REPORT

Example of a run of the code

optimal k 4
Silhoutte coefficient 0.33615612680651386

Clustering in kmeans.txt for provided shillbid.csv

0,1,2,3,4,5,6,7,8,10,11,12,13,14,15,16,17,18,22,23,24,25,26,35,36,37,38,40,45,46,53,54,56,58,
59,65,73,74,77,78,79,80,84,87,95,97,98,105,109,110,111,112,117,122,123,124,125,133,134,13
6,139,142,143,144,145,146,147,148,150,151,158,159,161,169,170,171,177,178,184,189,198,1
99,202,203,204,205,206,207,208,209,216,217,218,219,220,221,223,224,225,226,227,234,235,
236,242,243,254,255,256,257,258,259,260,265,266,267,268,269,278,279,283,284,292,293,302
,303,304,305,308,309,310,311,317,318,319,320,322,323,324,325,326,328,329,330,331,332,33
3,335,340,343,344,345,346,350,351,352,353,354,355,356,357,358,359,360,361,364,365,366,3
67,368,373,374,381,382,383,384,388,389,390,391,392,399,407,413,414,415,416,417,418,419,
421,425,426,427,428,429,434,435,436,437,438,439,442,444,451,452,463,464,468,473,477,480
,482,484,485,486,487,492,493,497,502,503,504,505,506,507,508,509,510,511,512,513,516,51
9,520,521,528,536,538,546,547,548,551,556,557,558,565,566,575,590,597,598,599,612,613,6
17,620,623,624,625,627

20,70,76,99,102,103,104,115,116,118,119,120,121,152,153,154,155,156,157,172,173,174,182,
194,197,200,201,228,230,231,232,233,244,246,247,248,249,275,276,277,281,282,341,348,371
,375,378,379,380,402,403,420,422,424,443,445,446,447,448,449,450,453,455,457,459,460,46
6,469,471,472,478,479,481,488,530,531,537,539,562,567,571,572,573,576,577,584,585,586,6
07,611,616,626

9,19,27,28,29,30,31,32,33,34,39,41,42,43,44,47,52,55,57,60,61,62,63,64,66,68,69,71,72,81,82,
83,88,89,90,91,92,93,94,96,100,101,106,107,108,114,126,127,128,129,130,131,135,137,138,1
49,160,162,163,164,165,175,176,183,185,186,187,188,190,191,192,193,210,211,212,213,214,
215,222,237,238,239,240,241,245,251,252,253,263,264,270,271,272,273,274,280,285,286,287
,288,289,290,291,294,295,296,297,298,299,300,306,307,321,327,337,338,339,342,347,349,36
2,363,369,370,376,377,385,386,387,393,394,395,396,397,398,400,401,408,409,410,411,412,4
30,431,432,441,461,462,465,470,474,476,483,489,490,491,494,495,496,498,499,500,501,514,
515,517,518,522,523,524,525,526,527,529,532,533,534,535,540,541,543,544,545,549,550,552
,553,554,559,560,563,564,568,569,570,574,578,579,580,581,582,583,587,588,589,591,592,60
2,603,604,605,606,608,609,610,618,619,621,622,628,629,630

21,48,49,50,51,67,75,85,86,113,132,140,141,166,167,168,179,180,181,195,196,229,250,261,2
62,301,312,313,314,315,316,334,336,372,404,405,406,423,433,440,454,456,458,467,475,542,
555,561,593,594,595,596,600,601,614,615

Clustering in divisive.txt for provided shillbid.csv

0,1,2,3,4,5,6,7,8,9,11,13,14,18,19,20,21,22,23,24,25,26,31,32,34,36,37,38,39,42,43,45,47,48,4
9,50,51,54,55,56,57,60,62,63,64,65,66,68,69,70,71,72,73,74,75,76,79,80,81,82,83,84,85,86,87,
88,89,90,91,92,93,94,95,96,98,99,100,101,102,103,104,105,108,110,111,112,113,116,117,118,1
19,120,121,123,124,128,133,135,137,138,140,141,143,144,147,148,149,152,153,154,155,156,
157,158,159,160,162,163,164,165,166,167,169,170,171,172,173,174,175,177,178,179,180,183
,184,186,187,188,189,190,191,192,196,197,199,201,202,203,204,205,206,207,208,209,211,21
2,213,214,215,218,219,220,221,222,224,226,227,228,229,230,233,234,235,236,237,238,239,2
40,241,242,243,244,246,248,249,251,252,253,254,256,257,258,259,261,262,263,264,265,268,
269,270,271,273,275,276,277,279,281,282,284,285,286,287,288,289,290,291,294,296,297,301
,302,304,305,308,309,311,312,313,314,315,316,317,318,319,320,322,323,324,326,327,329,33
1,332,334,335,336,337,338,339,340,341,342,343,349,351,352,353,356,357,359,360,361,364,3
65,366,367,368,369,370,371,372,373,375,377,378,379,380,383,384,386,387,389,391,392,394,
395,396,397,398,399,400,401,402,403,404,405,406,407,413,415,416,418,419,421,423,424,425
,426,427,428,429,433,434,435,436,437,438,439,440,441,442,444,445,446,447,448,449,451,45
2,453,454,455,457,459,460,461,464,465,467,468,469,471,472,473,474,475,476,477,478,481,4
82,484,485,486,487,488,491,492,493,495,496,497,498,499,500,501,502,503,504,506,507,508,
509,510,511,512,513,514,516,517,518,519,520,521,527,528,529,531,532,536,537,539,540,541
,542,543,544,545,546,547,548,549,553,554,555,556,557,558,559,560,562,563,564,565,566,56
8,572,573,574,575,576,577,578,579,581,586,587,588,589,590,591,593,594,595,596,597,598,5
99,600,601,607,608,611,612,613,614,615,616,617,618,619,620,622,624,625,626,627,630

