# Customer Segmentation--Unsupervised Clustering

In this project, I will perform unsupervised clustering on the customer records from a grocery firm's database. Customer segmentation involves separating customers into groups that reflect similarities among customers in each cluster. I will divide customers into segments to optimize the significance of each customer to the business, allowing for the modification of products according to the distinct needs and behaviors of the customers. This approach also helps the business address the concerns of different types of customers.

Table Of Contents



## Importing Libraries

In [2]:
```
import numpy as np
import pandas as pd
import datetime
import matplotlib
import matplotlib.pyplot as plt
from matplotlib import colors
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from yellowbrick.cluster import KElbowVisualizer
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt, numpy as np
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import AgglomerativeClustering
from matplotlib.colors import ListedColormap
from sklearn import metrics
import warnings
import sys
if not sys.warnoptions:
    warnings.simplefilter("ignore")
np.random.seed(42)
```

## Loading Data

In [3]:
```
data = pd.read_csv("../input/customer-personality-analysis/marketing_campaign.csv", sep="\t")
print("Number of datapoints:", len(data))
data.head()
```

```
Number of datapoints: 2240
```

Out[3]:

| | ID | Year_Birth | Education | Marital_Status | Income | Kidhome | Teenhome | Dt_Customer | Recency | MntWines | ... | NumWebVisitsMonth | A |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 5524 | 1957 | Graduation | Single | 58138.0 | 0 | 0 | 04-09-2012 | 58 | 635 | ... | 7 | |
| **1** | 2174 | 1954 | Graduation | Single | 46344.0 | 1 | 1 | 08-03-2014 | 38 | 11 | ... | 5 | |
| **2** | 4141 | 1965 | Graduation | Together | 71613.0 | 0 | 0 | 21-08-2013 | 26 | 426 | ... | 4 | |
| **3** | 6182 | 1984 | Graduation | Together | 26646.0 | 1 | 0 | 10-02-2014 | 26 | 11 | ... | 6 | |
| **4** | 5324 | 1981 | PhD | Married | 58293.0 | 1 | 0 | 19-01-2014 | 94 | 173 | ... | 5 | |

5 rows × 29 columns

## Data Cleaning

In [4]:
```
#Information on features
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 29 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   2240 non-null   int64  
 1   Year_Birth           2240 non-null   int64  
 2   Education            2240 non-null   object 
 3   Marital_Status       2240 non-null   object 
 4   Income               2216 non-null   float64
 5   Kidhome              2240 non-null   int64  
 6   Teenhome             2240 non-null   int64  
 7   Dt_Customer          2240 non-null   object 
 8   Recency              2240 non-null   int64  
 9   MntWines             2240 non-null   int64  
 10  MntFruits            2240 non-null   int64  
 11  MntMeatProducts      2240 non-null   int64  
 12  MntFishProducts      2240 non-null   int64  
 13  MntSweetProducts     2240 non-null   int64  
 14  MntGoldProds         2240 non-null   int64  
 15  NumDealsPurchases    2240 non-null   int64  
 16  NumWebPurchases      2240 non-null   int64  
 17  NumCatalogPurchases  2240 non-null   int64  
 18  NumStorePurchases    2240 non-null   int64  
 19  NumWebVisitsMonth    2240 non-null   int64  
 20  AcceptedCmp3         2240 non-null   int64  
 21  AcceptedCmp4         2240 non-null   int64  
 22  AcceptedCmp5         2240 non-null   int64  
 23  AcceptedCmp1         2240 non-null   int64  
 24  AcceptedCmp2         2240 non-null   int64  
 25  Complain             2240 non-null   int64  
 26  Z_CostContact        2240 non-null   int64  
 27  Z_Revenue            2240 non-null   int64  
 28  Response             2240 non-null   int64  
dtypes: float64(1), int64(25), object(3)
memory usage: 507.6+ KB
```

From the above output, we can conclude and observe the following:

There are missing values in the "income" column. The "Dt_Customer" column, which indicates the date a customer joined the database, is not parsed as DateTime. Some of our data frame's features are categorical (i.e., they are in dtype: object). We will need to encode them into numeric forms later.

Firstly, for handling the missing values, I will simply drop the rows that have missing income values.

