Input Data (provided under *Test*) consists of 10,740 images of handwritten digits (0-9). The images were scanned and scaled into 28x28 pixels. For every digit, each pixel can be represented as an integer in the range [0, 255] where 0 corresponds to the pixel being completely white, and 255 corresponds to the pixel being completely black. This gives us a 28x28 matrix of integers for each digit. We can then flatten each matrix into a 1x784 vector. No labels are provided.

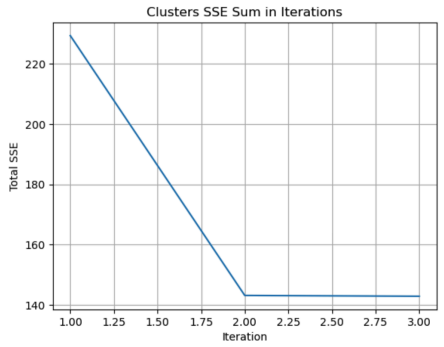
For the datasets in this homework, the numbers in the dataset are in string format after reading the text file with pandas “read_csv”, so I created a class called “RecordParser” to parse the records to float so that numerical computation can be done later in K-Means algorithm. For the Iris dataset, the dataset is very simple with only 4 features and 150 records that are on a similar scale. For the image dataset, the data is high dimensional and has a high number of total instances. In the preprocessing of my experiments, I used scaling techniques standard scaler or min max scaler, dimension reduction techniques primary component analysis or t-distributed stochastic neighbor embedding. I used standard scaler or min-max scaler to change data points to the same scale, primary component analysis to linearly transform the dataset to a smaller number of dimensions and pick n number of primary components. T-distributed stochastic neighbor embedding is a new technique I am trying after recommendation from the professor. After some research, I realized that t-distributed stochastic neighbor embedding is a better dimension reduction technique for our image dataset - In PCA, the aim is to keep the high eigenvalues that capture more variance in the data to reduce dimensionality. In t-SNE, the dimension is reduced more wisely by comparing the similarity between a data point to its neighbors, the number of neighbors is controlled by the perplexity parameter. Because the dataset is image data, it is more beneficial to use t-SNE because the local structure can be preserved during dimension reduction. For cluster validity, I chose to use the internal measure silhouette coefficient to measure both cohesion and separation of data points within clusters and between clusters. The goal is to maximize the difference between cohesion and separation, minimize cohesion, and produce an average silhouette as close to 1 as possible. The performance of the cluster is mainly measured by total cluster SSEs after the stopping condition is reached when centroids are no longer changing.

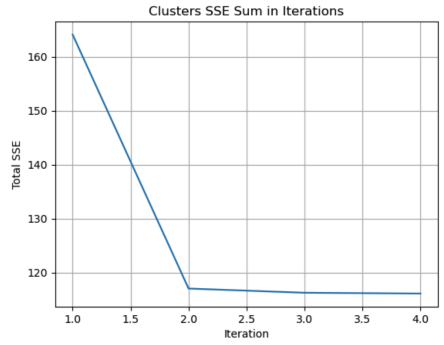
I created a class called “KMeans” to write the K-Means algorithm. There are a couple of different methods in this class. The first method is called compute Euclidean distance, this acts as a “utility” method to compute Euclidean distances from data points to cluster centroid and distances from a data point in a cluster to other data points in the same cluster in silhouette coefficient. The second method is called “computeWithinClusterDistances”, this method is used to compute data point distances to the cluster centroids to assign the data point to the closest centroid and cluster. Initially, the centroids are randomly chosen, and the distances are stored in hashmap “cluster_distances_map” with the keys as the centroid data points and values as the cluster data points distances to the corresponding centroid. For example “{(centroid data point): [dist1, dist2, dist3]} – {(1, 2, 3, 4): [0.2, 0.6, 1, 4], [7, 8, 9, 10]: [1.2, 3.1, 1.1]}”. The third method is called “collectClusterDataPoints”, this method has a clusters hashmap that has the keys as the centroids and values as the data points that are assigned to the cluster, for example “{(centroid data point): [datapoint1, datapoint2]} – {(1, 2, 3, 4): [[10, 11, 12, 13], [5, 6, 7, 8]]}”. The fourth method is “computeClusterSSE”, this method takes in cluster_distances_map and computes the sum of squared errors in each cluster and stores the SSE for each cluster in “cluster_sse_map”. For example “{(centroid data point): SSE value} – {(1, 2, 3, 4): 20.97}”. The method “computeTotalSSE” takes in the “cluster_sse_map” and sums the SSEs in clusters, and returns the total SSE. The method “computeNewCentroids” takes in the “clusters” hashmap and computes new centroids for each cluster by calculating the mean of the data points in that cluster, and returning the new centroids data points. The method “generateClusterAssignmentRes” takes in the data and “clusters” hashmap and generates the formatted output for the datapoint’s assigned cluster number such as 1, 2, or 3 if there are three clusters in total. The method “computeSilhouetteCoefficient” takes in the “clusters” hashmap and validate cluster accuracy by computing the average silhouette coefficient for all data point. In this method, I first calculate cohesion, a, the average distance of a data point to all other data points in its cluster. The result is stored in the “avg_within_cluster_distances” map in the corresponding index of the data point in the corresponding centroid key. The second part of this method is separation calculation, b, each data point’s minimum average distances to points in other clusters. The average distances from the data point to all data points in other clusters are stored in the “avg_other_cluster_distances” map. There are cases in which there is the same value of data points (key), so the distances for these different data points are all appended to the same key in the hashmap resulting in incorrect mapping. To prevent duplicate key issues, every cluster item is mapped to a unique index in incrementing order. I append the distances from the data point to other cluster data points with an associated index number of the key (data point that is computed against other data points), so it is easier to gather the average separation for each data point without duplicate key generates incorrect key distance mapping. More details about this can be found in the code. After this step,

