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

In [1]:
```
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
from sklearn.model_selection import ParameterGrid
import warnings
warnings.filterwarnings("ignore")

random_state = 42 # This variable will be used in all the procedure calls allowi
                  # 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 [2]:
```
X_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/00292/Wholesa
delimiter = ','
X0 = pd.read_csv(X_url, delimiter=delimiter)
X0.shape
```

Out[2]:
```
(440, 8)
```

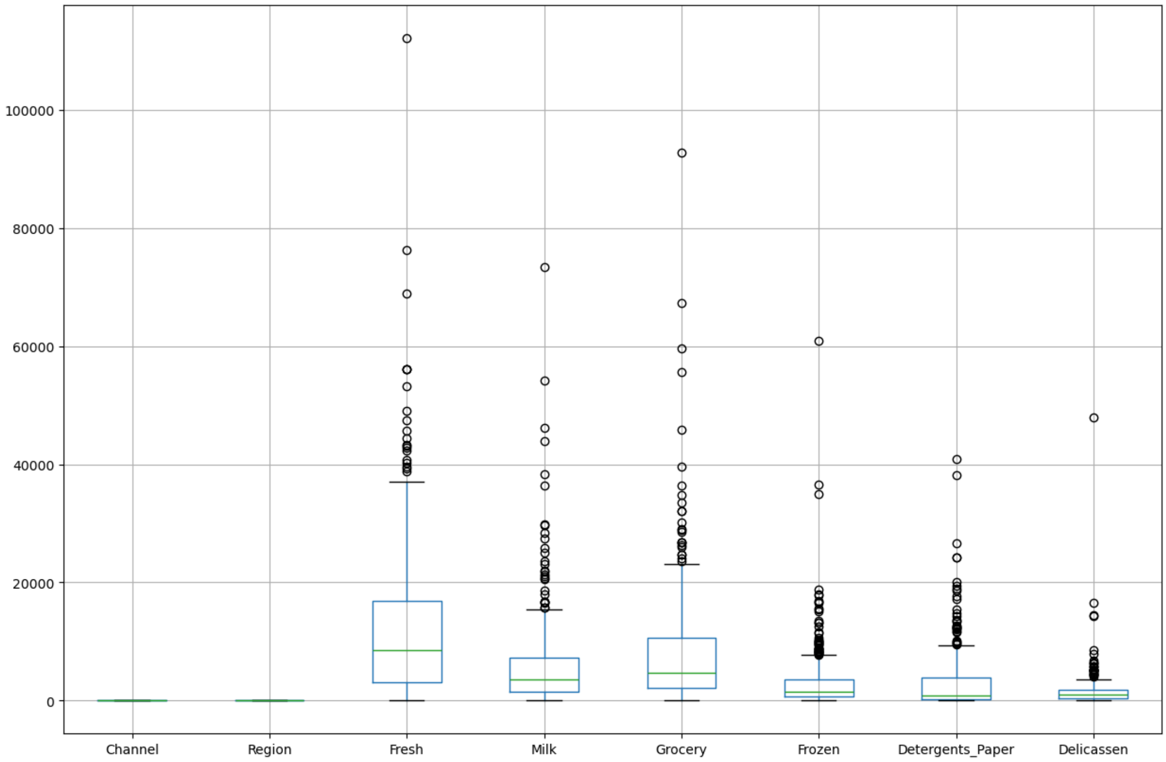
In [3]:
```
X0.head()
```

Out[3]:

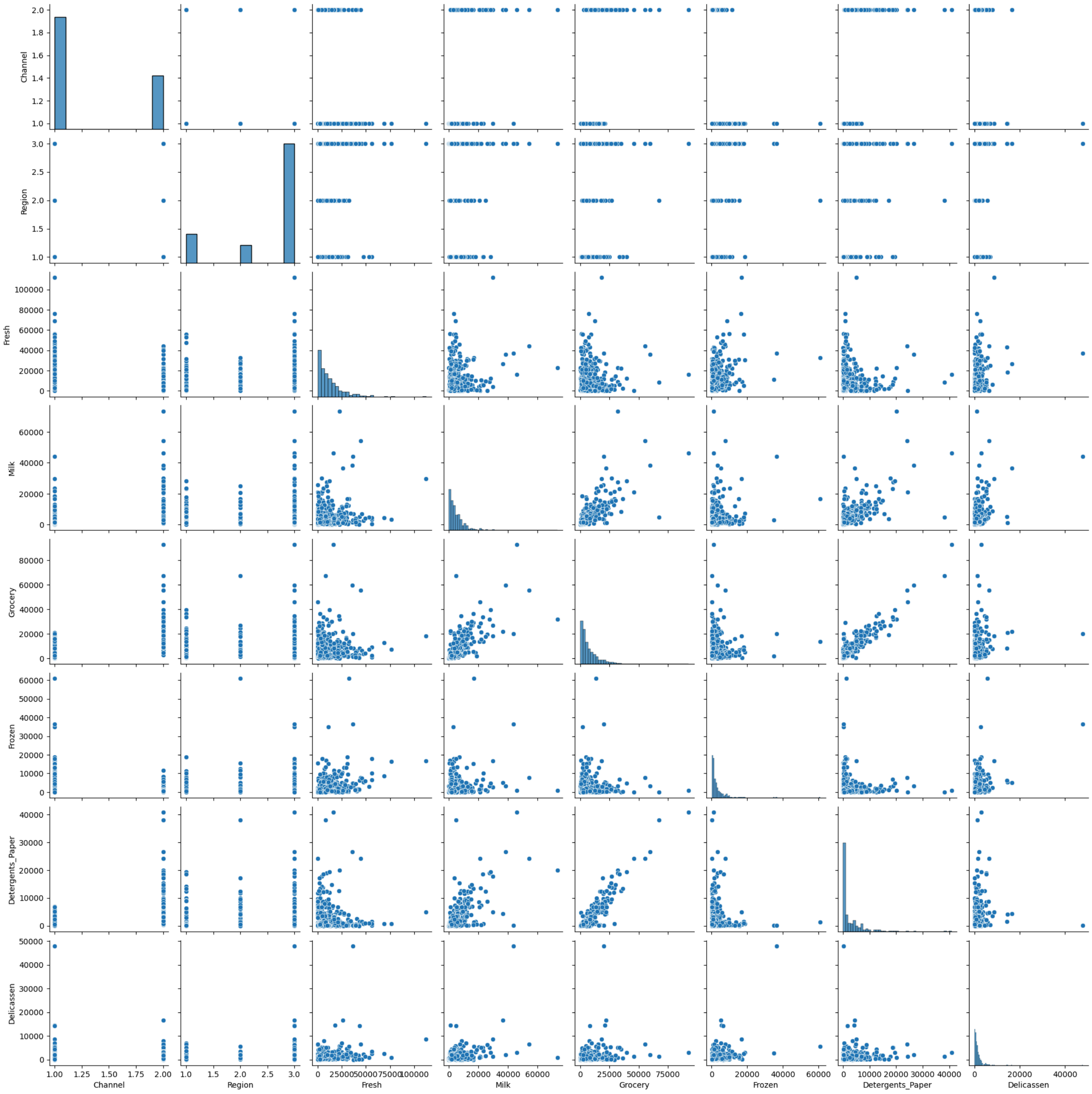
	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185

2. Observe the data distributions

In [4]:
```
X0.boxplot(figsize=(15,10));
```



