In [3]:
```
from datetime import datetime
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns

# Note the seperator keyword here, it is because the data in the file is tab separated rather than comma seperated
df=pd.read_csv('../input/customer-personality-analysis/marketing_campaign.csv',sep='\t')


## The following code creates features that would be helpful to describe the profile of the customer
#  creates a new field to store the age of the customer,
#  recodes the customer's education level to numeric form (0: high-school, 1: diploma, 2: bachelors, 3: masters, and 4: doctorates)
#  recodes the customer's marital status to numeric form (0: not living with a partner, 1: living with a partner)
#  creates a new field to store the number of children in the household
#  creates a new field to store the household size

df['Age']=2022-df['Year_Birth']
df["Education"].replace({"Basic":0,"2n Cycle":1, "Graduation":2, "Master":3, "PhD":4},inplace=True)
df['Marital_Status'].replace({"Married":1, "Together":1, "Absurd":0, "Widow":0, "YOLO":0, "Divorced":0, "Single":0,"Alone":0},inplace=True)
df['Children']=df['Kidhome']+df['Teenhome']
df['Family_Size']=df['Marital_Status']+df['Children']+1

## The following code creates features that would be helpful to describe the customer's purchasing preference and behavior
#  creates a new field to store the total spending of the customer
#  creates subsequent fields to store the spending proportion for each product by the customer

df['Total_Spending']=df["MntWines"]+ df["MntFruits"]+ df["MntMeatProducts"]+ df["MntFishProducts"]+ df["MntSweetProducts"]+ df["MntGoldProds"]
df['Prop_Wines']=df["MntWines"]/df["Total_Spending"]
df['Prop_Fruits']=df["MntFruits"]/df["Total_Spending"]
df['Prop_MeatProducts']=df["MntMeatProducts"]/df["Total_Spending"]
df['Prop_FishProducts']=df["MntFishProducts"]/df["Total_Spending"]
df['Prop_SweetProducts']=df["MntSweetProducts"]/df["Total_Spending"]
df['Prop_GoldProds']=df["MntGoldProds"]/df["Total_Spending"]

## The following code works out how long the customer has been with the company and store the total number of promotions the customers responded to
#df['Dt_Customer']=pd.to_datetime(df['Dt_Customer'])
df['Dt_Customer']=pd.to_datetime(df['Dt_Customer'], format='%d-%m-%Y')

today=datetime.today()
df['Days_as_Customer']=(today-df['Dt_Customer']).dt.days
df['Offers_Responded_To']=df['AcceptedCmp1']+df['AcceptedCmp2']+df['AcceptedCmp3']+df['AcceptedCmp4']+df['AcceptedCmp5']+df['Response']

## The following code remove outliers in the dataset that are plausibly caused by data-entry errors (improbable values)
#  Generally, we would remove outliers when we do customer segmentation, as we are more interested in the general population rather than the outliers
df = df[(df["Age"]<90)]
df = df[(df["Income"]<110000)]
df = df[(df["NumWebVisitsMonth"]<11)]
df = df[(df["NumWebPurchases"]<20)]
df = df[(df["NumCatalogPurchases"]<20)]

## Finally we are going to drop some of the fields that are no longer relevant / unhelpful for the clustering
#  Note: depending on the question we would like to have answered, there is a case we can make to keep
#  'MntFruits','MntWines','MntMeatProducts','MntFishProducts','MntSweetProducts' and 'MntGoldProds', however
#  because we are interested in the purchasing preferance rather than amount spent, the field for the spending
#  proportion we created above would be more helpful
#  We dropped the responses to the individual promotions because very few customers react to those promotions
#  (not enough to form their own clusters), so instead we look at the overall response, using the field created above
#  Less than 1% of customers filed a complaint. We drop the field as it wouldn't create a meaningful cluster
fields_to_drop=['ID','Year_Birth','Dt_Customer','Z_CostContact','Z_Revenue','AcceptedCmp1',
            'AcceptedCmp2','AcceptedCmp3','AcceptedCmp4','AcceptedCmp5','Response','Complain',
            'MntFruits','MntWines','MntMeatProducts','MntFishProducts','MntSweetProducts','MntGoldProds']
df.drop(fields_to_drop,axis=1,inplace=True)
df.dropna(inplace=True)

## Helper function for to do 3d scatter plots,
def scatter_3d(x,y,z,c=None):
    fig = plt.figure(figsize=(10,8))
    ax = plt.subplot(111, projection='3d', label="bla")
    ax.scatter(x, y, z, s=40, c=c, marker='o',cmap=plt.cm.viridis)
    ax.set_title("The Plot Of The Clusters")
    plt.show()
```

In [9]:
```
df[:5]
```

Out[9]:

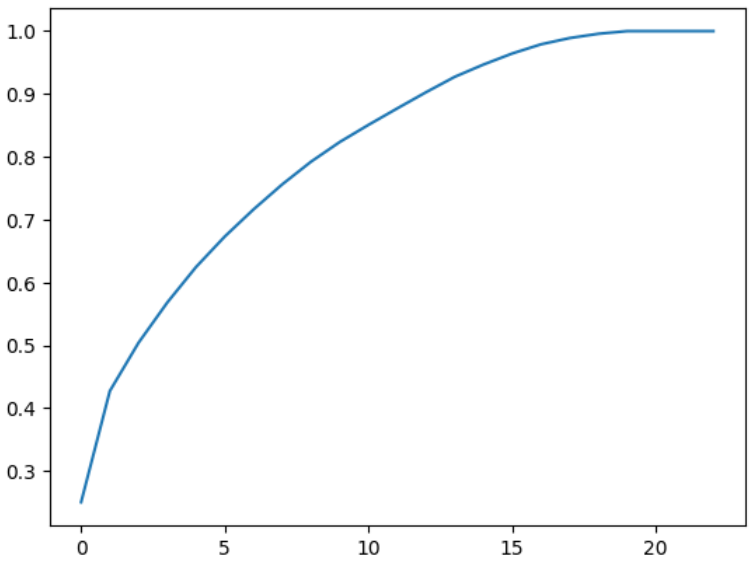
	Education	Marital_Status	Income	Kidhome	Teenhome	Recency	NumDealsPurchases	NumWebPurchases	NumCatalogPurchases	NumStorePurchases	...	F
0	2	0	58138.0	0	0	58	3	8	10	4	...	
1	2	0	46344.0	1	1	38	2	1	1	2	...	
2	2	1	71613.0	0	0	26	1	8	2	10	...	
3	2	1	26646.0	1	0	26	2	2	0	4	...	
4	4	1	58293.0	1	0	94	5	5	3	6	...	