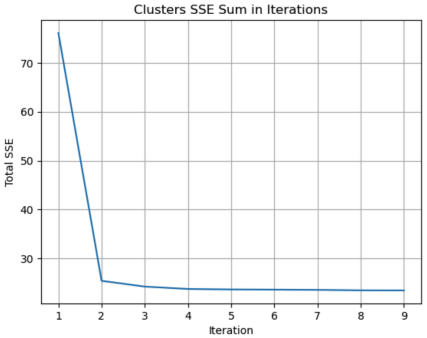
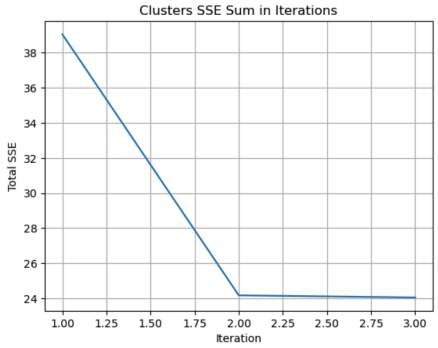
I find the minimum of the average distances for each data point to other clusters which looks like “b = min(average distance(c1, c2), average distance(c1, c3))”, and store them in “min_avg_distances” map. I append the minimum separation for each data point to the “separations_in_clusters” map to the corresponding index in the corresponding centroid. The silhouette coefficient is then calculated for each data point and the average silhouette coefficient is generated with the formula “b - a / max(a, b)”. The last method is called “runKMeans”, this method takes in the data and the initial centroids and runs the K-Means algorithm until the new centroid is equal to the previous centroid, if the stopping condition is not met, the centroid will be updated. The above methods are called in this method. When new centroids are equal to the previous centroids, the silhouette coefficient is generated, and the SSE changes for different iterations are plotted. The method “runKMeansKClusters” is used to plot the total SSE vs. number of clusters for the best results in the experiments, the reason to do so is to see if as the number of clusters increases, the total SSE decreases. The common knowledge is more number of clusters reduces the total SSE, and we can do so to validate this.

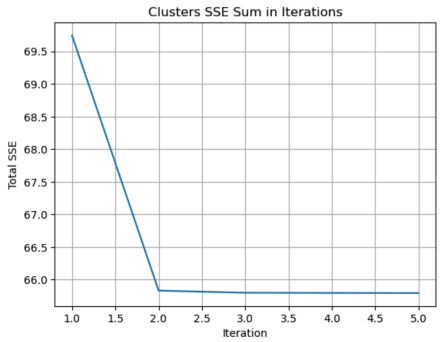
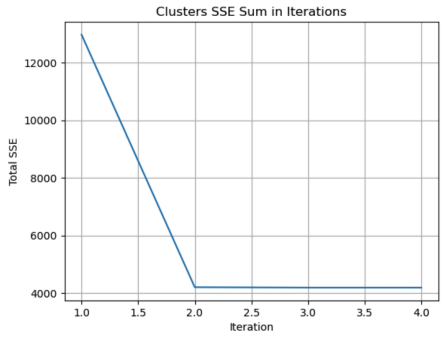
Every experiment below is associated with a unique seed number, every experiment has multiple runs because the initial centroid is randomly chosen. The total SSE result and average silhouette coefficient are observed, and the run with the lowest SSE is picked. The miner submission run is highlighted in yellow.

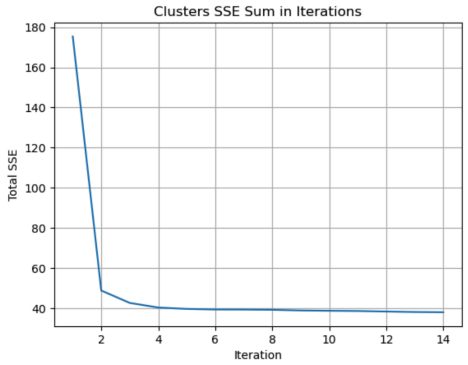
Part 1 - Iris Data						
Seed	Run No.	pre-processing	Initial Centroids	Total SSE Plot	Average Silhouette Coefficient	Final Total Cluster SSEs
74	1		[[4.5, 2.3, 1.3, 0.3], [6.7, 3.1, 4.7, 1.5], [5.1, 3.5, 1.4, 0.2]]		0.5415904929552017	78.94506582597731
	2		[[5.8, 2.6, 4.0, 1.2], [5.4, 3.4, 1.7, 0.2], [5.7, 4.4, 1.5, 0.4]]		0.49288553215414294	143.45373548406207

	3		[[5.0, 3.3, 1.4, 0.2], [4.8, 3.0, 1.4, 0.1], [6.2, 3.4, 5.4, 2.3]]		0.4960757260717737	142.87875
	4		[[6.5, 3.0, 5.5, 1.8], [5.8, 2.7, 5.1, 1.9], [6.7, 3.1, 5.6, 2.4]]	 no preprocessing is applied here, next experiment applying PCA to reduce the dimension to even smaller.	0.5414896370520781	78.94084142614601
75	1	PCA(n_components=2)	[[0.6404367495231468, -0.4173234829700222], [1.9708149459064834, -0.18112569470491022], [1.2847945878450728, 0.6854391861329235]]	 It seems like applying PCA increased SC and decreased SSE.	0.5905553228869362	63.873838060362274

	2		[[-3.22520044627498, -0.5032799094854222], [1.378736982775584, -0.42120513821462474], [-2.5065267893389023, 0.6519350136725757]]	 Next experiment I will apply scaling to see if that helps with the accuracy. Although the dataset is on similar scale.	0.5356860248464426	128.28242293863858
76	1	StandardScaler() PCA(n_components=2)	[[0.6621261375659381, -0.22434607094346454],[0.9739151144704745, -0.5711743763662664],[0.5738834855584674, -0.15371997408006546]]		0.4682513480929634	116.28662781711705
	2		[[0.7004727992084584, -0.06349392765170266],[1.3582398473872714, 0.3288202662704022], [-2.3887774935056423, 0.6747673967025153]]		0.47359348099641563	116.10924021401547

