MMAI5040 – W21 –Group3 – Assignment2

]Q1:

a)

For association rules, we considered only frequent item sets, based on the purchases of online courses. The criterion for frequency is support, i.e. the percent of purchases that include an itemset composed of courses: Intro, DataMining, Survey, Cat Data, Regression, Forecast, DOE and SW. Here, we used Apriori Algorithm to generate frequent item sets by generating all rules that meet specified minimum support.

I firstly create frequent itemsets and consider that frequent items are not enough. Therefore, I use min_support to generate enough rules.

	support	itemsets
0	0.394521	(Intro)
1	0.178082	(DataMining)
2	0.186301	(Survey)
3	0.208219	(Cat Data)
4	0.208219	(Regression)
5	0.139726	(Forecast)
6	0.172603	(DOE)
7	0.221918	(SW)
8	0.054795	(Intro, DataMining)
9	0.060274	(Intro, Survey)
10	0.071233	(Intro, Cat Data)

And apply the rules produced by apriori.

```
# Rules produced by apriori
rules = association_rules(itemsets , metric='confidence', min_threshold = 0.5)
rules.sort_values(by=['lift'], ascending=False).head(20)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
15	(DataMining, Regression)	(Cat Data)	0.043836	0.208219	0.027397	0.625000	3.001645	0.018270	2.111416
11	(Intro, DOE)	(SW)	0.046575	0.221918	0.030137	0.647059	2.915759	0.019801	2.204566
14	(Cat Data, Regression)	(DataMining)	0.054795	0.178082	0.027397	0.500000	2.807692	0.017639	1.643836
17	(Survey, Forecast)	(Cat Data)	0.038356	0.208219	0.021918	0.571429	2.744361	0.013931	1.847489
16	(Cat Data, Forecast)	(Survey)	0.043836	0.186301	0.021918	0.500000	2.683824	0.013751	1.627397
13	(Cat Data, DataMining)	(Regression)	0.049315	0.208219	0.027397	0.555556	2.668129	0.017129	1.781507
7	(Intro, DOE)	(Regression)	0.046575	0.208219	0.024658	0.529412	2.542570	0.014960	1.682534
3	(Intro, Survey)	(SW)	0.060274	0.221918	0.032877	0.545455	2.457912	0.019501	1.711781
9	(Intro, Regression)	(SW)	0.071233	0.221918	0.038356	0.538462	2.426401	0.022548	1.685845
0	(Intro, DataMining)	(Regression)	0.054795	0.208219	0.027397	0.500000	2.401316	0.015988	1.583562

Considering the top 10 rules with highest lift, and its support & confidence, we come up with these best rules (antecedent -> consequent):
(Regression, DataMining) -> Cat Data
(Intro, DOE) -> SW

(Regression, Cat Data) -> DataMining
(Survey, Forecast -> Cat Data
(Cat Data, Forecast) -> Survey

b)
First, I find the user who purchased both regression and forecast courses. That are No.21 and No.80.

	Intro	DataMining	Survey	Cat Data	Regression	Forecast	DOE	SW
21	0	0	0	0	1	1	0	0
80	0	0	0	0	1	1	0	0

For user-based collaborative Filtering, we created rows of triplets as userID, the course ID (name of course) and if course is purchased as its rating.

And convert the dataset into the format required by the surprise package and choose the rating scale from 0 to 1.
Apply the user-based filtering algorithm and compute cosine similarity between users.

Assuming the user we found (user No.21) in the first step who has purchased regression and forecast courses, and find the recommendation for this student.

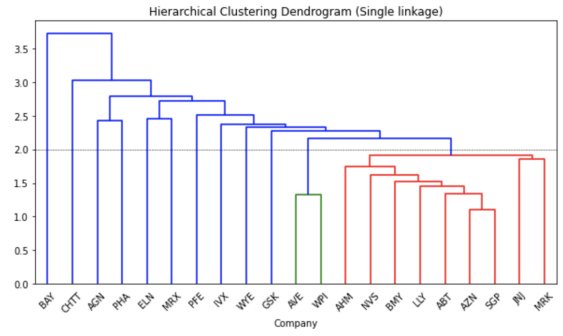
```
# Assuming that userID=21 is a new user, who has purchased Regression and Forecast courses
print('Intro:', algo.predict(21,'Intro').est,
'DataMining:',algo.predict(21,'DataMining').est,
'Survey:',algo.predict(21,'Survey').est,
'Cat Data:',algo.predict(21,'Cat Data').est,
'DOE:',algo.predict(21,'DOE').est,
'SW:',algo.predict(21,'SW').est)
```

