Practicum Problem

Problem 1

In [1]:

```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.cluster import AgglomerativeClustering
```

In [2]:

```
# import data (change '?' to np.Nan)
df = pd.read_csv('./data/auto-mpg.data', names = ["mpg", "cylinders", "displacement", "horsepower", "weight", "ac
             "origin", "car_name"], delim_whitespace=True , header=None, na_values='?')
```

In [3]:

```
df.loc[30:35,:]
```

Out[3]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
30	28.0	4	140.0	90.0	2264.0	15.5	71	1	chevrolet vega 2300
31	25.0	4	113.0	95.0	2228.0	14.0	71	3	toyota corona
32	25.0	4	98.0	NaN	2046.0	19.0	71	1	ford pinto
33	19.0	6	232.0	100.0	2634.0	13.0	71	1	amc gremlin
34	16.0	6	225.0	105.0	3439.0	15.5	71	1	plymouth satellite custom
35	17.0	6	250.0	100.0	3329.0	15.5	71	1	chevrolet chevelle malibu

In [4]:

```
# drop discrete columns
mpg_df = df.drop(["cylinders","model_year","origin", "car_name"],axis=1)

# mean-impute np.Nan fields
imp = SimpleImputer(missing_values = np.NaN, strategy='mean')
imp = imp.fit(mpg_df)
mpg_df = imp.transform(mpg_df)
mpg_df = pd.DataFrame(mpg_df)
mpg_df.columns = ["mpg", "displacement", "horsepower", "weight", "acceleration"]
```

In [5]:

```
mpg_df.loc[30:35,:]
```

Out[5]:

	mpg	displacement	horsepower	weight	acceleration
30	28.0	140.0	90.000000	2264.0	15.5
31	25.0	113.0	95.000000	2228.0	14.0
32	25.0	98.0	104.469388	2046.0	19.0
33	19.0	232.0	100.000000	2634.0	13.0
34	16.0	225.0	105.000000	3439.0	15.5
35	17.0	250.0	100.000000	3329.0	15.5

In [6]:

```
# fit clustering model
model = AgglomerativeClustering(linkage='average', affinity='euclidean', n_clusters=3)
cache = model.fit(mpg_df)
labels = cache.labels_
```

In [7]:

```
labels
```

Out[7]:

```
array([2, 2, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 2, 2, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
       2, 1, 1, 1, 1, 0, 2, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 2,
       1, 2, 1, 1, 1, 1, 1, 1, 2, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 2, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0,
       0, 2, 2, 2, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2,
       2, 0, 1, 1, 1, 1, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 1, 0, 2, 0, 0, 0, 0, 0,
       0, 2, 2, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 2, 1, 1, 2, 2, 0, 0, 0, 0,
       0, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 2, 0, 2, 0, 2, 2, 2, 2,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 2, 2,
       2, 2, 2, 1, 1, 2, 2, 0, 0, 0, 0, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0])
```

In [8]:

```
# Make a df with origin and cluster label
mpg_df['origin'] = df['origin']
```

```
mpg_df['cluster'] = labels
```

In [9]:
```
mpg_df
```

Out[9]:

	mpg	displacement	horsepower	weight	acceleration	origin	cluster
0	18.0	307.0	130.0	3504.0	12.0	1	2
1	15.0	350.0	165.0	3693.0	11.5	1	2
2	18.0	318.0	150.0	3436.0	11.0	1	2
3	16.0	304.0	150.0	3433.0	12.0	1	2
4	17.0	302.0	140.0	3449.0	10.5	1	2
...	...	...	...	...	...	...	...
393	27.0	140.0	86.0	2790.0	15.6	1	0
394	44.0	97.0	52.0	2130.0	24.6	2	0
395	32.0	135.0	84.0	2295.0	11.6	1	0
396	28.0	120.0	79.0	2625.0	18.6	1	0
397	31.0	119.0	82.0	2720.0	19.4	1	0

398 rows × 7 columns

In [10]:
```
# Compare cluster with class labels(origin)
group_by_origin = mpg_df[["mpg", "displacement", "horsepower", "weight", "acceleration", "origin"]].groupby(by=['
group_by_cluster = mpg_df[["mpg", "displacement", "horsepower", "weight", "acceleration", "cluster"]].groupby(by=
```