In [5]:
```
#To remove the NA values
data = data.dropna()
print("The total number of data-points after removing the rows with missing values are:", len(data))
```

```
The total number of data-points after removing the rows with missing values are: 2216
```

In the next step, I will create a feature based on "Dt_Customer" that indicates the number of days a customer has been registered in the firm's database. To keep it simple, I will calculate this value relative to the most recent customer in the records. Therefore, to obtain these values, I need to check the newest and oldest recorded dates.

In [6]:
```
data["Dt_Customer"] = pd.to_datetime(data["Dt_Customer"])
dates = []
for i in data["Dt_Customer"]:
    i = i.date()
    dates.append(i)
#Dates of the newest and oldest recorded customer
print("The newest customer's enrolment date in therecords:",max(dates))
print("The oldest customer's enrolment date in the records:",min(dates))
```

```
The newest customer's enrolment date in therecords: 2014-12-06
The oldest customer's enrolment date in the records: 2012-01-08
```

I will create a feature named "Customer_For," representing the number of days since the customers began shopping in the store relative to the last recorded date.

In [7]:
```
#Created a feature "Customer_For"
days = []
d1 = max(dates) #taking it to be the newest customer
for i in dates:
    delta = d1 - i
    days.append(delta)
data["Customer_For"] = days
data["Customer_For"] = pd.to_numeric(data["Customer_For"], errors="coerce")
```

We will now explore the unique values within the categorical features to gain a clearer understanding of the data.

In [8]:
```
print("Total categories in the feature Marital_Status:\n", data["Marital_Status"].value_counts(), "\n")
print("Total categories in the feature Education:\n", data["Education"].value_counts())
```

```
Total categories in the feature Marital_Status:
 Married     857
Together    573
Single      471
Divorced    232
Widow        76
Alone         3
Absurd        2
YOLO          2
Name: Marital_Status, dtype: int64

Total categories in the feature Education:
 Graduation    1116
PhD            481
Master         365
2n Cycle       200
Basic           54
Name: Education, dtype: int64
```

**In the upcoming steps, I will be performing the following operations to engineer some new features:**

Extract the customer's "Age" from the "Year_Birth," which indicates the birth year of each individual.

Create the "Spent" feature, which represents the total amount spent by the customer in various categories over a two-year span.

Utilize the "Marital_Status" to generate the "Living_With" feature, extracting information about the living situation of couples.

Create the "Children" feature to indicate the total number of children in a household, including kids and teenagers.

Introduce the "Family_Size" feature to gain a clearer understanding of household composition.

Create the "Is_Parent" feature to indicate parenthood status.

Finally, I will simplify the value counts of the "Education" feature into three categories and eliminate redundant features.

In [9]:
```
#Feature Engineering
#Age of customer today
data["Age"] = 2021-data["Year_Birth"]

#Total spendings on various items
data["Spent"] = data["MntWines"]+ data["MntFruits"]+ data["MntMeatProducts"]+ data["MntFishProducts"]+ data["MntSweetPr

#Deriving living situation by marital status"Alone"
data["Living_With"]=data["Marital_Status"].replace({"Married":"Partner", "Together":"Partner", "Absurd":"Alone", "Widow

#Feature indicating total children living in the household
data["Children"]=data["Kidhome"]+data["Teenhome"]

#Feature for total members in the householde
data["Family_Size"] = data["Living_With"].replace({"Alone": 1, "Partner":2})+ data["Children"]

#Feature pertaining parenthood
data["Is_Parent"] = np.where(data.Children> 0, 1, 0)

#Segmenting education levels in three groups
data["Education"]=data["Education"].replace({"Basic":"Undergraduate","2n Cycle":"Undergraduate", "Graduation":"Graduate

#For clarity
data=data.rename(columns={"MntWines": "Wines","MntFruits":"Fruits","MntMeatProducts":"Meat","MntFishProducts":"Fish","M

#Dropping some of the redundant features
to_drop = ["Marital_Status", "Dt_Customer", "Z_CostContact", "Z_Revenue", "Year_Birth", "ID"]
data = data.drop(to_drop, axis=1)
```

Now that new features have been added, let's examine the statistical summary of the data.

