Wholesale customers data

In [141…
```python
# import pandas library
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
import matplotlib.pyplot as plt
import seaborn as sns
```

In [142…
```python
# loading the dataset
data_0 = pd.read_csv("Wholesale customers data.csv")
```

In [143…
```python
# show the first 5 rows using dataframe.head() method
print("The first 5 rows of the dataframe")
data_0.head(5)
```

The first 5 rows of the dataframe

Out[143…

	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

In [144…
```python
missing_data = data_0.isnull()
missing_data.head(5)
```

Out[144…

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False

In [145…
```python
data_0.isnull().sum()
```

Out[145…
```
Channel             0
Region              0
Fresh               0
Milk                0
Grocery             0
Frozen              0
Detergents_Paper    0
Delicassen          0
dtype: int64
```

In [146…
```python
for column in missing_data.columns.values.tolist():
    print(column)
    print (missing_data[column].value_counts())
    print("")
```

```
Channel
False    440
Name: Channel, dtype: int64

Region
False    440
Name: Region, dtype: int64

Fresh
False    440
Name: Fresh, dtype: int64

Milk
False    440
Name: Milk, dtype: int64

Grocery
False    440
Name: Grocery, dtype: int64

Frozen
False    440
Name: Frozen, dtype: int64

Detergents_Paper
False    440
Name: Detergents_Paper, dtype: int64

Delicassen
False    440
Name: Delicassen, dtype: int64
```

In [147…
```python
data_0.dtypes
```

Out[147…
```
Channel             int64
Region              int64
Fresh               int64
Milk                int64
Grocery             int64
Frozen              int64
Detergents_Paper    int64
Delicassen          int64
dtype: object
```

In [148…
```python
data_0.describe()
```

Out[148…

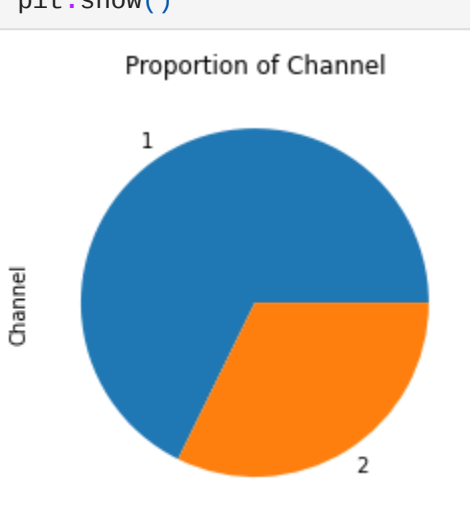
	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
count	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000
mean	1.322727	2.543182	12000.297727	5796.265909	7951.277273	3071.931818	2881.493182	1524.870455
std	0.468052	0.774272	12647.328865	7380.377175	9503.162829	4854.673333	4767.854448	2820.105937
min	1.000000	1.000000	3.000000	55.000000	3.000000	25.000000	3.000000	3.000000
25%	1.000000	2.000000	3127.750000	1533.000000	2153.000000	742.250000	256.750000	408.250000
50%	1.000000	3.000000	8504.000000	3627.000000	4755.500000	1526.000000	816.500000	965.500000
75%	2.000000	3.000000	16933.750000	7190.250000	10655.750000	3554.250000	3922.000000	1820.250000
max	2.000000	3.000000	112151.000000	73498.000000	92780.000000	60869.000000	40827.000000	47943.000000

