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.

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())

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

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: networkx==2.3 in /usr/local/lib/python3.7/dist-packages (2.3)
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.7/dist-packages (from networkx==2.3) (4.4.2)
```

```
!pip install stellargraph
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: stellargraph in /usr/local/lib/python3.7/dist-packages (1.2.1)
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.7/dist-packages (from stellargraph) (1.0.2)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.7/dist-packages (from stellargraph) (2.3)
Requirement already satisfied: numpy>=1.14 in /usr/local/lib/python3.7/dist-packages (from stellargraph) (1.21.6)
Requirement already satisfied: matplotlib>=2.2 in /usr/local/lib/python3.7/dist-packages (from stellargraph) (3.2.2)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.7/dist-packages (from stellargraph) (1.3.5)
Requirement already satisfied: tensorflow>=2.1.0 in /usr/local/lib/python3.7/dist-packages (from stellargraph) (2.8.0+zzzcolab20220506162203)
```

```
Requirement already satisfied: scipy>=1.1.0 in
/usr/local/lib/python3.7/dist-packages (from stellargraph) (1.4.1)
Requirement already satisfied: gensim>=3.4.0 in
/usr/local/lib/python3.7/dist-packages (from stellargraph) (3.6.0)
Requirement already satisfied: smart-open>=1.2.1 in
/usr/local/lib/python3.7/dist-packages (from gensim>=3.4.0-
>stellargraph) (6.0.0)
Requirement already satisfied: six>=1.5.0 in
/usr/local/lib/python3.7/dist-packages (from gensim>=3.4.0-
>stellargraph) (1.15.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib>=2.2-
>stellargraph) (1.4.2)
Requirement already satisfied: python-dateutil>=2.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib>=2.2-
>stellargraph) (2.8.2)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.7/dist-packages (from matplotlib>=2.2-
>stellargraph) (0.11.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!
=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from
matplotlib>=2.2->stellargraph) (3.0.9)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.7/dist-packages (from kiwisolver>=1.0.1-
>matplotlib>=2.2->stellargraph) (4.2.0)
Requirement already satisfied: decorator>=4.3.0 in
/usr/local/lib/python3.7/dist-packages (from networkx>=2.2-
>stellargraph) (4.4.2)
Requirement already satisfied: pytz>=2017.3 in
/usr/local/lib/python3.7/dist-packages (from pandas>=0.24-
>stellargraph) (2022.1)
Requirement already satisfied: joblib>=0.11 in
/usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20-
>stellargraph) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20-
>stellargraph) (3.1.0)
Requirement already satisfied: wrapt>=1.11.0 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0-
>stellargraph) (1.14.1)
Requirement already satisfied: libclang>=9.0.1 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0-
>stellargraph) (14.0.1)
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0-
>stellargraph) (1.1.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0-
>stellargraph) (0.26.0)
Requirement already satisfied: astunparse>=1.6.0 in
```

```
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0-
>stellargraph) (1.6.3)
Requirement already satisfied: opt-einsum>=2.3.2 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0-
>stellargraph) (3.3.0)
Requirement already satisfied: gast>=0.2.1 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0-
>stellargraph) (0.5.3)
Requirement already satisfied: keras<2.9,>=2.8.0rc0 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0-
>stellargraph) (2.8.0)
Requirement already satisfied: google-pasta>=0.1.1 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0-
>stellargraph) (0.2.0)
Requirement already satisfied: absl-py>=0.4.0 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0-
>stellargraph) (1.0.0)
Requirement already satisfied: flatbuffers>=1.12 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0-
>stellargraph) (2.0)
Requirement already satisfied: tensorboard<2.9,>=2.8 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0-
>stellargraph) (2.8.0)
Requirement already satisfied: protobuf>=3.9.2 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0-
>stellargraph) (3.17.3)
Requirement already satisfied: keras-preprocessing>=1.1.1 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0-
>stellargraph) (1.1.2)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0-
>stellargraph) (57.4.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0-
>stellargraph) (1.46.1)
Requirement already satisfied: h5py>=2.9.0 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0-
>stellargraph) (3.1.0)
Requirement already satisfied: tf-estimator-
nightly==2.8.0.dev2021122109 in /usr/local/lib/python3.7/dist-packages
(from tensorflow>=2.1.0->stellargraph) (2.8.0.dev2021122109)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/usr/local/lib/python3.7/dist-packages (from astunparse>=1.6.0-
>tensorflow>=2.1.0->stellargraph) (0.37.1)
Requirement already satisfied: cached-property in
/usr/local/lib/python3.7/dist-packages (from h5py>=2.9.0-
>tensorflow>=2.1.0->stellargraph) (1.5.2)
Requirement already satisfied: google-auth<3,>=1.6.3 in
/usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8-
>tensorflow>=2.1.0->stellargraph) (1.35.0)
```