In [11]:
```
print('Mean and Variance of our Clusters')
print('\nMean:\n', group_by_cluster.mean())
print('\nVariance:\n', group_by_cluster.var())
```

```
Mean and Variance of our Clusters

Mean:
                 mpg  displacement  horsepower       weight  acceleration
cluster
0         27.365414    131.934211   84.300061  2459.511278     16.298120
1         13.889062    358.093750  167.046875  4398.593750     13.025000
2         17.510294    278.985294  124.470588  3624.838235     15.105882

Variance:
                 mpg  displacement  horsepower         weight  acceleration
cluster
0         41.976309   2828.083391  369.143491  182632.099872      5.718298
1          3.359085   2138.213294  756.521577   74312.340278      3.591429
2          8.829892   2882.492318  713.088674   37775.809263     10.556980
```

In [12]:
```
print('Mean and Variance of our Class Labels (Origin)')
print('\nMean:\n', group_by_origin.mean())
print('\nVariance:\n', group_by_origin.var())
```

```
Mean and Variance of our Class Labels (Origin)

Mean:
               mpg  displacement  horsepower       weight  acceleration
origin
1       20.083534    245.901606  118.814769  3361.931727     15.033735
2       27.891429    109.142857   81.241983  2423.300000     16.787143
3       30.450633    102.708861   79.835443  2221.227848     16.172152

Variance:
               mpg  displacement   horsepower         weight  acceleration
origin
1       40.997026   9702.612255  1569.532304  631695.128385      7.568615
2       45.211230    509.950311   410.659789  240142.328986      9.276209
3       37.088685    535.465433   317.523856  102718.485881      3.821779
```

Let's do a cross tab of our cluster class labels to understand the cluster distribution.

In [13]:
```
pd.crosstab(mpg_df["cluster"], mpg_df["origin"])
```

Out[13]:

origin	**1**	**2**	**3**
cluster			
0	120	67	79
1	64	0	0
2	65	3	0

Conclusion: From the crosstab data of cluster ids and class label it can be observed that almost all records of Origin 2 & 3 along with few records from 1 are clustered in cluster with id 0. This means that the cluster with ids 1 & 2 are majorly sub-clusters of origin 1 records.

Comparing the mean and vairance for our clusters and class labels, it is observed that there is not much similarity in these values. But something that is observed is that since we are doing a Hierarchical Clustering using linkage as average and affinity as euclidean, the clusters in heirarchy minimize upon euclidean(variance) which is why the variance differs significantly from that of class labels. And since our cluster considers almost every origin in cluster 0, we can't get any further important information from the comparison.

Problem 2

In [14]:
```
import sklearn.datasets as dataset
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

In [15]:
```
data = dataset.load_boston()
df = pd.DataFrame(data.data, columns=data.feature_names)
```

In [16]:
```
df
```

Out[16]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88

506 rows × 13 columns

Scale data

In [17]:
```
scaler = StandardScaler()
df = pd.DataFrame(scaler.fit_transform(df.values),columns=df.columns)
```

In [18]:
```
df
```

Out[18]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	-0.419782	0.284830	-1.287909	-0.272599	-0.144217	0.413672	-0.120013	0.140214	-0.982843	-0.666608	-1.459000	0.441052	-1.075562
1	-0.417339	-0.487722	-0.593381	-0.272599	-0.740262	0.194274	0.367166	0.557160	-0.867883	-0.987329	-0.303094	0.441052	-0.492439
2	-0.417342	-0.487722	-0.593381	-0.272599	-0.740262	1.282714	-0.265812	0.557160	-0.867883	-0.987329	-0.303094	0.396427	-1.208727
3	-0.416750	-0.487722	-1.306878	-0.272599	-0.835284	1.016303	-0.809889	1.077737	-0.752922	-1.106115	0.113032	0.416163	-1.361517
4	-0.412482	-0.487722	-1.306878	-0.272599	-0.835284	1.228577	-0.511180	1.077737	-0.752922	-1.106115	0.113032	0.441052	-1.026501
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	-0.413229	-0.487722	0.115738	-0.272599	0.158124	0.439316	0.018673	-0.625796	-0.982843	-0.803212	1.176466	0.387217	-0.418147
502	-0.415249	-0.487722	0.115738	-0.272599	0.158124	-0.234548	0.288933	-0.716639	-0.982843	-0.803212	1.176466	0.441052	-0.500850
503	-0.413447	-0.487722	0.115738	-0.272599	0.158124	0.984960	0.797449	-0.773684	-0.982843	-0.803212	1.176466	0.441052	-0.983048
504	-0.407764	-0.487722	0.115738	-0.272599	0.158124	0.725672	0.736996	-0.668437	-0.982843	-0.803212	1.176466	0.403225	-0.865302
505	-0.415000	-0.487722	0.115738	-0.272599	0.158124	-0.362767	0.434732	-0.613246	-0.982843	-0.803212	1.176466	0.441052	-0.669058

506 rows × 13 columns

Make K-means model for k = 2 to 6 and find silhouette score.

In [19]:
```
score_lst = []