```
Intro: 0.0965082604954195 DataMining: 0.10024073045959112 Survey: 0 Cat Data: 0.07611366533573624 DOE: 0.0278595350
8802649 SW: 0.05198660021188137
```

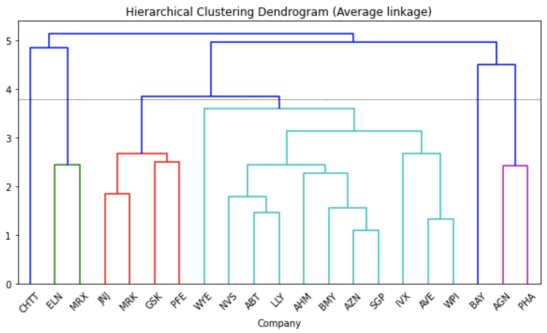
Based on the rating, the best item with the highest rating is DataMining. Therefore, the recommendation for the new user is DataMining. All recommendations will be 1 because we created the rating from 0 to 1 based on the binary variable 'purchase?', i.e. if purchase is 1 or 0.

Q2:

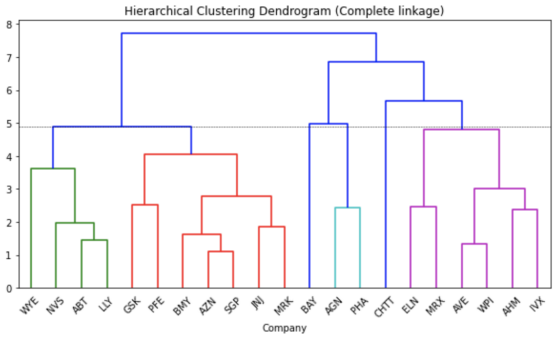
A)
After reading data from Pharmaceuticals.csv into dataframe, and deleting last categorical columns, the normalized euclidean distance between each two company calculated and stored in d_norm. Because there is no priority of one feature over the other, all features have same weight for clustering.
We clustered the data with hierarchical and K-mean algorithms. For Hierarchical the data was clustered using Single, Complete, Average, and Centroid Linkage method.