10,35,40,53,58,107,109,114,125,126,136,145,168,185,195,216,217,223,225,255,280,283,292,3
03,310,328,330,333,346,347,350,358,374,385,388,390,450,505,523,524,525,551,621,623

12,33,46,59,151,193,198,231,232,298,354,381,382,628,629

15,16,17,27,28,29,30,41,44,52,61,67,77,78,97,106,115,122,127,129,130,131,132,134,139,142,
146,150,161,176,181,182,194,200,210,245,247,250,260,266,267,272,274,278,293,295,299,300
,306,307,321,325,344,345,348,355,362,363,376,393,408,409,410,411,412,414,417,420,422,43
0,431,432,443,456,458,462,463,466,470,479,480,483,489,490,494,515,522,526,530,533,534,5
35,538,550,552,561,567,569,570,571,580,582,583,584,585,592,602,603,604,605,606,609,610

Jaccard Similarity matrix

```
[[0.3740458  0.10909091 0.02985075 0.0778098 ]
 [0.15271967 0.00740741 0.01904762 0.08465608]
 [0.26579926 0.04313725 0.02155172 0.23161765]
 [0.10042735 0.02040816 0.         0.04320988]]
```

Result

Cluster 0 in k-means corresponds to cluster 0 in hierarchical clustering with Jaccard similarity score of 0.374
Cluster 1 in k-means corresponds to cluster 3 in hierarchical clustering with Jaccard similarity score of 0.153
Cluster 2 in k-means corresponds to cluster 1 in hierarchical clustering with Jaccard similarity score of 0.266
Cluster 3 in k-means corresponds to cluster 2 in hierarchical clustering with Jaccard similarity score of 0.100

Approximate Total time taken: 50.37 seconds

If the dataset has larger instances time taken increases exponentially. To check the output faster try reducing the instance size

BRIEF EXPLAINATION OF FUNCTIONS USED

`cosine_similarity(x, y):`
function computes the cosine similarity between two vectors x and y. Cosine similarity is a measure of similarity between two non-zero vectors that calculates the cosine of the angle between them. It ranges from -1 (opposite directions) to 1 (same direction), with 0 indicating orthogonality. The function computes the dot product of the two vectors x and y, and divides it by the product of their Euclidean norms, as per the definition of cosine similarity.

`k_means_cosine(X, k, max_iter=20):`
function clusters the data X into k clusters using the K-means algorithm with cosine similarity as the distance measure. K-means is a popular clustering algorithm that partitions data into k clusters by minimizing the sum of squared distances between each data point and the centroid

of its assigned cluster. This function initializes k centroids randomly, then iteratively assigns each data point to the nearest centroid based on cosine similarity, and updates the centroid positions based on the mean of the assigned data points. The process is repeated for a maximum of max_iter iterations or until convergence. The function returns an array of cluster labels for each data point.

`jaccard_similarity(set1, set2):`
function takes in two sets set1 and set2, and computes the Jaccard similarity between them. The Jaccard similarity is a measure of similarity between two sets, defined as the ratio of the size of the intersection of the sets to the size of the union of the sets. The function computes the size of the intersection and union of the sets using the intersection and union methods of Python's built-in set class, respectively. If the size of the union is zero, the function returns zero to avoid division by zero.

`euclidean_distance(x1, x2):`
function takes in two vectors x1 and x2, and computes the Euclidean distance between them. The Euclidean distance is a measure of the distance between two points in n-dimensional space. The function computes the distance between the two points using the formula sqrt(sum((x1 - x2)^2)), where ^ denotes the element-wise power operator and sqrt is the square root function. This is equivalent to taking the norm of the difference between the two vectors. The function uses the numpy library to perform the element-wise subtraction and summation.

`complete_linkage(X, k):`
function is a hierarchical clustering algorithm that implements the complete linkage method for clustering. The input X is a matrix of shape (n_samples, n_features) containing the data points to be clustered, and k is the number of clusters to obtain. The function computes the pairwise distances between all data points in X and then merges the closest two clusters until only k clusters remain. The clustering is performed using the complete linkage method, which defines the distance between two clusters as the maximum distance between any two points in the clusters.

`get_cluster_bounds(labels):`
function takes the output of a clustering algorithm (i.e., an array of cluster labels) and returns the lower and upper bounds of each cluster. Specifically, it returns a list of tuples, where each tuple represents the lower and upper bound of a cluster in the original data array. The bounds are defined as the indices of the first and last elements of each cluster in the original data array