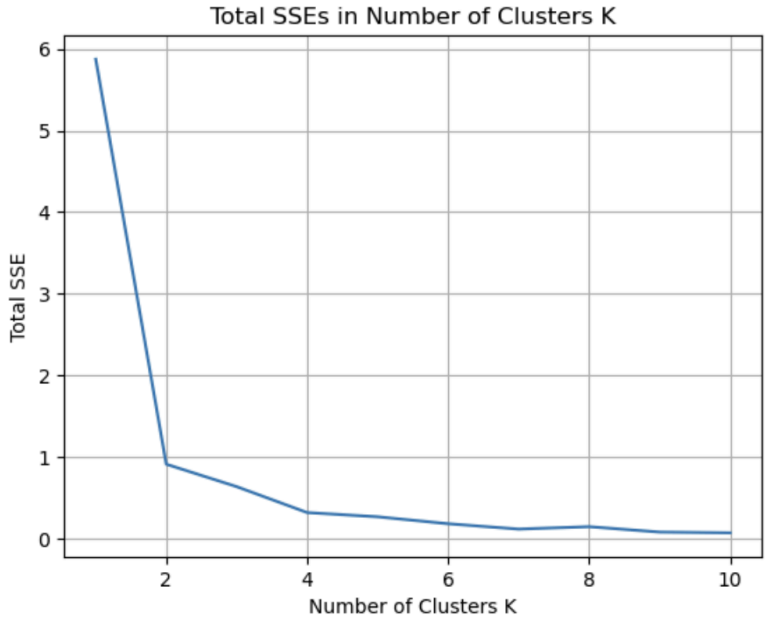
	3		[[1.3470444243548014, 0.422255965895547], [-2.0444165193198014, 0.6849564262949365], [1.1112717256759697, -0.29598610243039075]]	 It seems like scaling and PCA did not help with SC and SEE. Next experiment I will adjust n_component in PCA to see if anything improves.	0.47769830220942605	116.24247259315817
77	1	StandardScaler() PCA(n_components=1)	[[0.7324814400086799] ,[-2.1886757552808693],[0.5516339814225102]]	 The SSE is small here and the SC is much higher, this seems like a good result.	0.6812379328749264	23.46785440260509
	2		[[-1.9145563949595175],[0.9592985756160378],[1.7042407085196043]]		0.6929920089081274	24.04861222468255

	3		[[0.9262223675330981],[-2.4379508590951477],[-2.251465214584068 3]]	 Generally, the SSE and SC both improved in this experiment. Next experiment will try t-SNE to see if that helps with the SC and SSE.	0.6298790781371117	65.79428622498625
78	1	StandardScaler() TSNE(n_components=1, learning_rate='auto', init='random', perplexity=20, random_state=1)	[[-28.022689819335938], [-28.05878257751465], [15.815011978149414]]		0.7435246149334601	4190.579900224389
	2		[[-3.792764186859131], [-31.251739501953125], [2.912609100341797]]		0.73264839906926	1498.0758261482902

	3		[[6.17685604095459], [-1.0002344846725464], [9.901451110839844]]	 It seems like the quality of clusters improved with TSNE, but the SSE is very big. Next experiment I will apply only PCA with n_components of 1 without the scaling.	0.7318123791479659	1495.4273761524537
79	1	PCA(n_components=1, random_state=2)	[[0.23454058625983698], [0.5193832450849375], [-2.1990779614307625]]	 comparing to seed 77, scaling before applying PCA works better than without scaling.	0.6906008159936661	37.91846745046853
80	1	TSNE(n_components=1, learning_rate='auto', init='random', perplexity=55, random_state=3)	[[-5.3470635414123535], [5.845524311065674], [-5.634232997894287]]		0.6806910065234304	345.22193220325

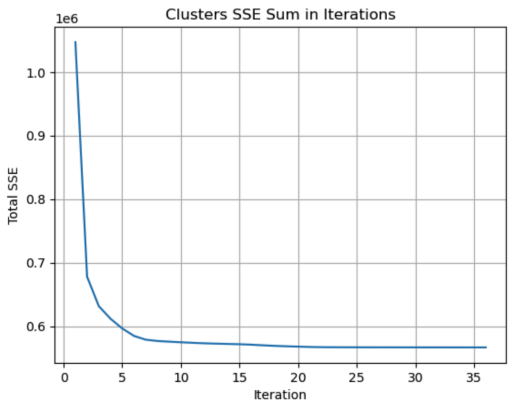
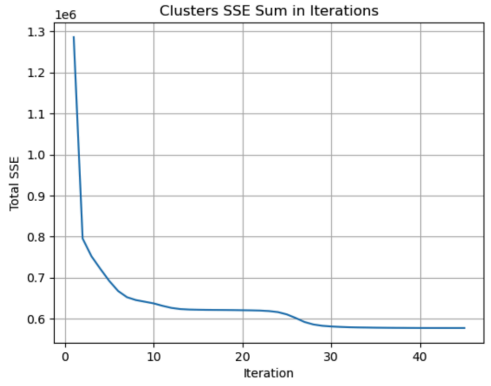
	2		[[2.892836809158325], [-5.126648426055908], [0.486527681350708]]	 It seems like seed 77 produced the best result so far. Next experiment will try min max scaler to see if it works better than standard scaler.	0.7035818378642182	103.80788570378363
91	1	MinMaxScaler() PCA(n_components=1)	[[-0.6256439567320374], [0.13534399328622046], [0.7578695970831767]]		0.7130490805855013	1.9003954081870633
	2		[[-0.17304986188184301], [0.623664456580685], [-0.5908477996600503]]	 It seems like MinMaxScaler produces significantly low total SSE and relatively high average SC. These preprocessing choices generate the best result so far.	0.7072897492795645	1.9011195391321893