```
Requirement already satisfied: werkzeug>=0.11.15 in
/usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8-
>tensorflow>=2.1.0->stellargraph) (1.0.1)
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0
in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8-
>tensorflow>=2.1.0->stellargraph) (0.6.1)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8-
>tensorflow>=2.1.0->stellargraph) (3.3.7)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in
/usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8-
>tensorflow>=2.1.0->stellargraph) (0.4.6)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8-
>tensorflow>=2.1.0->stellargraph) (2.23.0)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in
/usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8-
>tensorflow>=2.1.0->stellargraph) (1.8.1)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (0.2.8)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in
/usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (4.2.4)
Requirement already satisfied: rsa<5,>=3.1.4 in
/usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (4.8)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/usr/local/lib/python3.7/dist-packages (from google-auth-
oauthlib<0.5,>=0.4.1->tensorboard<2.9,>=2.8->tensorflow>=2.1.0-
>stellargraph) (1.3.1)
Requirement already satisfied: importlib-metadata>=4.4 in
/usr/local/lib/python3.7/dist-packages (from markdown>=2.6.8-
>tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (4.11.3)
Requirement already satisfied: zipp>=0.5 in
/usr/local/lib/python3.7/dist-packages (from importlib-metadata>=4.4-
>markdown>=2.6.8->tensorboard<2.9,>=2.8->tensorflow>=2.1.0-
>stellargraph) (3.8.0)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in
/usr/local/lib/python3.7/dist-packages (from pyasn1-modules>=0.2.1-
>google-auth<3,>=1.6.3->tensorboard<2.9,>=2.8->tensorflow>=2.1.0-
>stellargraph) (0.4.8)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1
in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (2022.5.18.1)
Requirement already satisfied: idna<3,>=2.5 in
/usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0-
```

```
>tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (3.0.4)
Requirement already satisfied: oauthlib>=3.0.0 in
/usr/local/lib/python3.7/dist-packages (from requests-oauthlib>=0.7.0-
>google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.9,>=2.8-
>tensorflow>=2.1.0->stellargraph) (3.2.0)
```

```
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
from datetime import datetime
```

```
URL =
'https://drive.google.com/file/d/1F83qXMeHv3CIESBSj6jnXi0wdG2Kz3Ux/
view?usp=sharing'
path = 'https://drive.google.com/uc?
export=download&id='+URL.split('/')[-2]
```

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

	movie	actor
0	m1	a1
1	m2	a1
2	m2	a2
3	m3	a1
4	m3	a3

```
data.describe()
```

	movie	actor
count	9650	9650
unique	1292	3411
top	m1094	a973
freq	77	197