In [10]:
```
data.describe()
```

Out[10]:

| | Income | Kidhome | Teenhome | Recency | Wines | Fruits | Meat | Fish | Sweets | Gold |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 2216.000000 | 2216.000000 | 2216.000000 | 2216.000000 | 2216.000000 | 2216.000000 | 2216.000000 | 2216.000000 | 2216.000000 | 2216.000000 |
| **mean** | 52247.251354 | 0.441787 | 0.505415 | 49.012635 | 305.091606 | 26.356047 | 166.995939 | 37.637635 | 27.028881 | 43.965253 |
| **std** | 25173.076661 | 0.536896 | 0.544181 | 28.948352 | 337.327920 | 39.793917 | 224.283273 | 54.752082 | 41.072046 | 51.815414 |
| **min** | 1730.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 35303.000000 | 0.000000 | 0.000000 | 24.000000 | 24.000000 | 2.000000 | 16.000000 | 3.000000 | 1.000000 | 9.000000 |
| **50%** | 51381.500000 | 0.000000 | 0.000000 | 49.000000 | 174.500000 | 8.000000 | 68.000000 | 12.000000 | 8.000000 | 24.500000 |
| **75%** | 68522.000000 | 1.000000 | 1.000000 | 74.000000 | 505.000000 | 33.000000 | 232.250000 | 50.000000 | 33.000000 | 56.000000 |
| **max** | 666666.000000 | 2.000000 | 2.000000 | 99.000000 | 1493.000000 | 199.000000 | 1725.000000 | 259.000000 | 262.000000 | 321.000000 |

8 rows × 28 columns

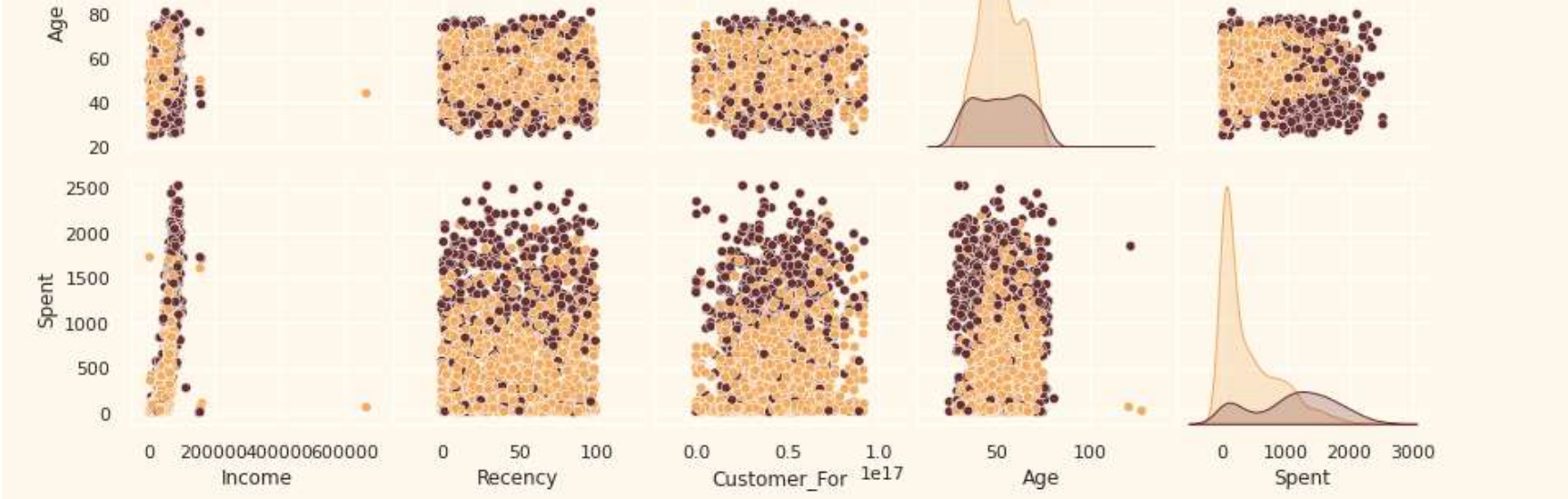
The statistics presented above reveal discrepancies in both mean income and age, as well as in the maximum values of income and age.