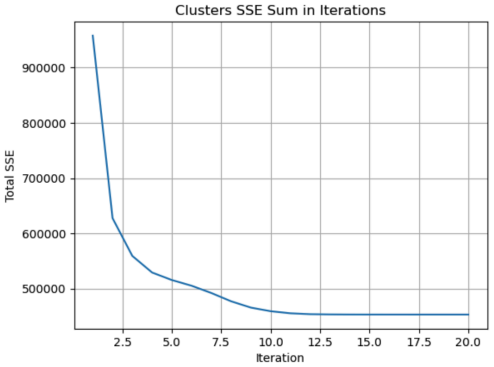
```
Total Cluster SSEs for 2 clusters: 5.87469178202524
average silhouette coefficient:  0.7586883202759617
Total Cluster SSEs for 4 clusters: 0.910071215078189
average silhouette coefficient:  0.6912294802976517
Total Cluster SSEs for 6 clusters: 0.6346786545020423
average silhouette coefficient:  0.5567486312886641
Total Cluster SSEs for 8 clusters: 0.3169511529507251
average silhouette coefficient:  0.6734158707335722
Total Cluster SSEs for 10 clusters: 0.2654188318713512
average silhouette coefficient:  0.5565104040388039
Total Cluster SSEs for 12 clusters: 0.18032191602908687
average silhouette coefficient:  0.5678216703683271
Total Cluster SSEs for 14 clusters: 0.11461245857288588
average silhouette coefficient:  0.5577873562399271
Total Cluster SSEs for 16 clusters: 0.14427639979491727
average silhouette coefficient:  0.5532075197410777
Total Cluster SSEs for 18 clusters: 0.07818718322406558
average silhouette coefficient:  0.5618454749078394
Total Cluster SSEs for 20 clusters: 0.0688111108824708
average silhouette coefficient:  0.5974041883404224
```



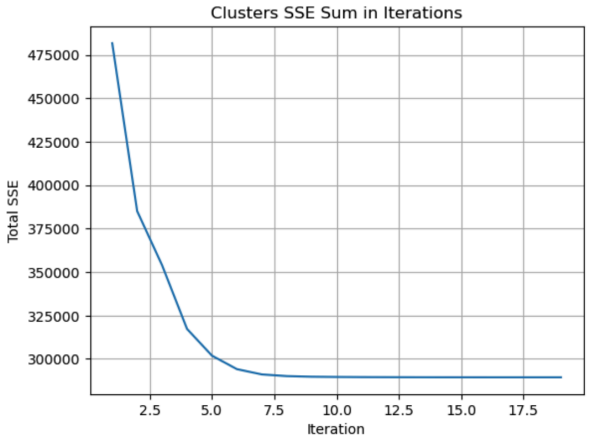
Based on the total cluster SSEs and the average silhouette coefficient, seed 91 generated the best result. The above plot for Total SSE vs. K Clusters is for seed 91. Based on the plot, as the number of clusters goes up, the total SSE decreases which proves that increasing the number of clusters decreases the total SSE. The “elbow finding” is at 2 clusters which means 2 clusters is an optimal number of clusters.

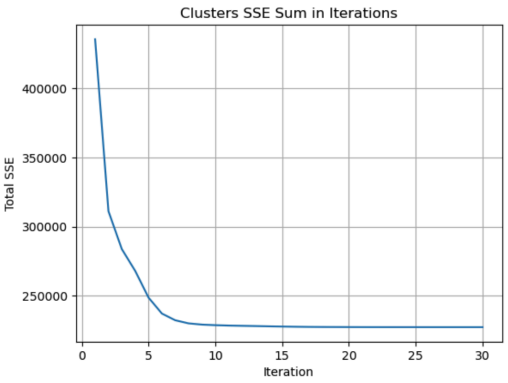
Part 2 - Image Data						
See	Run	pre-processing	Initial Centroids	Total SSE Plot / Interpretation	Average Silhouette	Final Total Cluster SSEs

d	No.				Coefficient	
81	1	StandardScaler() TSNE(n_components=3, learning_rate='auto', init='random', perplexity=50, random_state=5)	[[-4.362462997436523, 8.415939331054688, -11.174600601196289], [2.5119130611419678, -16.666147232055664, 4.290916919708252], [21.305927276611328, 13.326613426208496, 3.5207366943359375], [1.4578825235366821, 11.632572174072266, -3.877708673477173], [5.809217929840088, 0.5345432758331299, -19.609643936157227], [-0.6351838111877441, -0.6118497848510742, 15.208481788635254], [-0.9458160996437073, -15.186185836791992, -18.8242244720459], [-5.1339850425720215, -0.7986196875572205, 22.279041290283203], [-4.111088275909424, 1.8991625308990479, 6.847727298736572], [-10.029145240783691, 3.333775043487549, 2.917454242706299]]		0.3395042002095166 6	566831.0714954137
	2		[[9.988402366638184, -15.503031730651855, 5.895402908325195], [13.813042640686035, 10.077898025512695, -3.876455783843994], [0.21716047823429108, -6.454451560974121, 12.269683837890625], [-5.1715593338012695, -18.15504264831543, -16.95201873779297], [9.733298301696777, 10.89058780670166, -3.9977874755859375],	 The SSE is really high and the SC is really low, next experiment I will remove scaling.	0.3544084427742143	576971.4464817736