5 rows × 23 columns

In [7]:
```
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
df_scaled[:5]
```

Out[7]:
```
array([[-0.45432982, -1.35083021,  0.30773769, -0.82440588, -0.93141916,
         0.309307  ,  0.38515418,  1.49240533,  2.67632197, -0.57633328,
         0.74759881,  1.01783812, -1.26767782, -1.7644021 ,  1.67532915,
        -0.29771875,  0.09021044,  0.73250209,  0.44165647,  0.06813859,
        -0.61059243,  1.52616824,  0.61111078],
       [-0.45432982, -1.35083021, -0.26935211,  1.0366628 ,  0.9079239 ,
        -0.38238954, -0.1684576 , -1.1790211 , -0.59962764, -1.1971306 ,
        -0.12990405,  1.27425705,  1.40291303,  0.4475335 , -0.96674711,
        -0.23314743, -0.22225076, -0.2149497 ,  0.02837753, -0.2319575 ,
         0.96737653, -1.19265665, -0.50518491],
       [-0.45432982,  0.74028548,  0.96708015, -0.82440588, -0.93141916,
        -0.79740746, -0.72206937,  1.49240533, -0.23563324,  1.28605868,
        -0.56865548,  0.33405429, -1.26767782, -0.6584343 ,  0.27785359,
         0.38847121,  0.24698375, -0.69558919,  0.91092953, -0.40414957,
        -0.61339531, -0.20893637, -0.50518491],
       [-0.45432982,  0.74028548, -1.2331909 ,  1.0366628 , -0.93141916,
        -0.79740746, -0.1684576 , -0.79738876, -0.96362204, -0.57633328,
         0.30884738, -1.2899323 ,  0.06761761,  0.4475335 , -0.92354335,
        -1.11076391,  0.46854738,  1.05830037,  1.49494536,  0.10580447,
