Working with Unlabeled Data – Cluster Analysis

Find the best number of clusters with **k_means** and **agglomerative clustering**

Overview

1. Load the data file
    - check the shape and plot the content
2. Observe the pair plot and comment the shapes in view of clustering
    A. if necessary, transform the data
3. Use the elbow method to find the optimal number of clusters, to do this test `KMeans` with varying number of clusters, from 2 to 10: for each value of `k`
    - fit the data
    - compute the **inertia** and the **silhouette score**
    - store them for plot
4. Plot inertia and silhouette score versus `k`
5. Choose the optimal number of clusters looking at the plots
6. Cluster the data using the optimal number, plot the cluster assignment
    - in the plot choose the features that seem to be most promising
7. For comparison, repeat the same operation with the `AgglomerativeClustering`

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, silhouette_samples
from mpl_toolkits import mplot3d

random_state = 42 # This variable will be used in all the procedure calls allowing
                  # in this way the running can be perfectly reproduced
                  # just change this value for a different experiment
```

1. Load the data file

Check the shape and plot the content

In [ ]:

```
X_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/00292/Wholesale%
```

Out[ ]:

```
(440, 8)
```

In [ ]:

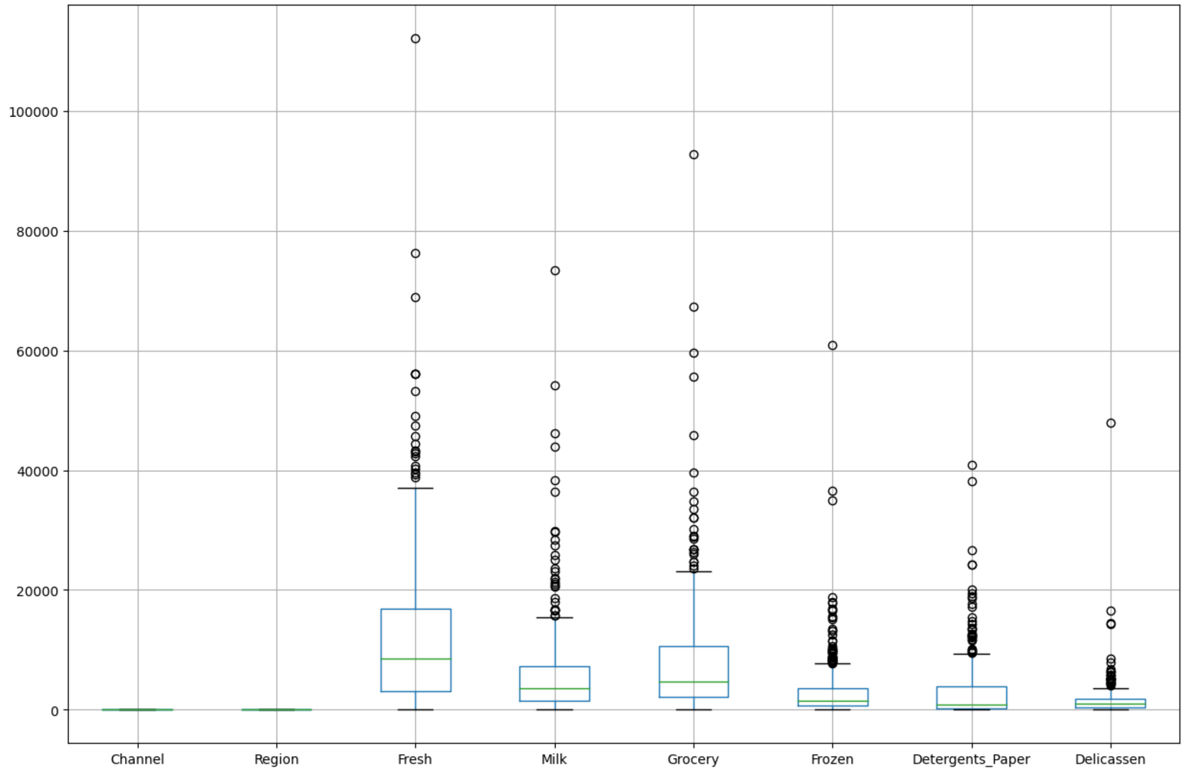
```

```

Out[ ]:

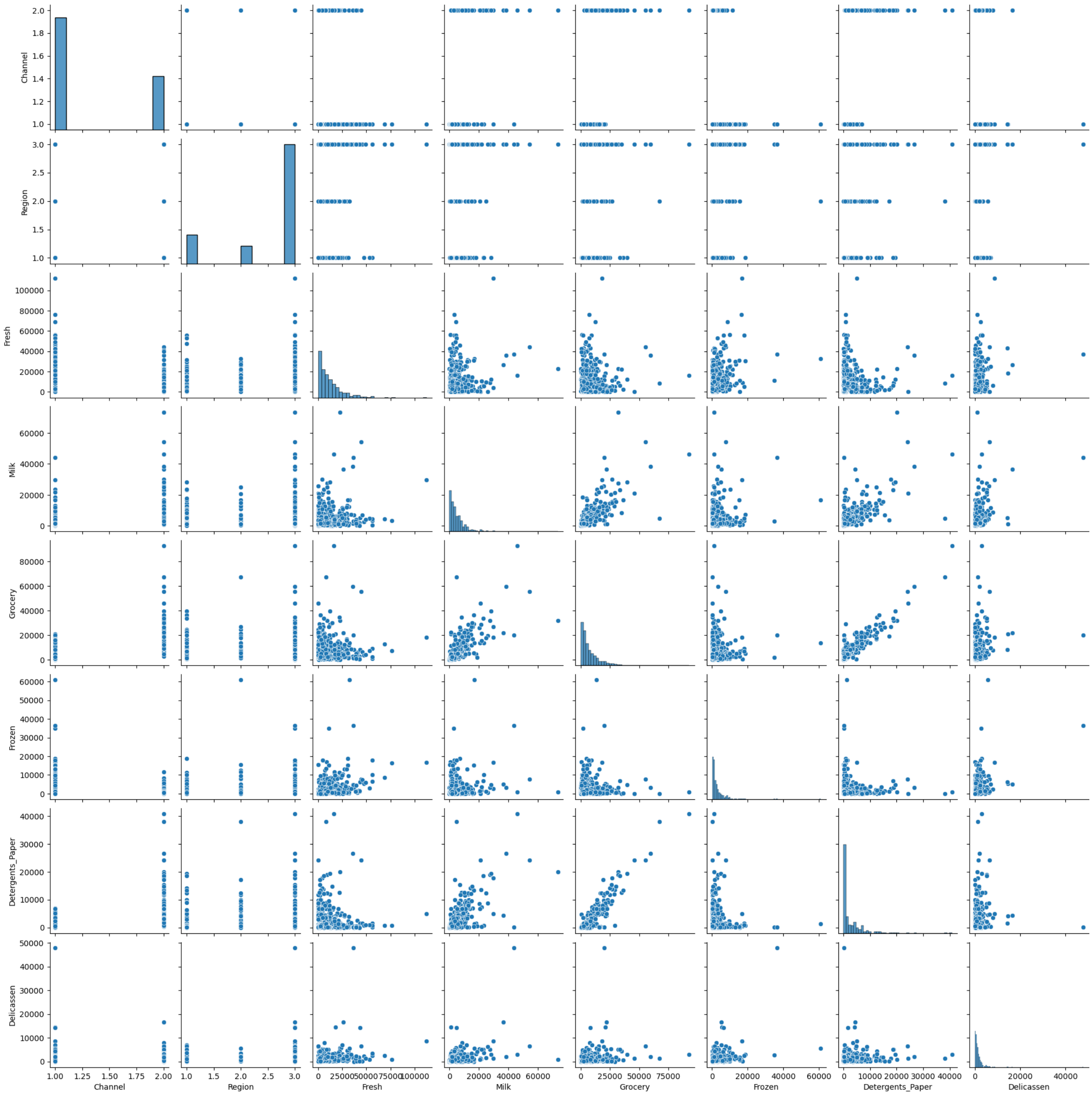
	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185

2. Observe the data distributions

In [ ]:



In [ ]:



We observe that the distributions of values are definitely *skewed*: in the columns from `Fresh` to `Delicassen` the values are highly concentrated on the right, but there are always outliers, frequently in a very large range.