```
data[data.duplicated()]
```

```
Empty DataFrame
Columns: [movie, actor]
Index: []
```

There are no duplicated rows in the dataset

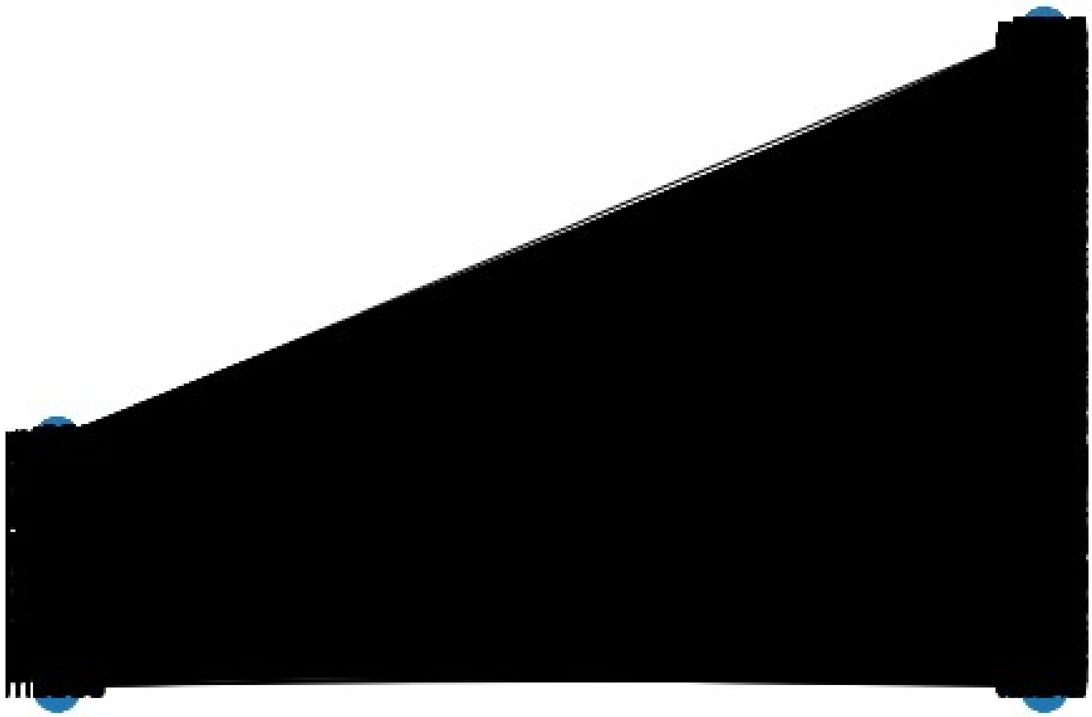
```
# edge/link/relationship: connects two nodes in a graph
edges = [tuple(x) for x in data.values.tolist()]
print("Number of edges = ",len(edges))
```

```
Number of edges =  9650
```

```
B = nx.Graph() #Creating an empty graph with no nodes and edges

#nodes/entities/vertex : objects that are connected by edges
B.add_nodes_from(data['movie'].unique(), bipartite=0, label='movie')
#adding unique values from dataset as nodes to graph by function
"add_nodes_from"
B.add_nodes_from(data['actor'].unique(), bipartite=1, label='actor')


B.add_edges_from(edges, label='acted') # adding edges to the graph by
function "add_edges_from"

# info about bipartite
graph:https://en.wikipedia.org/wiki/Bipartite_graph#:~:text=In%20the
%20mathematical%20field%20of,the%20parts%20of%20the%20graph.

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

print("number of nodes", A.number_of_nodes()) #contains  unique values
of movie,actor
print("number of edges", A.number_of_edges()) # number of connections
between movie and actor
```

```
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()
```



```python
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
```

```python
# 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)))
print('--'*15)
print("Sample random walk =")
print(walks[1])
```

```
Number of random walks: 4703
------------------------------
Sample random walk =
['m2', 'a2', 'm2', 'a2', 'm2', 'a2', 'm2', 'a1', 'm4', 'a10', 'm4',
'a1', 'm2', 'a1', 'm5', 'a13', 'm1206', 'a1381', 'm1231', 'a3446',
'm1231', 'a813', 'm176', 'a768', 'm181', 'a830', 'm185', 'a773',
'm188', 'a768', 'm212', 'a18', 'm5', 'a12', 'm1200', 'a969', 'm1195',
'a2271', 'm941', 'a1244', 'm593', 'a1629', 'm593', 'a1329', 'm1225',
'a3431', 'm1230', 'a3440', 'm1230', 'a3432', 'm1230', 'a3438',
'm1230', 'a1622', 'm1172', 'a1622', 'm1154', 'a1037', 'm356', 'a966',
'm515', 'a973', 'm755', 'a973', 'm681', 'a967', 'm719', 'a964',
'm792', 'a973', 'm1334', 'a1028', 'm693', 'a1016', 'm536', 'a1020',
'm1324', 'a205', 'm1325', 'a960', 'm294', 'a1003', 'm312', 'a977',
'm702', 'a1028', 'm306', 'a1025', 'm1349', 'a204', 'm1323', 'a3614',
'm1323', 'a3373', 'm1207', 'a3373', 'm1323', 'a3373', 'm1219',
'a3373']
```