        -0.23521123, -1.06413038, -0.50518491],
       [ 1.5394459 ,  0.74028548,  0.31532196,  1.0366628 , -0.93141916,
         1.55436077,  1.49237772,  0.34750829,  0.12836116,  0.04446404,
        -0.12990405, -1.03351336,  0.06761761,  0.4475335 , -0.31038225,
        -0.22197108,  0.94347482,  0.25613814,  0.47537473,  0.23315125,
        -0.78810635, -0.95537739, -0.50518491]])
```

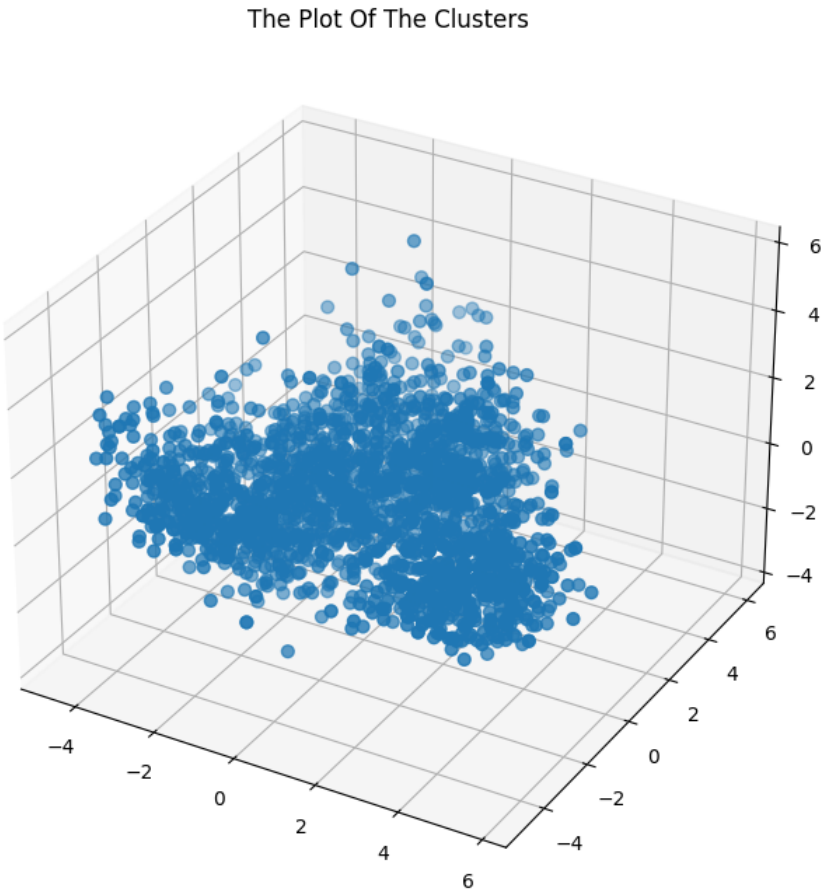
In [12]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=23)
X_reduced = pca.fit_transform(df_scaled)
# plt.plot(X_reduced)
# plt.show()
exp_var_pca = pca.explained_variance_ratio_
#
# Cumulative sum of eigenvalues; This will be used to create step plot
# for visualizing the variance explained by each principal component.
#
cum_sum_eigenvalues = np.cumsum(exp_var_pca)
plt.plot(cum_sum_eigenvalues)
plt.show()
```



In the above, it looks like 80% of the variance is explained at roughly 8 componants.

In [15]:
```
pca8 = PCA(n_components=8)
df_transformed = pca8.fit_transform(df_scaled)
scatter_3d(df_transformed[:,0], df_transformed[:,1], df_transformed[:,2])
```

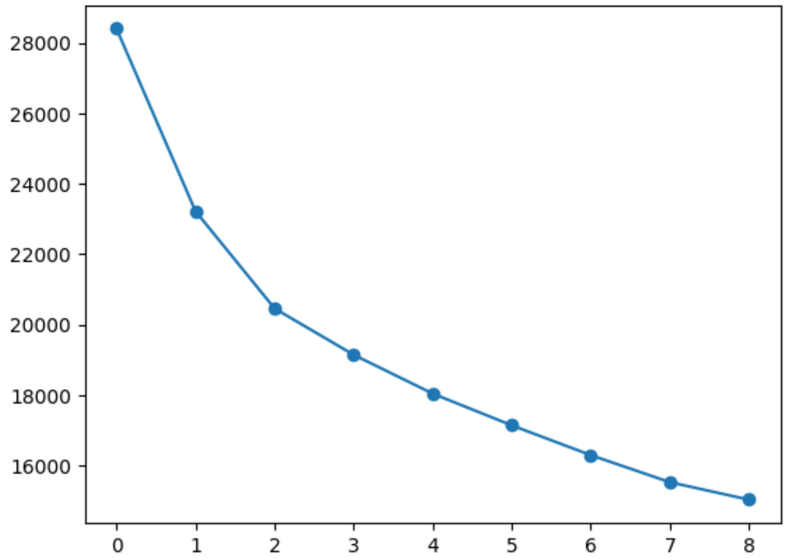
```
/tmp/ipykernel_32/2085842839.py:73: UserWarning: No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  ax.scatter(x, y, z, s=40, c=c, marker='o',cmap=plt.cm.viridis)
```



In [18]:
```
inertia_scores = []
for i in range(2, 11):
    kmeans = KMeans(n_clusters=i, random_state=1569660)
