Assignment 3

Ao Peng 300145650

Part A. Association Analysis

1. This part is to load data which is from the *groceries* dataset and do the preprocessing part. I removed all null values and then fulfill the Apriori algorithm.

Data Loading and Preprocessing

In [263]:
```
with open("C:\\Users\\admin\\Desktop\\work\\assignment\\Dataset 3\\groceries_new.csv", 'r+') as f:
    content = f.read()
    f.seek(0,0)
    str1 = ""
    for i in range(32):
        if (i<31):
            str1 = str1+str(i)+','
        else:
            str1 = str1+str(i)+'\n'
    f.write(str1+content)

```

In [284]:
```
df1 = pd.read_csv(".\\Dataset 3\\groceries_new.csv")
df1 = df1.fillna(0)
test1 = df1.values.tolist()
```

In [286]:
```
for i in range(len(test1)):
    test1[i] = list(set(test1[i]))
    for j in test1[i]:
        if (j):
            continue
        else:
            test1[i].remove(j)
```

In [316]:
```
def newApriori(df, min_sup, min_conf):
    '''
    df: dataset
    min_sup: minimum support
    min_conf: minimum confidence
    '''
    te = TransactionEncoder()
    te_ary = te.fit(df).transform(df)
    df = pd.DataFrame(te_ary, columns=te.columns_)
    fi = apriori(df, min_support=min_sup, use_colnames=True, max_len=3,verbose=0)
    rule = association_rules(fi, metric="confidence", min_threshold=min_conf)
#     rule.sort_values('lift', ascending=False).head(5)
#     print(rule.sort_values('lift', ascending=False).head(5))
    return rule
```

2. Get the best rule with the minimum support of 0.5% minimum confidence of 50%, and length of 2. The support table shows below.

	support	itemsets
0	0.008032	(Instant food products)
1	0.033449	(UHT-milk)
2	0.017690	(baking powder)
3	0.052460	(beef)
4	0.033245	(berries)

The best rule is

(tropical fruit, curd)---->(yogurt)

The lift is about 3.691.

```
r1 = newApriori(test1, 0.005, 0.5)
r1.sort_values('lift',ascending=False).head()
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
34	(tropical fruit, curd)	(yogurt)	0.010269	0.139502	0.005287	0.514851	3.690645	0.003855	1.773680
67	(pip fruit, whipped/sour cream)	(other vegetables)	0.009253	0.193493	0.005592	0.604396	3.123610	0.003802	2.038671
63	(root vegetables, onions)	(other vegetables)	0.009456	0.193493	0.005694	0.602151	3.112008	0.003864	2.027167
22	(citrus fruit, root vegetables)	(other vegetables)	0.017692	0.193493	0.010371	0.586207	3.029608	0.006948	1.949059
72	(tropical fruit, root vegetables)	(other vegetables)	0.021047	0.193493	0.012303	0.584541	3.020999	0.008231	1.941244

3. Get the best rule minimum support of 0.1%, minimum confidence of 50% and length of 2.

	support	itemsets
0	0.008032	(Instant food products)
1	0.033449	(UHT-milk)
2	0.003558	(abrasive cleaner)
3	0.003253	(artif. sweetener)
4	0.017690	(baking powder)

The best rule is

(Instant food products, soda)---->(hamburger meat)

The lift is about 18.995.

```
r2 = newApriori(test1, 0.001, 0.5)
r2.sort_values('lift',ascending=False).head()
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
11	(Instant food products, soda)	(hamburger meat)	0.001932	0.033249	0.001220	0.631579	18.995654	0.001156	2.624040
1358	(soda, popcorn)	(salty snack)	0.001932	0.037824	0.001220	0.631579	16.697793	0.001147	2.611620
32	(baking powder, flour)	(sugar)	0.001830	0.033859	0.001017	0.555556	16.408075	0.000955	2.173818
980	(ham, processed cheese)	(white bread)	0.003050	0.042095	0.001932	0.633333	15.045491	0.001803	2.612469
12	(whole milk, Instant food products)	(hamburger meat)	0.003050	0.033249	0.001525	0.500000	15.038226	0.001424	1.933503