In [5]:
```
sns.pairplot(X0);
plt.show()
```



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` with PowerTransformer
2. remap all the variables in the range `0:1`

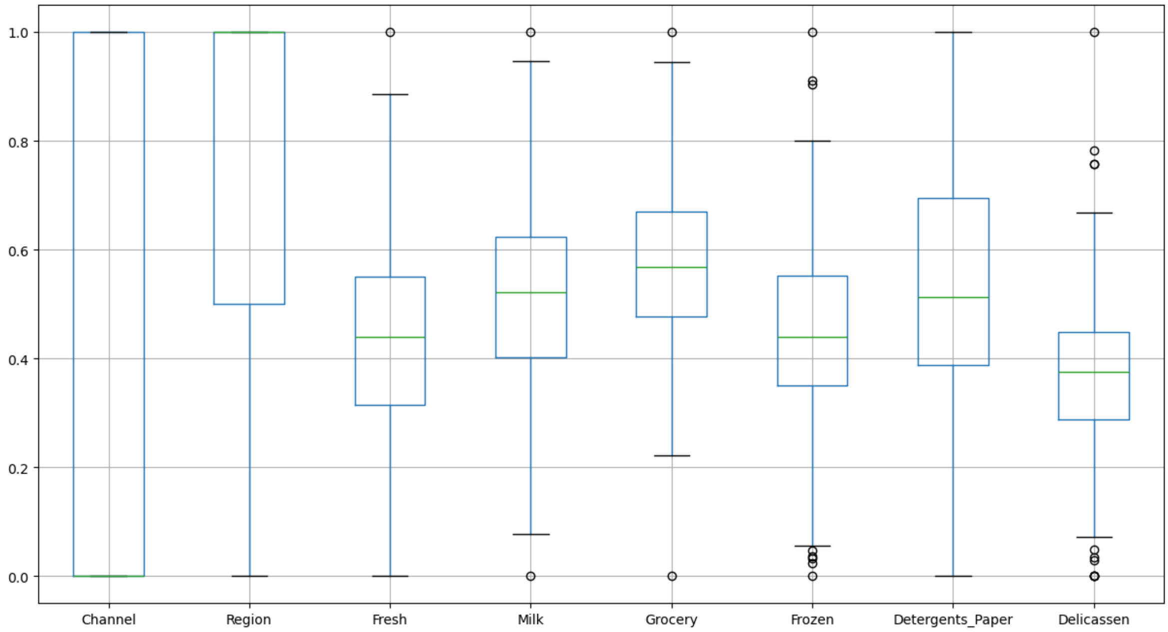
In [6]:
```
from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer()
X_pt = pd.DataFrame(pt.fit_transform(X0.iloc[:,2:]),columns=X0.columns[2:])
X_trasf = pd.concat([X0.iloc[:,:2],X_pt],axis=1)
from sklearn.preprocessing import MinMaxScaler
mms = MinMaxScaler()
X = pd.DataFrame(mms.fit_transform(X_trasf), columns = X_trasf.columns)
X.head()
```

Out[6]:

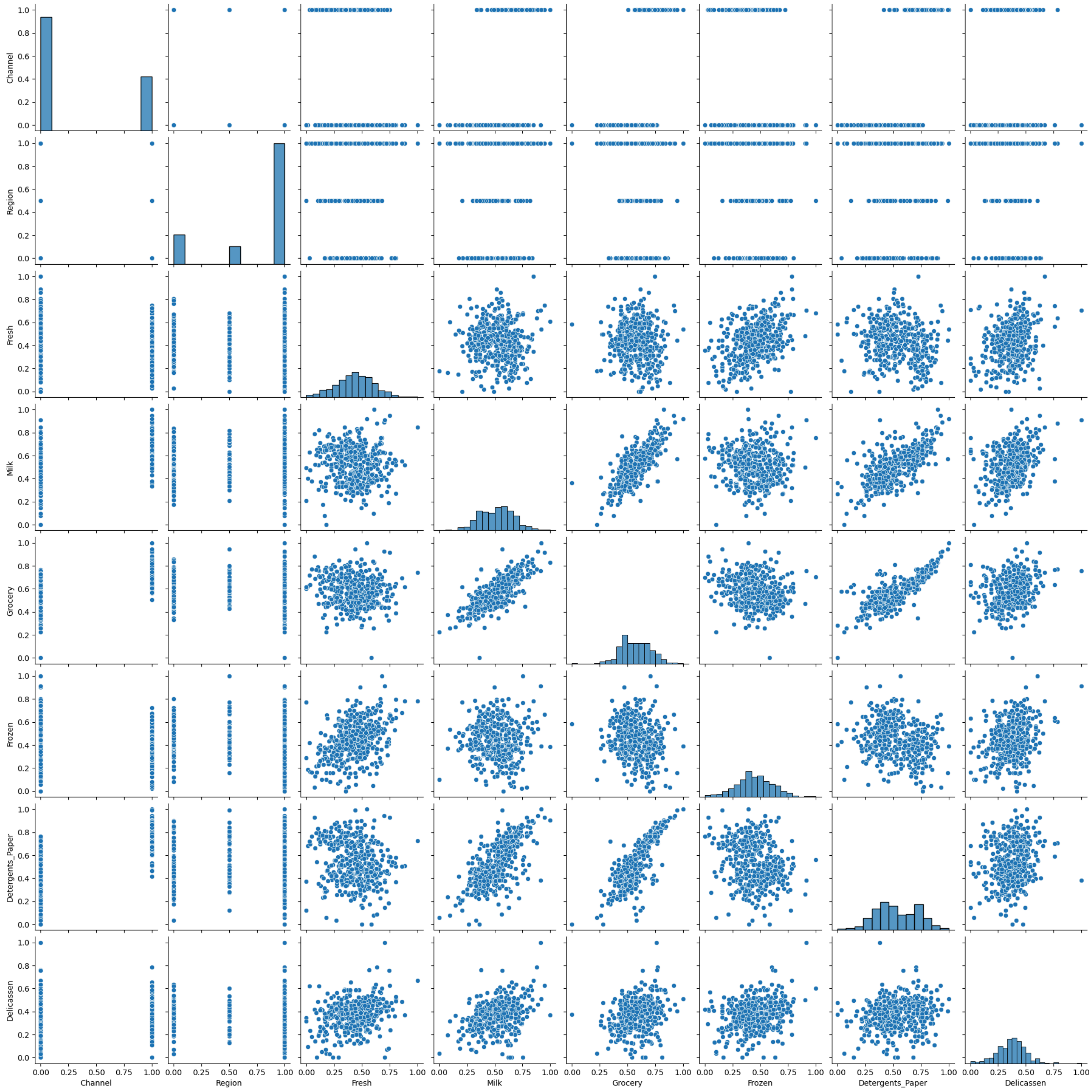
	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicass
0	1.0	1.0	0.501828	0.667606	0.625238	0.208640	0.649941	0.4120
1	1.0	1.0	0.414266	0.670028	0.655690	0.458800	0.674852	0.4456
