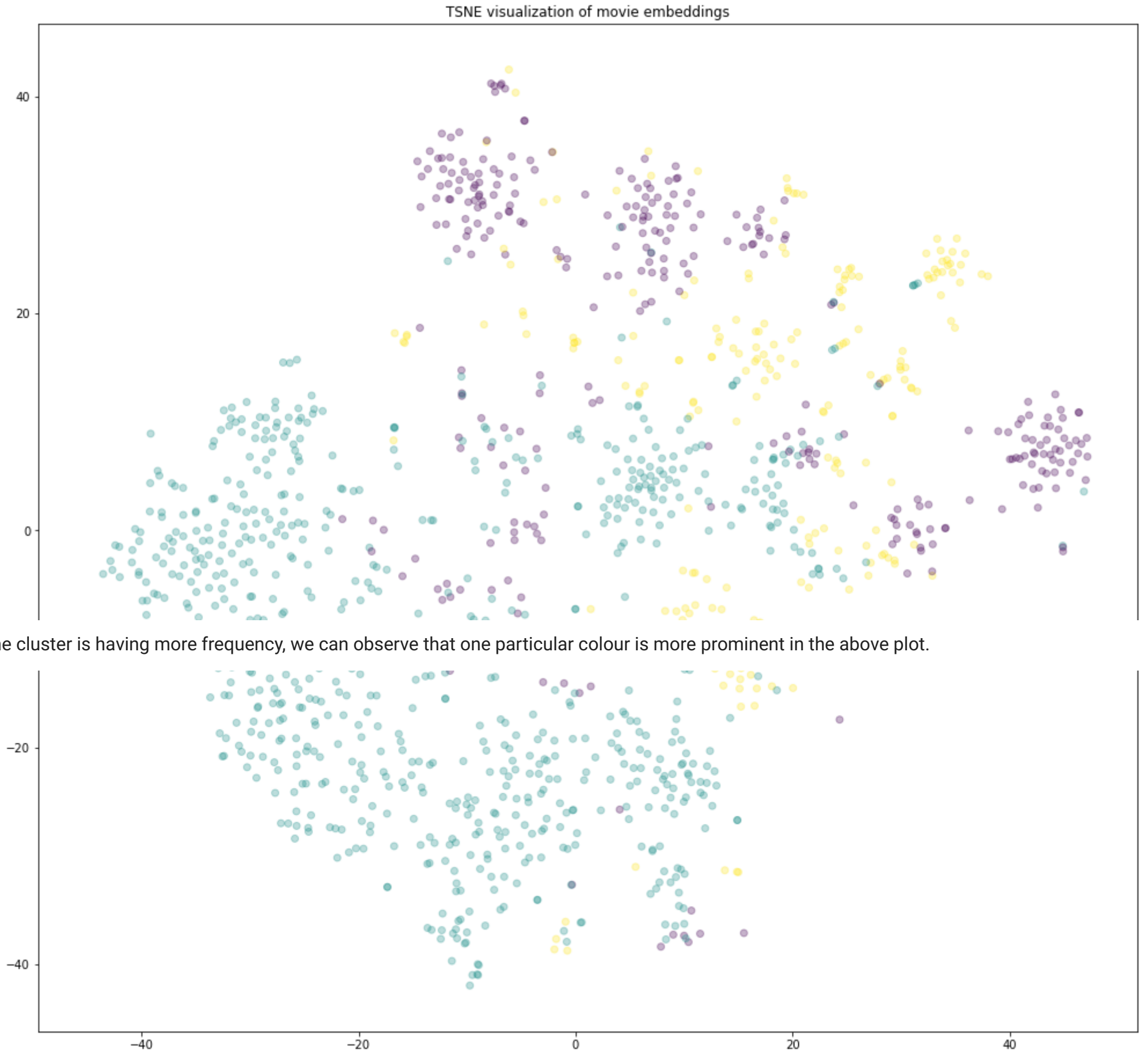
```
from sklearn.manifold import TSNE
transform = TSNE #PCA
trans = transform(n_components=2)
movie_embeddings_2d = trans.fit_transform(movie_embeddings)
```

```
import numpy as np
# draw the points

label_map = { l: label[i] for i, l in enumerate((movie_nodes))}
#print(label_map)
node_colours = [ label_map[target] for target in movie_nodes]
print(set(node_colours))
count = {i:node_colours.count(i) for i in (set(node_colours))}
print(count)
plt.figure(figsize=(20,16))
plt.axes().set(aspect="equal")
plt.scatter(movie_embeddings_2d[:,0],
            movie_embeddings_2d[:,1],
            c=node_colours, alpha=0.3)
plt.title('{} visualization of movie embeddings'.format(transform.__name__))

plt.show()
```

```
{0, 1, 2}
{0: 305, 1: 763, 2: 224}
```



As one cluster is having more frequency, we can observe that one particular colour is more prominent in the above plot.