HW7 : K-Means

By : 6338125721 Boonthicha Sae-jia

Code

https://colab.research.google.com/drive/1ZUhH71iJu74mgCmT_mm7FX9xMAv9Hq31?usp=sharing

1 Data Preparation and Cleaning

index	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_1_facebook_likes	gross	num_voted_users
0	723.0	178.0	0.0	855.0	1000.0	760505847.0	886204.0
1	302.0	169.0	563.0	1000.0	40000.0	309404152.0	471220.0
2	602.0	148.0	0.0	161.0	11000.0	200074175.0	275868.0
3	813.0	164.0	22000.0	23000.0	27000.0	448130642.0	1144337.0
4	140.1942719807731	107.2010739856802	131.0	645.0097609561753	131.0	48468407.52680933	8.0
5	462.0	132.0	475.0	530.0	640.0	73058679.0	212204.0
6	392.0	156.0	0.0	4000.0	24000.0	336530303.0	383056.0
7	324.0	100.0	15.0	284.0	799.0	200807262.0	294810.0
8	635.0	141.0	0.0	19000.0	26000.0	458991599.0	462669.0
9	375.0	153.0	282.0	10000.0	25000.0	301956980.0	321795.0
10	673.0	183.0	0.0	2000.0	15000.0	330249062.0	371639.0
11	434.0	169.0	0.0	903.0	18000.0	200069408.0	240396.0
12	403.0	106.0	395.0	393.0	451.0	168368427.0	330784.0
13	313.0	151.0	563.0	1000.0	40000.0	423032628.0	522040.0
14	450.0	150.0	563.0	1000.0	40000.0	89289910.0	181792.0
15	733.0	143.0	0.0	748.0	15000.0	291021565.0	548573.0
16	258.0	150.0	80.0	201.0	22000.0	141614023.0	149922.0
17	703.0	173.0	0.0	19000.0	26000.0	623279547.0	995415.0
18	448.0	136.0	252.0	1000.0	40000.0	241063875.0	370704.0
19	451.0	106.0	188.0	718.0	10000.0	179020854.0	268154.0
20	422.0	164.0	0.0	773.0	5000.0	255108370.0	354228.0

a. Select attribute : use only numerical data and some promising features

1 Data Preparation and Cleaning


```
col = df.columns
for i in col:
  if df[i].dtype!='O':
    arr = np.array(df[i])
    imp = SimpleImputer(missing_values=np.nan, strategy='mean')
    imp.fit(arr.reshape(-1, 1))
    SimpleImputer()
    df[i]=imp.transform(arr.reshape(-1, 1))
df
```

	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name	actor_1_facebook_likes	gross
0	James Cameron	723.000000	178.000000	0.000000	855.000000	Joel David Moore	1000.0	7.605058e+08
1	Gore Verbinski	302.000000	169.000000	563.000000	1000.000000	Orlando Bloom	40000.0	3.094042e+08
2	Sam Mendes	602.000000	148.000000	0.000000	161.000000	Rory Kinnear	11000.0	2.000742e+08
3	Christopher Nolan	813.000000	164.000000	22000.000000	23000.000000	Christian Bale	27000.0	4.481306e+08
4	Doug Walker	140.194272	107.201074	131.000000	645.009761	Rob Walker	131.0	4.846841e+07
...	...	...	...	...	...	...	...	...
5038	Scott Smith	1.000000	87.000000	2.000000	318.000000	Daphne Zuniga	637.0	4.846841e+07
5039	NaN	43.000000	43.000000	686.509212	319.000000	Valorie Curry	841.0	4.846841e+07
5040	Benjamin Roberds	13.000000	76.000000	0.000000	0.000000	Maxwell Moody	0.0	4.846841e+07
5041	Daniel Hsia	14.000000	100.000000	0.000000	489.000000	Daniel Henney	946.0	1.044300e+04
5042	Jon Gunn	43.000000	90.000000	16.000000	16.000000	Brian Herzlinger	86.0	8.522200e+04

b. Deal with missing values : use SimpleImputer() to change np.nan into mean of each column