4. I will choose the **second** rule. The main reason is based on the definitions of these 3 measures which are the support, confidence and lift. Support is a probability of the number of times this rule appears which is to judge whether it

should be considered as a possible connection or not. If the rule does not satisfy the minimum support, it will not be considered anymore. Confidence is the measure to indicate the probability of the connection among these commodities. Of course, the higher the value of confidence is, the stronger the connection is. Lift is the combination of these two measures. It illustrates the connection between antecedent and consequent. When lift = 1, these 2 conditions are **independent**. When lift > 1, they have a positive dependence between them.

$$lift(A -> C) = confidence(A -> C) / support(C)$$

Based on these definitions, I will choose the second rule as it has a higher lift value which is more possible to be a possible rule.

Part B Clustering

1. Get the 2 SSE and compare them.
   Firstly, I create the original data set and get the initial mean point (centroid). The initial SSE is **7.5**.

Get the initial centroid for cluseter_A & cluseter_B

n [2]:

```
cluster_A = [[1, 0], [2, 1]]
cluster_B = [[0, 1], [3, 3]]

def SSE(a, b):
    A = np.array(a)
    B = np.array(b)
    A_Mean, B_Mean = cal_Mean(A, B)
    e1 = e2 = 0
    for i in A:
        e1 += np.sum((i-A_Mean)**2)
    for i in B:
        e2 += np.sum((i - B_Mean)**2)
    sse = e1 + e2
    return sse

def cal_Mean(a, b):
    s_a = s_b = 0
    for i in a:
        s_a += i
    aMean = s_a / len(a)
    for j in b:
        s_b += j
    bMean = s_b / len(b)
    return aMean, bMean

print(SSE(cluster_A, cluster_B))
```

```
7.5
```

Then this is the first iteration. The SSE is almost **2.667.** The algorithm split this dataset into 2 clusters. “t1” is the label of clusters and the value is 1 belonging to c1 while the c2 includes the data whose value is 0.

n [3]:
```
1 X = [[1, 0], [0, 1], [2, 1], [3, 3]]
```

Create K-Means model

n [4]:
```
1 km = KMeans(n_clusters=2, max_iter=1)
```

The first iteration

n [5]:
```
1 km.fit(X)
2 t1 = km.predict(X)
3 t1
```

ut[5]:
```
array([1, 1, 1, 0], dtype=int32)
```

n [6]:
```
1 c1 = [[1, 0], [0, 1], [2, 1]]
2 c2 = [[3, 3]]
3 print(SSE(c1, c2))
```

```
2.666666666666667
```

The final step is to get the result after the second iteration. The SSE is the same as the former one which is **2.667**. However, the label changed.

The second iteration

:
```
1 km.fit(X)
2 t2 = km.predict(X)
3 t2
```

:
```
array([0, 0, 0, 1], dtype=int32)
```

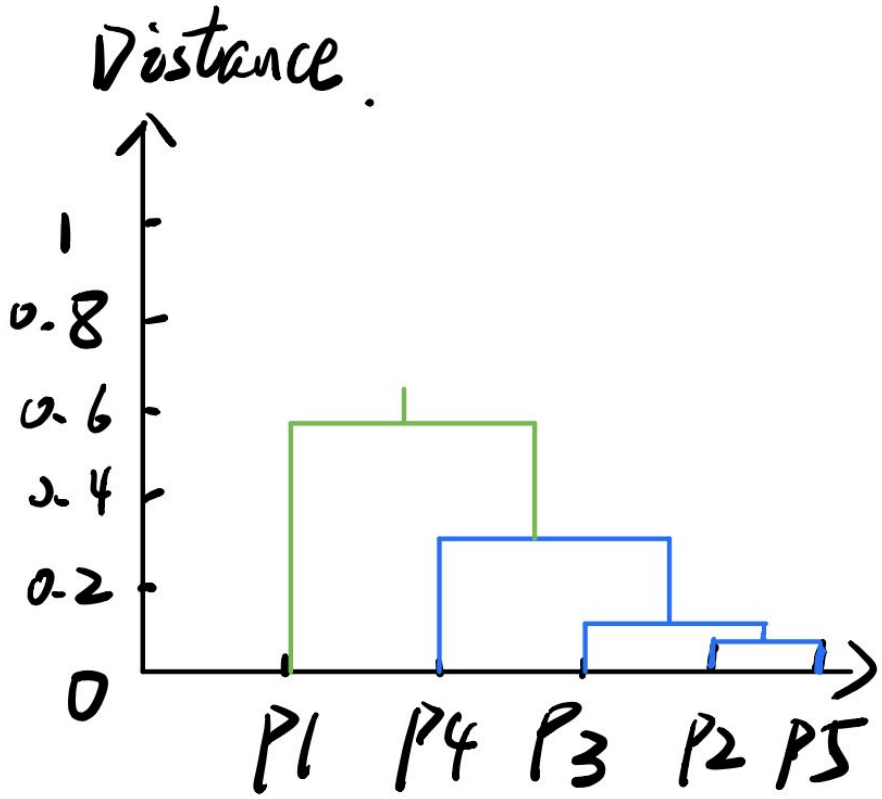
:
```
1 c1 = [[3, 3]]
2 c2 = [[1, 0], [0, 1], [2, 1]]
3 print(SSE(c1, c2))
```

```
2.666666666666667
```