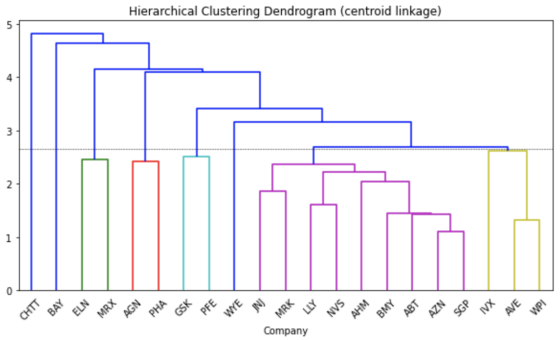
Short Linkage



Average Linkage



Complete Linkage

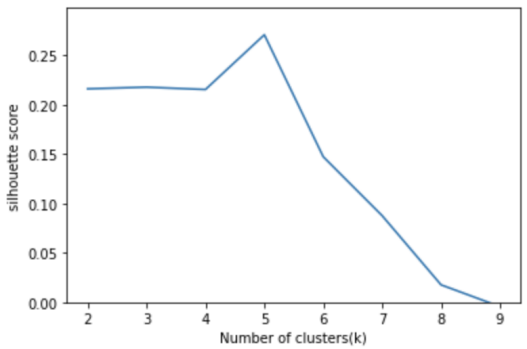


Centroid Linkage

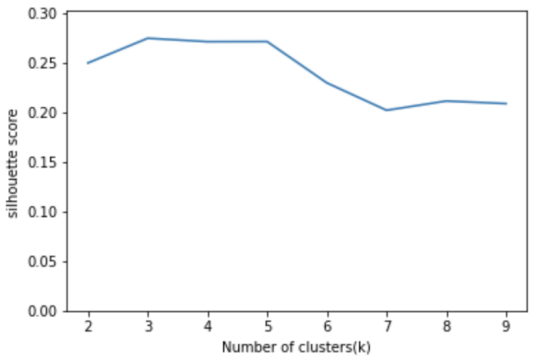
Also k-mean were executed on the data for K from 3 to 10.

For comparing these methods and select the optimum number of clusters, we chose the **Silhouette Score.** This criteria get maximized when the distances between nodes within a cluster are close while the distance between clusters are far.

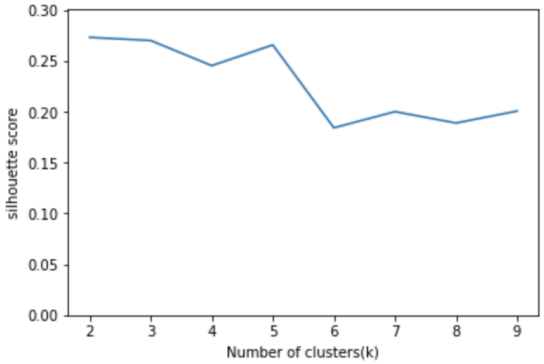
For each of methods, the **Silhouette Score** were calculated for number of clusters of 3 to 10, and the **Silhouette Score** drawn on a chart based on the number of clusters.



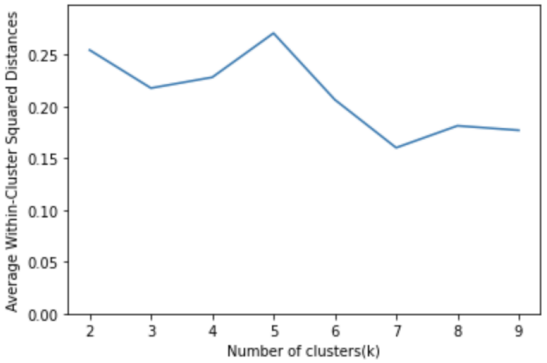
Short Linkage



Average Linkage



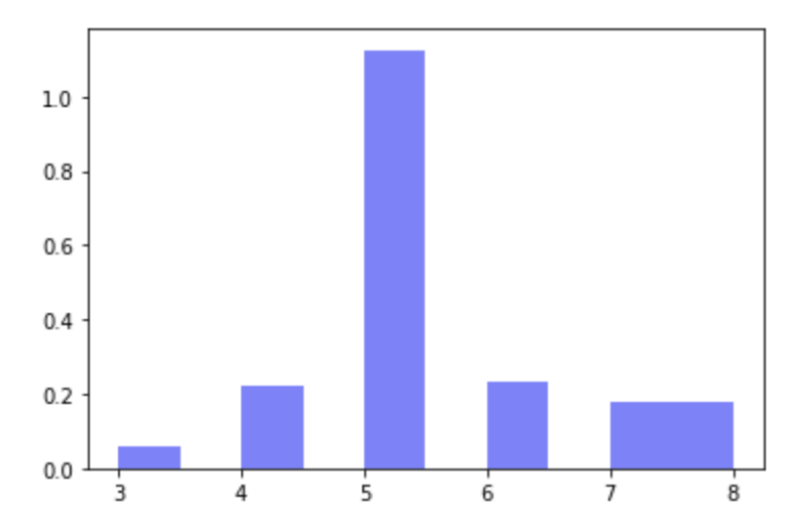
Complete Linkage



Centroid Linkage