In [149…
```python
data_0.info()
```

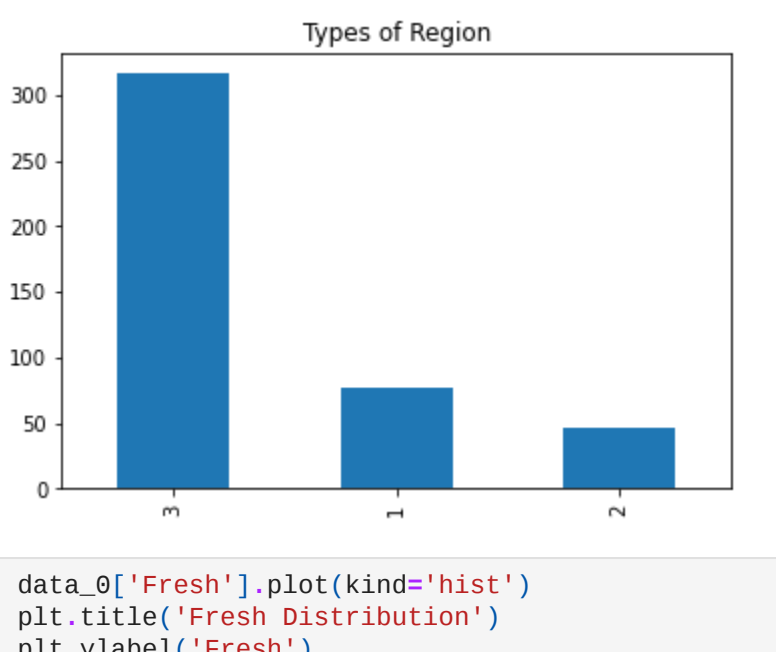
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 440 entries, 0 to 439
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Channel           440 non-null    int64
 1   Region            440 non-null    int64
 2   Fresh             440 non-null    int64
 3   Milk              440 non-null    int64
 4   Grocery           440 non-null    int64
 5   Frozen            440 non-null    int64
 6   Detergents_Paper  440 non-null    int64
 7   Delicassen        440 non-null    int64
dtypes: int64(8)
memory usage: 27.6 KB
```

visulization

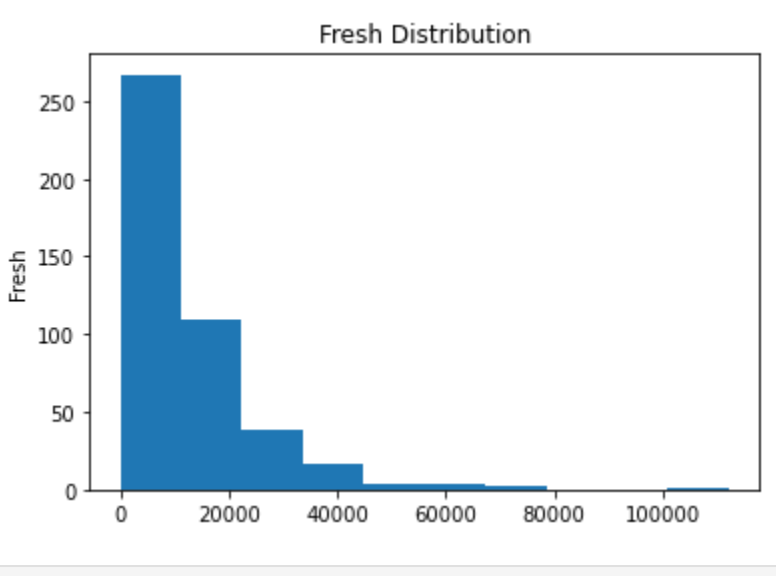
In [118…
```python
data_0['Channel'].value_counts().plot(kind='pie')
plt.title('Proportion of Channel')
plt.show()
```



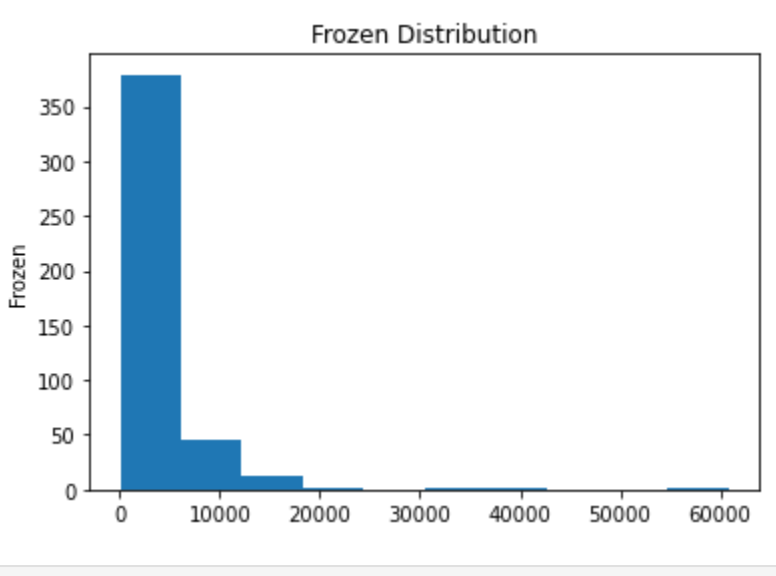
In [119…
```python
data_0['Region'].value_counts().plot(kind='bar')
plt.title('Types of Region')
plt.show()
```



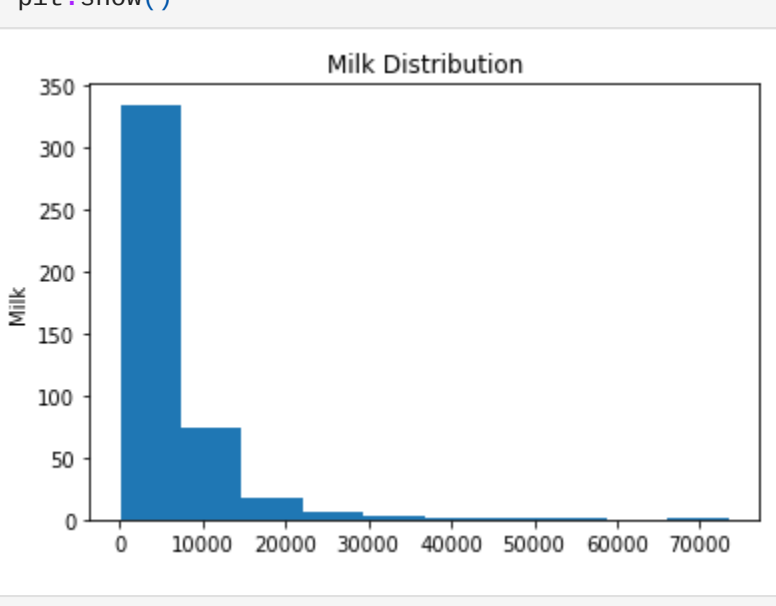
In [120…