    kmeans.fit(df_transformed)
    inertia_scores.append(kmeans.inertia_)

plt.plot(inertia_scores, marker='o')
plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` w
ill change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` w
ill change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` w
ill change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` w
ill change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` w
ill change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` w
ill change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` w
ill change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` w
ill change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` w
ill change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

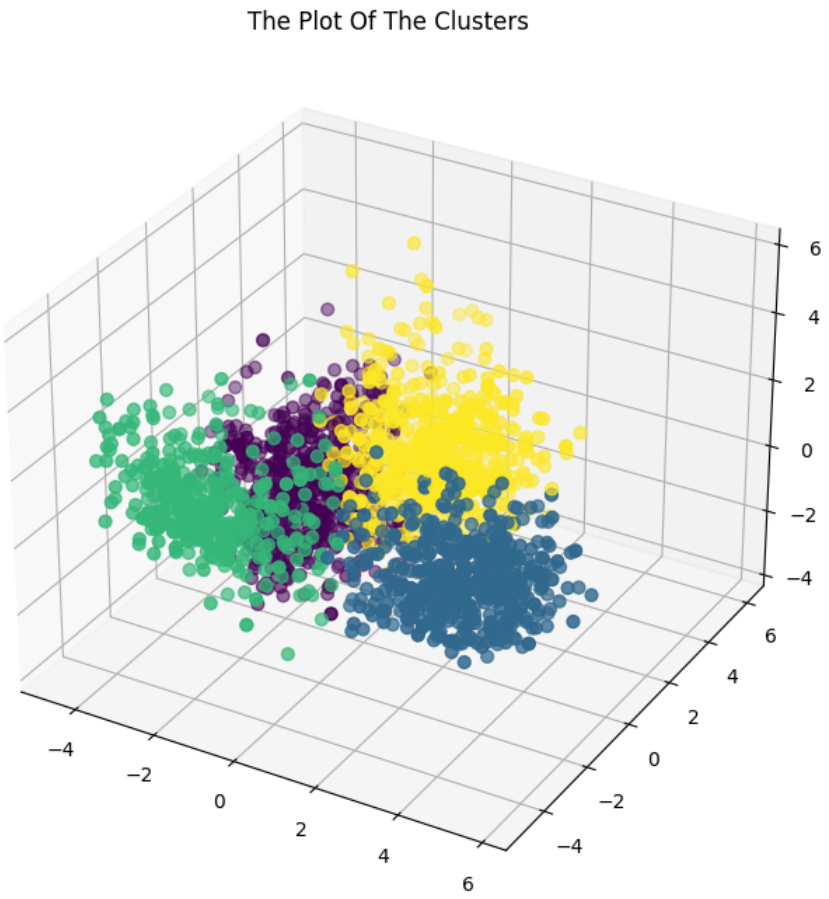


This one is tricky. But I think where the decrease begins to slow is at 2 clusters (or, the "elbow")

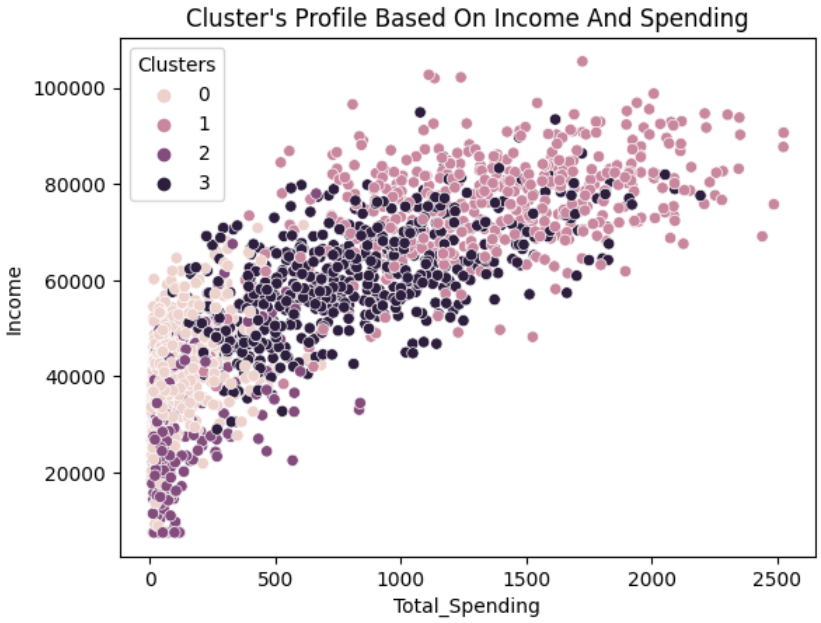
In [19]:
```
kmeans = KMeans(n_clusters=4, random_state=1569660).fit(df_transformed)
df['Clusters'] = kmeans.labels_
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` w
ill change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

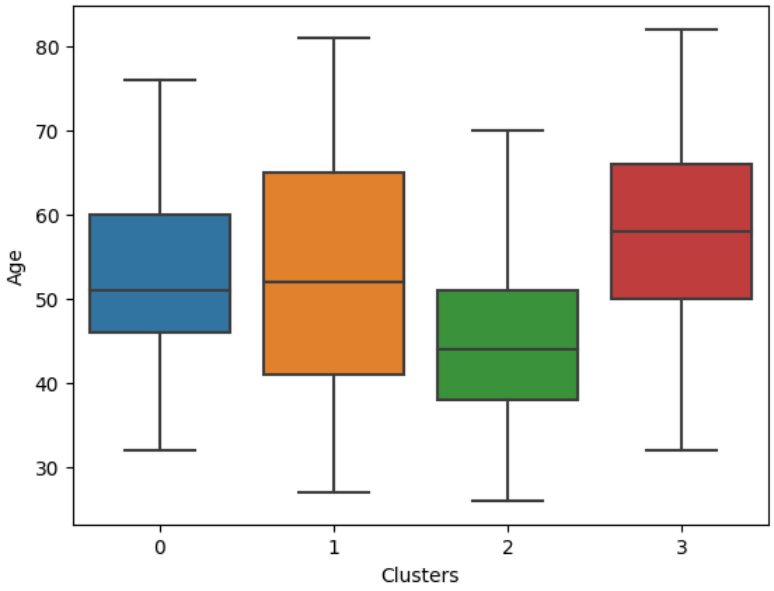
In [20]:
```
scatter_3d(df_transformed[:,0], df_transformed[:,1], df_transformed[:,2],df['Clusters'])
```

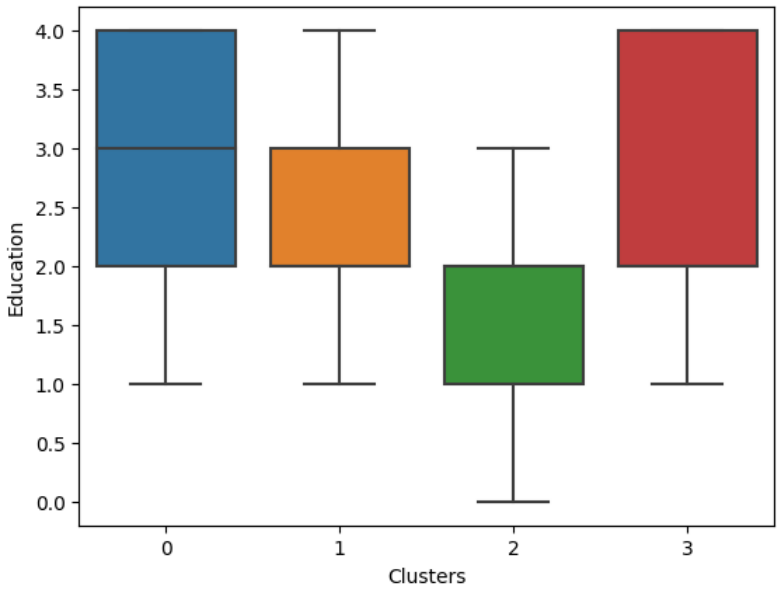


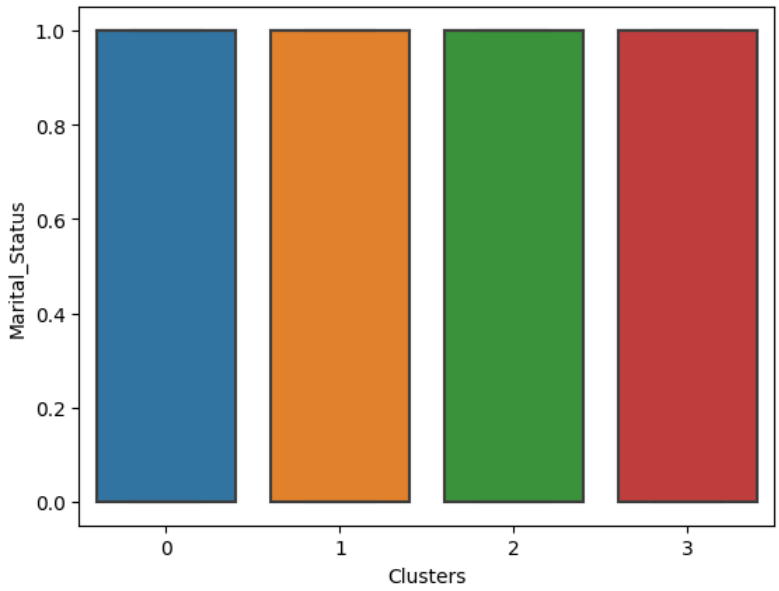
In [21]:
```
pl = sns.scatterplot(data=df, x="Total_Spending", y="Income",hue="Clusters")
pl.set_title("Cluster's Profile Based On Income And Spending")
plt.show()
```

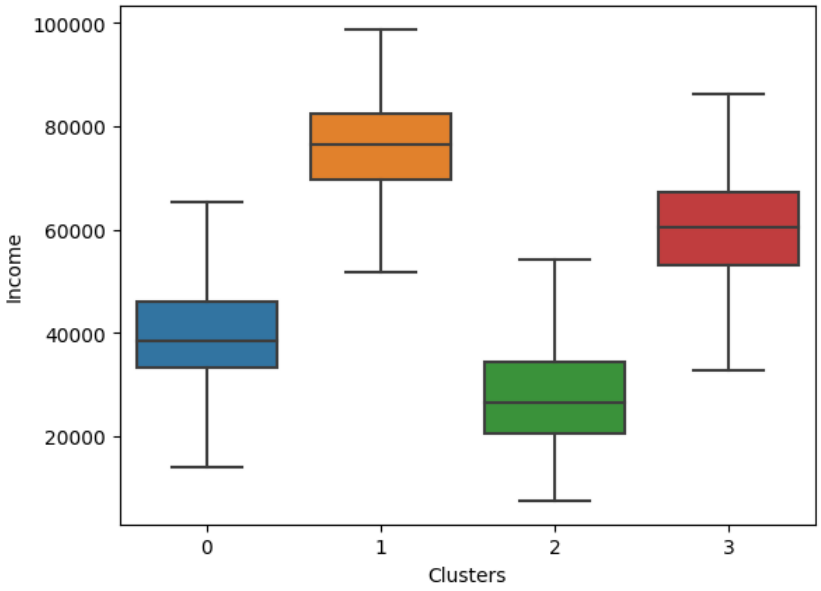


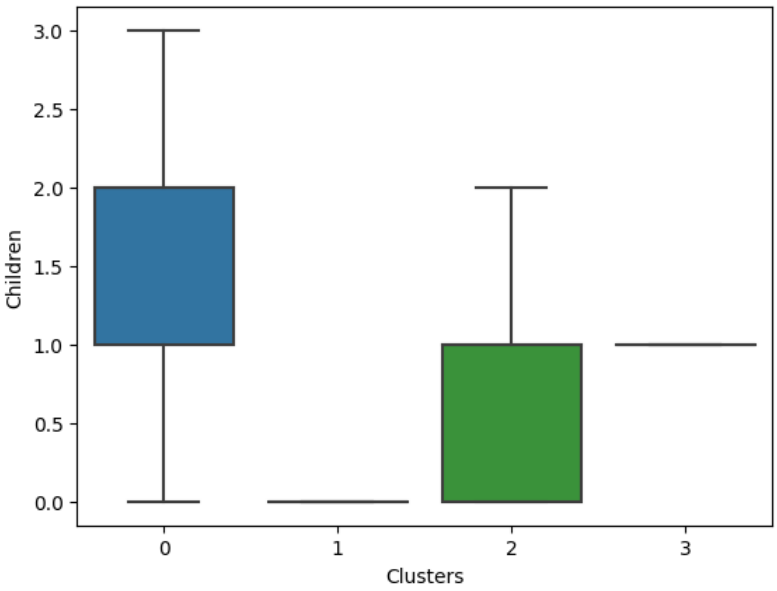
In [22]:
```
attributes = ['Age', 'Education', 'Marital_Status', 'Income', 'Children', 'Family_Size', 'Kidhome', 'Teenhome']
for attr in attributes:
    sns.boxplot(data=df, x='Clusters', y=attr, showfliers=False)
    plt.show()
```

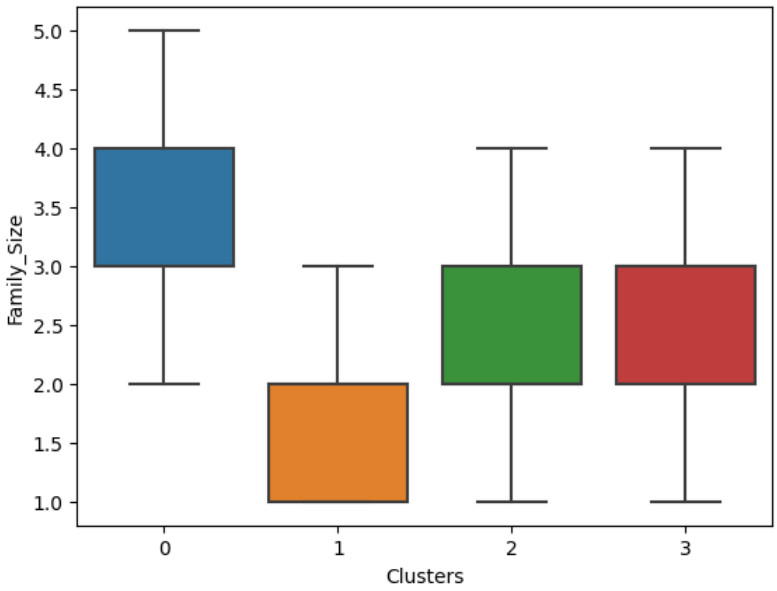

Age
80
70
60
50
40
30
0
1
2
3
Clusters

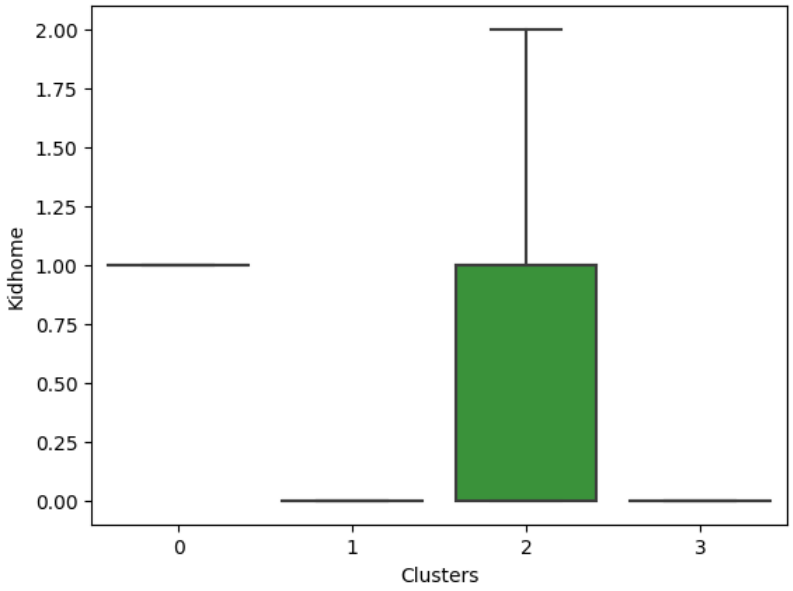

Education
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
3
Clusters

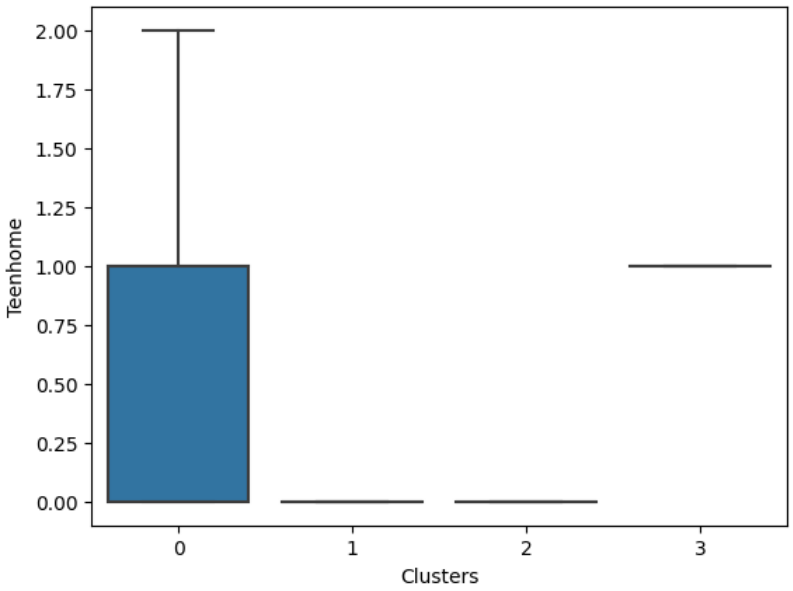

Marital_Status
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
Clusters


100000
80000
60000
40000
20000
Income
0
1
2
3
Clusters


3.0
2.5
2.0
1.5
1.0
0.5
0.0
Children
0
1
2
3
Clusters

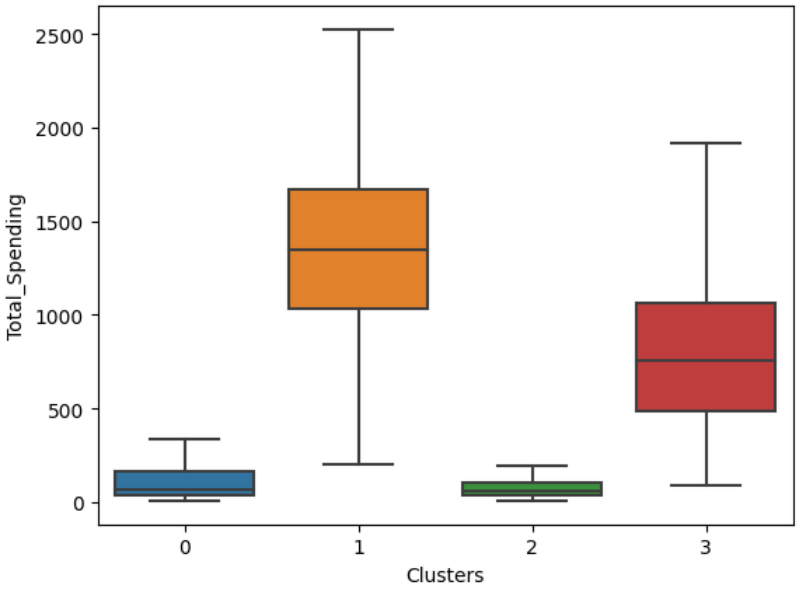

5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
Family_Size
0
1
2
3
Clusters

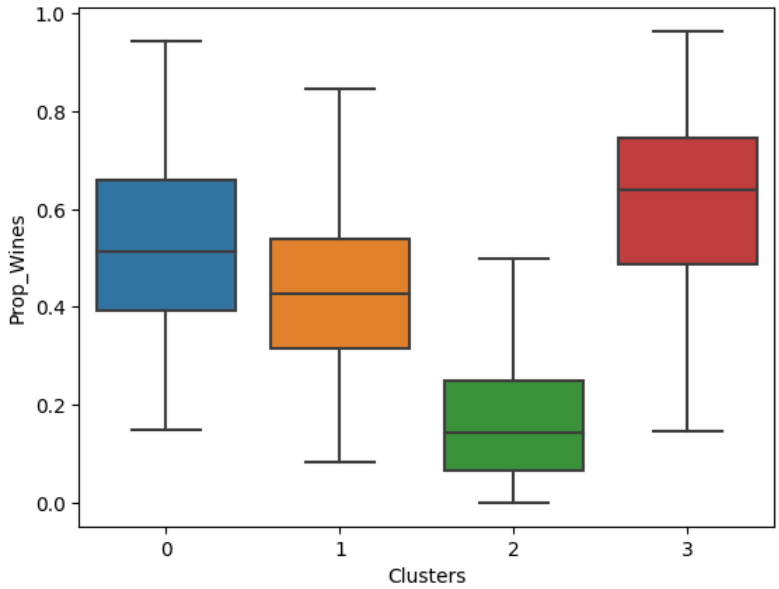


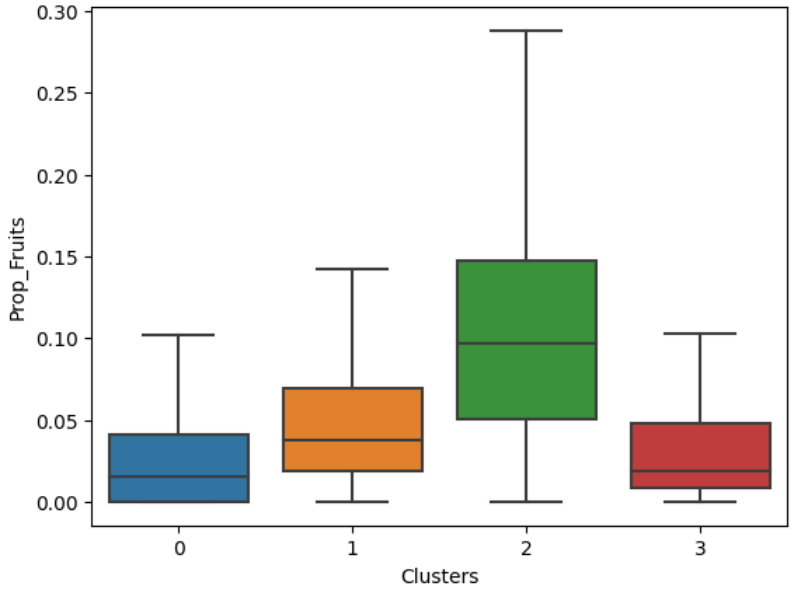


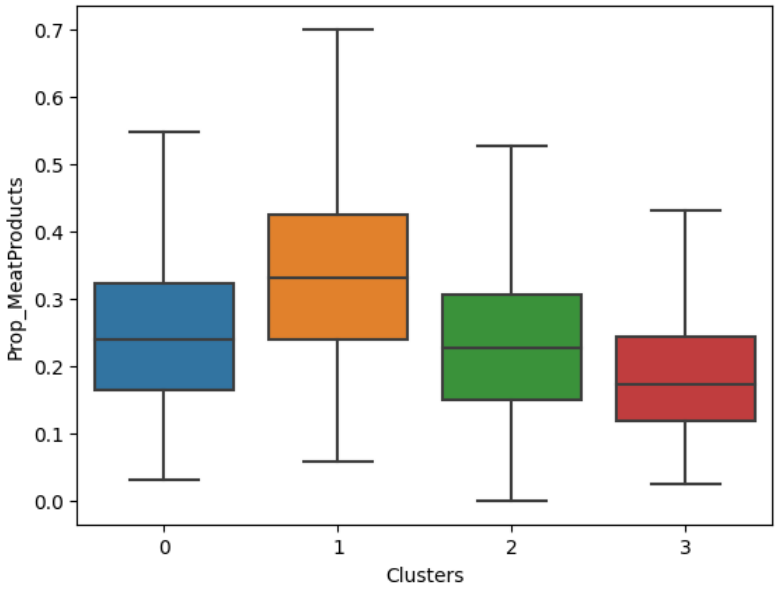
One thing we can see for sure, is that the highest income also has no kids (Orange and Red). High education also seems to be correlated with income too. The older people with kids also have a higher number of teens, while the younger group with kids have a "kidhome".