Discussion:

The clustering results are different as the initial centroid maybe not the same between these two iterations. The KMeans method will update the centroid every iteration. That is the working principle of it.

2. The detail can be seen in the file named ***manual_part.pdf*** which includes the more clear figure of this part. I first use the similarity to analyze this part and then I transfer the similarity to the dissimilarity and follow the steps discussed in the class to draw the plot. The plot can be seen below which is also included in that file.



3. DBSCAN algorithm.

1). p1 is a noise as it's not the neighbourhood of a core (there is no neighbour of it that satisfies the threshold and the MinPts requirements).

2). p2 is a core as it satisfies both similarity threshold 0.8 and MinPts >=2 (p2, p5)

3). p3 is a core because of p3 and p5 so it satisfies the MinPts is greater than 2.

4). p4 is a noise because there is no point near it that can satisfy the given requirements.
5). p5 is a core as the circle includes p2, p3 and p5 in the ratio with 0.8.

4. Processing with the *Absenteeism_at_work* dataset.
   1). Load data

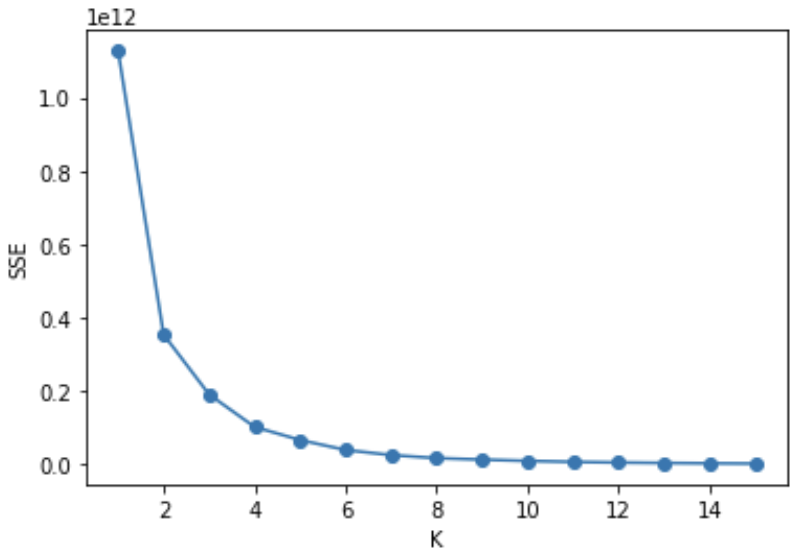
```
1 df = pd.read_excel("/Users/aopeng/D
2
```

```
1 df
```

	Age	Work load Average/day
0	33	239554
1	50	239554
2	38	239554
3	39	239554
4	33	239554