```python
data_0['Fresh'].plot(kind='hist')
plt.title('Fresh Distribution')
plt.ylabel('Fresh')
plt.show()
```



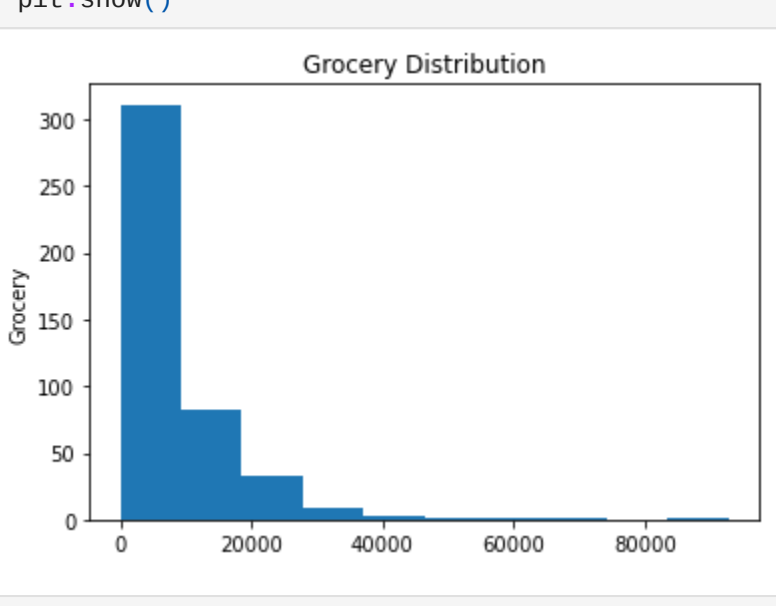
In [121…
```python
data_0['Frozen'].plot(kind='hist')
plt.title('Frozen Distribution')
plt.ylabel('Frozen')
plt.show()
```



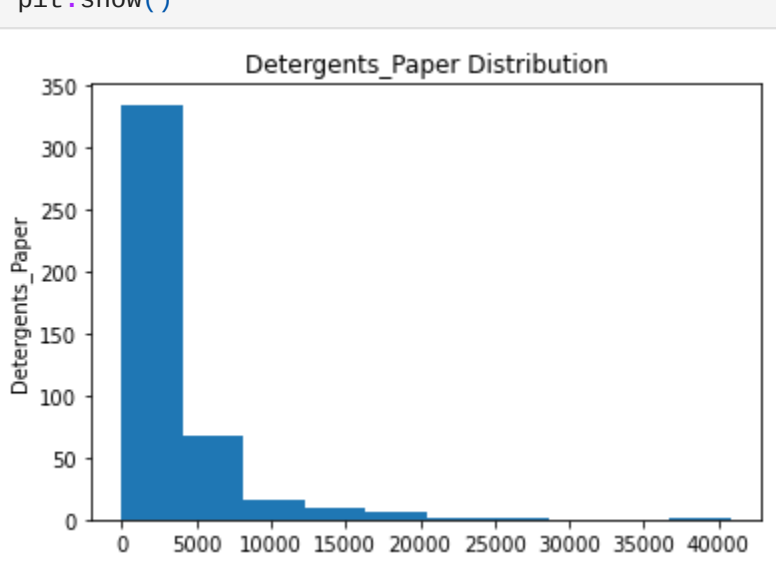
In [237…
```python
data_0['Milk'].plot(kind='hist')
plt.title('Milk Distribution')
plt.ylabel('Milk')
plt.show()
```



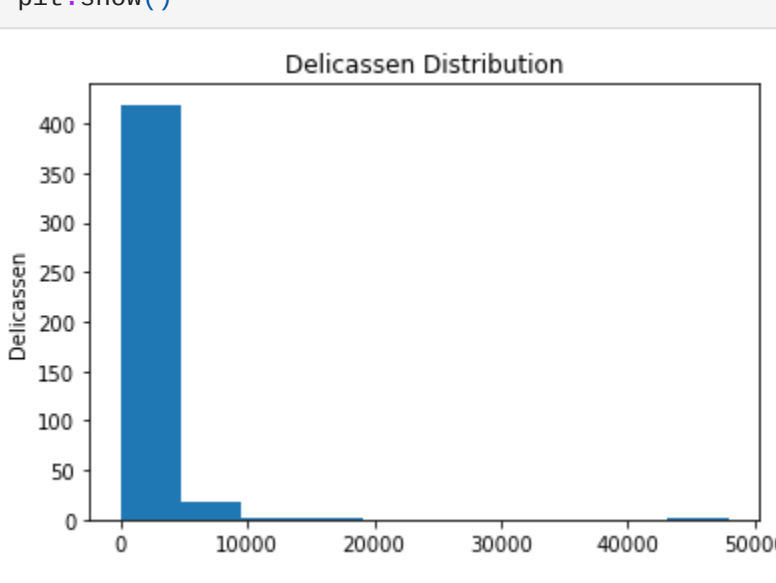
In [122…
```python
data_0['Grocery'].plot(kind='hist')
plt.title('Grocery Distribution')
plt.ylabel('Grocery')
plt.show()
```



In [123…
```python
data_0['Detergents_Paper'].plot(kind='hist')
plt.title('Detergents_Paper Distribution')
plt.ylabel('Detergents_Paper')
plt.show()
```



In [124…
```python
data_0['Delicassen'].plot(kind='hist')
plt.title('Delicassen Distribution')
plt.ylabel('Delicassen')
plt.show()
```



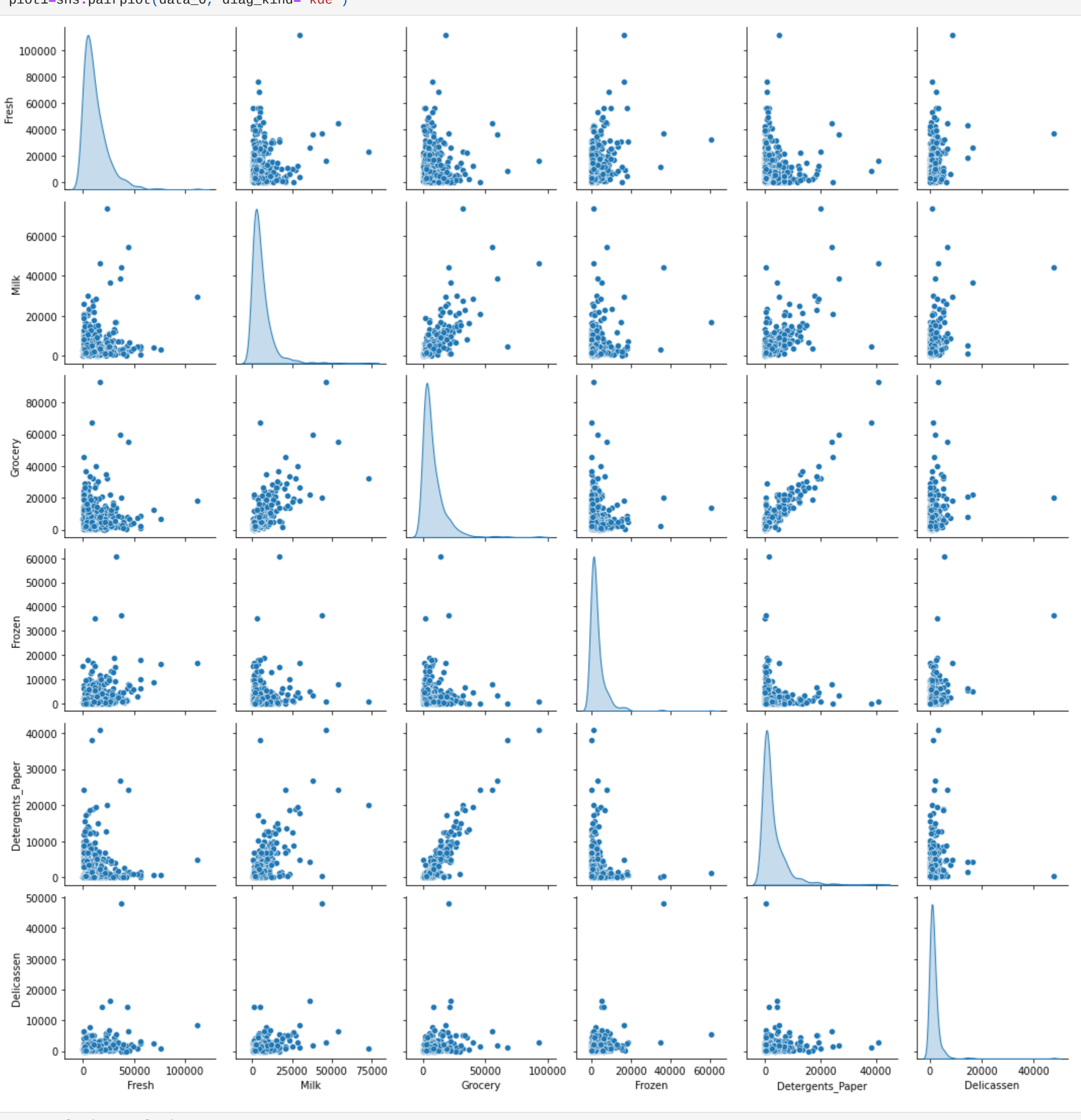
In [170…
```python
