Clustering Assignment

There will be some functions that start with the word "grader" ex: grader_actors(), grader_movies those function definition.

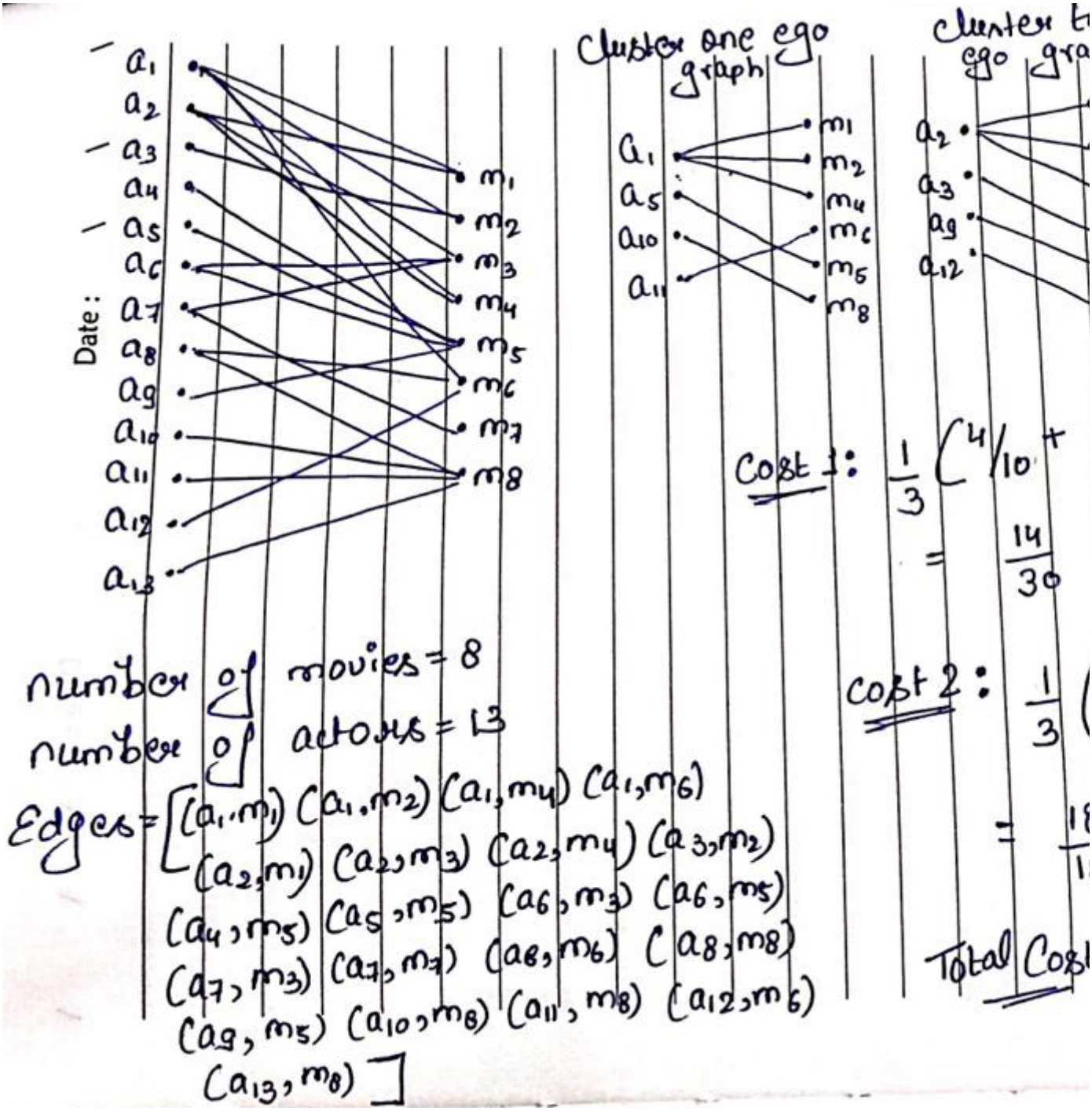
Every Grader function has to return True.