It's important to note that the maximum recorded age is 128 years, which suggests that I calculated the ages based on the assumption that the data is from the year 2021, and the data may be outdated.

To gain a broader perspective of the data, I will proceed to plot some selected features.

In [11]:
```
#To plot some selected features
#Setting up colors prefrences
sns.set(rc={"axes.facecolor":"#FFF9ED","figure.facecolor":"#FFF9ED"})
pallet = ["#682F2F", "#9E726F", "#D6B2B1", "#B9C0C9", "#9F8A78", "#F3AB60"]
cmap = colors.ListedColormap(["#682F2F", "#9E726F", "#D6B2B1", "#B9C0C9", "#9F8A78", "#F3AB60"])
#Plotting following features
To_Plot = [ "Income", "Recency", "Customer_For", "Age", "Spent", "Is_Parent"]
print("Reletive Plot Of Some Selected Features: A Data Subset")
plt.figure()
sns.pairplot(data[To_Plot], hue= "Is_Parent",palette= (["#682F2F","#F3AB60"]))
#Taking hue
plt.show()
```

```
Reletive Plot Of Some Selected Features: A Data Subset
<Figure size 576x396 with 0 Axes>
```




It's evident that there are outliers in both the Income and Age features. As a solution, I will proceed to remove these outliers from the dataset.

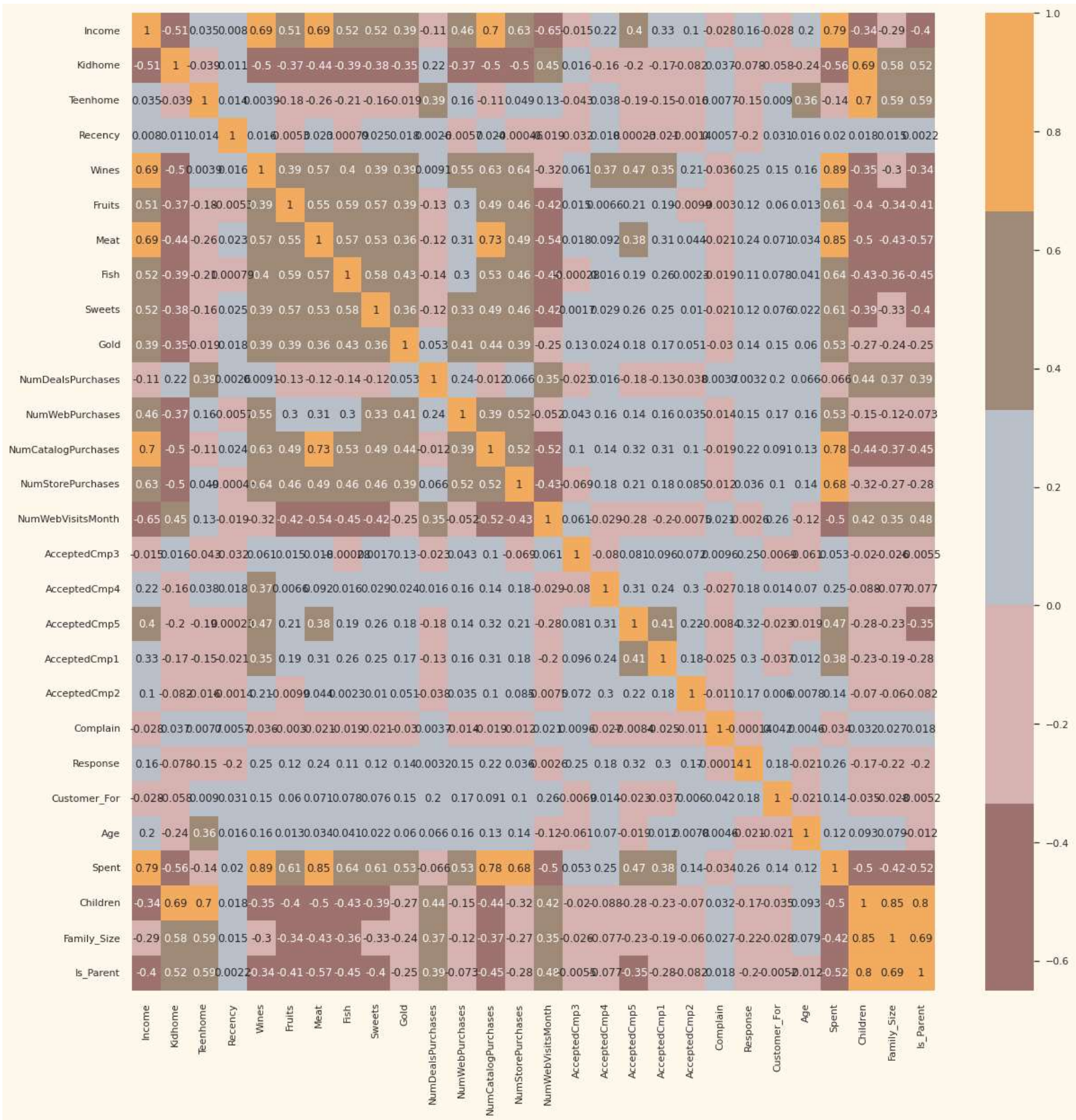
In [12]:
```
#Dropping the outliers by setting a cap on Age and income.
data = data[(data["Age"]<90)]
data = data[(data["Income"]<600000)]
print("The total number of data-points after removing the outliers are:", len(data))
```

```
The total number of data-points after removing the outliers are: 2212
```

