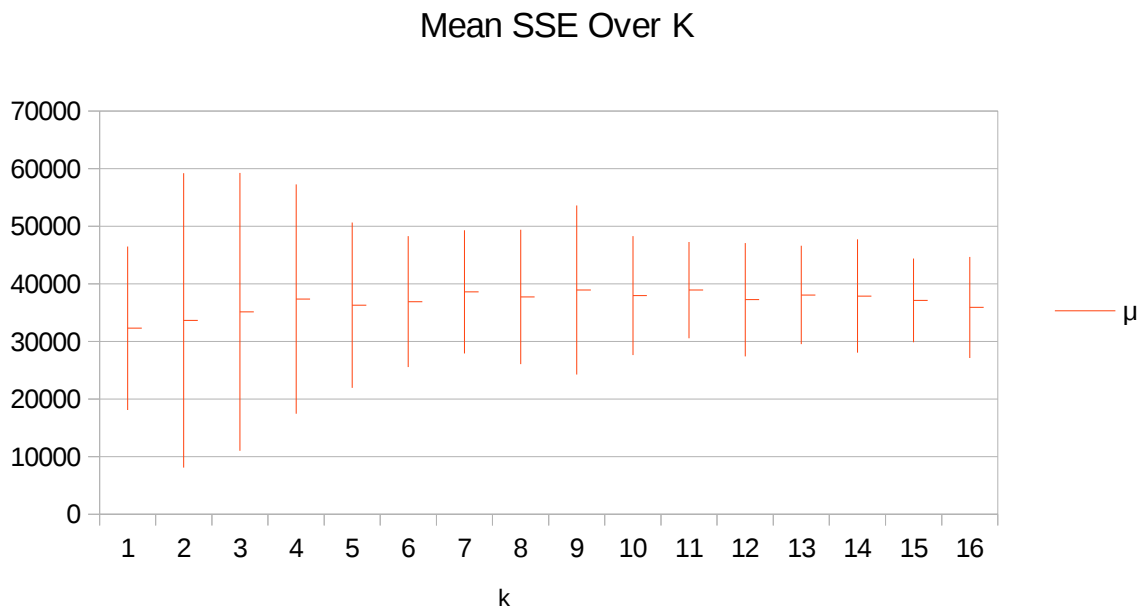
Benjamin Limoges
Assignment 7

Table:

k	μ-2σ	μ+2σ	μ
1	10697	43110	26904
2	-743	57679	28468
3	2555	57732	30143
4	9948	55424	32686
5	15054	47869	31461
6	19181	45164	32172
7	21912	46325	34119
8	19796	46449	33122
9	17697	51247	34472
10	21557	45196	33377
11	24917	44029	34473
12	21306	43792	32549
13	23722	43249	33485
14	22054	44520	33287
15	24110	40721	32416
16	21012	41032	31022