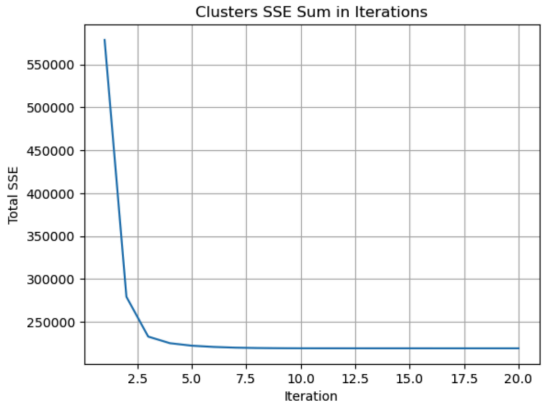
			[-3.9586968421936035, -17.13099479675293, -14.273725509643555], [-12.760273933410645, 2.4130396842956543, -11.76315975189209], [-4.261356353759766, -18.18511962890625, -15.55452823638916], [8.052326202392578, 13.311427116394043, -3.1309280395507812], [2.2043638229370117, -13.79829216003418, -11.091588020324707]]			
82	1	TSNE(n_components=3, learning_rate='auto', init='random', perplexity=50, random_state=6)	[[-18.816272735595703, 8.576919555664062, -5.259861469268799], [14.216898918151855, -14.761919021606445, -3.441166877746582], [-10.133437156677246, -5.964501857757568, -0.7163533568382263], [-9.55116081237793, 2.7974815368652344, -7.283871650695801], [-9.288122177124023, 3.769473075866699, 3.953503131866455], [15.684388160705566, 5.0264387130737305, -9.083136558532715], [13.58538818359375, -13.155280113220215, 3.159348726272583], [11.508049011230469, 7.970358371734619, -4.05140495300293], [1.3996915817260742, 6.549544811248779, 16.583755493164062], [7.768166542053223, -1.2984304428100586, -0.04256720095872879]]		0.40056536019378813	452848.01009250403

	2		[[-1.2197660207748413, -1.8514689207077026, -15.04919719696045], [-9.484627723693848, 1.8222013711929321, 3.7106685638427734], [7.249090194702148, -7.850967884063721, 10.243032455444336], [21.663772583007812, -0.3610471189022064, -9.329562187194824], [-13.92473316192627, 10.694684028625488, 4.205991744995117], [-9.040042877197266, -4.474287033081055, 8.025867462158203], [18.07091522216797, 9.520544052124023, -10.852327346801758], [-14.680047035217285, -6.763497352600098, 5.168874263763428], [-0.14507736265659332, -16.465505599975586, -5.151956558227539], [-0.4795327484607696 5, 4.934739589691162, 1.180207371711731]]		0.3878625336164127 6	451110.68861014384
	3		[[-5.33727502822876, -4.966237545013428, -8.269036293029785], [-1.623985767364502, -4.1701555252075195, -14.113212585449219], [15.0390625, 4.804385662078857, -12.860013961791992], [9.797309875488281, -3.9276015758514404, -10.840121269226074], [-8.633082389831543, 4.991453647613525,		0.3878159444565148 5	451117.97682640475

			-0.6242240071296692], [6.0813093185424805, 8.580733299255371, 0.8642632365226746], [-9.32388687133789, 14.769373893737793, -4.7810163497924805], [-8.897210121154785, -8.516839981079102, 7.752476692199707], [9.498943328857422, 6.704414367675781, 4.433642387390137], [-9.350461959838867, -1.9095929861068726, 5.0184898376464841]	It seems like the parameters in TSNE is not good, going to different perplexity.		
83	1	TSNE(n_components=3, learning_rate='auto', init='random', perplexity=250, random_state=8)	[[6.488522529602051, -2.0926103591918945, 0.20483070611953735], [4.81536865234375, 4.621217727661133, 14.570356369018555], [-2.1589252948760986, -14.588318824768066, -10.078678131103516], [2.884786367416382, -6.859468460083008, -2.323586940765381], [7.678153038024902, 8.00413990020752, 0.16501861810684204], [-2.1509077548980713, -4.722622394561768, -5.974485874176025], [-0.830449640750885, 5.480724334716797, 7.8328351974487305], [-0.5425418019294739, 4.898514270782471, 0.7636023759841919], [3.4304723739624023, 7.323925018310547, 4.520482063293457], [-1.6627742052078247, 14.91784381866455,		0.31377527300021507	289452.2896391974

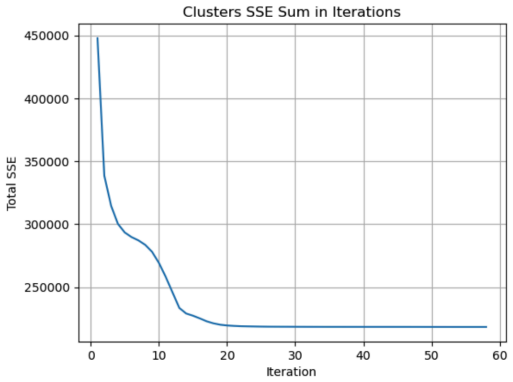
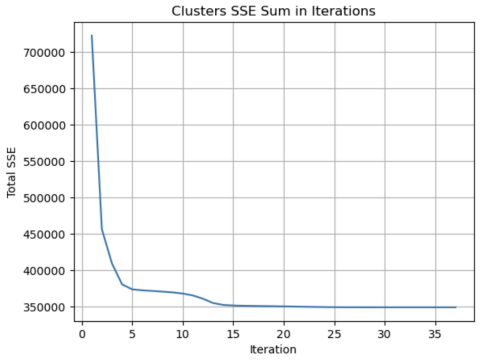
			6.040406227111816]]			
	2		[[-1.7591627836227417, -13.620737075805664, -2.377864122390747], [6.04260778427124, 4.663503646850586, 9.090254783630371], [-10.208724021911621, -17.127262115478516, -2.523061990737915], [8.641295433044434, 7.458371162414551, 3.77372407913208], [-10.27718448638916, -9.061979293823242, -6.587792873382568], [1.3853777647018433, -3.452169418334961, 5.278581619262695], [-4.972595691680908, 6.039614200592041, -5.610734939575195], [5.363536834716797, 9.63963508605957, 9.080191612243652], [12.860066413879395, 7.60385799407959, 6.691082000732422], [4.986866474151611, 4.1510419845581055, -4.62326192855835]]		0.3781522496556291 6	227234.10345959663
	3		[[-10.964117050170898, -8.392838478088379, -5.41792631149292], [-1.4707932472229004, -3.7345669269561768, -2.094139814376831], [-2.338507652282715, -9.582348823547363, -13.449580192565918], [0.9892698526382446, 3.9565823078155518, 10.695898056030273], [-1.2265188694000244,		0.3782164821160777	227233.85706878453