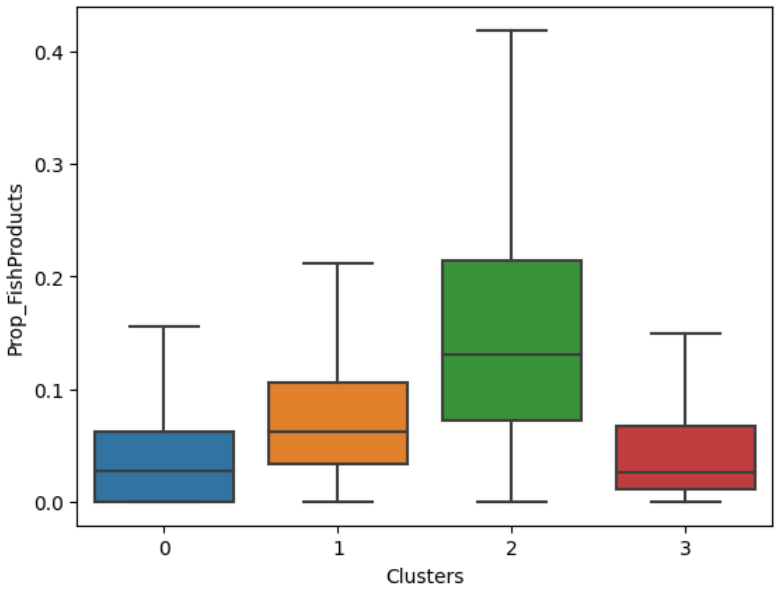
In [23]:
```
preferences = ['Total_Spending', 'Prop_Wines', 'Prop_Fruits', 'Prop_MeatProducts', 'Prop_FishProducts', 'Prop_SweetProducts', 'Prop_GoldProds']
for pref in preferences:
    sns.boxplot(data=df, x='Clusters', y=pref, showfliers=False)
    plt.show()
```

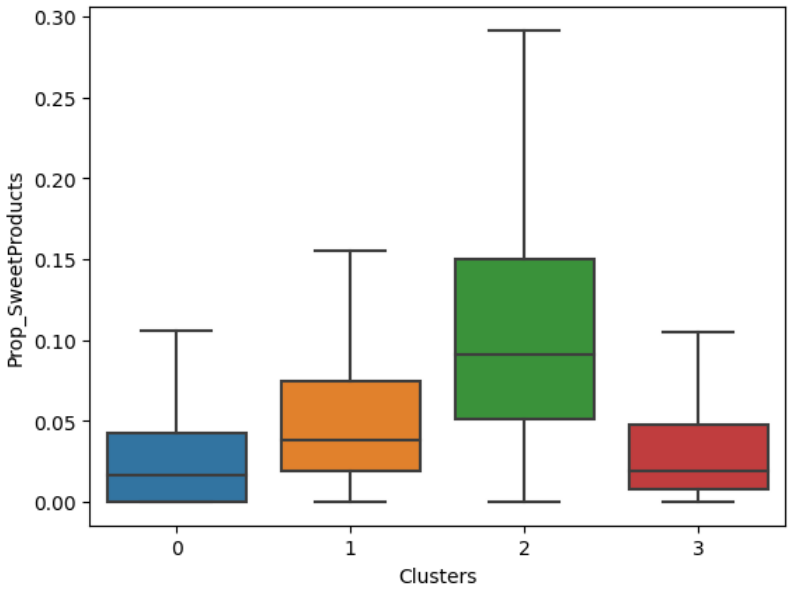

Total_Spending
2500
2000
1500
1000
500
0
0
1
2
3
Clusters

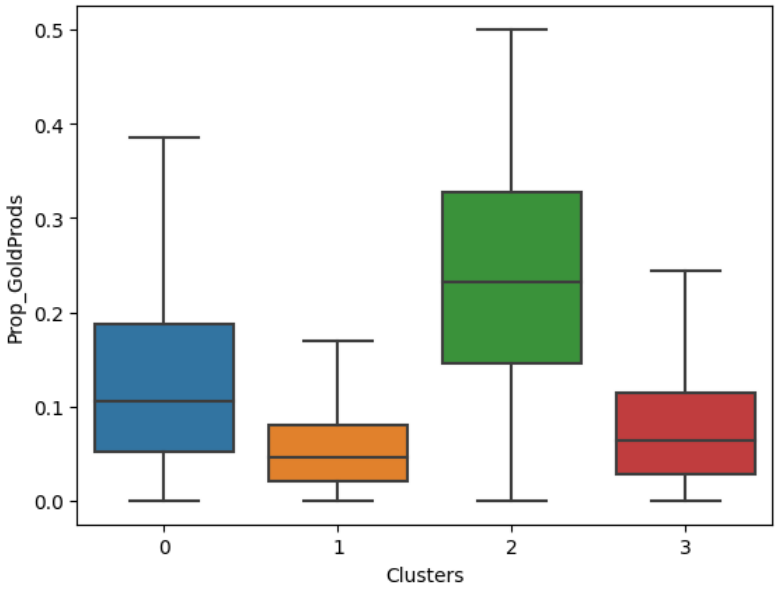

Prop_Wines
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
Clusters


Prop_Fruits
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
1
2
3
Clusters