Please check clustering assignment helper functions notebook before attempting this assignme

- 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 vecto Clustering_Assignment_Reference.ipynb]
- Split the dense representation into actor nodes, movies nodes.(Write you code in def data_sp

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 = $\frac{1}{N}\sum_{\text{each cluster i}} \frac{(\text{number of nodes in the largest connected component in the graph with the act}}{(\text{total number of nodes in that clust}}$
   where N= number of clusters
   (Write your code in def cost1())
5. Cost2 = $\frac{1}{N}\sum_{\text{each cluster i}} \frac{(\text{sum of degress of actor nodes in the graph with the actor nodes and its movie}}{(\text{number of unique movie nodes in the graph with the actor nodes and its movi}}$
   clusters
   (Write your code in def cost2())
6. Fit the clustering algorithm with the opimal number_of_clusters and get the cluster number f
7. Convert the d-dimensional dense vectors of nodes into 2-dimensional using dimensionality r
8. Plot the 2d scatter plot, with the node vectors after step e and give colors to nodes such that



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 = $\frac{1}{N}\sum_{\text{each cluster i}} \frac{(\text{number of nodes in the largest connected component in the graph with the mo}}{(\text{total number of nodes in that clust}}$

   where N= number of clusters

   (Write your code in def cost1())

4. Cost2 = $\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(sum of degress of movie nodes in the graph with the movie nodes and its actor}}{\text{(number of unique actor nodes in the graph with the movie nodes and its actor}}$ 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
    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
        cost2=cost2(graph1)+cost2(graph2)+cost2(graph3)
    computer the metric Cost = Cost1*Cost2
return number_of_clusters which have maximum Cost
```

```
!pip install pynput
```

```
Collecting pynput
  Downloading https://files.pythonhosted.org/packages/33/0a/ea13c055a90b1aff5945e7eb3
     |████████████████████████████████| 92kB 3.3MB/s
Collecting python-xlib>=0.17; "linux" in sys_platform
  Downloading https://files.pythonhosted.org/packages/33/10/2eb938852a9bdf6745808f141
     |████████████████████████████████| 184kB 5.7MB/s
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from py
Installing collected packages: python-xlib, pynput
Successfully installed pynput-1.6.8 python-xlib-0.27
```

```
!pip install stellargraph
```

```
Collecting stellargraph
  Downloading https://files.pythonhosted.org/packages/62/5f/62ae75e9c157a012cc560984e
     |████████████████████████████████| 419kB 4.7MB/s
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: matplotlib>=2.2 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: numpy>=1.14 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: gensim>=3.4.0 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: tensorflow>=2.1.0 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: wheel>=0.26; python_version >= "3" in /usr/local/lib/p
Requirement already satisfied: tensorboard<2.3.0,>=2.2.0 in /usr/local/lib/python3.6/
Requirement already satisfied: astunparse==1.6.3 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: grpcio>=1.8.6 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: tensorflow-estimator<2.3.0,>=2.2.0 in /usr/local/lib/p
Requirement already satisfied: wrapt>=1.11.1 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: absl-py>=0.7.0 in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: gast==0.3.3 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: google-pasta>=0.1.8 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: h5py<2.11.0,>=2.10.0 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: protobuf>=3.8.0 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: keras-preprocessing>=1.1.0 in /usr/local/lib/python3.6
Requirement already satisfied: boto3 in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (fr
Requirement already satisfied: boto in /usr/local/lib/python3.6/dist-packages (from s
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python
Requirement already satisfied: google-auth<2,>=1.6.3 in /usr/local/lib/python3.6/dist
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/pyt
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in /usr/local/lib/python3.6/dis
Requirement already satisfied: botocore<1.17.0,>=1.16.23 in /usr/local/lib/python3.6/
Requirement already satisfied: s3transfer<0.4.0,>=0.3.0 in /usr/local/lib/python3.6/d
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.6/dist
```

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

```
Collecting networkx==2.3
  Downloading https://files.pythonhosted.org/packages/85/08/f20aef11d4c343b557e5de6b9
     |████████████████████████████████| 1.8MB 4.6MB/s
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.6/dist-pack
Building wheels for collected packages: networkx
  Building wheel for networkx (setup.py) ... done
  Created wheel for networkx: filename=networkx-2.3-py2.py3-none-any.whl size=1556408
  Stored in directory: /root/.cache/pip/wheels/de/63/64/3699be2a9d0ccdb37c7f16329acf3
```