			-16.988155364990234, -13.571538925170898], [-2.9806790351867676, 4.218292236328125, -0.2907271385192871], [2.1832664012908936, -8.584368705749512, -4.137304782867432], [2.3381335735321045, 9.188650131225586, 4.894195556640625], [-2.1847281455993652, 3.380561590194702, 3.274754762649536], [3.542116165161133, 14.748555183410645, 1.529593825340271]]	This experiment shows that higher perplexity decrease total SSE. The SC is still very similar to experiment seed 82. Next experiment I will increase the perplexity and see if further increasing makes the results better.		
84	1	TSNE(n_components=3, learning_rate='auto', init='random', perplexity=350, random_state=9)	[[-1.6252886056900024, -6.070270538330078, -5.346400737762451], [-6.586874485015869, -8.425541877746582, 1.3213939666748047], [-4.167906761169434, -5.5160698890686035, -0.19658342003822327], [-6.8885369300842285, 5.637436866760254, 4.943836212158203], [1.4521379470825195, -4.942769527435303, -0.08668452501296997], [-0.08201657235622406, -7.289196491241455, 4.86399507522583], [3.421640157699585, 13.85307502746582, 8.79636001586914], [-12.9214448928833, 2.209547996520996, -3.249208688735962], [8.815271377563477, 8.991310119628906,		0.35771213737817237	215365.4947894038

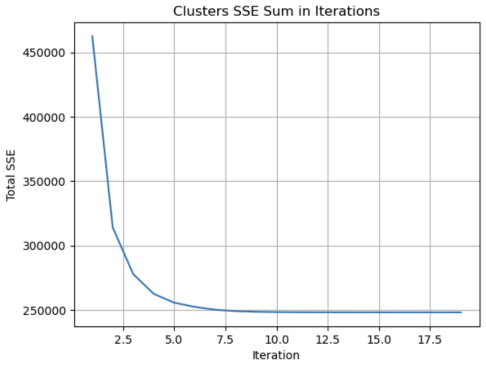
			-1.5307070016860962], [-6.872860908508301, 10.531994819641113, 6.076694488525391]]			
	2		[[-6.821990966796875, 5.8918843269348145, 7.39335823059082], [3.35626482963562, 1.595435619354248, 14.3589448928833], [2.666459798812866, 14.218631744384766, 5.165430545806885], [-6.314855575561523, -5.291659832000732, -3.253720760345459], [-3.289766311645508, -4.727995872497559, -1.5655932426452637], [-4.231493949890137, -11.103683471679688, 0.23079997301101685] , [4.8213019371032715, -2.2340762615203857, 7.401984691619873], [-4.256531238555908, 4.829395294189453, 8.489357948303223], [5.958770275115967, 3.8139092922210693, 8.946341514587402], [2.759877920150757, 13.049784660339355, 11.74701976776123]]		0.3573208686835267	219358.35138534696

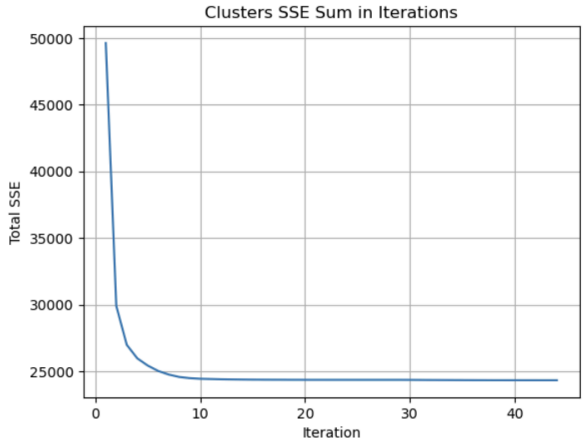
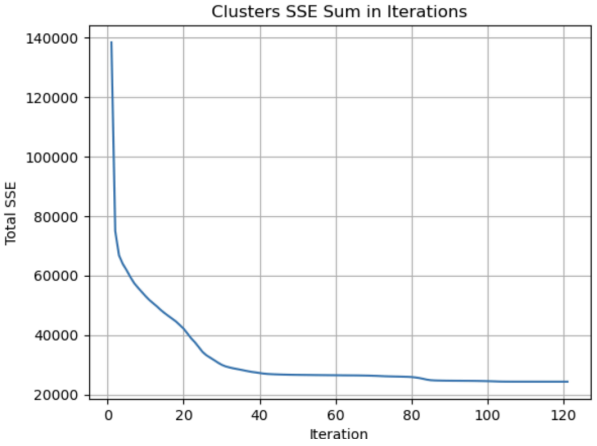
	3		[[6.400210857391357, 0.29408878087997437, 1.5152064561843872], [8.45903491973877, 5.985864162445068, 2.927959680557251], [2.8005447387695312, -1.3928099870681763, 0.9021463394165039], [3.3699729442596436, -0.06024805083870888, 2.736462354660034], [4.473725318908691, -5.0457611083984375, 0.3547155261039734], [-5.802370071411133, -9.06456470489502, -7.106898784637451], [-12.809651374816895, -2.9516282081604004, -9.522708892822266], [-3.6160895824432373, -13.71955680847168, -4.979814529418945], [-3.229032278060913, -2.1261496543884277, 3.0647575855255127], [-0.46626436710357666, 9.629544258117676, 8.14352798461914]]	 It looks like increasing perplexity decreased the total SSE. The next experiment will increase the perplexity even more and see if SSE keeps decreasing. After all experiments, it seems like this parameter produces the best result.	0.3579502961018043	215322.86752045114
85	1	TSNE(n_components=3, learning_rate='auto', init='random', perplexity=450, random_state=10)	[[6.727671146392822, -1.2766348123550415, -3.5054025650024414], [-5.229009628295898, 8.892516136169434, 6.044737815856934], [-5.789258003234863, -0.5211054086685181, -9.72125244140625], [6.6036057472229, 8.066794395446777, 5.184606552124023], [0.42423561215400696, -3.0925891399383545, -1.2565906047821045], [-5.351006031036377,		0.3479067127860909	218513.53080266053