2	1.0	1.0	0.400077	0.653586	0.627297	0.499856	0.682752	0.6542
3	0.0	1.0	0.509368	0.369264	0.553550	0.636716	0.461095	0.4464
4	1.0	1.0	0.604755	0.580657	0.618985	0.566470	0.601884	0.5902

Show the result of the transformation

In [7]:
```
X.boxplot(figsize=(15,8));
plt.show()
```



In [8]:
```
sns.pairplot(X);
plt.show()
```



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 [9]:
```
k_range = list(range(2,11)) # set the range of k values to test
parameters_km = [{'n_clusters': k_range}]
pg = list(ParameterGrid(parameters_km))
inertias_km = []
silhouette_scores_km = []
for i in range(len(pg)):
    km = KMeans(**(pg[i]), random_state=random_state)
    y_km = km.fit_predict(X)
    inertias_km.append(km.inertia_)
    silhouette_scores_km.append(silhouette_score(X,y_km))
```

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

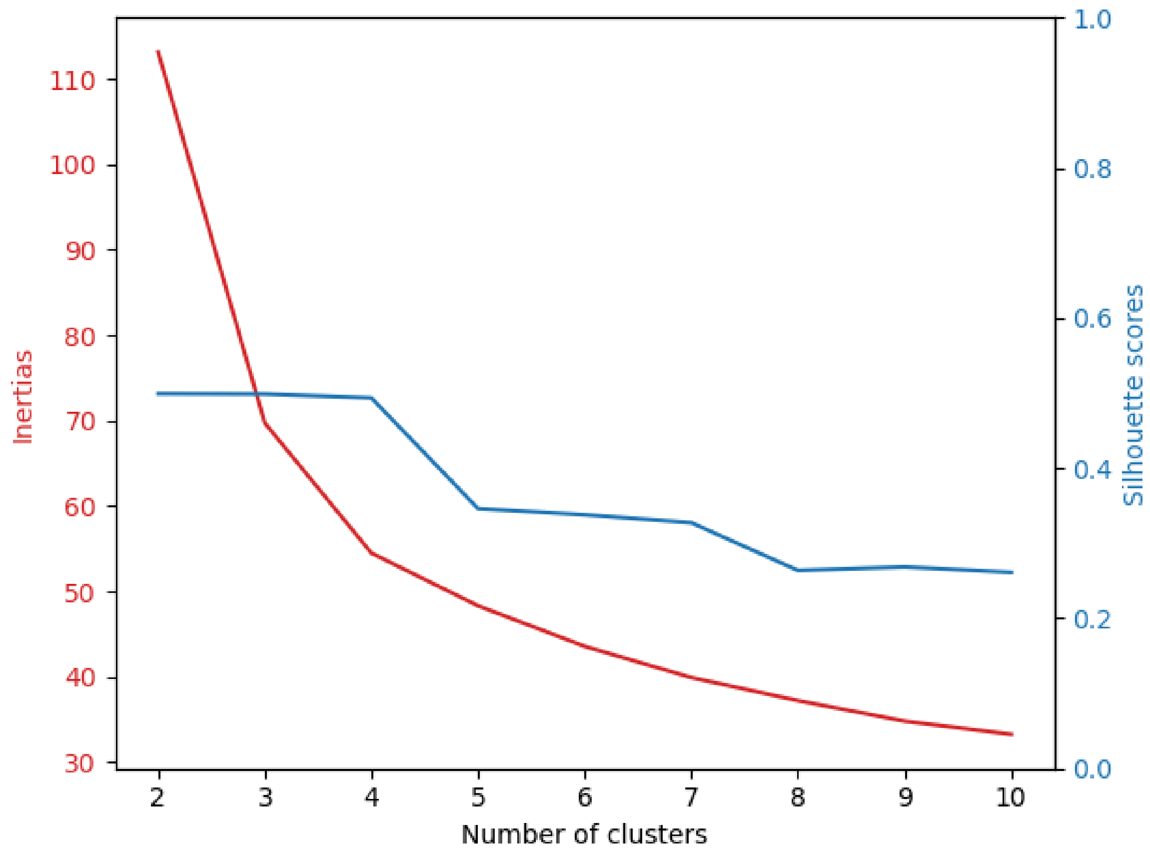
In [10]:
```
def two_plots(x, y1, y2, xlabel, y1label, y2label):
    fig, ax1 = plt.subplots()

    color = 'tab:red'
    ax1.set_xlabel(xlabel)
    ax1.set_ylabel(y1label, color=color)
    ax1.plot(x, y1, color=color)
    ax1.tick_params(axis='y', labelcolor=color)
    ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis

    color = 'tab:blue'
    ax2.set_ylabel(y2label, color=color)  # we already handled the x-label with
    ax2.plot(x, y2, color=color)
    ax2.tick_params(axis='y', labelcolor=color)
    ax2.set_ylim(0,1) # the axis for silhouette is [0,1]

    fig.tight_layout()  # otherwise the right y-label is slightly clipped
    plt.show()
```

In [11]:
```
two_plots(x=k_range, y1=inertias_km, y2=silhouette_scores_km