```
from gensim.models import Word2Vec
model = Word2Vec(walks, size=128, window=5) #conveting each walk into
vector format with 128 dimension

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='')
```

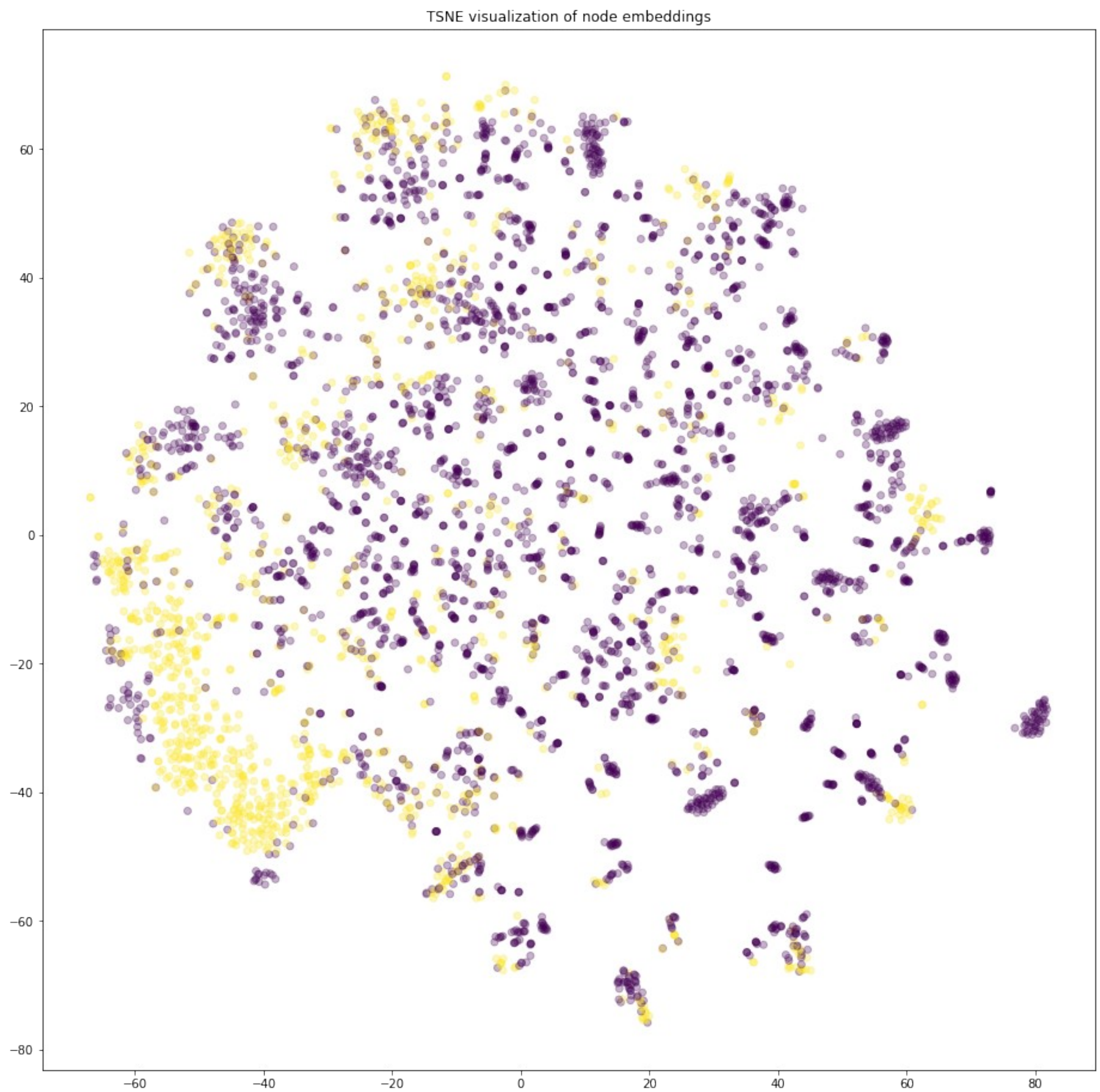
```
['a973', 'a967', 'a964', 'a1731', 'a969', 'a970', 'a1028', 'a1057', 'a965', 'a1003', 'm1094', 'a966', 'm67', 'a988', 'm1111']
```

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

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