Next, we will examine the correlation among the features, excluding the categorical attributes for now.

In [13]:
```
#correlation matrix
corrmat= data.corr()
plt.figure(figsize=(20,20))
sns.heatmap(corrmat,annot=True, cmap=cmap, center=0)
```

Out[13]:
```
<AxesSubplot:>
```




The data has been cleaned, and the new features have been incorporated. I will now move on to the next step, which involves preprocessing the data.

## Data Preprocessing

In this section, we will preprocess the data in preparation for clustering operations.

The following steps will be applied to preprocess the data:

Label encoding the categorical features. Scaling the features using the standard scaler. Creating a subset dataframe for dimensionality reduction.

In [14]:
```
#Get list of categorical variables
s = (data.dtypes == 'object')
object_cols = list(s[s].index)

print("Categorical variables in the dataset:", object_cols)
```

```
Categorical variables in the dataset: ['Education', 'Living_With']
```

In [15]:
```
#Label Encoding the object dtypes.
LE=LabelEncoder()
for i in object_cols:
    data[i]=data[[i]].apply(LE.fit_transform)

print("All features are now numerical")
```

```
All features are now numerical
```

In [16]:
```
#Creating a copy of data
ds = data.copy()
# creating a subset of dataframe by dropping the features on deals accepted and promotions
cols_del = ['AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1','AcceptedCmp2', 'Complain', 'Response']
ds = ds.drop(cols_del, axis=1)
#Scaling
scaler = StandardScaler()
scaler.fit(ds)
```

```
scaled_ds = pd.DataFrame(scaler.transform(ds),columns= ds.columns )
print("All features are now scaled")
```

```
All features are now scaled
```

In [17]:
```
#Scaled data to be used for reducing the dimensionality
print("Dataframe to be used for further modelling:")
scaled_ds.head()
```

```
Dataframe to be used for further modelling:
```

Out[17]:

| | Education | Income | Kidhome | Teenhome | Recency | Wines | Fruits | Meat | Fish | Sweets | ... | NumCatalogPurchases | Nu |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | -0.893586 | 0.287105 | -0.822754 | -0.929699 | 0.310353 | 0.977660 | 1.552041 | 1.690293 | 2.453472 | 1.483713 | ... | 2.503607 | |
| **1** | -0.893586 | -0.260882 | 1.040021 | 0.908097 | -0.380813 | -0.872618 | -0.637461 | -0.718230 | -0.651004 | -0.634019 | ... | -0.571340 | |
| **2** | -0.893586 | 0.913196 | -0.822754 | -0.929699 | -0.795514 | 0.357935 | 0.570540 | -0.178542 | 1.339513 | -0.147184 | ... | -0.229679 | |
| **3** | -0.893586 | -1.176114 | 1.040021 | -0.929699 | -0.795514 | -0.872618 | -0.561961 | -0.655787 | -0.504911 | -0.585335 | ... | -0.913000 | |
| **4** | 0.571657 | 0.294307 | 1.040021 | -0.929699 | 1.554453 | -0.392257 | 0.419540 | -0.218684 | 0.152508 | -0.001133 | ... | 0.111982 | |

5 rows × 23 columns

In [ ]:
```

```

## Dimensionality Reduction

In this problem, the final classification is based on numerous factors, which are essentially attributes or features. The more features there are, the more challenging it becomes to work with the data. Many of these features are correlated and, therefore, redundant. To address this, I will perform dimensionality reduction on the selected features before applying a classifier.