```
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
# you need to have tensorflow
from stellargraph.data import UniformRandomMetaPathWalk
from stellargraph import StellarGraph
```

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

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

```
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]
```

```
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)
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()
```

```
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
```

```
# 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
```

```
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
```

```
# 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 embeddin
node_targets = [ A.node[node_id]['label'] for node_id in node_ids]


print(len(node_targets[1]))
print(len(node_embeddings))
```

```
5
4703
```

```
type(node_targets[11])
```

```
str
```

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

```
['a973', 'a967', 'a964', 'a1731', 'a969', 'a970', 'a1028', 'a1057', 'a965', 'a1003', 'm1094',
```

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

```
['actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'm
```

```
def data_split(node_ids,node_targets,node_embeddings):
    '''In this function, we will split the node embeddings into actor_embeddings , movie_e
    actor_nodes,movie_nodes=[],[]
    actor_embeddings,movie_embeddings=[],[]
    for i in range(0,4703):

      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])



    # 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 e
    # By using node_ids and node_targets, we can extract actor_nodes and movie nodes

    return actor_nodes,movie_nodes,actor_embeddings,movie_embeddings



actor_nodes,movie_nodes,actor_embeddings,movie_embeddings=data_split(node_ids,node_targets
```

Graded function - 1

```
def grader_actors(data):
```

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

```
True
```

Graded function - 2

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

```
True
```

Calculating cost1

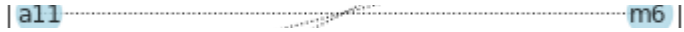
Cost1 = $\frac{1}{N}\sum_{\text{each cluster i}} \frac{(\text{number of nodes in the largest connected component in the graph with the actor nod}}{(\text{total number of nodes in that cluster i})}$

number of clusters

```
def cost1(graph,number_of_clusters):
    '''In this function, we will calculate cost1'''
    cost1=(1/number_of_clusters)*(len(max(nx.connected_component_subgraphs(graph), key=len
    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
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'),
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,
```

Graded 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

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

```
def cost2(graph,number_of_clusters):
    '''In this function, we will calculate cost1'''
    cont=[]
    for i in graph.nodes():
      if 'm' in i:
         cont.append(i)
    cost2=(1/number_of_clusters)*(graph.number_of_edges()/len(cont))

    return cost2
```

```
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

```
matrics_cost=0
for k in [3, 5, 10,15,18,25,50,120]:
  algo = KMeans(n_clusters=k,init='k-means++', random_state=0)
  algo.fit(actor_embeddings)
  clusture_no_for_every_datapoint=algo.labels_
  unique_clusture_no = np.unique(clusture_no_for_every_datapoint)
  dict_actor_nodes=dict(zip(actor_nodes,clusture_no_for_every_datapoint))
  k_no_of_cluster=[]
  for n in unique_clusture_no:
```

```
      clusture=[]
      for actor_node,clusture_no in dict_actor_nodes.items():
        if clusture_no==n:
          clusture.append(actor_node)
      k_no_of_cluster.append(clusture)
  cost1_new=0
  cost2_new=0
  for every_cluster in k_no_of_cluster:
    new= nx.Graph()
    for node_in_cluster  in every_cluster:
      ego = nx.ego_graph(B,node_in_cluster)
      new.add_nodes_from(ego.nodes())
      new.add_edges_from(ego.edges())
    temp1=cost1(new,k)
    temp2=cost2(new,k)
    cost1_new=cost1_new+temp1
    cost2_new=cost2_new+temp2
  matrics_cost=cost1_new*cost2_new
  print("for k = ",k, "matric_cost = ",matrics_cost)
```