for k in range(2,7):
    model = KMeans(n_clusters=k)
    cache = model.fit_predict(df)
    score = silhouette_score(df,cache)
    score_lst.append(score)
    print(k, "cluster silhouette score is:",score)
```

```
2 cluster silhouette score is: 0.36011768587358606
3 cluster silhouette score is: 0.2574894522739469
4 cluster silhouette score is: 0.28812346566702973
```

```
5 cluster silhouette score is: 0.2817144677423954
6 cluster silhouette score is: 0.298235231885957
```

Since 2 clusters give us the best silhouette score and has small k value, it happens to be the optimal K for our K-means clustering.

In [20]:
```
optimal_model = KMeans(n_clusters = 2)
cache = optimal_model.fit_predict(df)
df["cluster"] = cache
```

coordinates of centroid

In [21]:
```
optimal_model.cluster_centers_
```

Out[21]:
```
array([[-0.39012396,  0.26239167, -0.62036759,  0.00291182, -0.58467512,
         0.24331476, -0.43510819,  0.45722226, -0.58380115, -0.63145993,
        -0.28580826,  0.32645106, -0.44642061],
       [ 0.72514566, -0.48772236,  1.15311264, -0.00541237,  1.086769  ,
        -0.45226302,  0.80876041, -0.8498651 ,  1.0851445 ,  1.1737306 ,
         0.53124811, -0.60679321,  0.82978746]])
```

mean for each feature w.r.t clusters

In [22]:
```
df.groupby(by="cluster").mean()
```

Out[22]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTA
cluster													
0	-0.390124	0.262392	-0.620368	0.002912	-0.584675	0.243315	-0.435108	0.457222	-0.583801	-0.631460	-0.285808	0.326451	-0.4464
1	0.725146	-0.487722	1.153113	-0.005412	1.086769	-0.452263	0.808760	-0.849865	1.085145	1.173731	0.531248	-0.606793	0.8297

Conclusion: We observe that the means of all the features with respect to the clusters happen to be the centroids of the cluster.

Problem 3

In [23]:
```
from sklearn.metrics import homogeneity_score, completeness_score
```

In [24]:
```
data = dataset.load_wine()
df = pd.DataFrame(data.data, columns = data.feature_names)
```

In [25]:
```
df
```

Out[25]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32
...	...	...	...	...	...	...	...	...	...	...
173	13.71	5.65	2.45	20.5	95.0	1.68	0.61	0.52	1.06	7.70
174	13.40	3.91	2.48	23.0	102.0	1.80	0.75	0.43	1.41	7.30