As we can see, in all methods, number of **clusters of 3 and 5** have the highest score. Also the Average Linkage overally, for all number of clusters, cluster data with more **Silhouette Score.**

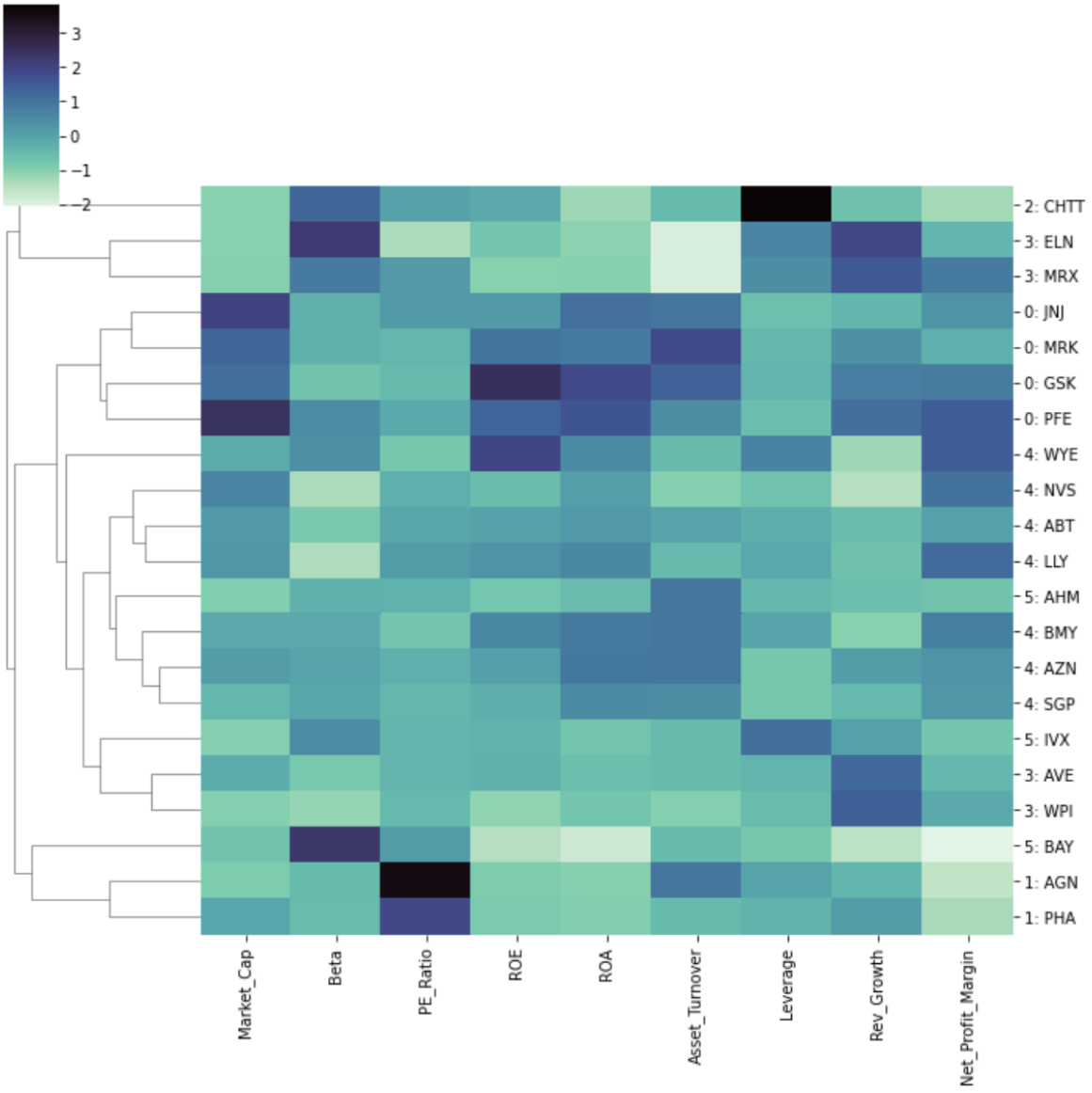
Also for the K-Mean method the evaluation was run for K of 3 to 10 and because this method highly depends on the initial random clustering of nodes, for each K, clustering executed 300 times. The maximum score for each run was stored in an array and the histogram of that array plotted.



This histogram also shows that the optimal number of cluster is 5. But because the number of companies are limited in this problem, and the complexity of algorithm doesn't matter with this number of nodes, we chose the **Average Linkage Method** which return single result for every time that we run the clustering. Also it cluster data with most **Silhouette Score.**

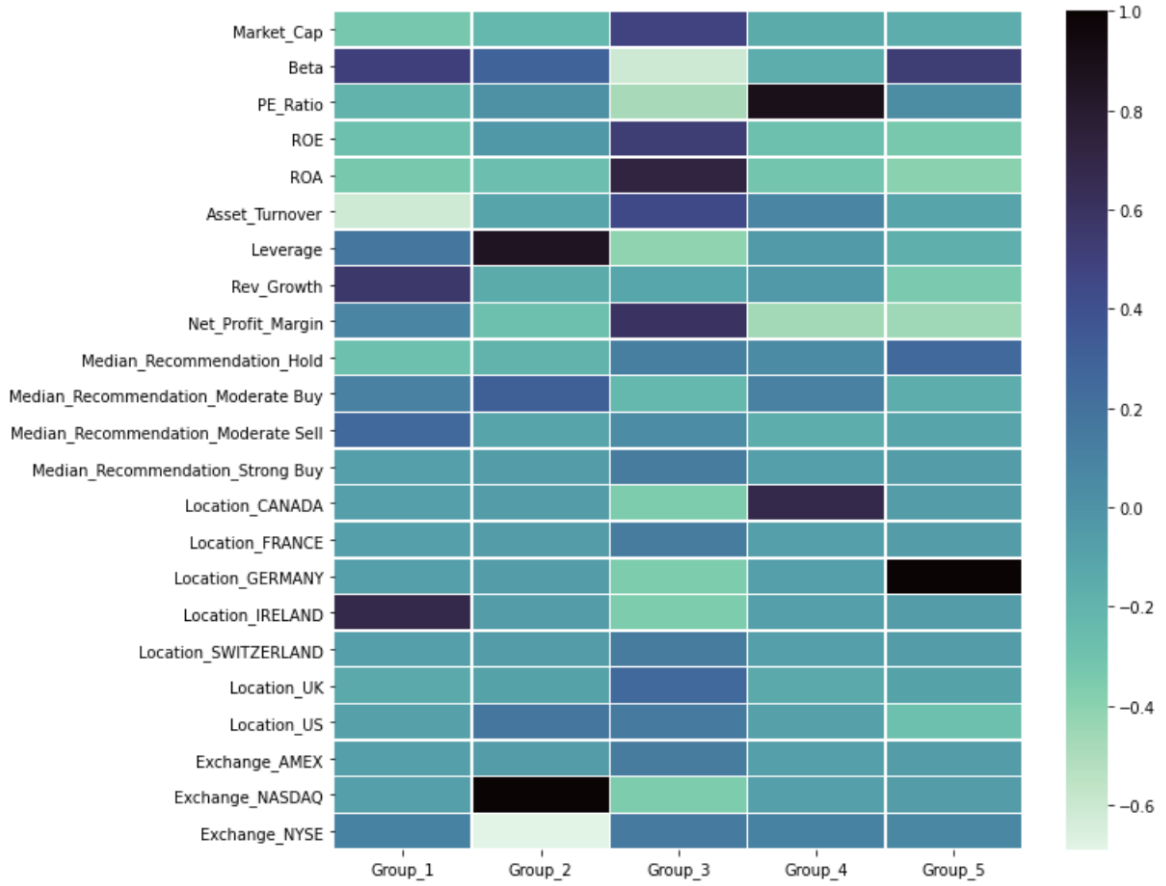
Average linkage cluster membership:

```
1 :  ELN, MRX
2 :  CHTT
3 :  ABT, AHM, AZN, AVE, BMY, LLY, GSK, IVX, JNJ, MRK, NVS, PFE, SGP, WPI, WYE
4 :  AGN, PHA
5 :  BAY
silhouette score:0.2708859373526622
```



B)

The correlations of each feature with each group were plotted in a heatmap.



It shows that:
Stocks in Group 1 have high Beta and Revenue Growth and Assets Turnover.
Stocks in Group 2 have high Leverage and relatively high beta
Stocks in Group 3 have high ROA, ROE, Market Cap,Net Profit Margin. and Low Beta
Stocks in Group 4 have high PE Ratio and Low Net Profit
Stocks in Group 5 have high Beta and Low Net Profit

C)
Based on the heatmap in section B,
The Stocks in Group 1 are mostly in Irland, Stocks in Group 4 are mostly in Canada, and Stocks in Group 5 are mostly in Germany, The majority of Group 2 are in US and majority of group 3 are in Swiss, UK, France and US.

D)
Group 1 : High Risk / High Growth.
Group 2 : High leverage.
Group 3 : Large Market cap with low growth, High return on Equity and Assets (RoE, RoA) and High Turnover with High Net Profit Margin.
Group 4 : High Price to Earning Ratio with Medium Growth , and Low Profit (Expensive Stocks based on the growth)
Group 5 : High Risk , Low Growth and Profit

Q3:

Preprocess the data by removing all cereals with missing values