1 Data Preparation and Cleaning


```
9 df_new.imdb_score = df_new.imdb_score.round().astype('int32')
```

imdb_score
8
7
7
8
7
7
6
8
8
8
7
6
7
7
6
7
7
8
7
7

c. Calculate imdb_1 by int (imdb_score)

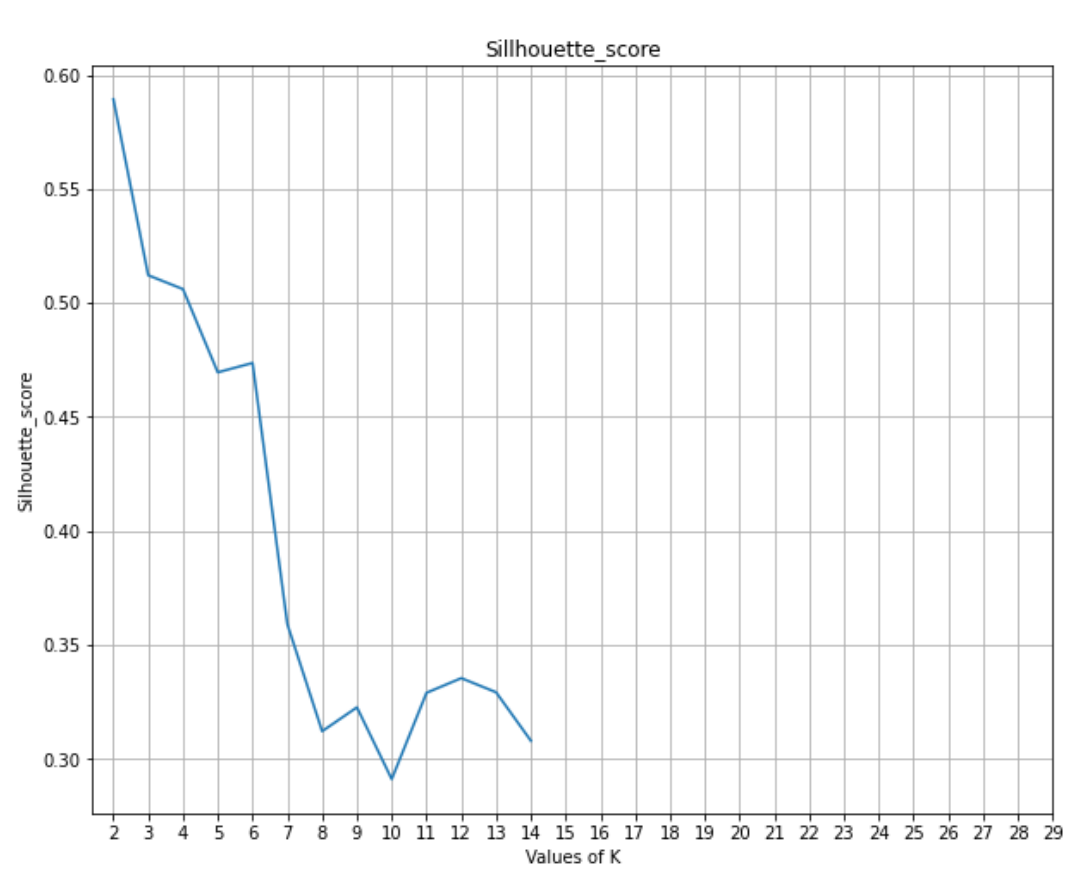
1 Data Preparation and Cleaning

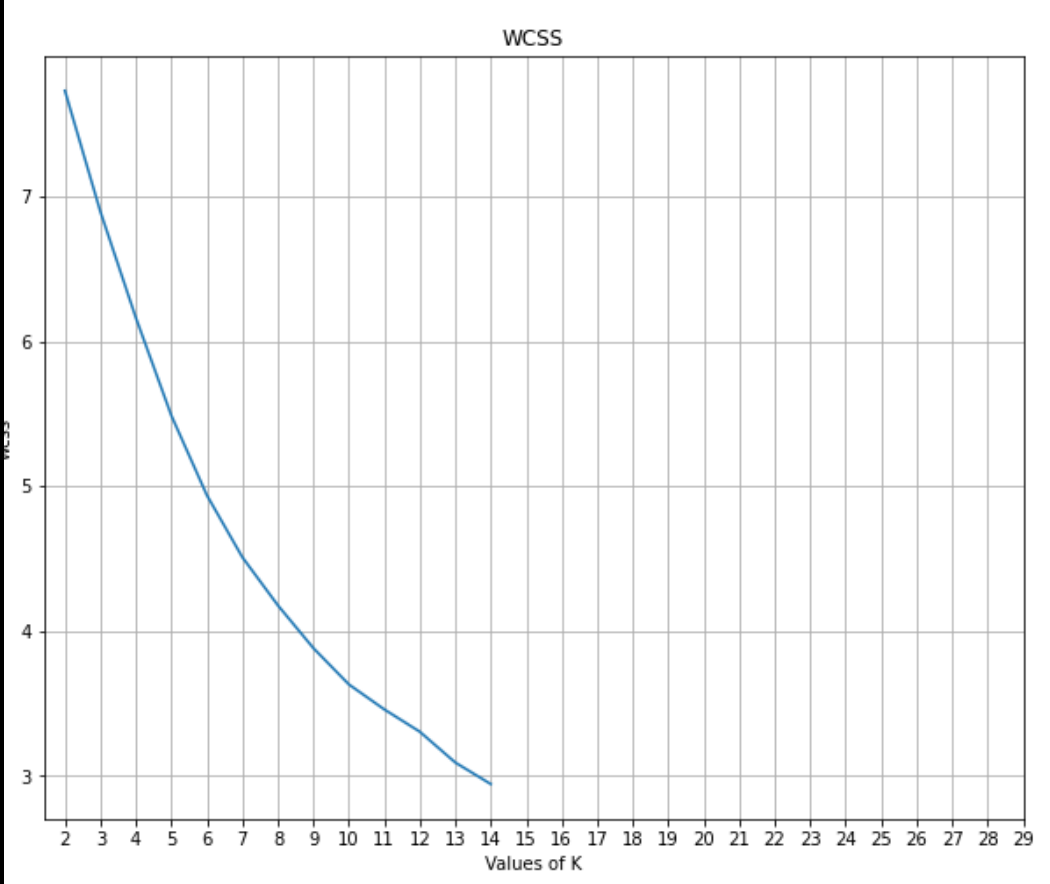

```
for i in x.columns:
  x_array = np.array(x[i])
  normalized_arr = normalize([x_array])
  x[i]=normalized_arr[0]
x
```