175	13.27	4.28	2.26	20.0	120.0	1.59	0.69	0.43	1.35	10.20
176	13.17	2.59	2.37	20.0	120.0	1.65	0.68	0.53	1.46	9.30
177	14.13	4.10	2.74	24.5	96.0	2.05	0.76	0.56	1.35	9.20

178 rows × 13 columns

Scale data

In [26]:
```
scaler = StandardScaler()
df = pd.DataFrame(scaler.fit_transform(df.values),columns=df.columns)
```

In [27]:
```
df
```

Out[27]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_inte
0	1.518613	-0.562250	0.232053	-1.169593	1.913905	0.808997	1.034819	-0.659563	1.224884	0.25
1	0.246290	-0.499413	-0.827996	-2.490847	0.018145	0.568648	0.733629	-0.820719	-0.544721	-0.29
2	0.196879	0.021231	1.109334	-0.268738	0.088358	0.808997	1.215533	-0.498407	2.135968	0.26
3	1.691550	-0.346811	0.487926	-0.809251	0.930918	2.491446	1.466525	-0.981875	1.032155	1.18
4	0.295700	0.227694	1.840403	0.451946	1.281985	0.808997	0.663351	0.226796	0.401404	-0.31
...	...	...	...	...	...	...	...	...	...	

173	0.876275	2.974543	0.305159	0.301803	-0.332922	-0.985614	-1.424900	1.274310	-0.930179	1.14
174	0.493343	1.412609	0.414820	1.052516	0.158572	-0.793334	-1.284344	0.549108	-0.316950	0.96
175	0.332758	1.744744	-0.389355	0.151661	1.422412	-1.129824	-1.344582	0.549108	-0.422075	2.22
176	0.209232	0.227694	0.012732	0.151661	1.422412	-1.033684	-1.354622	1.354888	-0.229346	1.83
177	1.395086	1.583165	1.365208	1.502943	-0.262708	-0.392751	-1.274305	1.596623	-0.422075	1.79

178 rows × 13 columns

Fit Model

In [28]:
```
model = KMeans(n_clusters = 3)
cache = model.fit_predict(df)
```

Homogeneity score

In [29]:
```
homogeneity_score(cache, data.target)
```

Out[29]:
```
0.8729636016078731
```

Completeness score

In [30]:
```
completeness_score(cache, data.target)
```

Out[30]:
```
0.8788432003662366
```

Conclusion:

- Homogeneity score is the representation of how close cluster(s) are to containing objects from only one class. Since we have a score of 87.29%, our clusters are well formed.
- Completeness score is ratio the assignment of samples belonging to the same class in a cluster. Since we have a score of 87.88% in this, we can infer that our clusters almost represent the complete classes.

Recitation Questions

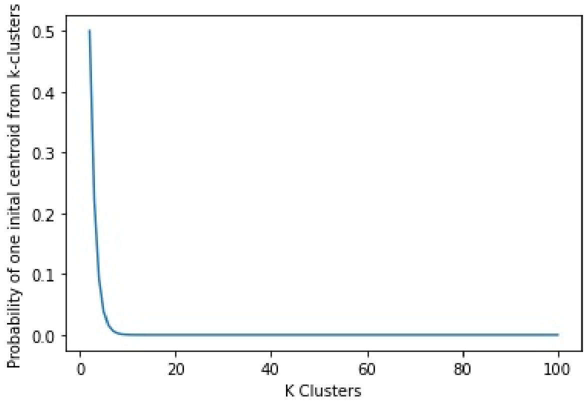
In [15]:
```
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Q4