Dimensionality reduction is the process of reducing the number of random variables under consideration by obtaining a set of principal variables.

Principal Component Analysis (PCA) is a technique used to reduce the dimensionality of such datasets, making them more interpretable while minimizing information loss.

Steps in this section:

Dimensionality reduction with PCA. Plotting the reduced dataframe.

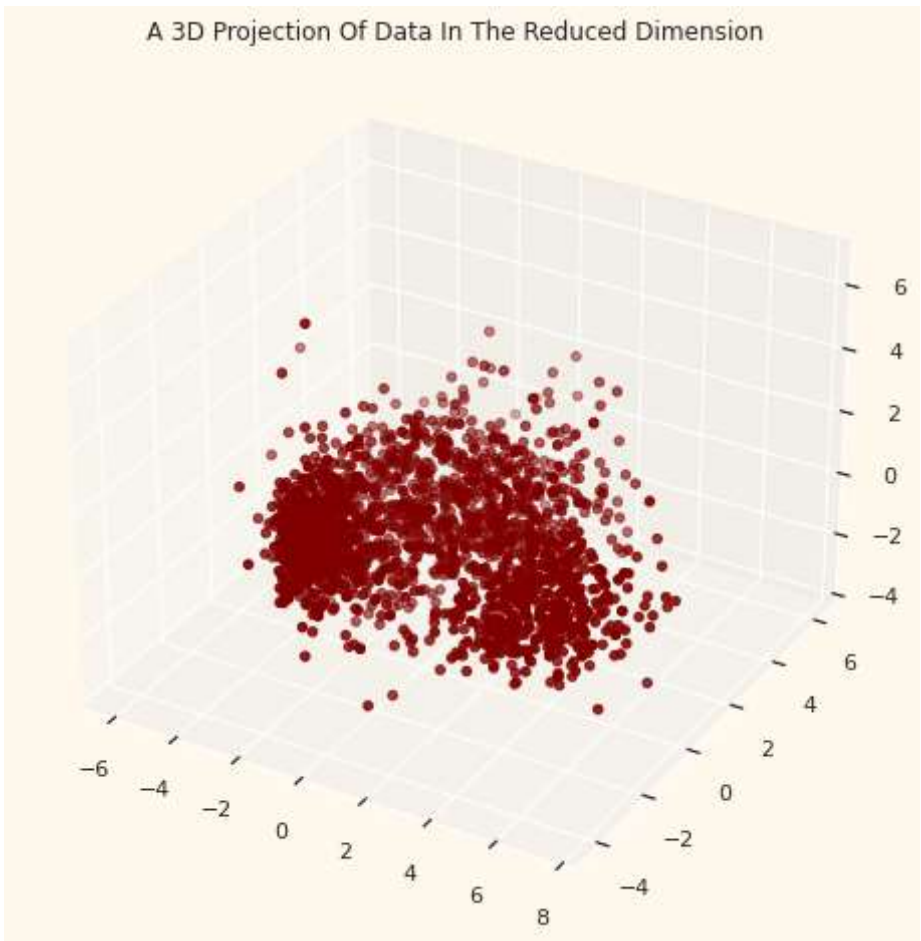
For this project, I will reduce the dimensions to 3.