```
from sklearn.manifold import TSNE
transform = TSNE #PCA

trans = transform(n_components=2)
node_embeddings_2d = trans.fit_transform(node_embeddings)

import numpy as np
```

```python
# draw the points

label_map = { l: i for i, l in enumerate(np.unique(node_targets))}
node_colours = [ label_map[target] for target in node_targets]

plt.figure(figsize=(20,16))
plt.axes().set(aspect="equal")
plt.scatter(node_embeddings_2d[:,0],
            node_embeddings_2d[:,1],
            c=node_colours, alpha=0.3)
plt.title('{} visualization of node 
embeddings'.format(transform.__name__))

plt.show()
```




```python
# split the node_embeddings into actor_embeddings,movie_embeddings 
based on node_ids
```

```python
# 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
start = datetime.now()
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=[],[]

    for index,node in enumerate(node_ids):
      if 'a' in node:
        actor_nodes.append(node)
        actor_embeddings.append(node_embeddings[index])
      else :
        movie_nodes.append(node)
        movie_embeddings.append(node_embeddings[index])
    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)

print("Time taken = ", datetime.now()-start)
```

```
Time taken =  0:00:00.004010
```

Grader function - 1

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

```
True
```

Grader function - 2

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

```
True
```

Calculating cost1

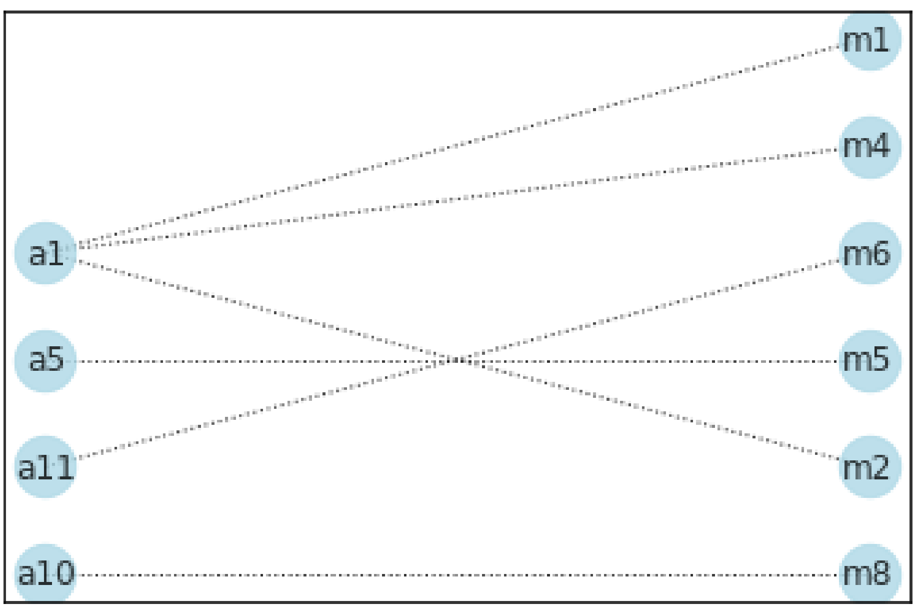
Cost1 =

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

where N= number of clusters

```
def cost1(graph,number_of_clusters):
    '''In this function, we will calculate cost1'''
    max_nodes = len(max(nx.connected_components(graph),key=len))
    total_nodes = graph.number_of_nodes()
    cost1= max_nodes/(total_nodes*number_of_clusters)
    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 =
$$\frac{1}{N}\sum_{\text{each cluster i}}\frac{\text{(sum of degress of actor nodes in the graph with the actor nodes and its movie neighbours in cluster i)}}{\text{(number of unique movie nodes in the graph with the actor nodes and its movie neighbours in cluster}}$$
where N= number of clusters

```python
def cost2(graph,number_of_clusters):
    '''In this function, we will calculate cost1'''
    num=sum([graph.degree(i) for i in graph.nodes if "a" in i ])
    den=len([j for j in graph.nodes if "m" in j])
    cost2= num/(den*number_of_clusters)

    return cost2
```