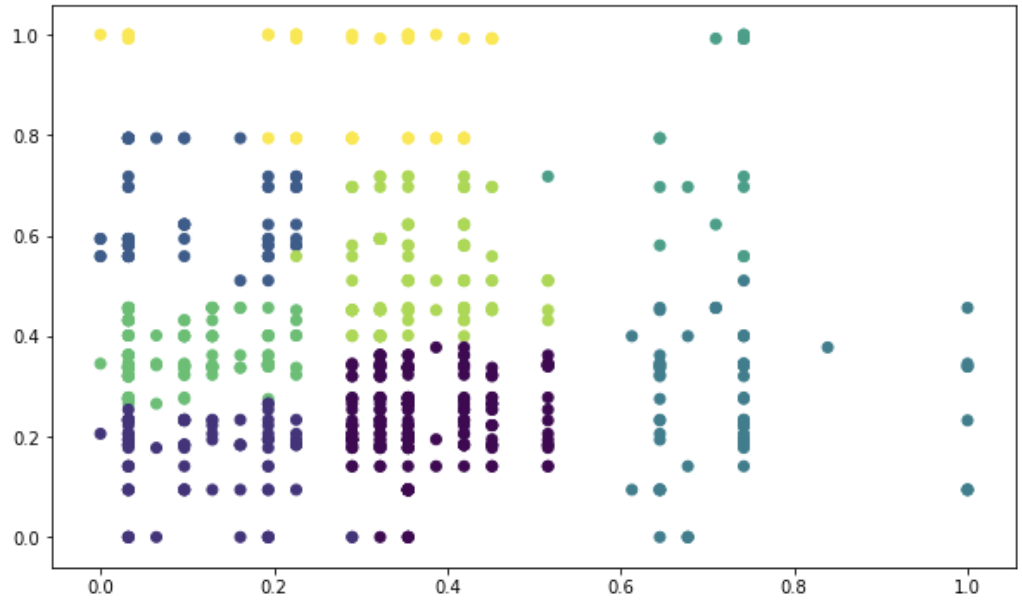
   2). KMeans with elbow method

   By using the elbow method, I try to use 1 to 15 as the K value. The result shows below. We can see the elbow is almost 8. So I choose K=8 as the optimal value.



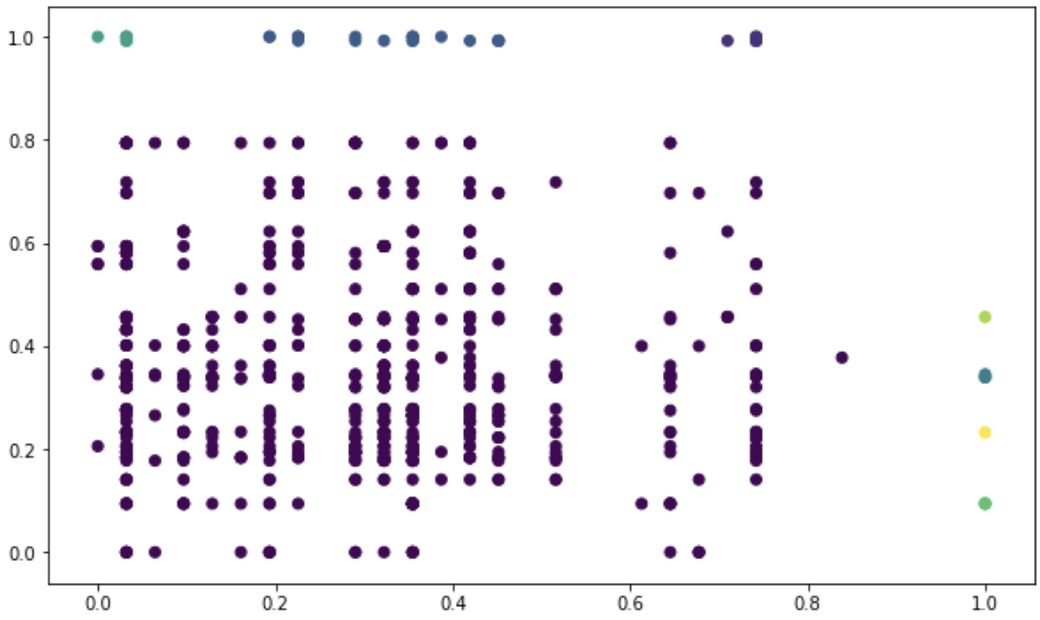
3). KMeans with the optimal K after standardizing the dataset

Firstly, I use ***MinMaxScaler()*** to standardize the dataset with the min_max method. Then I build the KMeans model and run it with K=8. The plot shows below.