Prop_MeatProducts
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
1
2
3
Clusters

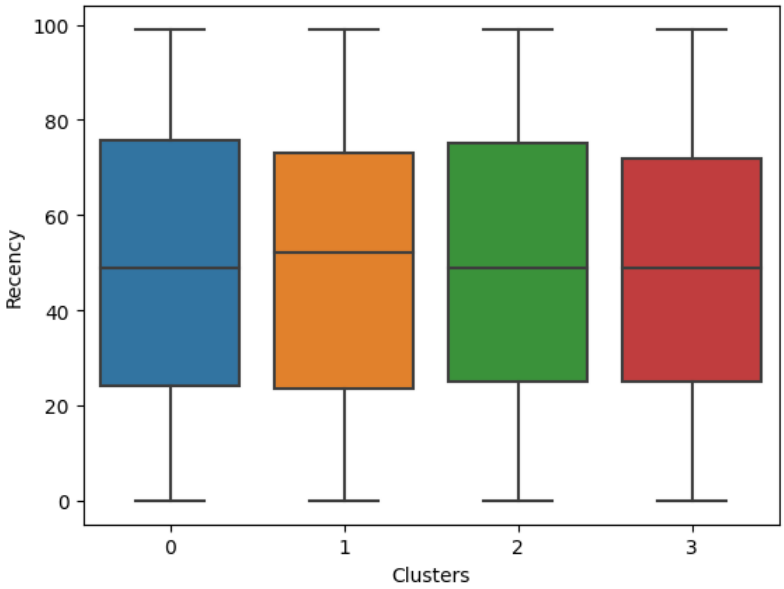

Prop_FishProducts
0.0
0.1
0.2
0.3
0.4
0
1
2
3
Clusters

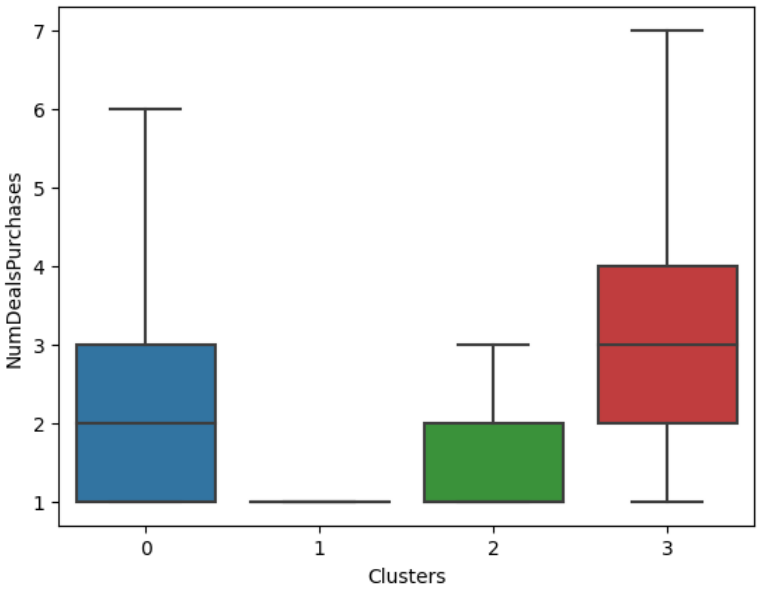

Prop_SweetProducts
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0
1
2
3
Clusters

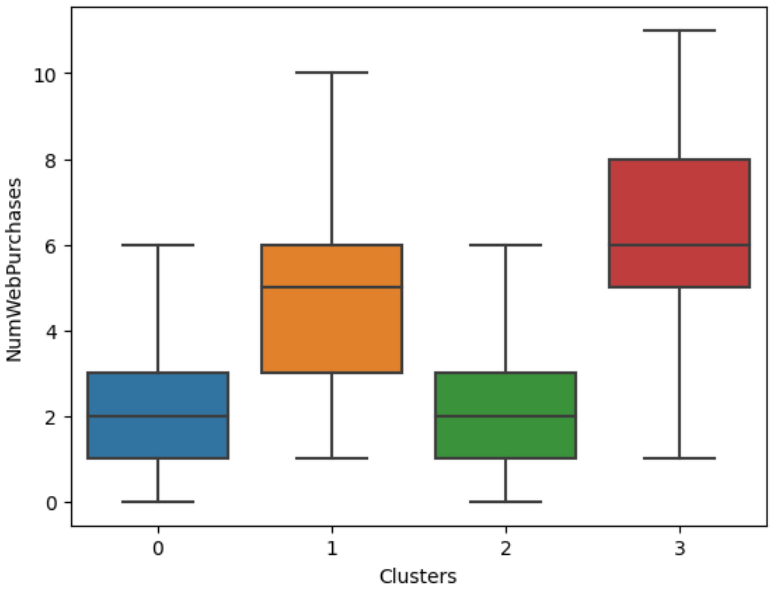


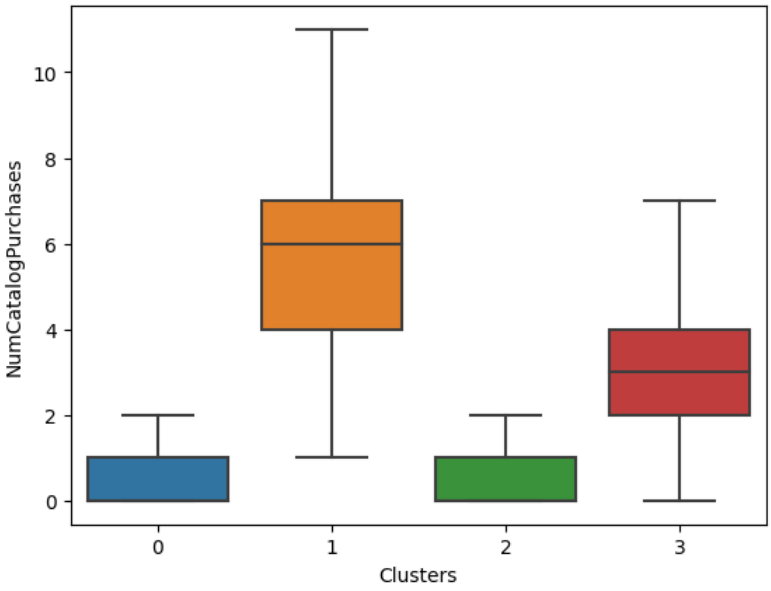
The groups with the higher income from the attributes, also seem to spend the most. Also fruit, fish and sweet products seem to be correlated somehow? They all have a similar distribution between each cluster.