          , xlabel='Number of clusters', y1label='Inertias', y2label='Silhouette
         )
```



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.

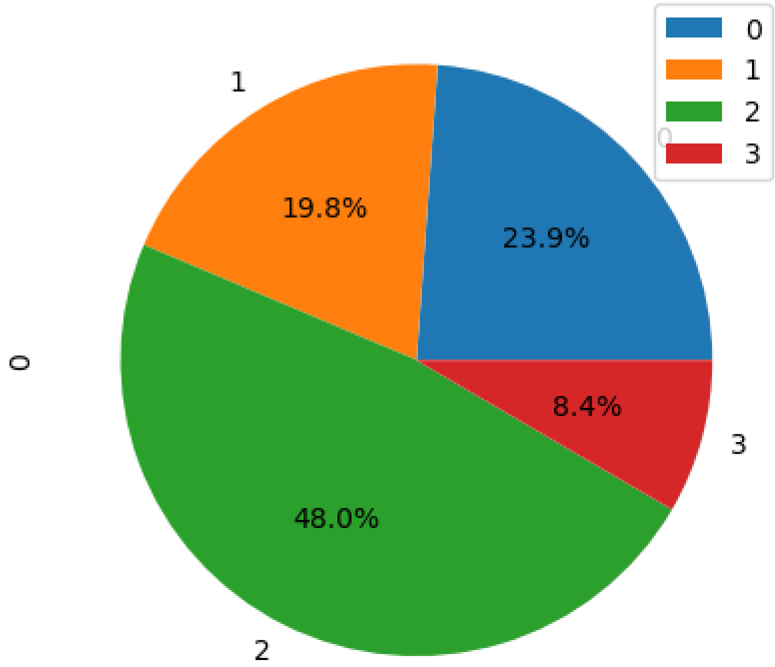
We will choose k=4

In [12]:
```
k=4
km = KMeans(n_clusters=k,
            random_state=random_state)
y_km = km.fit_predict(X)
print("Number of clusters = {}\t- Distortion = {:6.2f}\t- Silhouette score = {:4
    .format(k,inertias_km[k_range.index(k)],silhouette_scores_km[k_range.index(k
```

```
Number of clusters = 4  - Distortion =  54.49   - Silhouette score = 0.49
```

Show the distribution of samples in the clusters with a pie chart

In [13]:
```
clust_sizes_km = np.unique(y_km,return_counts=True)
pd.DataFrame(clust_sizes_km[1]).plot.pie(y=0, autopct='%1.1f%%', );
plt.show()
```



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.