```
[5] cereals_df = pd.read_csv('Cereals.csv')
    cereals_df=cereals_df.dropna()
    cereals_df=cereals_df.drop(columns=['mfr', 'type'])
    print(cereals_df.head())
    cereals_df.set_index('name', inplace=True)
    print (cereals_df.dtypes)
```

```
                        name  calories  protein  ...  weight  cups     rating
0                  100%_Bran        70        4  ...     1.0  0.33  68.402973
1          100%_Natural_Bran       120        3  ...     1.0  1.00  33.983679
2                   All-Bran        70        4  ...     1.0  0.33  59.425505
3  All-Bran_with_Extra_Fiber        50        4  ...     1.0  0.50  93.704912
5    Apple_Cinnamon_Cheerios       110        2  ...     1.0  0.75  29.509541

[5 rows x 14 columns]
calories      int64
protein       int64
fat           int64
sodium        int64
fiber       float64
carbo       float64
sugars      float64
potass      float64
vitamins      int64
shelf         int64
weight      float64
cups        float64
rating      float64
dtype: object
```

```
[6] cereals_df['calories'] = cereals_df['calories'].astype(float)
    cereals_df['protein'] = cereals_df['protein'].astype(float)
    cereals_df['fat'] = cereals_df['fat'].astype(float)
    cereals_df['sodium'] = cereals_df['sodium'].astype(float)
    cereals_df['vitamins'] = cereals_df['vitamins'].astype(float)
    cereals_df['shelf'] = cereals_df['shelf'].astype(float)



    print(cereals_df.dtypes)
    #changing all datatypes to float
```

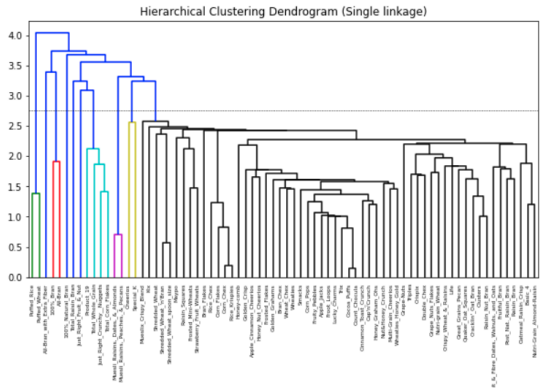
```
calories    float64
protein     float64
fat         float64
sodium      float64
fiber       float64
carbo       float64
sugars      float64
potass      float64
vitamins    float64
shelf       float64
weight      float64
cups        float64
rating      float64
dtype: object
```

I removed all columns with missing values and dropped the MFR and TYPE columns to get all integer values. I also changed all data types from integer to float.

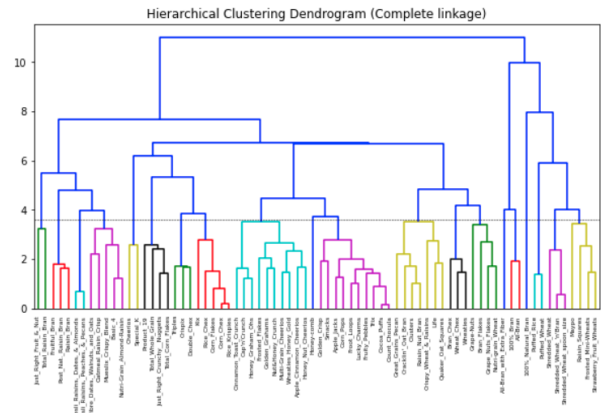
a. Apply hierarchical clustering to the data using Euclidean distance to the normalized measurements. Compare the dendrograms from single linkage and complete linkage and look at cluster centroids. Comment on the structure of the clusters and their stability. (Hint: To obtain cluster centroids for hierarchical clustering, computer the average values of each cluster members, using groupby( ) with the cluster centers followed by mean: dataframe.groupby(clusterlabel).mean().)

```
[11] #single linkage hierachal clustering
     Z = linkage(cereals_df_norm, method='single')

     fig = plt.figure(figsize=(10, 6))
     fig.subplots_adjust(bottom=0.23)
     plt.title('Hierarchical Clustering Dendrogram (Single linkage)')
     plt.xlabel('name')
     dendrogram(Z, labels=cereals_df_norm.index, color_threshold=2.75)
     plt.axhline(y=2.75, color='black', linewidth=0.5, linestyle='dashed')
     plt.show()
```




```
#complete linkage hierachal clustering
Z = linkage(cereals_df_norm, method='complete')
fig = plt.figure(figsize=(10, 6))
fig.subplots_adjust(bottom=0.23)
plt.title('Hierarchical Clustering Dendrogram (Complete linkage)')
plt.xlabel('Company')
dendrogram(Z, labels=cereals_df_norm.index, color_threshold=3.6)
plt.axhline(y=3.6, color='black', linewidth=0.5, linestyle='dashed')
plt.show()
```



Based on the dendrograms, there are more cereals in each cluster overall for complete linkage than than for single linkage. For single linkage, there are 2 clusters (cluster 1 and 2) with 2 cereals each and 2 clusters (cluster 3 and 6) with 1 cereal each, and the rest of the cereals are in clusters 4 and 5. For complete linkage, only cluster 6 has 1 cereal, and the rest of the clusters have more than 2 cereals in each cluster. All bran, all bran with extra fibre, and 100% natural bran, are all grouped into one cluster in complete linkage whereas in single linkage, 100% natural bran is seperate from the two and in its one cluster. I noticed that 100% natural bran is always its own cluster in both single and complete linkage.

