Clustering Assignment

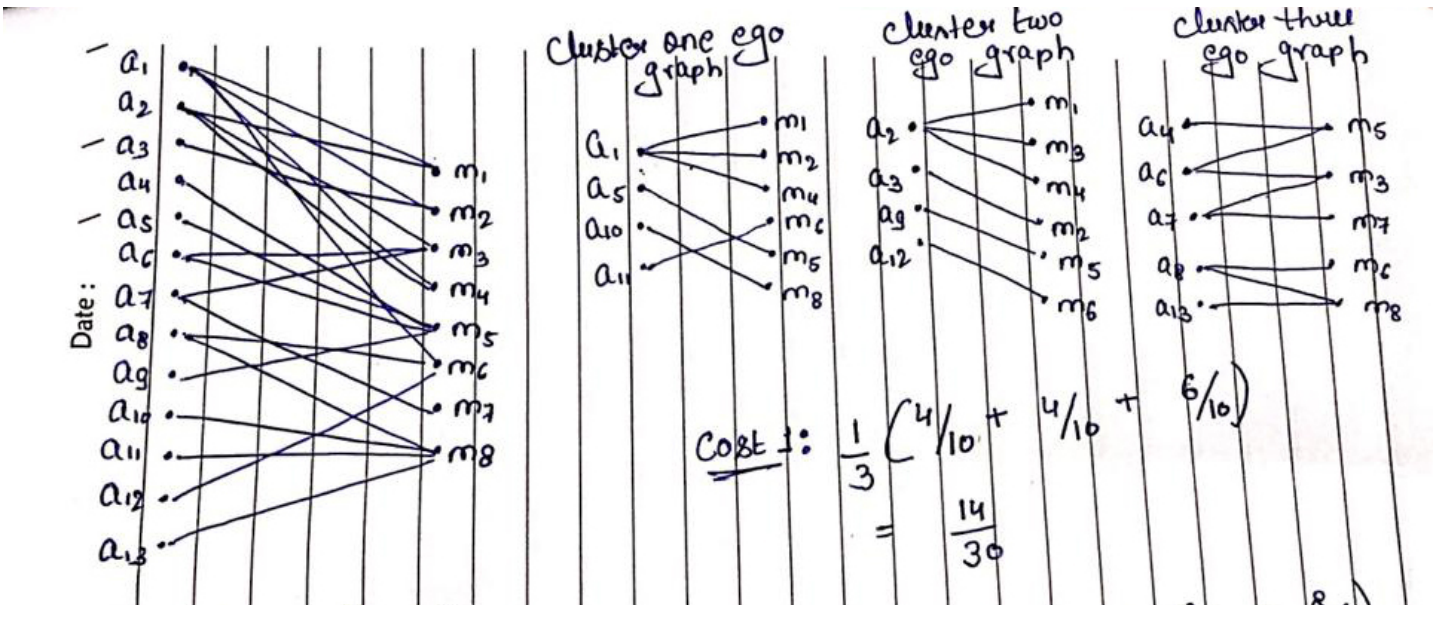
There will be some functions that start with the word "grader" ex: grader_actors(), grader_movies(), grader_cost1() etc, you should not change those function definition.

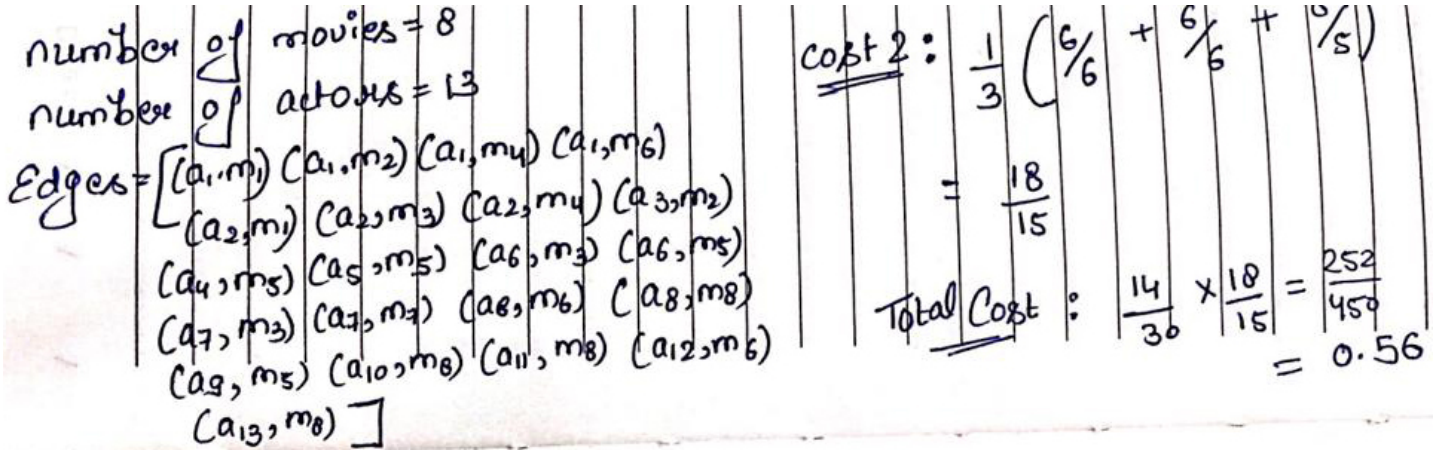
Every Grader function has to return True.