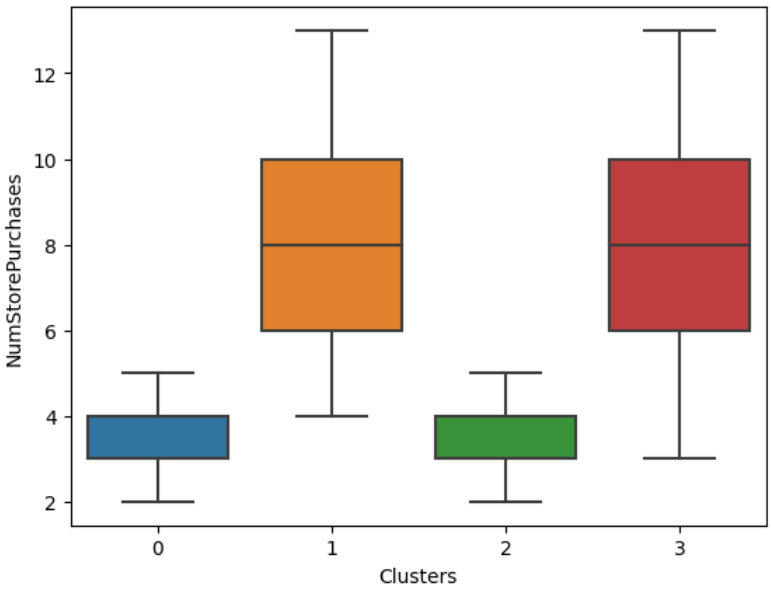
In [24]:
```
behaviors = ['Recency', 'NumDealsPurchases', 'NumWebPurchases', 'NumCatalogPurchases', 'NumStorePurchases', 'NumWebVisitsMonth', 'Days_as_Customer', 'Offers_Responded_To']
for behav in behaviors:
    sns.boxplot(data=df, x='Clusters', y=behav, showfliers=False)
    plt.show()
```

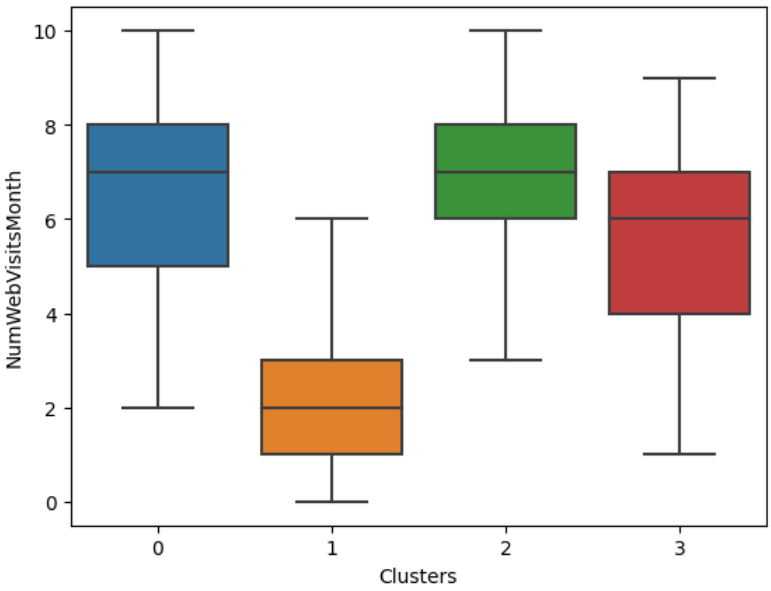

Recency
Clusters
0
1
2
3
0
20
40
60
80
100

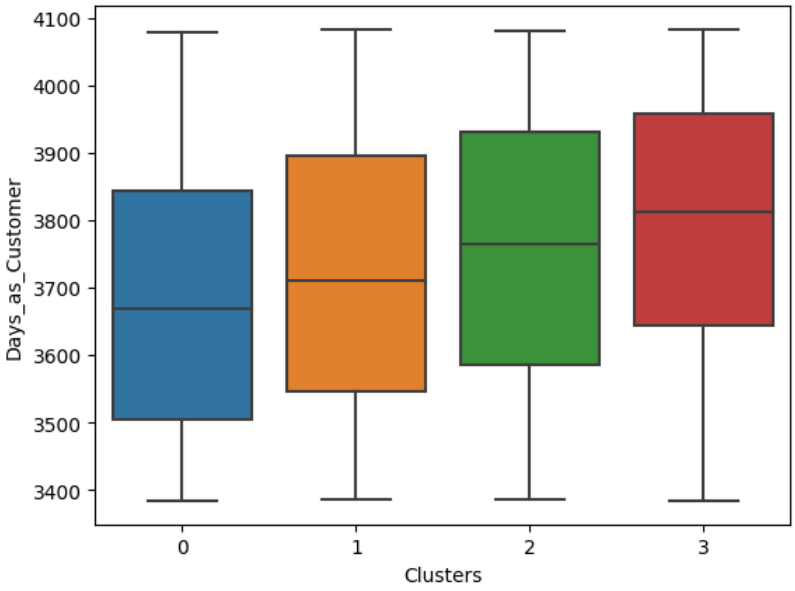

NumDealsPurchases
Clusters
0
1
2
3
1
2
3
4
5
6
7

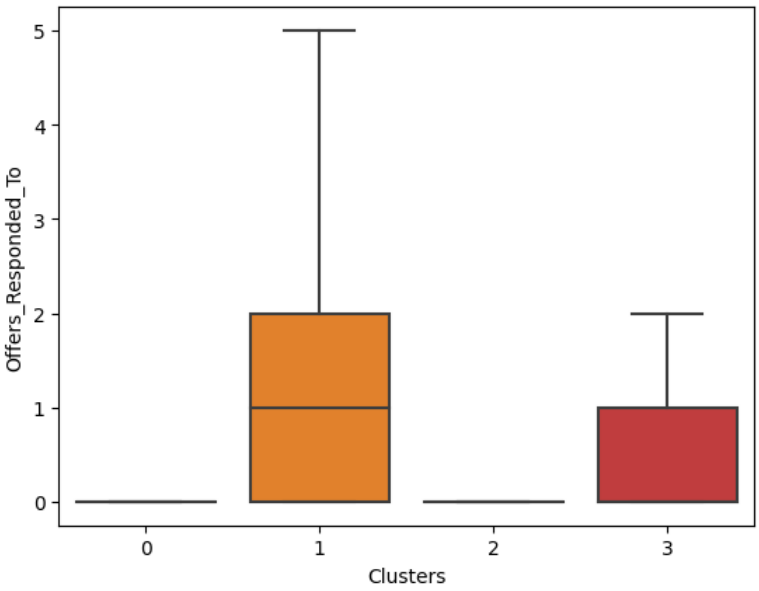

NumWebPurchases
Clusters
0
1
2
3
0
2
4
6
8
10


NumCatalogPurchases
0
2
4
6
8
10
0
1
2
3
Clusters


NumStorePurchases
2
4
6
8
10
12
0
1
2
3
Clusters


NumWebVisitsMonth
0
2
4
6
8
10
0
1
2
3
Clusters





Interestingly here, although orange and red are big spenders on web purchases, they also seem to visit the website the least. Perhaps they are better time management or more descisive, or maybe they don't need to look twice before purchasing?

In [29]:
```
df.groupby('Clusters').mean().T
```

Out[29]:

Clusters	0	1	2	3
Education	2.752427	2.508671	1.648456	2.659306
Marital_Status	0.674757	0.595376	0.624703	0.673502
Income	39498.135922	75617.423892	28162.149644	60159.176656
Kidhome	0.930421	0.023121	0.679335	0.154574
Teenhome	0.692557	0.048170	0.097387	0.971609
Recency	49.276699	49.308285	49.192399	48.545741
NumDealsPurchases	2.475728	1.053950	1.824228	3.479495
NumWebPurchases	2.412621	4.895954	2.152019	6.350158
NumCatalogPurchases	0.616505	5.868979	0.581948	3.361199
NumStorePurchases	3.411003	8.287091	3.273159	7.966877
NumWebVisitsMonth	6.396440	2.660886	6.553444	5.545741
Age	52.972492	53.129094	45.779097	58.033123
Children	1.622977	0.071291	0.776722	1.126183
Family_Size	3.297735	1.666667	2.401425	2.799685
Total_Spending	117.694175	1356.583815	109.482185	806.891167
Prop_Wines	0.524920	0.433207	0.164406	0.616673
Prop_Fruits	0.025657	0.049245	0.106268	0.034917
Prop_MeatProducts	0.251281	0.333279	0.233586	0.185989
Prop_FishProducts	0.041378	0.074425	0.152205	0.046109
Prop_SweetProducts	0.027006	0.051076	0.106694	0.035527
Prop_GoldProds	0.129759	0.058767	0.236841	0.080785
Days_as_Customer	3686.951456	3723.412331	3756.247031	3788.506309
Offers_Responded_To	0.184466	1.073218	0.180523	0.386435

In [ ]:
```

```