# Drpping Channel and Region
data_0.drop("Channel", axis=1, inplace = True)
```

In [171…
```python
data_0.drop("Region", axis=1, inplace = True)
```

In [172…
```python
print(data_0)
```

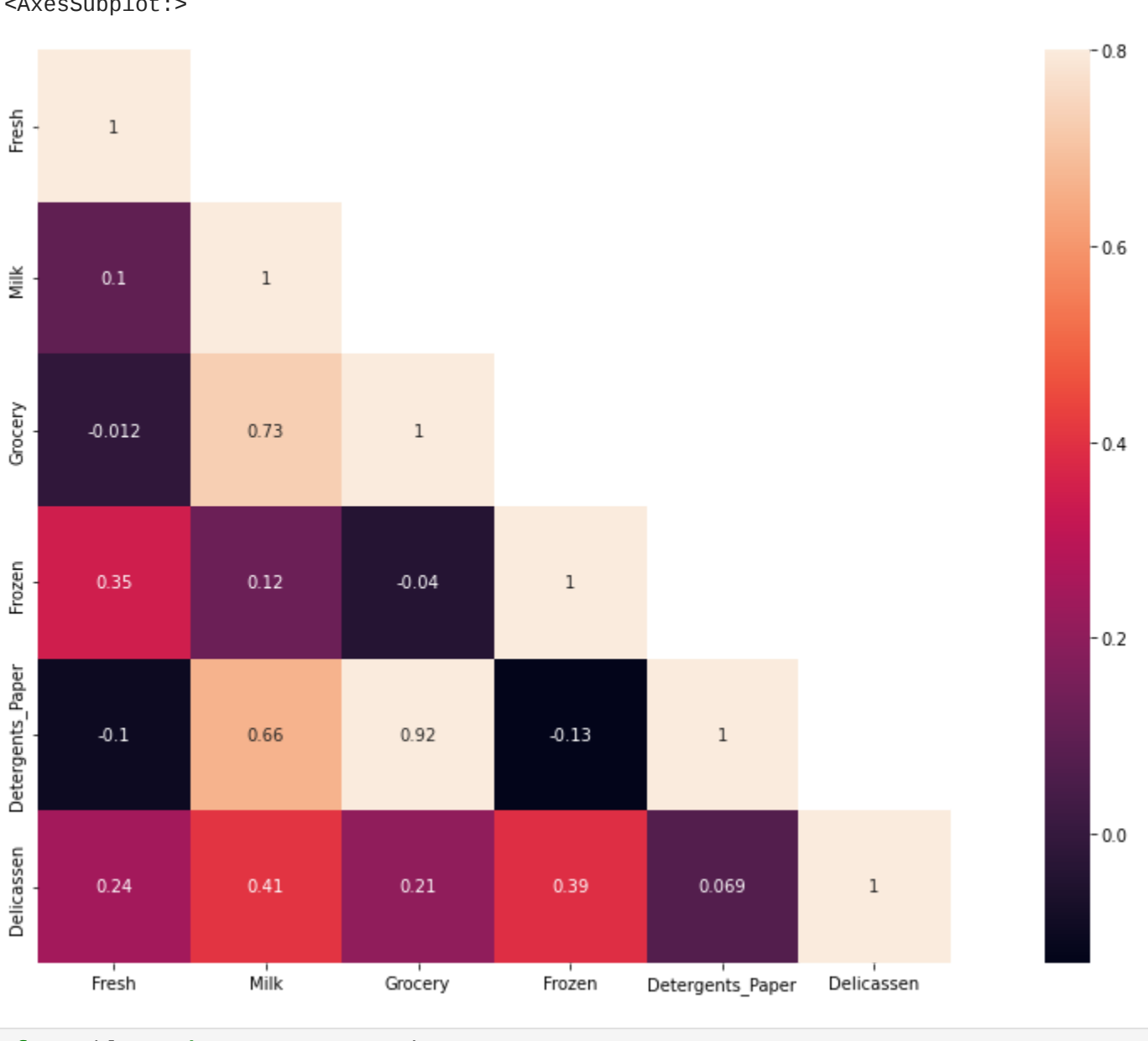
```
     Fresh   Milk  Grocery  Frozen  Detergents_Paper  Delicassen
0    12669   9656     7561     214              2674        1338
1     7057   9810     9568    1762              3293        1776
2     6353   8808     7684    2405              3516        7844
3    13265   1196     4221    6404               507        1788
4    22615   5410     7198    3915              1777        5185
..     ...    ...      ...     ...               ...         ...
435  29703  12051    16027   13135               182        2204
436  39228   1431      764    4510                93        2346
437  14531  15488    30243     437             14841        1867
438  10290   1981     2232    1038               168        2125
439   2787   1698     2510      65               477          52