index	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_1_facebook_likes	gross	num_voted_users	cast_total_facebook_likes
0	0.05497961668327644	0.02276342106462555	0.0	0.006756714478189327	0.0008596973391478823	0.13586841957725615	0.07713440996385908	0.003306093092290767
1	0.02296520641542114	0.021612461572593923	0.0027649898842652	0.007902590032969973	0.03438789356591529	0.05527669946090649	0.041014570757037516	0.033067770172167686
2	0.045778325371137504	0.018926889424520123	0.0	0.0012723169953081656	0.009456670730626705	0.035744316842147	0.02401130598362214	0.00800192163421638
3	0.06182355236999135	0.02097303963257635	0.10804578588602913	0.18175957075830937	0.023211828156992822	0.08006092567580372	0.09960207728109172	0.07301514117498345
4	0.010646122179334304	0.013683629516376033	0.0006433635432304462	0.005097170571265633	0.0001126203514283725 8	0.008659139025024772	6.963129027976319e-07	9.780126441820018e-05
5	0.0351322031918032	0.01688073921646389	0.0023328067407210836	0.0041883727174740855	0.0005502062970546447	0.013052322071275373	0.018470047903158586	0.001280991386400622
6	0.02980914210213605	0.019949964528548237	0.0	0.03161036013187989	0.020632736139549177	0.06012293079512003	0.03334085441175621	0.031498162466994474
7	0.024638168472173674	0.012788438800351434	7.366758128592896e-05	0.0022443355693634722	0.0006868981739791579	0.03587528673869091	0.025660000859221235	0.0013924711493388503
8	0.048287768456266304	0.01803169870849552	0.0	0.1501492106264295	0.02235213081784494	0.08200129348297791	0.0402702993030597	0.0629210931921288
9	0.028516398694645458	0.019566311364537695	0.0013849505281754643	0.07902590032969972	0.021492433478697056	0.053946222523810705	0.028008751319470497	0.04018264117736025
10	0.05117743019065704	0.023402843004643122	0.0	0.015805180065939946	0.012895460087218235	0.059000753640242924	0.03234712886035114	0.01672196444073423
11	0.033002978755936344	0.021612461572593923	0.0	0.007136038799771886	0.01547455210466188	0.035743465192210735	0.02092385457261744	0.020511592455707984
12	0.030645623130512317	0.013555745128372518	0.0019399129738627958	0.003105717882957199	0.0003877234999556949 3	0.03007991611562011	0.028791145904876483	0.001383580125300832
13	0.023801687443797407	0.019310542588530665	0.0027649898842652	0.007902590032969973	0.03438789356591529	0.07557703181731529	0.04543789847205947	0.033160783962103875
14	0.03421967843357455	0.01918265820052715	0.0027649898842652	0.007902590032969973	0.03438789356591529	0.015952117927497587	0.015823014403173387	0.03129435283904605
15	0.05574005398180032	0.01828746748450255	0.0	0.005911137344661539	0.012895460087218235	0.05199255239841663	0.04774730725330067	0.014017041358398692
16	0.019619282301916074	0.01918265820052715	0.0003929893766858287 76	0.0015884205966269645	0.01891334146125341	0.025300099362664336	0.013049077876653322	0.015523044045453774
17	0.05345874208622868	0.02212399912460798	0.0	0.1501492106264295	0.02235213081784494	0.1113522102949961	0.0866400385172881	0.059978164235544765
18	0.03406759097386977	0.01739227676847795	0.0012376153656036065	0.007902590032969973	0.03438789356591529	0.04306734503438897	0.032265747289836666	0.03698871177293371
19	0.03429572216342693	0.013555745128372518	0.0009233003521169763	0.00567405964367244	0.008596973391478822	0.0319830289277851	0.023339886267099524	0.0085983041696896
20	0.032090453997707685	0.02097303963257635	0.0	0.006108702095485789	0.004298486695739411	0.04557646885893029	0.030831690866524944	0.0062592809227648115
21	0.04555019418158034	0.019566311364537695	0.002278783847778069	0.0076101942017500835	0.012895460087218235	0.04681317336747656	0.039324532302834815	0.019484337216853883