In [18]:
```
#Initiating PCA to reduce dimentions aka features to 3
pca = PCA(n_components=3)
pca.fit(scaled_ds)
PCA_ds = pd.DataFrame(pca.transform(scaled_ds), columns=(["col1","col2", "col3"]))
PCA_ds.describe().T
```

Out[18]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **col1** | 2212.0 | -1.116246e-16 | 2.878377 | -5.969394 | -2.538494 | -0.780421 | 2.383290 | 7.444305 |
| **col2** | 2212.0 | 1.105204e-16 | 1.706839 | -4.312196 | -1.328316 | -0.158123 | 1.242289 | 6.142721 |
| **col3** | 2212.0 | 3.049098e-17 | 1.221956 | -3.530416 | -0.829067 | -0.022692 | 0.799895 | 6.611222 |

In [19]:
```
#A 3D Projection Of Data In The Reduced Dimension
x =PCA_ds["col1"]
y =PCA_ds["col2"]
z =PCA_ds["col3"]
#To plot
fig = plt.figure(figsize=(10,8))
ax = fig.add_subplot(111, projection="3d")
ax.scatter(x,y,z, c="maroon", marker="o" )
ax.set_title("A 3D Projection Of Data In The Reduced Dimension")
plt.show()
```




## Clustering

Now that I have reduced the attributes to three dimensions, I will proceed with clustering using the Agglomerative method. Agglomerative clustering is a hierarchical approach that involves merging examples until the desired number of clusters is achieved.

Steps Involved in Clustering:

Employing the Elbow Method to determine the number of clusters to be formed.

Implementing Clustering via Agglomerative Clustering.

Visualizing the formed clusters through a scatter plot examination.

In [20]:
```
# Quick examination of elbow method to find numbers of clusters to make.
print('Elbow Method to determine the number of clusters to be formed:')
Elbow_M = KElbowVisualizer(KMeans(), k=10)
Elbow_M.fit(PCA_ds)
Elbow_M.show()
```

```
Elbow Method to determine the number of clusters to be formed:
```


Out[20]:
```
<AxesSubplot:title={'center':'Distortion Score Elbow for KMeans Clustering'}, xlabel='k', ylabel='distortion score'>
```

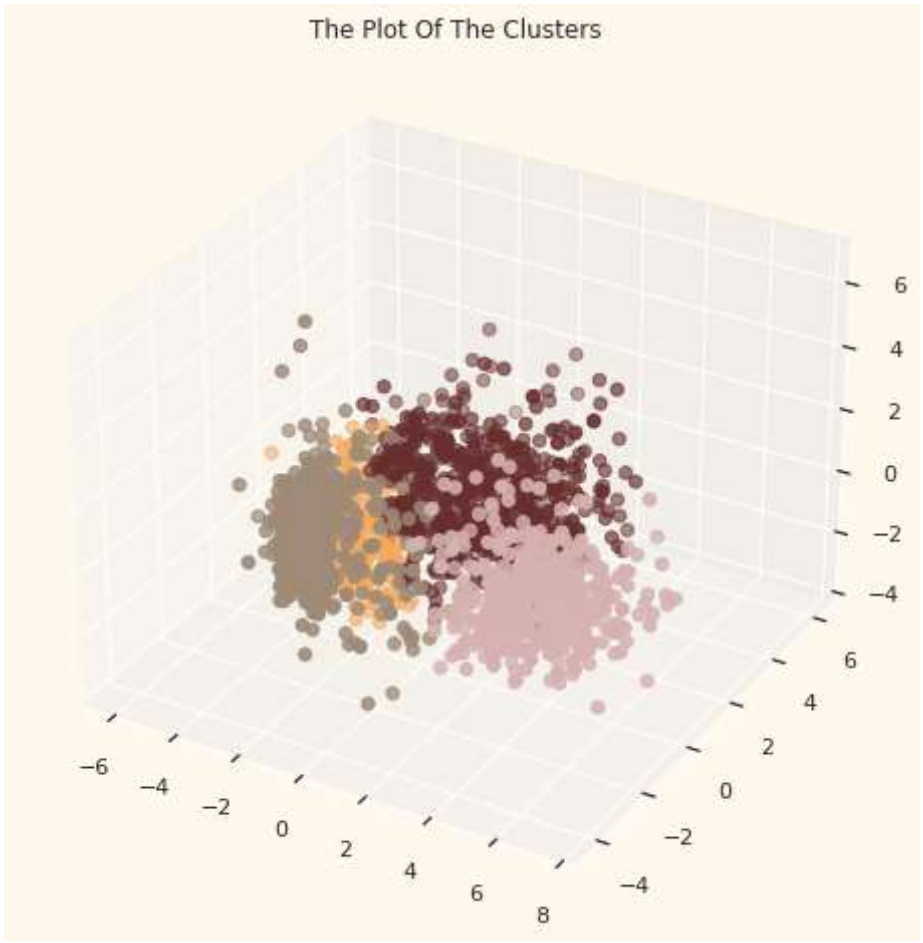
The previous analysis suggests that four clusters will be the optimal choice for this dataset. Next, we will fit the Agglomerative Clustering Model to obtain the final clusters.