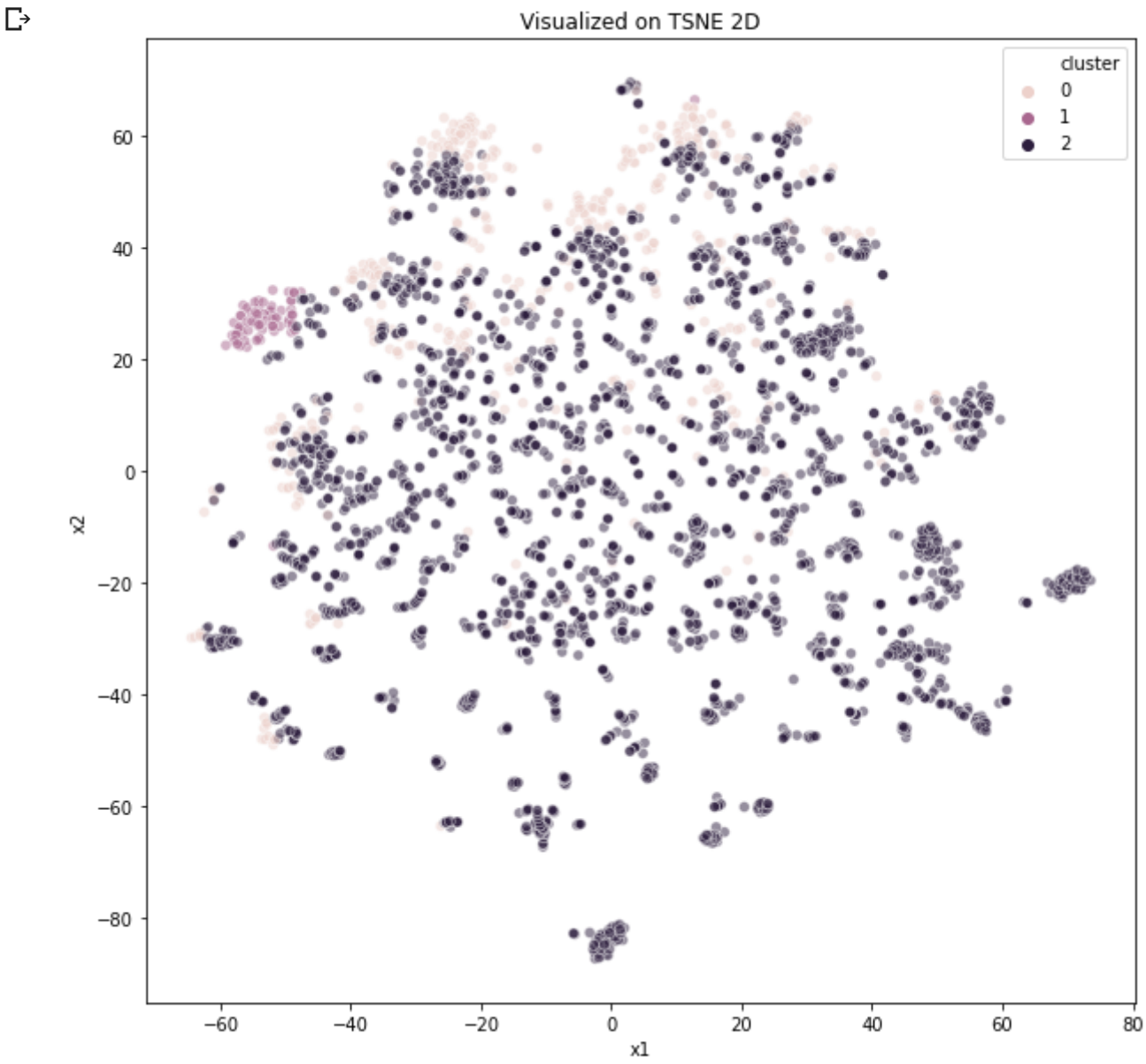
```
for k =  3 matric_cost =  3.8176562439630652
for k =  5 matric_cost =  3.1119483680181865
for k =  10 matric_cost =  2.2580616101021693
for k =  15 matric_cost =  2.0196895329388904
for k =  18 matric_cost =  1.9444747154866706
for k =  25 matric_cost =  1.7724891851426308
for k =  50 matric_cost =  1.5408944319606732
for k =  120 matric_cost =  1.6782105987944902
```