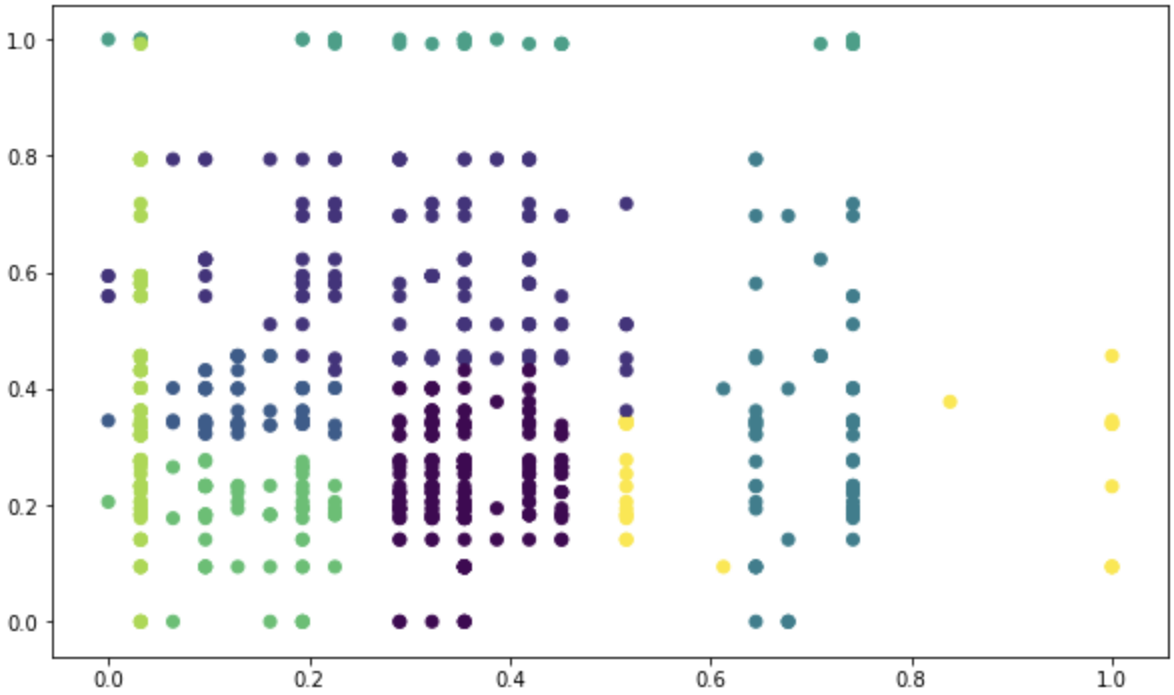
4). Run agglomerative clustering on the data, using single linkage.

I use ***AgglomerativeClustering()*** to build the model and set the parameter to use single linkage. The figure shows below.



5). Gaussian mixture model

I use ***GaussianMixture()*** to create the model and set K equals 8. The plot shows below.



6). Comparison

I think the best method, in this case, is the agglomerative clustering. Based on the distribution of this dataset (Histogram shows below), we can find the main parts are [28,38] for age and [22500, 27500] for workload average/day. In my opinion, the purpose of the cluster is to distinguish the possible groups in the dataset. From the figures, we can see that the agglomerative clustering model illustrates the main distribution comparing to the other 2 methods. It also may because of the original structure of the agglomerative clustering. All clusters will be merged into one cluster eventually so that the main part can be seen as one cluster.

	Age	Work load Average/day
count	740.000000	740.000000
mean	36.450000	271490.235135
std	6.478772	39058.116188
min	27.000000	205917.000000
25%	31.000000	244387.000000
50%	37.000000	264249.000000
75%	40.000000	294217.000000
max	58.000000	378884.000000

By the way, KMeans is more sensitive to the outliers which means it may see the outliers as one part of one cluster. It is a common and efficient method as its time complexity. And the Gaussian mixture algorithm is based on Mahalanobis distances to centers. It is good for density estimation while it cannot be scalable.