Grader function - 4

```python
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
```

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

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
</pre>
```

```
#info about KMeans : https://stackabuse.com/k-means-clustering-with-
scikit-learn/

start = datetime.now()
final_cost = dict() #creating an empty dictionary to store
cluster_number as "key" and total_cost(cost1*cost2) as "value"

for k in [3, 5, 10, 30, 50, 100, 200, 500]: #range of cluster values
   algo = KMeans(n_clusters=k,random_state=0)
   algo.fit(actor_embeddings)               # matrix of size N*d where
N number of actor nodes and d is dimension from gensim

   cluster_cost1 = 0
   cluster_cost2 = 0

   for cluster_num in range(0,k): # itering through each "k-cluster"
range
   # extracting particular actor_node data belonging to a label
     cluster_nodes = np.array(actor_nodes).reshape(len(actor_nodes),)
[algo.labels_== cluster_num].tolist()
     cluster_graph = nx.Graph() #empty graph

     for each_node in cluster_nodes: # iterating through each node in
cluster_nodes
```

```
        cluster_graph.add_nodes_from(nx.ego_graph(B,each_node).nodes)
#adding nodes to empty graph
        cluster_graph.add_edges_from(nx.ego_graph(B,each_node).edges())
#adding edges to empty graph

      cluster_cost1 = cluster_cost1 + cost1(cluster_graph, k) # after
each cluster value calculating costs
      cluster_cost2 = cluster_cost2 + cost2(cluster_graph ,k)

    print(f"Total cost for cluster {k} is
{cluster_cost1*cluster_cost2}")

    final_cost[str(k)] = cluster_cost1*cluster_cost2

print('---'*15)
kmax_cluster = max(final_cost, key=final_cost.get)
print("Number of cluster with maximum cost is ",kmax_cluster)
print("maximum cost is ",final_cost[kmax_cluster])
print('---'*15)
print(" Total time taken = " , datetime.now()-start)
```