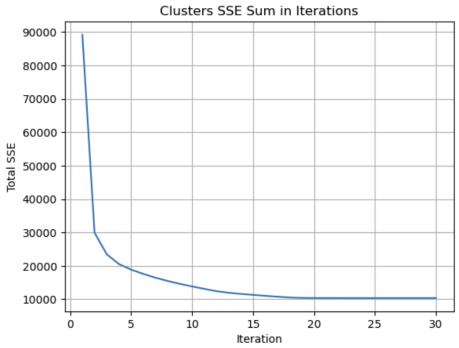
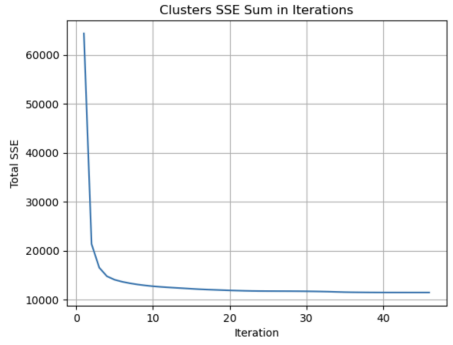
			-13.868536949157715, 1.6186814308166504], [-7.2596917152404785, -9.834341049194336, -5.158730506896973], [1.4117968082427979, 3.005951166152954, 8.25523853302002], [6.39180850982666, 5.188699245452881, -5.045945644378662], [-2.472606658935547, -2.9770004749298096, -8.269400596618652]]			
	2		[[-8.26598834991455, 7.12450647354126, 6.569612503051758], [-4.477186679840088, 2.8766205310821533, -0.37208548188209534], [-6.251322269439697, 8.659124374389648, 10.029833793640137], [-10.959444999694824, -10.766304016113281, -3.2044832706451416], [5.3347015380859375, -1.2890453338623047, 2.1897082328796387], [-3.734576463699341, -5.497470378875732, -1.411848783493042], [4.287235260009766, 5.148699760437012, -5.743025302886963], [2.687910556793213, 11.099952697753906, 4.294515132904053], [5.135231018066406, 0.3830163776874542, -3.778355360031128], [-5.314864635467529, 8.036186218261719, -2.345923900604248]]		0.3410323411172379	223141.095985796

	3		[[4.58882999420166, 10.634196281433105, -2.2113940715789795], [-6.2828898429870605, 1.1963903903961182, 6.530126571655273], [3.1332616806030273, 1.1257513761520386, 9.945514678955078], [0.5300735235214233, 5.260720252990723, -1.3173565864562988], [3.0609631538391113, 1.8781545162200928, -4.738558769226074], [7.271354675292969, 1.0047706365585327, -0.2610110342502594], [-1.1524641513824463, 11.332873344421387, -1.7120332717895508], [-1.0905256271362305, 8.684524536132812, -3.8036370277404785], [-12.859895706176758, 6.247151851654053, -1.1607208251953125], [-4.546169757843018, -12.435802459716797, 3.7476861476898193]]	 It looks like further increasing perplexity did not decrease SSE. I have been using init=”random” in TSNE parameter, but init=“pca” is more stable. The next experiment will adjust the init to “pca”, and keep other parameters the same from seed 84.	0.3474334835487443 7	218480.2812049198
86	1	TSNE(n_components=3, learning_rate='auto', init='pca', perplexity=350, random_state=11)	[[-12.227723121643066, 6.344151020050049, 0.5424076914787292], [22.191978454589844, -7.461008071899414, -5.048150539398193], [7.3020501136779785, -0.5327677130699158, -0.3652942478656769], [5.648404598236084, -0.39568498730659485, 12.987274169921875], [-15.806148529052734, 4.339147567749023, 2.1030004024505615],		0.347546620175304	349132.0527464356

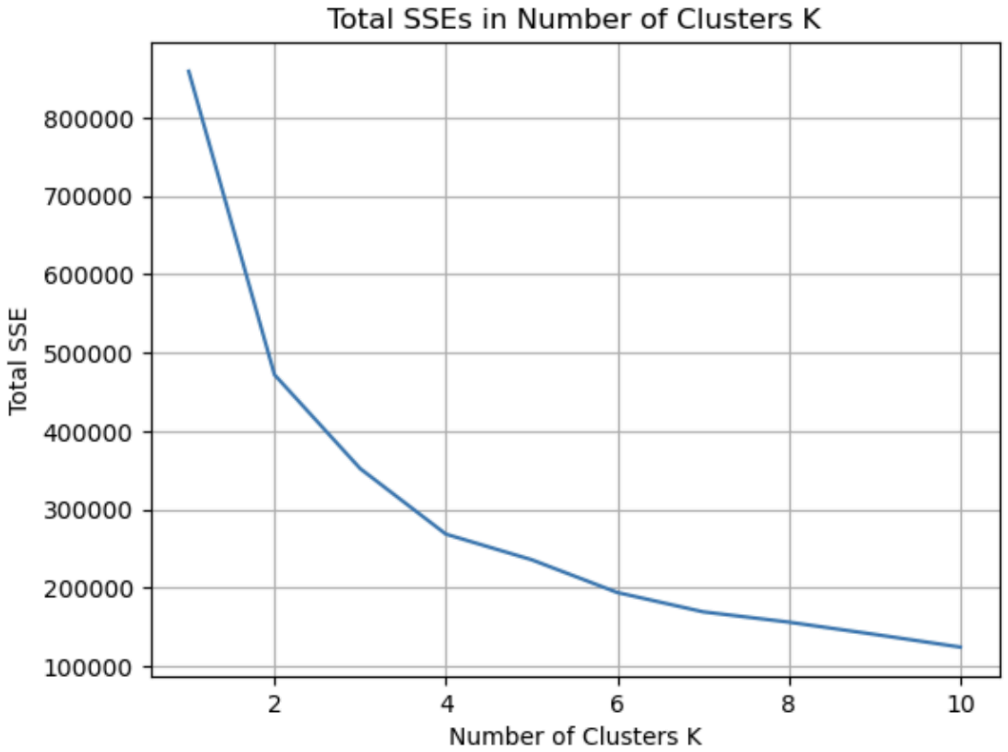
			[6.66001558303833, -4.135474681854248, 2.900871515274048], [8.960783004760742, 8.840009689331055, 13.318812370300293], [10.49425983428955, -4.476538181304932, 5.0475077629089355], [13.047249794006348, -14.41474723815918, -14.933512687683105], [-6.022294521331787, 2.7771215438842773, -0.9828327298164368]]		
	2		[[20.648513793945312, -10.573799133300781, -1.616604208946228], [4.155029773712158, 9.431520462036133, 12.007688522338867], [14.210494041442871, 0.9669098258018494, 8.500822067260742], [3.2410597801208496, 4.166337490081787, 11.928366661071777], [-10.668756484985352, 10.76205062866211, -3.097686767578125], [-12.73373794555664, 6.971416473388672, 5.940825939178467], [-12.746455192565918, 5.3590497970581055, -1.6550467014312744], [-2.1307525634765625, 2.516178846359253, -2.6465044021606445], [-0.30406180024147034, 5.856456756591797, -6.303934097290039], [12.679292678833008, -7.139547824859619, 6.046748161315918]]	 It seems like init pca did not perform better than init random. It seems like seed 84 is the best result so far. The next experiment will tune the perplexity parameters in seed 84 with all other variables staying the same. But with more experiments to hopefully have better random centroids as they can drastically affect the quality of clustering.	0.35153301241124074	342648.31450355676