Clustering is more effective in absence of outliers and with all the variables distributed in similar ranges, for this reason, we will execute two transformations:

1. transform all the variables from the column `Fresh` to the column `Delicassen` computing the *square root*
2. remap all the variables in the range `0:1`

In [ ]:
```
# square root transformation - the first two columns are not transformed
from math import sqrt

# remap on the 0:1 range with MinMaxScaler
from sklearn.preprocessing import MinMaxScaler
```

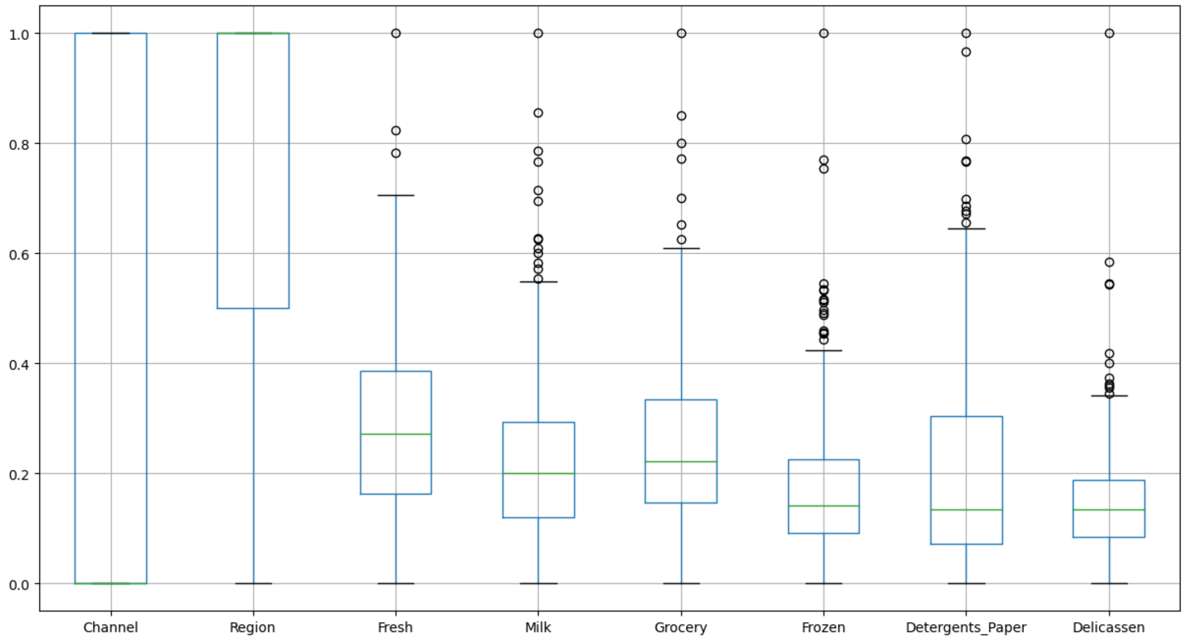
Out[ ]:

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	1.0	1.0	0.332649	0.344530	0.281385	0.039835	0.249488	0.160416
1	1.0	1.0	0.246952	0.347490	0.317250	0.152973	0.277812	0.186029
2	1.0	1.0	0.234044	0.327791	0.283711	0.182200	0.287352	0.399740
3	0.0	1.0	0.340505	0.103027	0.208796	0.310384	0.103755	0.186684
4	1.0	1.0	0.446188	0.250813	0.274408	0.238171	0.201784	0.323509