[440 rows x 6 columns]
```

In [213…
```python
plot1=sns.pairplot(data_0, diag_kind='kde')
```



In [214…
```python
# Correlation analysis
corrMatt = data_0.corr()
mask = np.array(corrMatt)
mask[np.tril_indices_from(mask)] = False
fig,ax= plt.subplots()
fig.set_size_inches(20,10)
sns.heatmap(corrMatt, mask=mask,vmax=.8, square=True,annot=True)
```

Out[214…
```
<AxesSubplot:>
```



In [235…
```python
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
data = preprocessing.normalize(data_0, norm='max', axis=0, copy=True)
print(data)
```

```
[[0.11296377 0.13137772 0.08149386 0.00351575 0.06549587 0.02790814]
 [0.06292409 0.13347302 0.10312567 0.02894741 0.08065741 0.03704399]
 [0.05664684 0.11984    0.08281957 0.03951108 0.08611948 0.16361095]
 ...
 [0.12956639 0.21072682 0.32596465 0.00717935 0.36350944 0.03894208]
 [0.0917513  0.02695311 0.02405691 0.01705302 0.00411492 0.04432347]
 [0.02485042 0.02310267 0.02705324 0.00106787 0.01168344 0.00108462]]
```

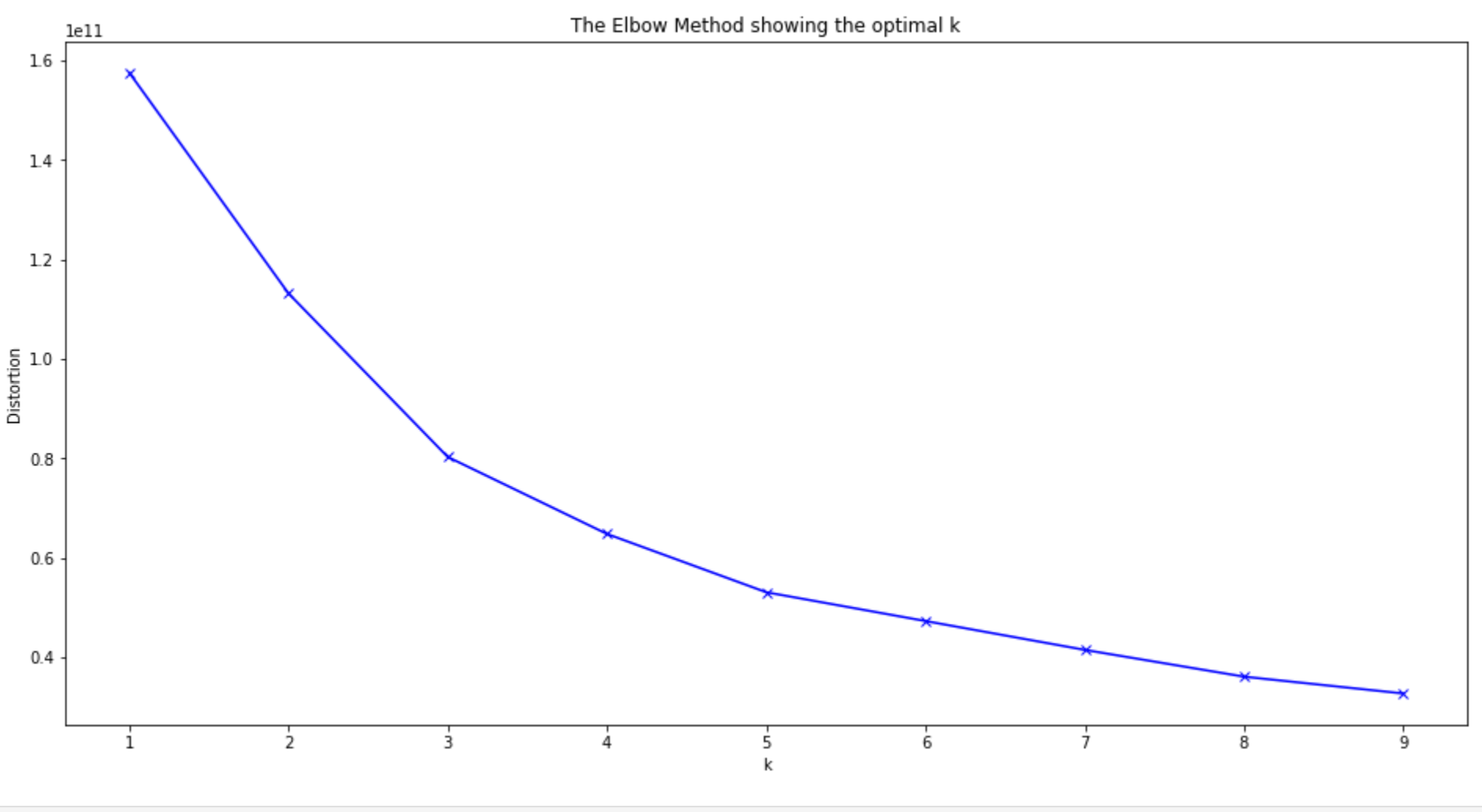
k-Means

In [174…
```python
np.random.seed(0)
```

In [212…
```python
# Using elbow method check best number of clusters
# run K-Means for a range of clusters using a for loop and collecting the distortions into a list.
distortions = []
K = range(1,10)
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(data)
    distortions.append(kmeanModel.inertia_)
```