1.

In [16]:
```
cache = pd.DataFrame([(k, math.factorial(k) * (10**k) / ((k*10) ** k)) for k in range(2, 101)])
cache.columns = ["Clusters", "Probability"]
plt.xlabel("Clusters")
plt.ylabel("Probability of one inital centroid from k-clusters")
plt.plot(results["Clusters"], results["Probability"])
```

Out[16]:
```
[<matplotlib.lines.Line2D at 0x7ff91a269250>]
```



2.

In [17]:
```
cache = pd.DataFrame([(k, math.factorial(k) * ((2*k)**k) / ((k*(2*k)) ** k)) for k in [10, 100, 1000]])
cache.columns = ["Clusters", "Probability"]
print(cache)
```

```
   Clusters   Probability
0        10  3.628800e-04
1       100  9.332622e-43
2      1000  0.000000e+00
```

Q7

More centroids should be allocated to the less dense region because For points spread over a large area, many centroids would help in providing less distance to the points and becoming accurate calculations.

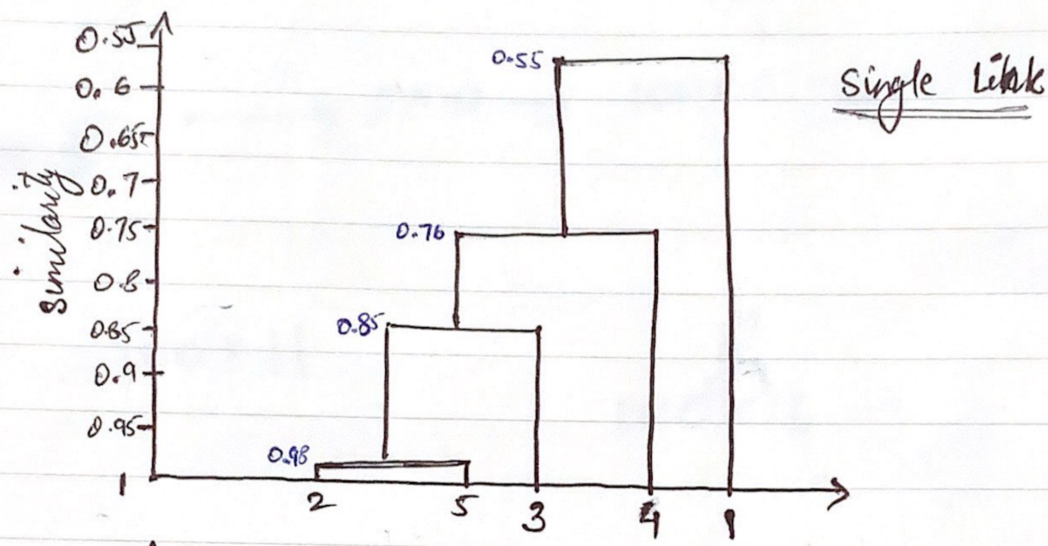
Q11

If the SSE of one attribute is low for all clusters, then the variable is possibly a constant and not much useful in dividing the data into groups, thus it can be dropped. Alternatively if SSE of one attribute is relatively low for just one cluster, then that attribute helps to define the cluster.

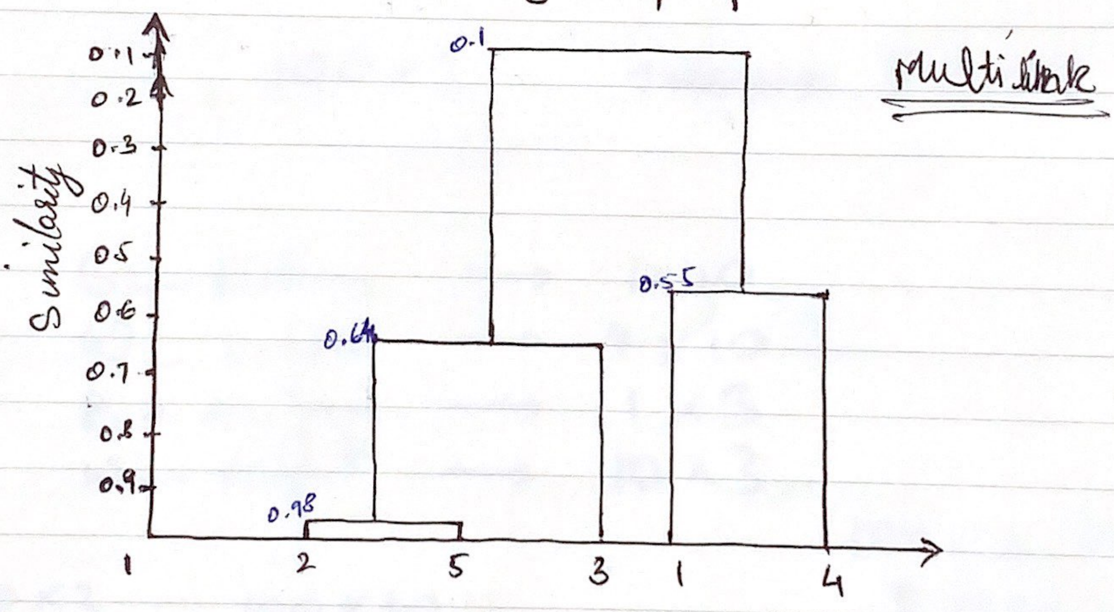
If the SSE of an attribute is relatively high for all clusters, then it might mean that the attribute is random/noise. Alternatively if SSE of an attribute is relatively high for one cluster, then it is not at par with the information provided by the attributes with low SSE that define the cluster. This attribute does not help define the cluster.