Show the result of the transformation

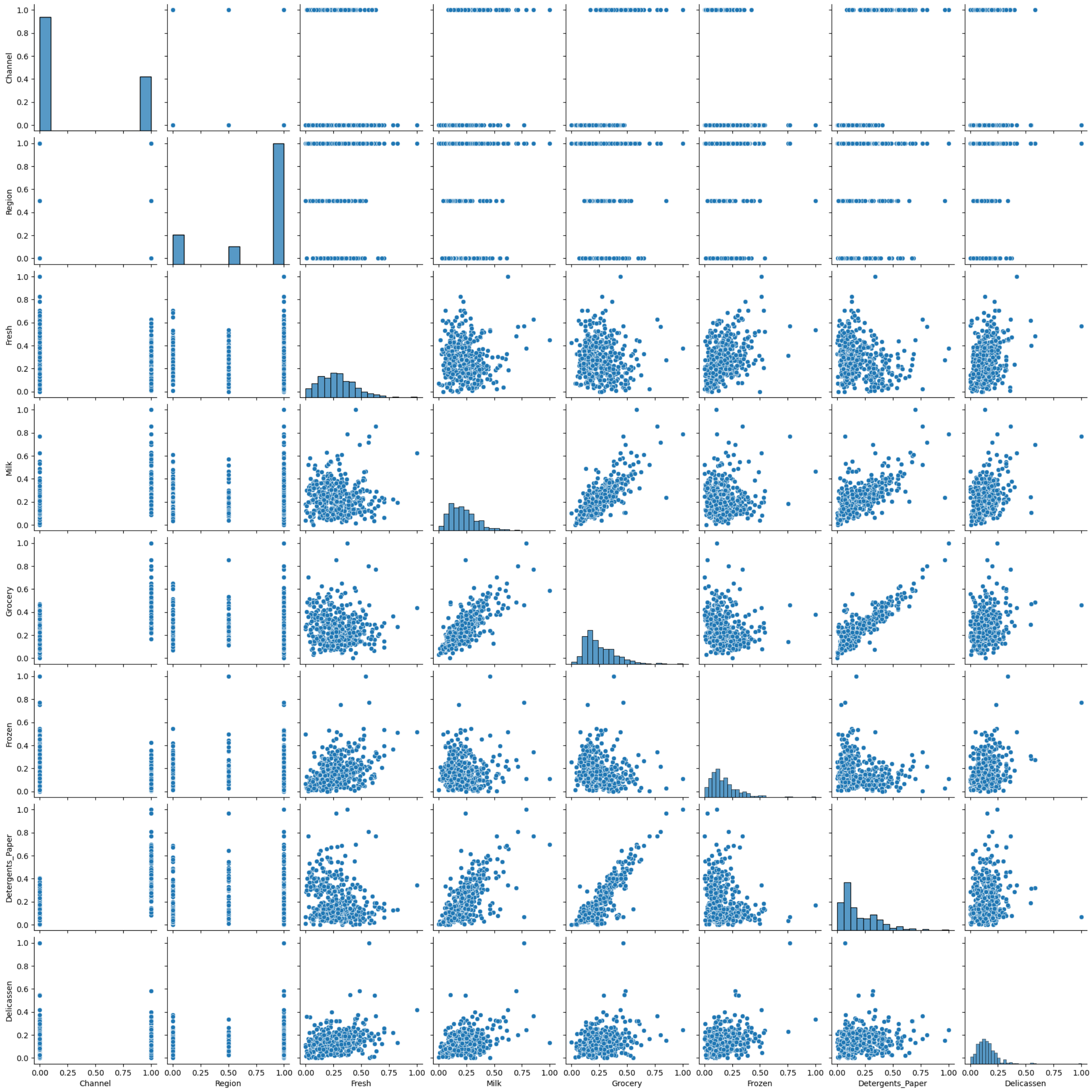
In [ ]:

```

```



In [ ]:

```

```



Now the effect of outliers is reduced, and we compute the clustering

3. Use the elbow method to find the optimal number of clusters

Test `KMeans` with varying number of clusters, from 2 to 10

Prepare the results list that will contain pairs of `inertia` and `silhouette_score` for each value of `k`, then, **for each value** of `k`

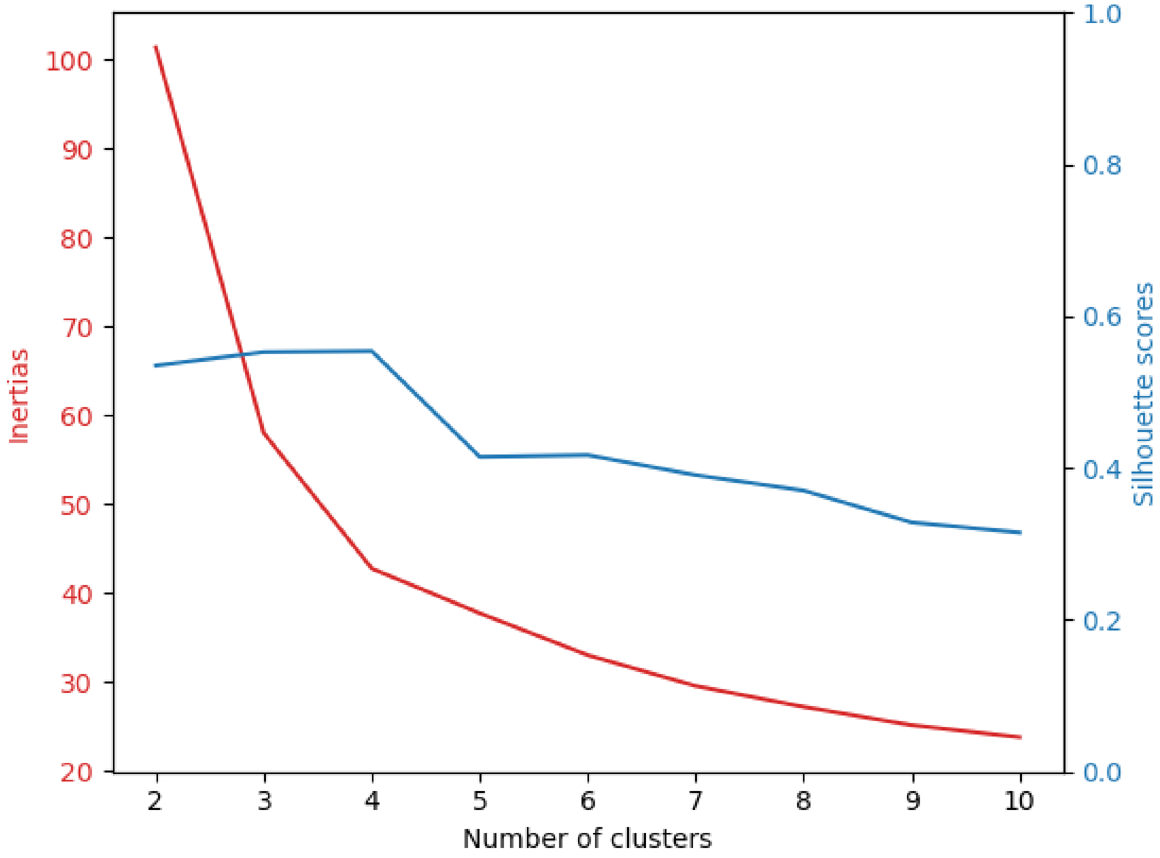
- initialize an estimator for `KMeans`
- fit the data and predict the cluster assignment for each individual with `fit` and `predict`
- the **inertia** is provided in the attribute `inertia_` of the fitted model
- compute the **silhouette score** using the function `silhouette_score` from `sklearn.metrics` using as arguments the data and the fitted labels, we will fill the variable `silhouette_scores`
- store the two values above in the list created at the beginning

In [ ]:
```
from sklearn.model_selection import ParameterGrid
```

```
k_range = list(range(2,11)) # set the range of k values to test
```

4. Plot **inertia** and **silhouette score** versus `k`

In [ ]:



5. Cluster with the optimal number

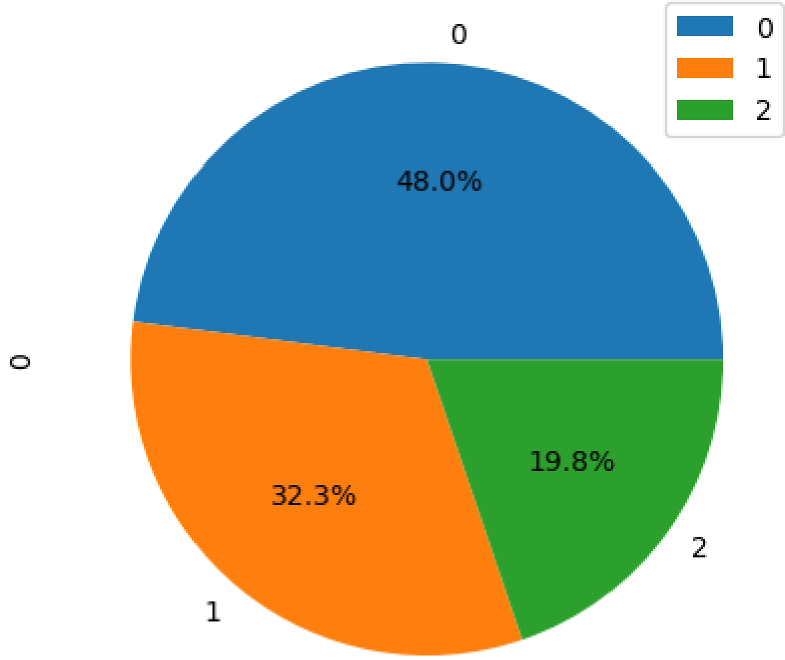
The two *elbow* points of inertia would suggest as cluster number 3 or 4, slightly more pronounced in 3. Silhouette has a maximum on 4, but the increase with respect to 3 is very small.

We will choose k=3

In [ ]:

```
Number of clusters = 3  - Distortion =  58.00   - Silhouette score = 0.55
```

In [ ]:



Comments

The **silhouette score** ranges from `-1` (worst) to `1` (best); as a rule of thumb, a value greater than `0.5` should be considered acceptable.

Agglomerative clustering

We will try a grid of parameter configurations, with the number of clusters in the range `2:10` and the four linkage methods available in the *sklearn* implementation of *AgglomerativeClustering*.