```
Total cost for cluster 3 is 3.8127442267567453
Total cost for cluster 5 is 2.9060869609140974
Total cost for cluster 10 is 2.2434608266542537
Total cost for cluster 30 is 1.771617621193852
Total cost for cluster 50 is 1.5117437436345245
Total cost for cluster 100 is 1.6945709313621242
Total cost for cluster 200 is 1.6118779719372072
Total cost for cluster 500 is 1.8791949082665356
---------------------------------------------
Number of cluster with maximum cost is  3
maximum cost is  3.8127442267567453
---------------------------------------------
 Total time taken =  0:00:38.672811
```

Grouping similar actors

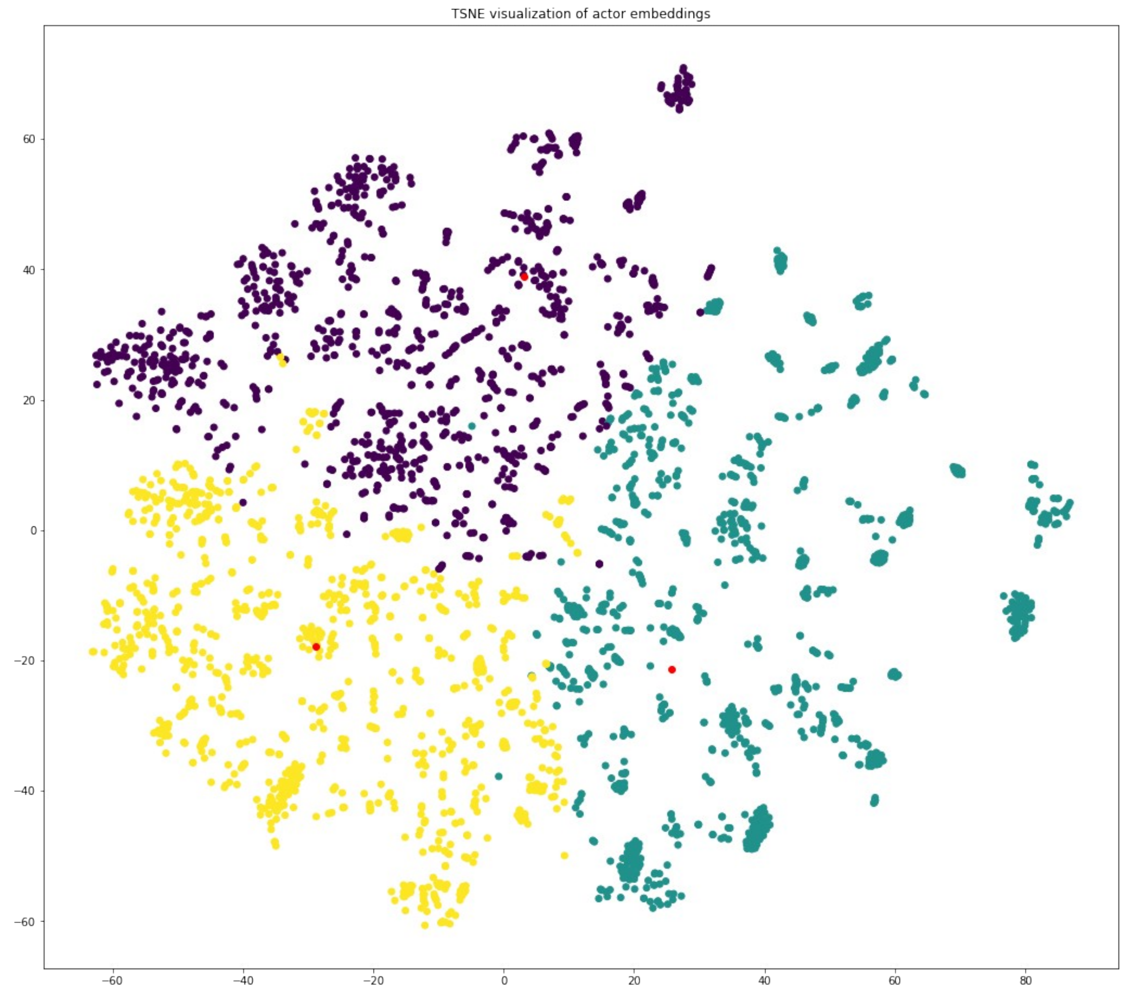
```
kmeans = KMeans(n_clusters=int(kmax_cluster),
random_state=0).fit(actor_embeddings)
actor_nodes=np.array(actor_nodes)
actor_data=np.vstack((actor_nodes,kmeans.labels_))
actors_df=pd.DataFrame(actor_data.T,columns=["actor_node","labels"])
print(actors_df.head())
print()
print('****'*15)
print( )
print(actors_df.describe())
```

```
  actor_node labels
0       a973      1
```

```
1       a967      1
2       a964      1
3       a970      1
4      a1731      2

************************************************************

       actor_node labels
count        3411   3411
unique       3411      3
top          a973      0
freq            1   2992
```

Displaying similar actor clusters

```
from sklearn.manifold import TSNE
transform = TSNE #PCA

trans = transform(n_components=2)
actor_embeddings_2d = trans.fit_transform(actor_embeddings)

import numpy as np
# draw the points

label_map = { l: i for i, l in enumerate(np.unique(kmeans.labels_))}
node_colours = [ label_map[target] for target in kmeans.labels_]

plt.figure(figsize=(20,16))
plt.axes().set(aspect="equal")
plt.scatter(actor_embeddings_2d[:,0],
            actor_embeddings_2d[:,1],
            c=node_colours)
plt.scatter(kmeans.cluster_centers_[:, 0],
            kmeans.cluster_centers_[:, 1],
            c='red')
plt.title('{} visualization of actor
embeddings'.format(transform.__name__))

plt.show()
```



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 =
   $$\frac{1}{N}\sum_{\text{each cluster i}}\frac{(\text{number of nodes in the largest connected component in the graph with the movie nodes and it}}{(\text{total number of nodes in that cluster i})}$$
   where N= number of clusters (Write your code in def cost1())

3. Cost2 =
   $$\frac{1}{N}\sum_{\text{each cluster i}}\frac{(\text{sum of degress of movie nodes in the graph with the movie nodes and its actor neighbours in}}{(\text{number of unique actor nodes in the graph with the movie nodes and its actor neighbours in c}}$$
   where N= number of clusters (Write your code in def cost2())