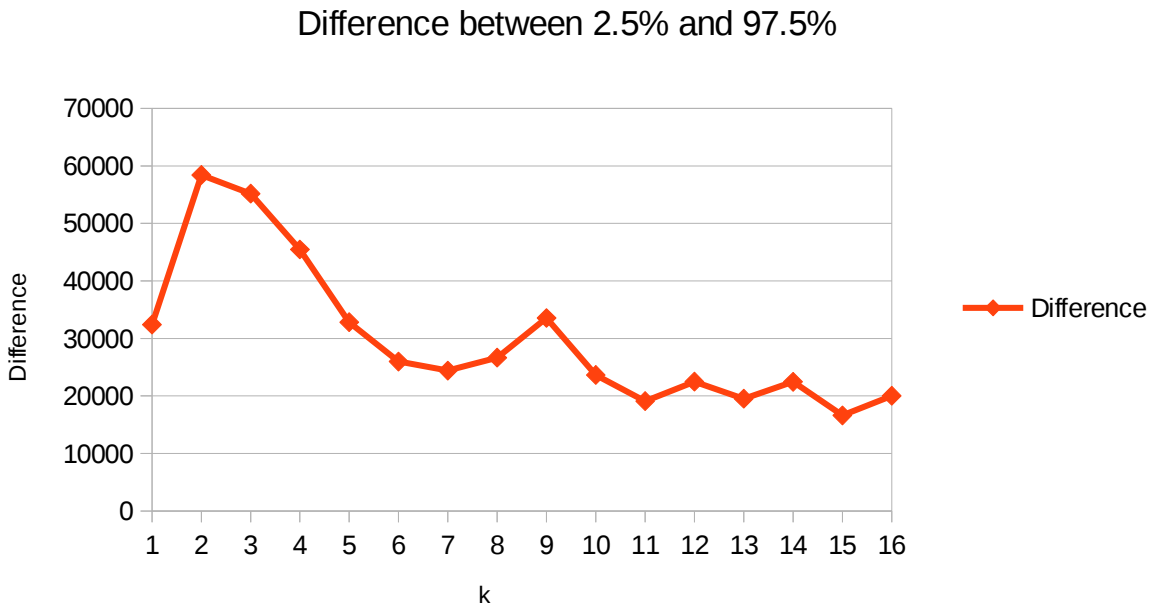
Graph:



N.B. : I could not get the lines to connect – no matter what configuration I gave Excel, it always produces stock diagrams as seperate instances with error bars.

As the number of clusters (*k*) increases to the number of data points, the SSE metric will become decrease until it reaches 0 (at *n* clusters). This is because we're looking at the sum of the squared distances between each point in each cluster and the cluster mean. When there are *n* clusters, the distance between each point in each cluster and the cluster mean is 0, since each point is by definition its own cluster mean. Since SSE will decrease as we increase *k*, if we do not bound the maximum value that *k* can take on, then if we try to minimize SSE without cost to pick the optimal *k*, we will have *n* clusters.

I would argue that we shouldn't necessarily think that there should be an “elbow” when the number of clusters is equal to the number of classes. This depends on the crucial assumption that the class labels (unseen) are not only perfect, but that they are relatively spread out in space, non-overlapping, and compact. But there are plenty of datasets where these assumptions are violated, and so a chart of the variance of the SSE is a bad criteria for these specific datasets. After looking at a graph of the difference between the 2.5% level and 97.5% level, I can see two candidates for “elbows”: 6 and 10.



My methodology takes the first feature, sorts it (after normalization) and then uniformly splits the dataset depending on the value of k. So if I'm looking for 4 clusters, then I would split the dataset into fifths, and the indexes from the original dataset are returned for the value at the 1/5th, 2/5th, 3/5th, 4/5th points. For ease, I only did it on the first feature in the dataset; to be properly exhaustive, I should have done this for each feature. Overall it typically performed on the lower side of extrema of the random starts method.

I originally thought that this method would produce better since I would guarantee that I would not just pick randomly pick k points in one “true” cluster (unless that feature is not related to the cluster). By spreading the points out this way, I'm assuring that if clusters exist with respect to this feature, then I would hopefully be able to spread them out better than random chance. If I did this for all the features, then I would be able to properly identify the “best” feature to spread the data on, assuming the data is clustered relatively spherically.

k	Own Method	Min Random	Max Random
1	33508.63222	9448.56674	53825.81982
2	23535.64798	20434.00802	67176.5987
3	23694.80582	20856.31529	64565.42008
4	21351.247	21411.97913	57768.3225
5	33413.85246	20348.81418	50063.51821
6	31218.38807	21410.0835	44931.19215
7	22648.40558	20580.62591	45251.7577
8	32614.56584	22766.53182	48258.08825
9	36973.55431	22977.3316	53150.91186
10	23850.2962	21323.09106	42663.59299
11	31541.48616	26408.43008	49251.61801
12	30846.94093	25613.19528	53007.19249
13	32652.89408	22858.10041	42070.36419
14	35265.39299	26234.50474	51242.07091
15	30296.07871	22700.06282	40602.53253
16	33765.68466	24756.08166	46961.05188