```
[14] #single linkage
     memb_df=pd.DataFrame(memb)
     memb_df=memb_df.rename(columns={0:'label'})
     memb_norm=pd.concat([cereals_df_norm,memb_df],axis=1)
     centroids=memb_norm.groupby(['label']).mean()
     centroids
```

label	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf	weight	cups	rating
1	-2.873782	-0.942101	-9.932203e-01	-1.961644	-0.691459	-0.829907	-1.630632	-0.931359	-1.303202	0.941971	-3.459955	0.756753	1.390159
2	-1.865915	1.381748	0.000000e+00	0.394288	3.022345	-2.243181	-0.368907	2.842695	-0.181842	0.941971	-0.200832	-2.085658	1.535050
3	-2.873782	1.381748	-9.932203e-01	-0.270206	4.879247	-1.729263	-1.630632	3.265954	-0.181842	0.941971	-0.200832	-1.364449	3.657844
4	0.485774	0.142362	-1.655367e-01	0.555377	-0.141266	0.883151	-0.177737	-0.037814	3.182239	0.941971	0.668267	0.579987	-0.304512
5	0.141690	-0.057539	3.581365e-18	0.029888	-0.139047	0.069448	0.104702	-0.118976	-0.236102	-0.182317	0.059897	-0.003458	-0.114250
6	0.653751	0.452208	3.972881e+00	-1.780419	-0.072492	-1.729263	0.204604	0.514774	-1.303202	0.941971	-0.200832	0.756753	-0.597711

```
[17] #Complete linkage
     memb_df=pd.DataFrame(memb)
     memb_df=memb_df.rename(columns={0:'label'})
     memb_norm=pd.concat([cereals_df_norm,memb_df],axis=1)
     centroids=memb_norm.groupby(['label']).mean()
     centroids
```

label	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf	weight	cups	rating
1	1.409651	0.607132	0.662147	0.218097	0.563669	0.230048	0.873701	1.037968	0.378838	0.841819	1.673163	-0.268495	-0.442940
2	0.005837	0.120230	-0.496610	0.998373	-0.573560	1.280825	-0.926033	-0.639109	0.779324	-0.174010	-0.200832	0.871904	0.069518
3	0.005837	-0.371098	0.113511	0.061178	-0.267024	-0.404090	0.414345	-0.341821	-0.181842	-0.259854	-0.200832	-0.149909	-0.429404
4	-2.201871	1.381748	-0.331073	0.172790	3.641312	-2.071875	-0.789482	2.983781	-0.181842	0.941971	-0.200832	-1.845255	2.242648
5	-1.249997	-0.064202	-0.882862	-1.941508	-0.026642	0.155101	-1.095355	-0.112276	-0.804820	-0.259854	-1.048204	0.115679	1.471215
6	0.653751	0.452208	3.972881	-1.780419	-0.072492	-1.729263	0.204604	0.514774	-1.303202	0.941971	-0.200832	0.756753	-0.597711

The structure of the clusters is that there are the same number of clusters (6 clusters) for both single and complete linkage. For single linkage, there are 2 clusters (cluster 1 and 2) with 2 cereals each and 2 clusters (cluster 3 and 6) with 1 cereal each, and the rest of the cereals are in clusters 4 and 5. For complete linkage, only cluster 6 has 1 cereal, and the rest of the clusters have more than 2 cereals in each cluster. Complete linkage is better for stability because the distances between splits are larger and there are more cereals in each cluster and the cereals won't change much between clusters by changing measurements. Single linkage is not that stable because by changing measurements, cereals can easily move between clusters.

b. Which method leads to the most meaningful clusters?
The method leading to the most meaningful clusters is complete linkage because based on the dendrogram, there aren't a bunch of clusters squished together and complete linkage is more stable because the distances between splits are larger.

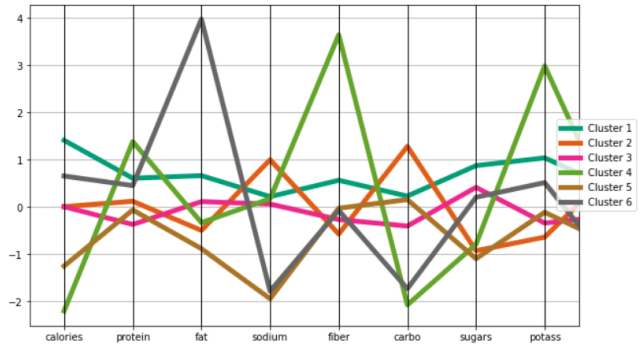
c. Choose one of the methods. How many clusters would you use? What distance is used for this cutoff? (Look at the dendrogram.)
For the complete linkage there are 6 clusters and distance is 3.5 for the cutoff.

d. The elementary public schools would like to choose a set of cereals to include in their daily cafeterias. Every day a different cereal is offered, but all cereals should support a healthy diet. For this goal, find a cluster of "healthy cereals". Should the data be normalized? If not, how should they be used in the cluster analysis?