d. Normalized attributed

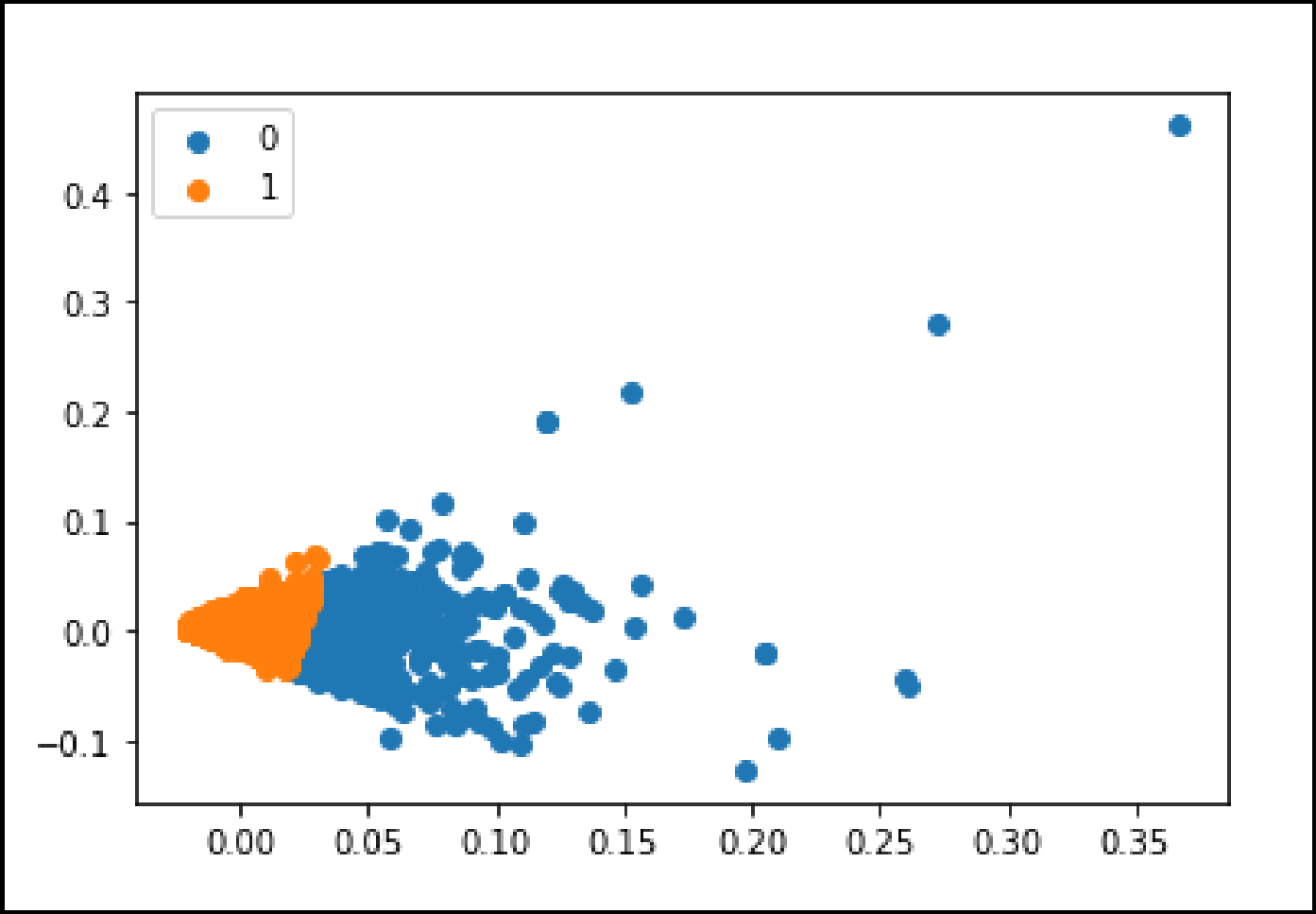
2 Data Visualization & Analysis





a. Select k : from Sillhouette_score, k=2 should be the best k

3 k-means Segmentation



a. Cluster : create 2-means model and plot the scatter plot with PCA(2)

3 k-means Segmentation


```
[ ]  1 compare[compare.label==0].imdb_score.value_counts()
```

```
8    231
7    182
6    110
9     24
5     12
4      3
Name: imdb_score, dtype: int64
```

```
 1 compare[compare.label==1].imdb_score.value_counts()
```

```
6     1517
7     1488
8      607
5      538
4      216
3       70
9       24
2       20
10       1
Name: imdb_score, dtype: int64
```

b. Check with imdb_score : the clusters didn’t group by only the label

4 Evaluation


```
1 c1 = []
2 c2 = []
3 for i in range(len(distances)):
4   distance_c1 = np.sqrt((distances[i][0]-centroids[0][0])**2+(distances[i][1]-centroids[0][1])**2)
5   distance_c2 = np.sqrt((distances[i][0]-centroids[1][0])**2+(distances[i][1]-centroids[1][1])**2)
6   c1.append(distance_c1)
7   c2.append(distance_c2)
```

```
[ ] 1 print(sum(c1)/len(c1))
    2 print(sum(c2)/len(c2))
    3
```

```
0.06654306027473829
0.07806994118700337
```

a. Average within centroid distance