```
C:\Users\LENOVO\.conda\envs\tf\lib\site-packages\sklearn\cluster\_kmeans.py:1036: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
```

In [176…
```python
# Plotting the distortions of K-Means
plt.figure(figsize=(16,8))
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('The Elbow Method showing the optimal k')
plt.show()
```



In [244…
```python
kmeans = KMeans(n_clusters = 3,random_state = 111)
y_means = kmeans.fit(data_0)
labels_1 = k_means_1.labels_
print(labels_1)
```

```
[0 0 0 0 2 0 0 0 0 1 0 0 2 0 2 0 0 0 0 0 0 0 2 1 2 0 0 0 1 2 0 0 0 2 0 0 2
 0 1 2 2 0 0 1 0 1 1 1 0 1 0 0 2 0 2 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0
 0 0 0 1 0 0 0 0 0 0 0 1 1 2 0 2 0 0 1 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 1 0
 1 0 0 0 0 0 0 0 0 0 0 0 0 2 2 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 2 2 0 0 1 0 0
 0 2 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 2 0 0 0 0 2 0 2 0
 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 1 1 2 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 2 2 2 0 0 0 0 0 0 0 0 0 1 0 2 0 2 0 0 2
 2 0 0 0 0 0 1 1 0 1 0 0 0 0 2 0 0 2 0 0 0 0 0 2 2 2 2 0 0 0 2 0 0 0 0 0 0
 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 2 1 0 0 0 0 0 0 1 0 0 0 0 2 2 0 0 0 0 0 1 0
 1 0 2 0 0 0 0 0 0 1 0 0 0 2 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 2 0 0 0 0 0 0 2 0 0 2 0 2 0 1 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 2 2 2 0 0 2
 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 2 0 0 0 0 0 2 0 0 0 0 0 0 0 0 2 2 1 0 0]
```

In [188…
```python
print('SSE: ', kmeans.inertia_)
print('\nCentroids: \n', kmeans.cluster_centers_)
```

```
SSE:  80332413843.01636

Centroids:
 [[ 8000.04       18511.42       27573.9         1996.68
  12407.36        2252.02      ]
 [ 8253.46969697  3824.6030303   5280.45454545  2572.66060606
   1773.05757576  1137.4969697 ]
 [35941.4        6044.45        6288.61666667  6713.96666667
   1039.66666667  3049.46666667]]
```

In [189…
```python
#count number of records in every cluster
pd.Series(kmeans.labels_).value_counts()
```

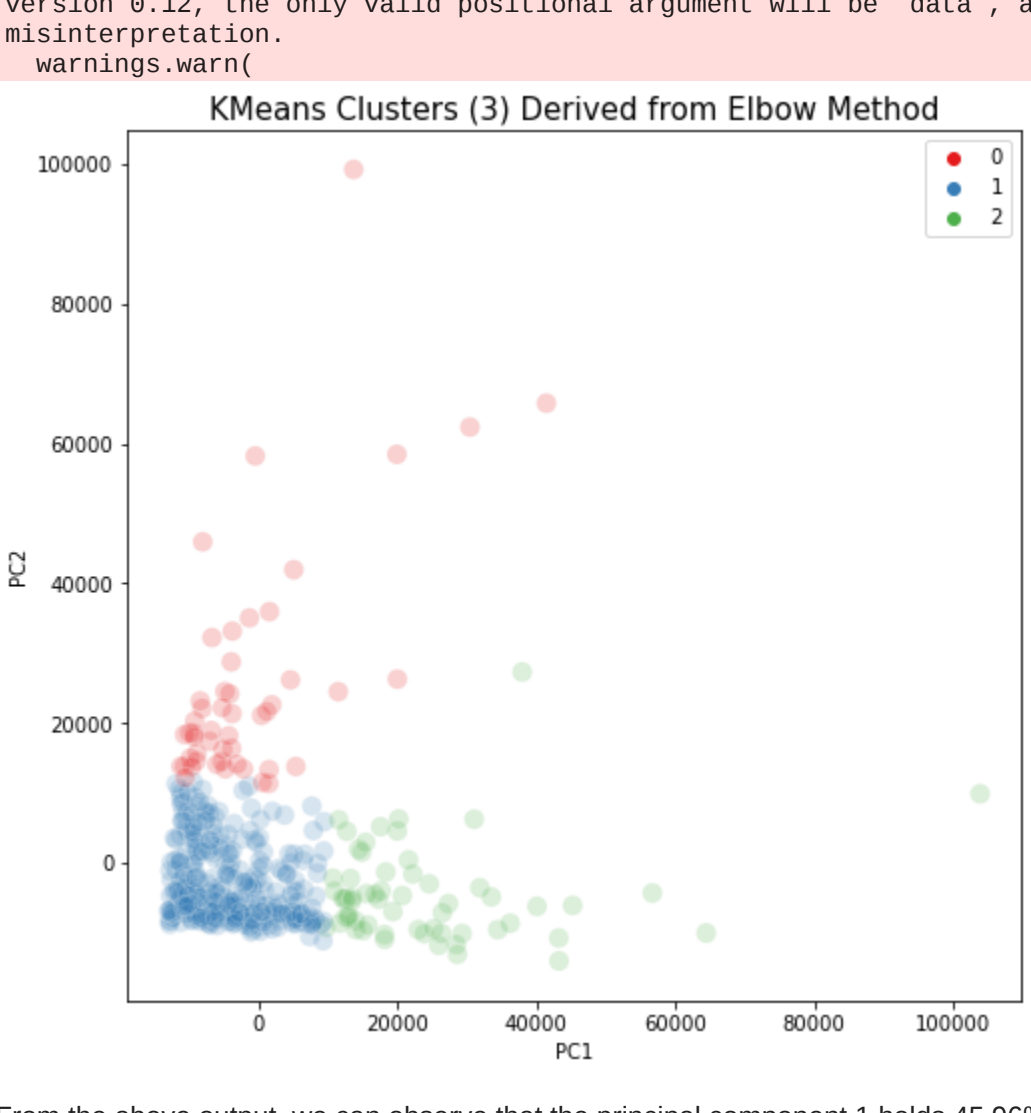
Out[189…
```
1    330
2     60
0     50
dtype: int64
```

In [193…
```python
pca2 = PCA(n_components=2).fit(data_0)
pca2d = pca2.transform(data_0)