By eliminating attributes with low or high SSE for all clusters, which are useless for clustering, we can improve the clustering.

16.



Single Link
Similarity
0.55
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95
1
0.55
0.76
0.85
0.98
2
5
3
4
1



Multi link
Similarity
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0.1
0.55
0.64
0.98
2
5
3
1
4

17.

a) i. $\{18, 45\}$

→ 1^{st} cluster, 18 : $\{6, 12, 18, 24, 30\}$

$$\text{Error} = (18-6)^2 + (18-12)^2 + (18-18)^2 + (24-18)^2 + (30-18)^2$$

$$\text{Error} = 12^2 + 6^2 + 0^2 + 6^2 + 12^2$$
$$= 288 + 72$$
$$= 360$$

→ 2^{nd} Cluster, 45 : $\{42, 48\}$

$$\text{Error} = 3^2 + 3^2 = 18$$

$$\text{Total error} = 360 + 18 = 378$$

17. a) ii. {15, 40}

- First cluster {6, 12, 18, 24}

Error $= (15-6)^2 + (12-15)^2 + (18-15)^2 + (24-15)^2$
$= 9^2 + 3^2 + 3^2 + 9^2$
$= 180$

- Second Cluster {30, 42, 48}

Error $= (40-30)^2 + (42-40)^2 + (48-40)^2$
$= 10^2 + 2^2 + 8^2$
$= 100 + 4 + 64$
$= 168$

Total error $= 180 + 168$
$= 348$

17. b) Yes, both centroids are stable solutions.

17. c) The two clusters produced by single link are
1 : {6, 12, 18, 24, 30}
2 : {42, 48}

Date / / No.

17. d) clusters by K-Means are

{6, 12, 18, 24} & {30, 42, 48}

for K-means distance between centroids is 25.

While for single link it is 27.

Thus single link seems to produce the "most ~~exact~~ natural" clustering.

17. e) Single link produces ~~the~~ contiguous clusters. Although centerbased & density would also be right answers.

17. f) K-means tries to minimize upon the ~~eucl~~ eucleadean distance and thus is not good at finding clusters of different sizes, specially when they are not separated. It breaks the larger cluster.

Date / / No.

21. **Cluster 1:**

Purity $= 676/693 = \underline{0.98}$

Entroy $= -\left[\frac{1}{693} \log_2\left(\frac{1}{693}\right) + \frac{1}{693} \log_2\left(\frac{1}{693}\right) + 0 + \frac{11}{693} \log_2\left(\frac{11}{693}\right) + \frac{4}{693} \log_2\left(\frac{4}{693}\right) + \frac{676}{693} \log_2\left(\frac{676}{693}\right) \right]$

$= \underline{0.20}$

Cluster 2:

Purity $= 827/1562 = \underline{0.53}$

Entropy $= -\left[\frac{27}{1562} \log_2\left(\frac{27}{1562}\right) + \frac{89}{1562} \log_2\left(\frac{89}{1562}\right) + \frac{333}{1562} \log_2\left(\frac{333}{1562}\right) + \frac{827}{1562} \log_2\left(\frac{827}{1562}\right) + \frac{253}{1562} \log_2\left(\frac{253}{1562}\right) + \frac{33}{1562} \log_2\left(\frac{33}{1562}\right) \right]$

$= \underline{1.84}$

Cluster 3:

$$\text{Purity} = 465 / 949 = 0.49$$

$$\text{Entropy} = -\Big[\frac{326}{949} \log_2\left(\frac{326}{949}\right) + \frac{465}{949} \log_2\left(\frac{465}{949}\right) + \frac{8}{949} \log_2\left(\frac{8}{949}\right) + \frac{105}{949} \log_2\left(\frac{105}{949}\right) + \frac{16}{949} \log_2\left(\frac{16}{949}\right) + \frac{29}{949} \log_2\left(\frac{29}{949}\right) \Big]$$

$$= 1.70$$

22. (a) Yes, random points can have different density in certain regions, while uniform one is evenly spread-out.

(b) The random points will have a smaller SSE as it can have tight regions.

(c) It will merge all points in one cluster or classify them as noise depending on threshold. Although it can always find clusters in random data due to variation in density.