```
#info about KMeans : https://stackabuse.com/k-means-clustering-with-
scikit-learn/
```

```
start = datetime.now()
final_cost = dict() #creating an empty dictionary to store
cluster_number as "key" and total_cost(cost1*cost2) as "value"

for k in [3, 5, 10, 30, 50, 100, 200, 500]: #range of cluster values
   algo = KMeans(n_clusters=k,random_state=0)
   algo.fit(movie_embeddings)               # matrix of size N*d where
N number of actor nodes and d is dimension from gensim

   cluster_cost1 = 0
   cluster_cost2 = 0

   for cluster_num in range(0,k): # itering through each "k-cluster"
range
   # extracting particular actor_node data belonging to a label
     cluster_nodes = np.array(movie_nodes).reshape(len(movie_nodes),)
[algo.labels_== cluster_num].tolist()
     cluster_graph = nx.Graph() #empty graph

     for each_node in cluster_nodes: # iterating through each node in
cluster_nodes
       cluster_graph.add_nodes_from(nx.ego_graph(B,each_node).nodes)
#adding nodes to empty graph
       cluster_graph.add_edges_from(nx.ego_graph(B,each_node).edges())
#adding edges to empty graph

     cluster_cost1 = cluster_cost1 + cost1(cluster_graph, k) # after
each cluster value calculating costs
     cluster_cost2 = cluster_cost2 + cost2(cluster_graph ,k)

   print(f"Total cost for cluster {k} is
{cluster_cost1*cluster_cost2}")

   final_cost[str(k)] = cluster_cost1*cluster_cost2
print('---'*15)
kmax_cluster = max(final_cost, key=final_cost.get)
print("Number of cluster with maximum cost is ",kmax_cluster)
print("maximum cost is ",final_cost[kmax_cluster])
print('---'*15)
print(" Total time taken = " , datetime.now()-start)
```

```
Total cost for cluster 3 is 8.714318964836597
Total cost for cluster 5 is 10.635902760986742
Total cost for cluster 10 is 8.846442409438822
Total cost for cluster 30 is 11.79072161639957
Total cost for cluster 50 is 14.459502050176162
Total cost for cluster 100 is 13.911764337352425
Total cost for cluster 200 is 12.72465925420378
```

```
Total cost for cluster 500 is 10.34457543892231
---------------------------------------------
Number of cluster with maximum cost is  50
maximum cost is  14.459502050176162
---------------------------------------------
 Total time taken =  0:00:22.794406
```

Grouping similar movies

```
kmeans = KMeans(n_clusters=int(kmax_cluster),
random_state=0).fit(movie_embeddings)
movie_nodes=np.array(movie_nodes)
movie_data=np.vstack((movie_nodes,kmeans.labels_))
movie_df=pd.DataFrame(movie_data.T,columns=["movie_nodes","labels"])
print(movie_df.head())
print()
print('****'*15)
print( )
print(movie_df.describe())
```

```
  movie_nodes labels
0       m1094     48
1         m67     32
2       m1111     28
3       m1100     43
4       m1095     36

************************************************************


       movie_nodes labels
count         1292   1292
unique        1292     50
top          m1094      1
freq             1    201
```

Displaying similar movie clusters

```
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: i for i, l in enumerate(np.unique(kmeans.labels_))}
node_colours = [ label_map[target] for target in kmeans.labels_]

plt.figure(figsize=(12,12))
plt.axes().set(aspect="equal")
```

```
plt.scatter(movie_embeddings_2d[:,0],
            movie_embeddings_2d[:,1],
            c=node_colours)
plt.scatter(kmeans.cluster_centers_[:, 0],
            kmeans.cluster_centers_[:, 1],
            c='red')
plt.title('{} visualization of movie
embeddings'.format(transform.__name__))

plt.show()
```