print("Explained variance is:",pca2.explained_variance_)
print("Explained variance ratio",pca2.explained_variance_ratio_)
print("Variance for 1st component is 45.96% & 2nd component is 40.51%")
```

```
Explained variance is: [1.64995904e+08 1.45452098e+08]
Explained variance ratio [0.45961362 0.40517227]
Variance for 1st component is 45.96% & 2nd component is 40.51%
```

In [234…
```python
pca4 = PCA(n_components=4).fit(data_0)
pca4d = pca4.transform(data_0)

print("Explained variance is:",pca4.explained_variance_)
print("Explained variance ratio is:",pca4.explained_variance_ratio_)

plt.figure(figsize = (8,8))
sns.scatterplot(pca4d[:,0], pca4d[:,1],
                hue=y_means.labels_,
                palette='Set1',
                s=100, alpha=0.2).set_title('KMeans Clusters (3) Derived from Elbow Method', fontsize=15)
plt.legend()
plt.ylabel('PC2')
plt.xlabel('PC1')
plt.show()
```

```
Explained variance is: [1.64995904e+08 1.45452098e+08 2.51399785e+07 1.58039005e+07]
Explained variance ratio is: [0.45961362 0.40517227 0.07003008 0.04402344]
C:\Users\LENOVO\.conda\envs\tf\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



From the above output, we can observe that the principal component 1 holds 45.96% of the information while the principal component 2 holds only 40.51% , 3 holds 7% and 4 holds 4.4%of the information.

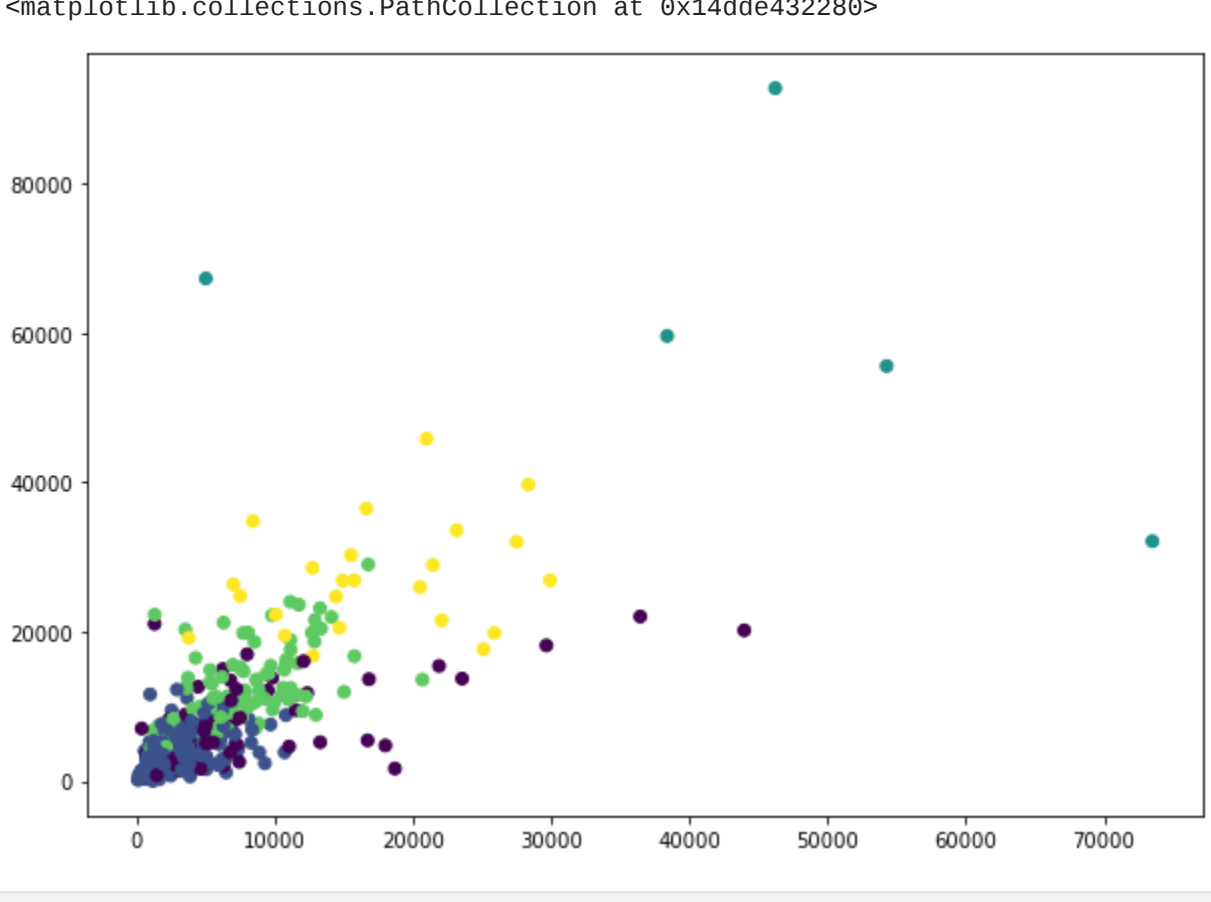
Hierarchical Clustering. Agglomerative (the bottom up approach)