87	1	TSNE(n_components=3, learning_rate='auto', init='random', perplexity=320, random_state=12)	[[-6.372227191925049, -2.551150321960449, 10.00829792022705], [3.060089588165283, 0.316343754529953, -4.420971870422363], [-6.4063639640808105, -1.5493075847625732, 15.682098388671875], [11.79458236694336, -2.0142831802368164, -7.739021301269531], [1.329126000404358, 8.006464958190918, 5.14825439453125], [-11.28390121459961, -1.1729214191436768, -1.0584746599197388], [0.6033880114555359, 1.4097782373428345, -18.12601089477539], [-9.603614807128906, -5.631997585296631, -9.814868927001953], [-5.861692428588867, 0.8566109538078308, 11.420513153076172], [-0.6360477805137634, -1.5112977027893066, -2.102100372314453]]		0.3618462109055911	248111.49260296178
	2		[[4.449072360992432, 8.932315826416016, 8.769105911254883], [-8.399085998535156, 9.816617965698242, 5.723801612854004], [-7.132967948913574, -0.933777928352356, 1.5125041007995605], [7.717006206512451, 3.8383243083953857, 8.815823554992676], [-5.861692428588867, 0.8566109538078308, 11.420513153076172], [-2.969677209854126, -7.230989456176758,	 it seems like tuning perplexity did not help with the total SSE or SC that much. Next experiment I will change the number of	0.3432707758402013	261909.3740462702

			3.8178670406341553], [4.618891716003418, 0.7773556113243103, -2.3302674293518066], [2.1014764308929443, 2.3974695205688477, 2.0438685417175293], [8.85661792755127, 7.503250598907471, 14.387982368469238], [-5.340523719787598, 4.1084418296813965, 7.2584757804870605]]	components to 1 with a slightly higher perplexity to compare more neighbors because we are only keeping one component.		
88	1	TSNE(n_components=1, learning_rate='auto', init='random', perplexity=420, random_state=13)	[[24.903518676757812], [-21.882692337036133], [19.42913055419922], [15.690125465393066], [7.281490802764893], [2.3023571968078613], [-7.091976642608643], [-11.107832908630371], [-28.310630798339844], [-8.269097328186035]]		0.586857846969897	24327.98840519711
	2		[[23.120267868041992], [13.745823860168457], [4.15481424331665], [-4.937835693359375], [17.91064453125], [-25.143749237060547], [14.157017707824707], [19.41477394104004], [1.5112552642822266], [12.372127532958984]]		0.5868590221016096	24327.95874846896

89	1	TSNE(n_components=1, learning_rate='auto', init='random', perplexity=700, random_state=14)	[[-2.1927101612091064], [-8.128716468811035], [-6.60947322845459], [3.8004446029663086], [0.8631519675254822], [13.735475540161133], [-3.289433479309082], [-3.709421396255493], [-7.676798343658447], [8.053791046142578]]		0.5856617039258601	10362.636404301198
	2		[[7.492152690887451], [6.124416828155518], [-8.354227066040039], [-7.061989784240723], [0.31566962599754333], [10.95419979095459], [10.189692497253418], [8.816688537597656], [11.667515754699707], [-15.525835990905762]]		0.5730388400982013	11469.732055749524

```
Total Cluster SSEs for 2 clusters: 858747.1254253855
Total Cluster SSEs for 4 clusters: 471972.19515468564
Total Cluster SSEs for 6 clusters: 352250.5941478379
Total Cluster SSEs for 8 clusters: 268698.03863281757
Total Cluster SSEs for 10 clusters: 236057.97500675466
Total Cluster SSEs for 12 clusters: 194271.81042825227
Total Cluster SSEs for 14 clusters: 169677.88014413975
Total Cluster SSEs for 16 clusters: 156509.80026496056
Total Cluster SSEs for 18 clusters: 140921.52424924864
Total Cluster SSEs for 20 clusters: 124733.082756365
```



Based on the total cluster SSEs and the average silhouette coefficient, seed 84 generated the best result. The above plot for Total SSE vs. K Clusters is for seed 84. Based on the plot, as the number of clusters goes up, the total SSE decreases which proves that increasing the number of clusters decreases the total SSE. The “elbow finding” is at 2 clusters which means 2 clusters is an optimal number of clusters.