In [ ]:
```
from sklearn.cluster import AgglomerativeClustering
```

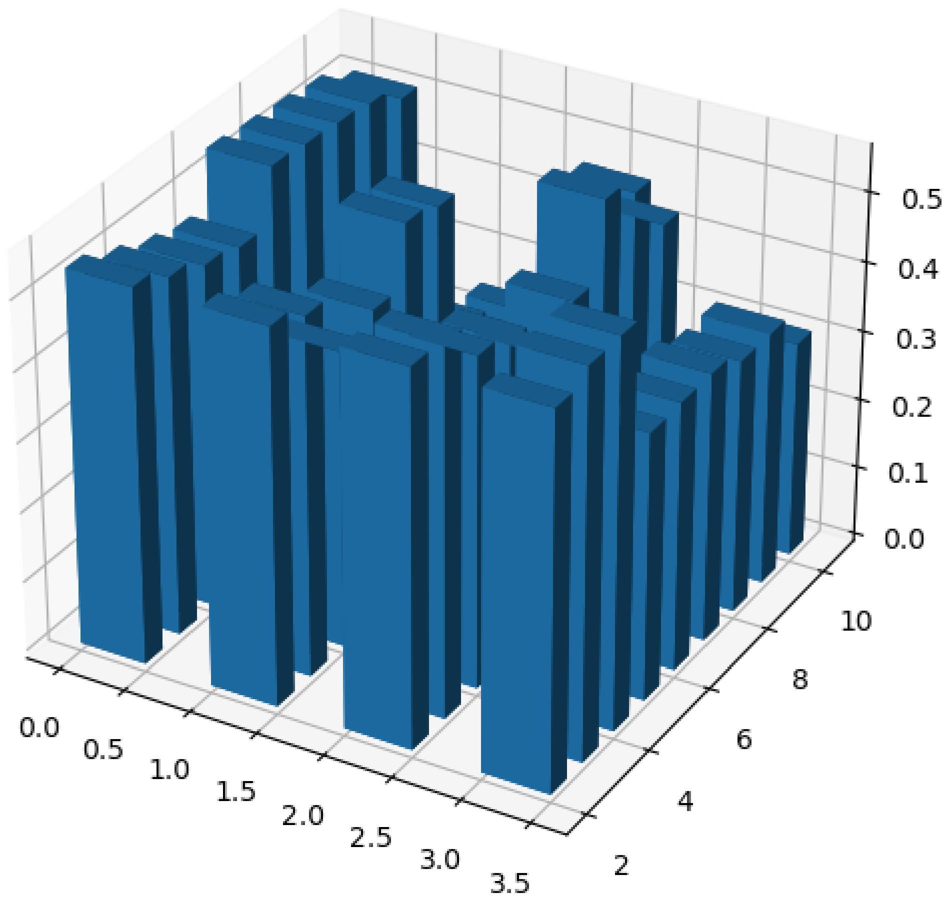
In [ ]:
```
```

Out[ ]:

	linkage	n_clusters	silhouette_score
22	average	6	0.555314
2	ward	4	0.553519
1	ward	3	0.552275
23	average	7	0.550038
24	average	8	0.545323

In [ ]:
```
```

In [ ]:
```
```



The top five results have a very similar silhouette score, we will choose the setting with 3 clusters, as for k-means, and the linkage giving the best result with 3 clustes, that is `ward`. This is the result record with `index 1` (the record index is the unnamed column at the very left of the dataframe output

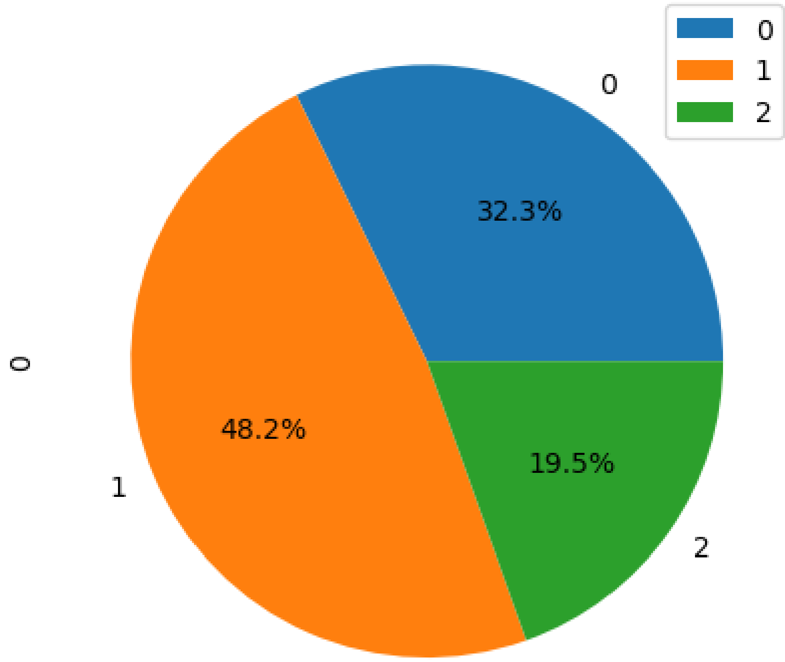
In [ ]:

```