In [238…
```python
from scipy.cluster import hierarchy
from scipy.spatial import distance_matrix
from matplotlib import pyplot as plt
from sklearn import manifold, datasets
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import AgglomerativeClustering
cluster = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
cluster.fit_predict(data)
```

Out[238…
```
array([1, 3, 3, 0, 0, 1, 1, 3, 3, 3, 3, 1, 0, 0, 0, 1, 3, 1, 1, 1, 1, 1,
       0, 0, 0, 1, 1, 1, 4, 0, 1, 1, 1, 0, 1, 3, 0, 3, 3, 0, 0, 1, 3, 3,
       3, 4, 3, 2, 3, 4, 1, 3, 0, 3, 1, 3, 4, 3, 1, 3, 1, 2, 3, 3, 1, 4,
       3, 0, 1, 1, 0, 0, 1, 0, 3, 1, 1, 4, 1, 1, 1, 3, 3, 1, 1, 2, 2, 0,
       1, 0, 1, 0, 4, 0, 3, 1, 3, 1, 1, 1, 3, 3, 3, 0, 1, 1, 3, 3, 1, 3,
       1, 3, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 3, 1, 1, 1, 1, 0, 0, 1, 1, 4, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 4, 3, 1, 3, 3, 3, 1, 1, 4, 3, 3, 3, 1, 1, 1, 3, 4, 3, 3, 1, 3,
       0, 1, 1, 1, 1, 0, 3, 0, 1, 1, 1, 3, 3, 3, 0, 1, 1, 3, 1, 0, 0, 3,
       1, 1, 3, 4, 0, 1, 1, 3, 1, 3, 3, 4, 1, 4, 1, 3, 3, 3, 4, 1, 3, 1,
       1, 3, 1, 1, 1, 1, 1, 1, 3, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 3, 3, 1, 1, 1, 1, 1, 4, 1, 0, 0, 1, 1, 3, 0, 0, 1, 1, 1, 3,
       3, 0, 3, 1, 3, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0,
       1, 0, 1, 0, 1, 1, 1, 3, 1, 3, 1, 1, 3, 1, 1, 3, 3, 3, 4, 3, 3, 1,
       1, 3, 0, 0, 4, 1, 1, 3, 1, 1, 1, 4, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 4, 1, 2, 0, 0, 1, 1, 0, 1, 3, 3, 3, 4, 1, 3, 3, 0, 1, 4, 1, 4,
       1, 3, 0, 1, 1, 3, 0, 1, 1, 1, 1, 3, 1, 3, 1, 1, 1, 1, 0, 0, 1, 1,
       1, 1, 3, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       3, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 3, 1, 1, 3, 1, 3, 0, 1, 3, 3, 3,
       3, 1, 3, 1, 1, 1, 1, 0, 3, 0, 1, 1, 3, 0, 1, 1, 1, 0, 0, 4, 1, 1],
      dtype=int64)
```

In [241…
```python
plt.figure(figsize=(10, 7))
plt.scatter(data_0['Milk'], data_0['Grocery'], c=cluster.labels_)
```

Out[241…
```
<matplotlib.collections.PathCollection at 0x14dde432280>
```



In [239…
```python
# Normalization - MinMaxScaler transforms
from sklearn.preprocessing import MinMaxScaler
x = data_0.values #returns a numpy array
min_max_scaler = MinMaxScaler()
feature_mtx = min_max_scaler.fit_transform(x)
feature_mtx [0:5]
```

Out[239…
```
array([[0.11294004, 0.13072723, 0.08146416, 0.0031063 , 0.0654272 ,
        0.02784731],
       [0.06289903, 0.13282409, 0.10309667, 0.02854842, 0.08058985,
        0.03698373],
       [0.05662161, 0.11918086, 0.08278992, 0.03911643, 0.08605232,
        0.16355861],
       [0.11825445, 0.01553586, 0.04546385, 0.10484189, 0.01234568,
        0.03723404],
       [0.20162642, 0.07291369, 0.07755155, 0.063934  , 0.04345483,
        0.10809345]])
```

In [240…
```python
from sklearn.metrics.pairwise import euclidean_distances
dist_matrix = euclidean_distances(feature_mtx,feature_mtx)
print(dist_matrix)
```

```
[[0.         0.06274669 0.15312258 ... 0.39409586 0.13744499 0.16152769]
 [0.06274669 0.         0.12961725 ... 0.37493175 0.15652802 0.16206847]
 [0.15312258 0.12961725 0.         ... 0.40775074 0.18651639 0.21663802]
 ...
 [0.39409586 0.37493175 0.40775074 ... 0.         0.50568523 0.51073072]
 [0.13744499 0.15652802 0.18651639 ... 0.50568523 0.         0.08174748]
 [0.16152769 0.16206847 0.21663802 ... 0.51073072 0.08174748 0.        ]]
```

In [243…
```python
import scipy.cluster.hierarchy as shc
plt.figure(figsize=(10, 7))
plt.title("Dendrograms")
dend = shc.dendrogram(shc.linkage(data_0, method='ward'))
```