- **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())
3. **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())

```
    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 a
nd 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
```

In [ ]:

```
pip install stellargraph
```

In [ ]:

```
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
```

In [ ]:

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

```
Mounted at /content/drive
```

In [ ]:

```
path = "drive/My Drive/Colab Notebooks/"
data=pd.read_csv(path+'movie_actor_network.csv', index_col=False, names=['movie','actor'
])
```

In [ ]:

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

In [ ]:

```
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')
```

In [ ]:

```
A = (B.subgraph(c) for c in nx.connected_components(B))
A = list(A)[0]
```

In [ ]:

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

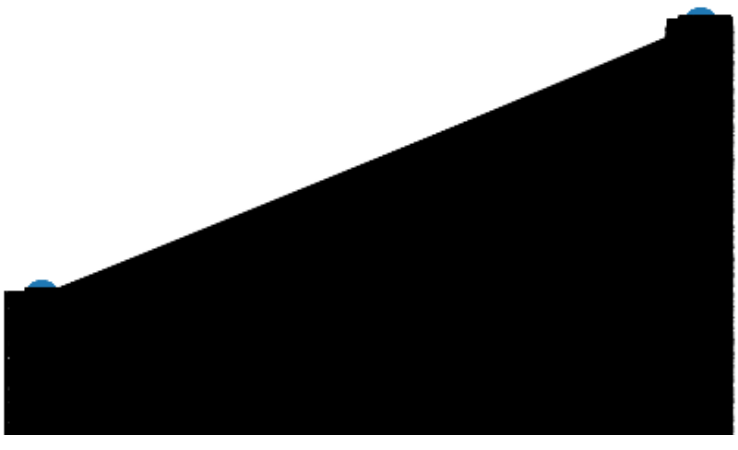
```
number of nodes 4703
number of edges 9650
```

In [ ]:

```
l, r = nx.bipartite.sets(A)
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)
plt.show()
```



In [ ]:

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

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

In [ ]:

```
# Create the random walker
rw = UniformRandomMetaPathWalk(StellarGraph(A))

# specify the metapath schemas as a list of lists of node types.
metapaths = [
    ["movie", "actor", "movie"],
    ["actor", "movie", "actor"]
]

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
```

In [ ]:

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

In [ ]:

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

Out[ ]:

```
(4703, 128)
```

In [ ]:

```
# 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 embeddi
ngs dimensionality
node_targets = [ A.nodes[node_id]['label'] for node_id in node_ids]
```

In [ ]:

```
actor_nodes=[i for i in node_ids if "a" in i]
movie_nodes=[j for j in node_ids if "m" in j]
len(actor_nodes),len(movie_nodes)
```

Out[ ]:

```
(3411, 1292)
```

In [ ]:

```
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 node_ids:
        if 'm' in i:
            movie_nodes.append(i)
        else:
            actor_nodes.append(i)

    for i in range(len(node_ids)):
        if 'm' in node_ids[i]:
            movie_embeddings.append(node_embeddings[i])
        else:
            actor_embeddings.append(node_embeddings[i])

    return actor_nodes,movie_nodes,actor_embeddings,movie_embeddings
```

In [ ]:

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

Out[ ]:

```
True
```

In [ ]:

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

Out[ ]:

```
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

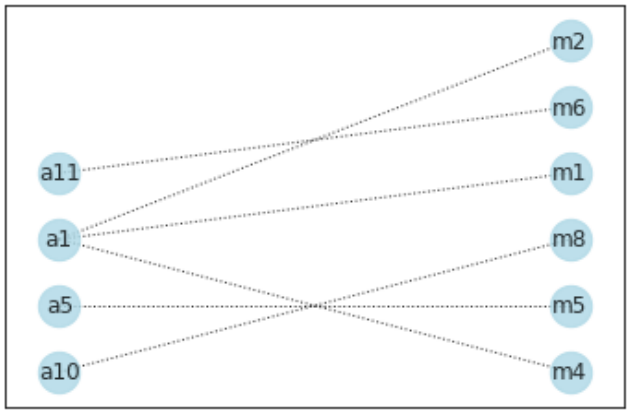
In [ ]:

```
def cost1(graph,number_of_clusters):
    '''In this function, we will calculate cost1'''
    max_nodes,total_nodes=len(max(nx.connected_components(graph),key=len)),graph.number_o
f_nodes()
    cost1= max_nodes/(total_nodes)
    return cost1/number_of_clusters
```

In [ ]:

```
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 attribut
e "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)
```



In [ ]:

```
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)
```

Out[ ]:

True

In [ ]:

```
def cost2(graph,number_of_clusters):
    '''In this function, we will calculate cost2'''
    num=0
    den=0
    for i in graph.nodes:
        if "a" in i:
            num+=graph.degree(i)
        else:
            den+=1
    cost2= num/(den)
    return cost2/number_of_clusters
```

In [ ]:

```
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)
```

Out[ ]:

True

Task 1 : Applying clustering algorithm to group similar actors

In [ ]:

```
def Kmeans_clustering(nodes,list_num_cluster,embeddings):
```

```
    all_cost=[]
    #Running loop for different number of cluster values
    for k in (list_num_cluster):
        kmeans = KMeans(n_clusters=k).fit(embeddings)
        currcost1=0
        currcost2=0
        #Finding cost function for each clusters formed from the above fitting of kmeans
        for i in range(0,k):
            current_nodes=np.array(nodes).reshape(len(nodes),)[kmeans.labels_==i].tolist
()
            current_graph=nx.Graph()
            #Creating the graph Cluster
            for j in current_nodes:
                current_graph.add_nodes_from(nx.ego_graph(B,j).nodes)
                current_graph.add_edges_from(nx.ego_graph(B,j).edges())
            currcost1=currcost1+cost1(current_graph,k)
            currcost2=currcost2+cost2(current_graph,k)
        all_cost.append(currcost1*currcost2)
    kmax=list_num_cluster[all_cost.index(max(all_cost))]
    return kmax,all_cost
```

In [ ]:

```
actor_nodes,movie_nodes,actor_embeddings,movie_embeddings=data_split(node_ids,node_target
s,node_embeddings)
```

In [ ]:

```
list_num_cluster=[3, 5, 10, 30, 50, 100, 200, 500]
kmax_actor,score_actor=Kmeans_clustering(actor_nodes,list_num_cluster,actor_embeddings)
print("The value of k is ",kmax_actor)
```

```
The value of k is  3
```

Fit the clustering algorithm with the opimal number_of_clusters and get the cluster number for each node

In [ ]:

```
kmeans = KMeans(n_clusters=3, random_state=0).fit(actor_embeddings)
```

Convert the d-dimensional dense vectors of nodes into 2-dimensional using (TSNE)

In [ ]:

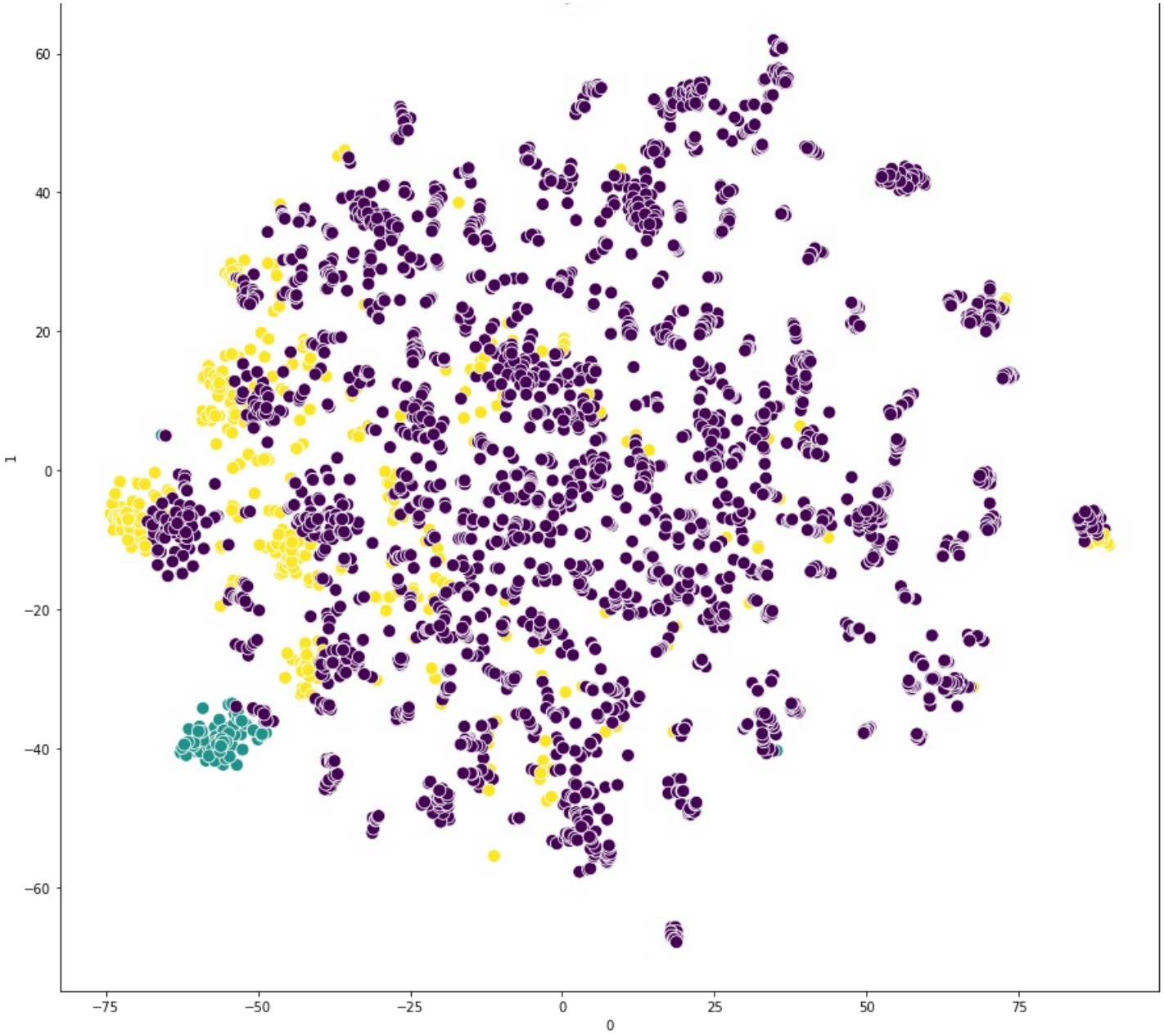
```
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2,n_jobs=-1)
actor_tsne_2d = tsne.fit_transform(actor_embeddings)
```

In [ ]:

```
import pandas as pd
df = pd.DataFrame(actor_tsne_2d)
df['labels'] = kmeans.labels_
```

In [ ]:

```
import seaborn as sns
plt.figure(figsize = (15,15))
sns.scatterplot(df.iloc[:,0],df.iloc[:,1],hue=df.labels,palette = 'viridis',s=100, alpha
='auto').set_title('Number of clusters =3', fontsize=15)
plt.legend()
plt.show()
```



Task 2

Fit the clustering algorithm with the opimal number_of_clusters and get the cluster number for each node

In [ ]:

```
list_num_cluster=[3, 5, 10, 30, 50, 100, 200, 500]
kmax_movie,score_movie=Kmeans_clustering(movie_nodes,list_num_cluster,movie_embeddings)
print("The value of k is ",kmax_movie)
```

```
The value of k is  50
```

In [ ]:

```
kmeans = KMeans(n_clusters=50, random_state=0).fit(movie_embeddings)
```

Convert the d-dimensional dense vectors of nodes into 2-dimensional using (TSNE)

In [ ]:

```
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2,n_jobs=-1)
movie_tsne_2d = tsne.fit_transform(movie_embeddings)
```

In [ ]:

```
import pandas as pd
df = pd.DataFrame(movie_tsne_2d)
df['labels'] = kmeans.labels_
```

Plot the 2d scatter plot

In [ ]:

```
import seaborn as sns
plt.figure(figsize = (15,15))
sns.scatterplot(df.iloc[:,0],df.iloc[:,1],hue=df.labels,palette = 'viridis',s=100, alpha
='auto').set_title('Number of clusters =3', fontsize=15)
plt.legend()
plt.show()
```