  linkage  n_clusters  silhouette_score  linkage_enc
1    ward           3          0.552275          3.0
```

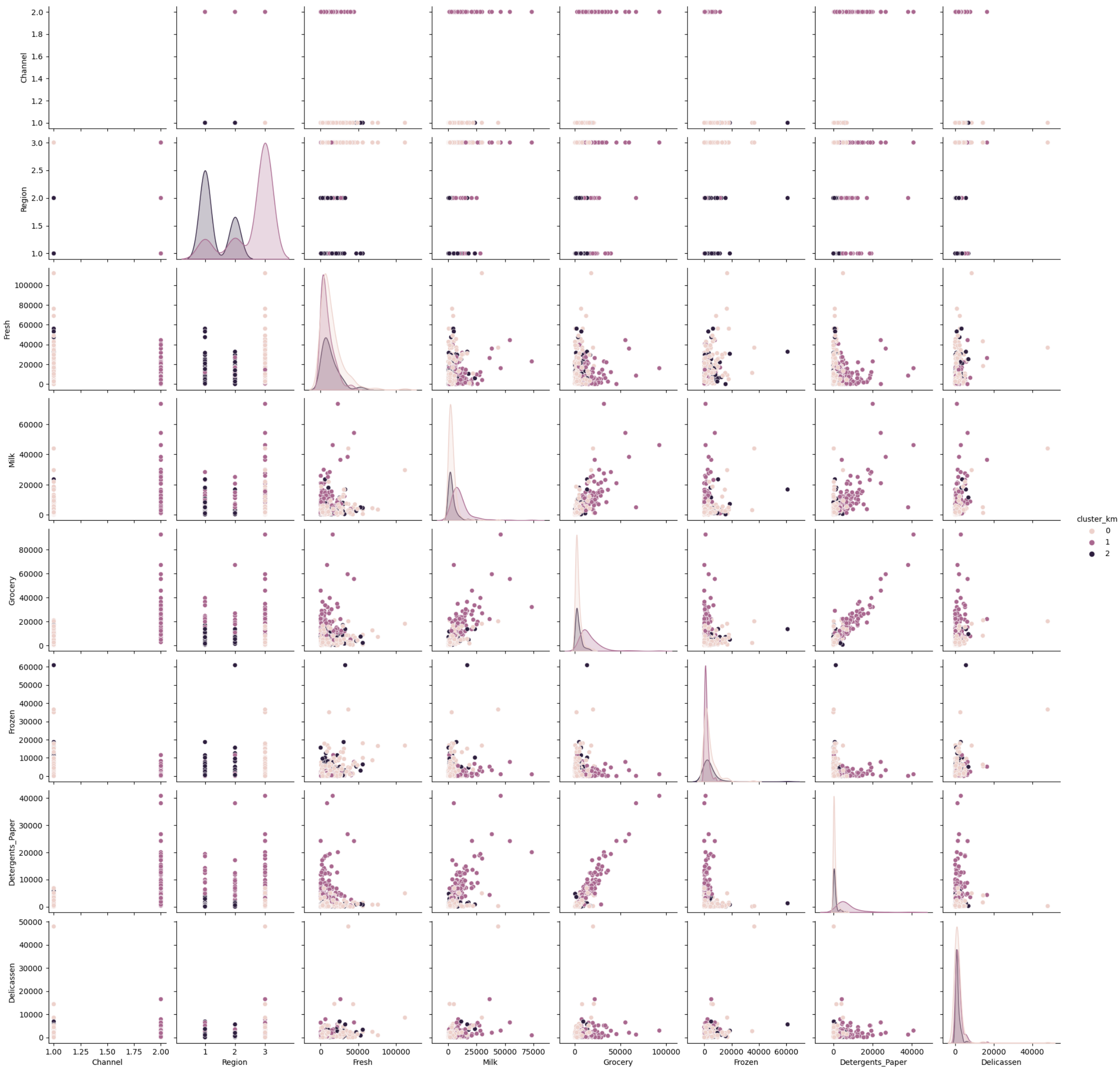
In [ ]:

Show the distribution of data in the three clusters

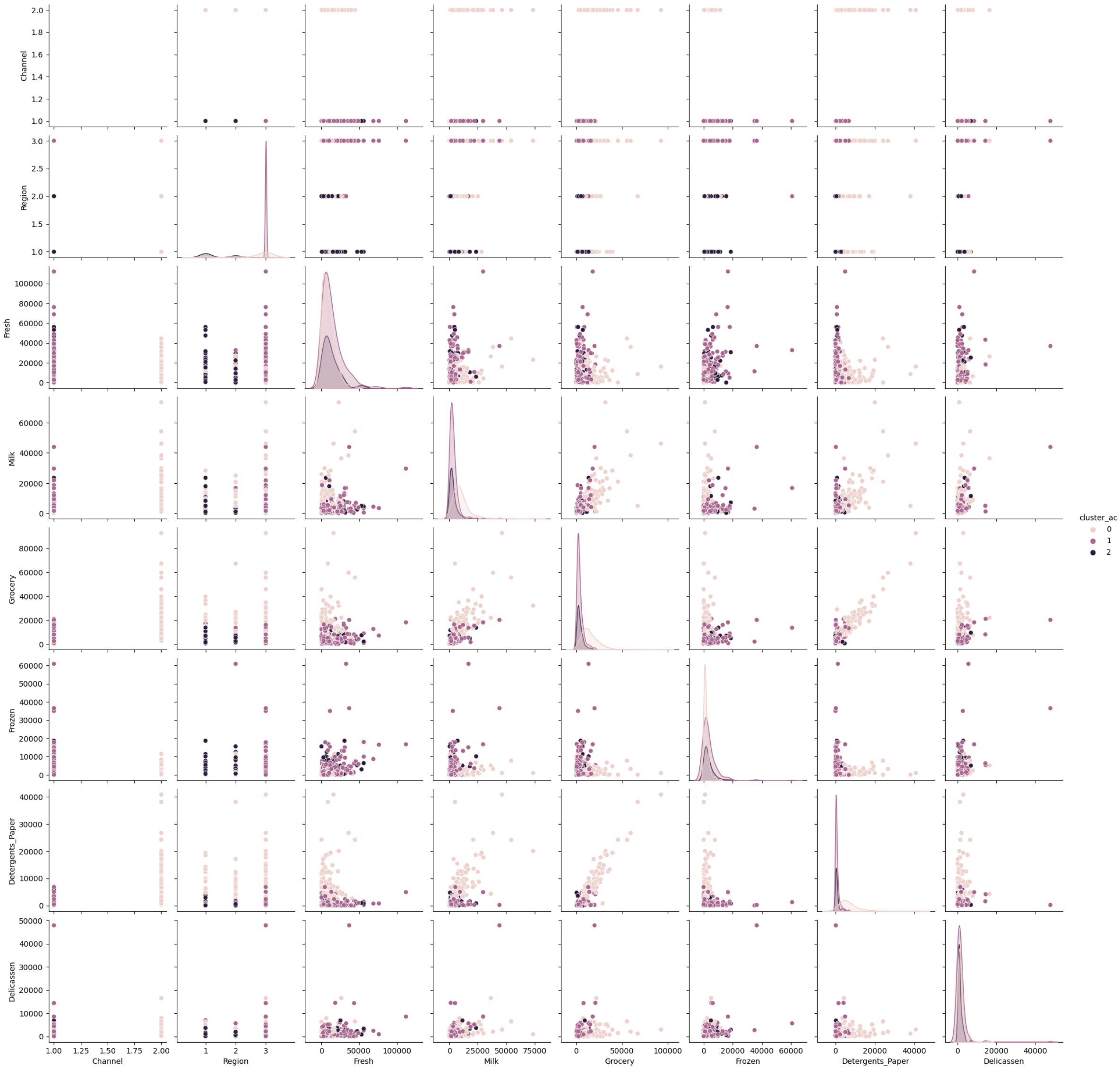
In [ ]:


0
1
2
0
32.3%
48.2%
1
19.5%
2
0

In [ ]:


Channel
Region
Fresh
Milk
Grocery
Frozen
Detergents_Paper
Delicassen
cluster_km
0
1
2

In [ ]:



Comments

The solution with the Agglomerative Clustering in this case provides a result very similar to that of kmeans.

It is interesting to compare more deeply the results of the two clustering models.

The function `pair_confusion_matrix` computes the number of pairs of objects that are in the same clusters or in different clusters in two different clustering schemes.

The result is given in a 2x2 matrix, the perfect match is when only the numbers in the main diagonal are non zero.

We present here the results normalized to 1

In [ ]:
```
from sklearn.metrics import pair_confusion_matrix
```

Out[ ]:
```
array([[0.62603023, 0.00218472],
       [0.00089045, 0.3708946 ]])
```

In [ ]:
```
```

```
The percentage of match between the two clustering schemes is  99.69%
```

Control questions

1. Repeat the experiments without the data transformations and comment the result
2. Repeat the final fittings with the numbers of clusters immediately before and after the chosen values and comment the results