Displaying similar actor clusters

```
#https://www.kaggle.com/aussie84/clustering-with-kmeans-pca-tsne
algo = KMeans(n_clusters=3,init='k-means++', random_state=0)
algo.fit(actor_embeddings)
clusture_no_for_every_datapoint= algo.fit_predict(actor_embeddings)


actor_labels = pd.DataFrame(actor_embeddings)
actor_labels['cluster'] = clusture_no_for_every_datapoint


import seaborn as sns
from sklearn.manifold import TSNE
X = actor_labels.iloc[:,:-1]
Xtsne = TSNE(n_components=2).fit_transform(X)
dftsne = pd.DataFrame(Xtsne)
dftsne['cluster'] = clusture_no_for_every_datapoint
dftsne.columns = ['x1','x2','cluster']
plt.figure(figsize=(10,10))
sns.scatterplot(data=dftsne,x='x1',y='x2',hue='cluster',legend="full",alpha=0.5,).set_titl
plt.show()
```



Grouping similar movies

```
def cost1(graph,number_of_clusters):
    '''In this function, we will calculate cost1'''
    cost1=(1/number_of_clusters)*(len(max(nx.connected_component_subgraphs(graph), key=len
    return cost1


def cost2(graph,number_of_clusters):
    '''In this function, we will calculate cost1'''
    cont=[]
    for i in graph.nodes():
      if 'a' in i:
         cont.append(i)
    cost2=(1/number_of_clusters)*(graph.number_of_edges()/len(cont))

    return cost2


matrics_cost=0
for k in [2, 5, 10,15,18,25,50,120]:
```

```
for k in [3, 5, 10,15,18,25,50,120]:
  algo = KMeans(n_clusters=k,init='k-means++', random_state=0)
  algo.fit(movie_embeddings)
  clusture_no_for_every_datapoint=algo.labels_
  unique_clusture_no = np.unique(clusture_no_for_every_datapoint)
  dict_movie_nodes=dict(zip(movie_nodes,clusture_no_for_every_datapoint))
  k_no_of_cluster=[]
  for n in unique_clusture_no:
      clusture=[]
      for movie_node,clusture_no in dict_movie_nodes.items():
        if clusture_no==n:
          clusture.append(movie_node)
      k_no_of_cluster.append(clusture)
  cost1_new=0
  cost2_new=0
  for every_cluster in k_no_of_cluster:
    new= nx.Graph()
    for node_in_cluster  in every_cluster:
      ego = nx.ego_graph(B,node_in_cluster)
      new.add_nodes_from(ego.nodes())
      new.add_edges_from(ego.edges())
    temp1=cost1(new,k)
    temp2=cost2(new,k)
    cost1_new=cost1_new+temp1
    cost2_new=cost2_new+temp2
  matrics_cost=cost1_new*cost2_new
  print("for k = ",k, "matric_cost = ",matrics_cost)
```

```
for k =  3 matric_cost =  2.7411597198287625
for k =  5 matric_cost =  2.621507378052418
for k =  10 matric_cost =  2.072028966046183
for k =  15 matric_cost =  2.418753176778979
for k =  18 matric_cost =  2.371243298961502
for k =  25 matric_cost =  2.1240471260932927
for k =  50 matric_cost =  1.903053418897081
for k =  120 matric_cost =  1.5207020626611851
```

Displaying similar movie clusters

```
algo = KMeans(n_clusters=3,init='k-means++', random_state=0)
algo.fit(actor_embeddings)
clusture_no_for_every_datapoint= algo.fit_predict(movie_embeddings)
```

```
actor_labels = pd.DataFrame(movie_embeddings)
actor_labels['cluster'] = clusture_no_for_every_datapoint
```

```
import seaborn as sns
from sklearn.manifold import TSNE
X = actor_labels.iloc[:,:-1]
Xtsne = TSNE(n_components=2).fit_transform(X)
dftsne = pd.DataFrame(Xtsne)
```

```
dftsne = pd.DataFrame(Xtsne)
dftsne['cluster'] = clusture_no_for_every_datapoint
dftsne.columns = ['x1','x2','cluster']
plt.figure(figsize=(10,10))
sns.scatterplot(data=dftsne,x='x1',y='x2',hue='cluster',legend="full",alpha=0.5,).set_titl
sns.palplot(sns.color_palette("husl", 8))
plt.show()
```