```
[ ] centroids['cluster'] = ['Cluster {}'.format(i) for i in centroids.index]

    plt.figure(figsize=(10,6))
    fig.subplots_adjust(right=3)
    ax = parallel_coordinates(centroids, class_column='cluster', colormap='Dark2', linewidth=5)
    plt.legend(loc='center left', bbox_to_anchor=(0.95, 0.5))
    plt.xlim(-0.5,7.5)
```

```
(-0.5, 7.5)
```



The healthiest cluster is cluster 4 as it is low in sugar, lowest in calories, lowest in carbohydrates, highest in protein, highest in fibre, and highest in potassium. Yes the data should be normalized because Euclidean distance is very sensitive to the changes in the differences and normalizing is essential to do before clustering.

Q4:

a. Use predictors Age_08_04, KM, Fuel_Type, HP, Automatic, Doors, Qarterly_Tax, Mfr_Guarantee, Guarantee_Period, Airco, Automatic_airco, CD_Player, Powered_Windows, Sport_Model, and Tow_Bar.

Used unnormalized data to predict the outcome using a single hidden layer with two nodes. Part of the predicted values is in the screenshot below:

```
array([ 9236.13264104, 14368.73945594, 11111.07269109,  3334.4369458 ,
       12893.75074012,  8510.84601753,  8120.68565449, 14573.7861481 ,
        4283.74929392, 10716.90001032,  6903.84263445,  8134.56948895,
        9597.40437801,  4906.2935626 ,  3828.30686088,  3912.02826081,
        8838.93455511,  6918.57089723,  8764.90065248,  8325.33120298,
        8404.18920265,  4416.41828654,  9647.11103773,  8544.29653142,
        6259.54463476,  7682.77546071,  5164.27471808,  6318.08341162,
        4963.3006877 ,  4700.99761383, 16366.40448678,  8366.94149819,
        3913.59599334,  6095.09674006,  3210.25666002,  5564.13450159,
        6679.26620367, 12239.2804427 , 13845.31361004,  3113.75005181,
       14400.98574659,  8211.77920422,  7291.01153982,  7005.27638823,
       14382.65572478,  4090.63459185,   997.60786105,  7156.98671082,
        7484.24856865,  7486.09047753,  6833.65577443, 12743.31206642,
        7215.16451995, 12487.1917332 ,  8479.70159009,  3668.31313593,
        5791.69173946,  4781.15103126,  4333.96730338, 21393.72303076,
        4707.94413034,  4948.35662022, 11421.05661555,  6735.03764602,
        5934.81080304,  4495.18433435,  7041.6675942 ,  2528.6314269 ,
```

b. Use the scikit-learn transformer MinMaxScaler() to scale the data to the range [0,1]. Use separate transformers for the input and output data. To create the dummy variables, use the pandas function pd.get_dummies().
The codes to generate such requirement is in the screenshot below:

```
[ ] min_max_scaler = preprocessing.MinMaxScaler()
    X_minmax = min_max_scaler.fit_transform(X)
    y_minmax = min_max_scaler.fit_transform(y.values.reshape(-1,1))
```

```
[ ] train_X2, valid_X2, train_y2, valid_y2 = train_test_split(X_minmax, y_minmax, test_size=0.4, random_state=1)
    print(type(train_y2))

    <class 'numpy.ndarray'>
```

c.Record the RMS error for the training data and the validation data.
The RMS of training data is:
0.34392802964796804
The RMS of validation data is:
0.4162694814518107

d.Repeat the process, changing the number of hidden layers and nodes to {single layer with five nodes}, {two layers, five nodes in each layer}.

i.What happens to the RMS error for the training data as the number of layers and nodes increase?

The RMS of training data original is:
0.34392802964796804

Single layer with five nodes
The RMS of training data is:
0.4308163114892777

two layers, five nodes in each layer
The RMS of training data is:
0.3664391964102709

ii.What happens to the RMS error of the validation data?

The RMS of validation data original is:
0.4162694814518107

single layer with five nodes
The RMS of validation data is:
0.3694974297591548

two layers, five nodes in each layer

The RMS of validation data is:
0.38182047894579835

iii. Comment on the appropriate number of layers and nodes for this application.

Based on the results, the single layer with five nodes, with activation function relu and solver adam has the lowest RMS on validation data. This would be the most appropriate only based on RMS comparison to the other two with their parameters. However the parameter change in each model could generate different outcomes, the answer to this question could change accordingly as well. So, a more appropriate model should do hyperparameter tuning in each model to get the best parameter and compare.