In [21]:
```
#Initiating the Agglomerative Clustering model
AC = AgglomerativeClustering(n_clusters=4)
# fit model and predict clusters
yhat_AC = AC.fit_predict(PCA_ds)
PCA_ds["Clusters"] = yhat_AC
#Adding the Clusters feature to the orignal dataframe.
data["Clusters"]= yhat_AC
```

To examine the clusters that have been formed, let's visualize the 3-D distribution of these clusters.

In [22]:
```
#Plotting the clusters
fig = plt.figure(figsize=(10,8))
ax = plt.subplot(111, projection='3d', label="bla")
```

```
ax.scatter(x, y, z, s=40, c=PCA_ds["Clusters"], marker='o', cmap = cmap )
ax.set_title("The Plot Of The Clusters")
plt.show()
```



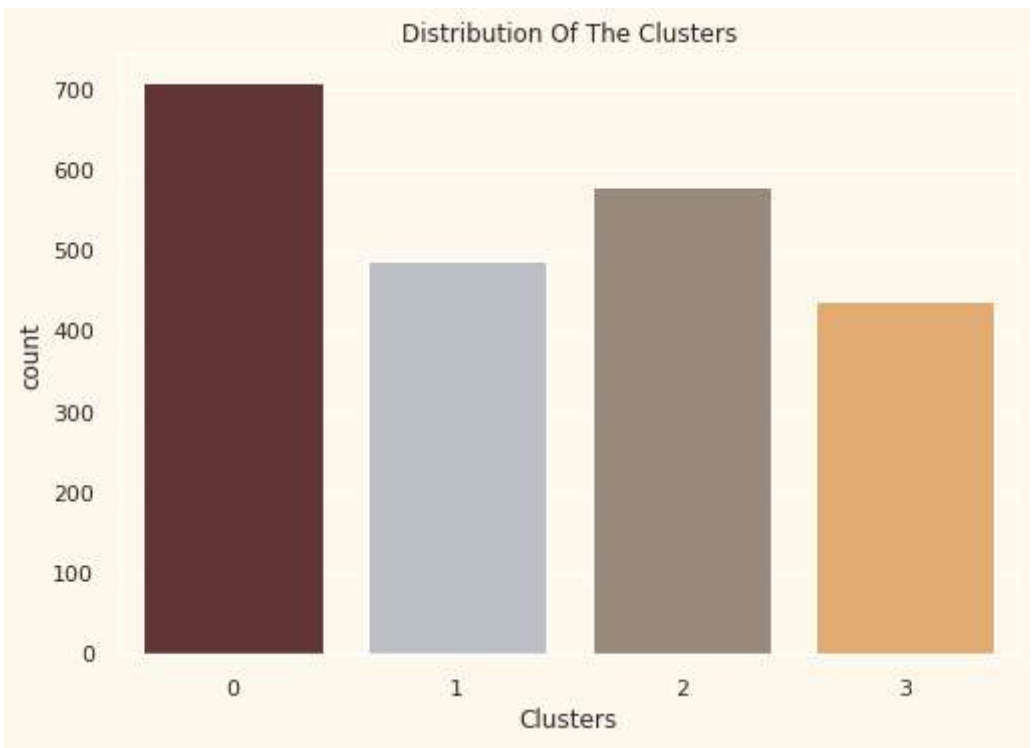

## Evaluating Models

As this is an unsupervised clustering task, we lack tagged features to evaluate or score our model. The primary aim of this section is to explore the patterns within the formed clusters and ascertain the nature of these patterns.

To accomplish this, we will inspect the data considering the clusters through exploratory data analysis, drawing conclusions based on our observations.

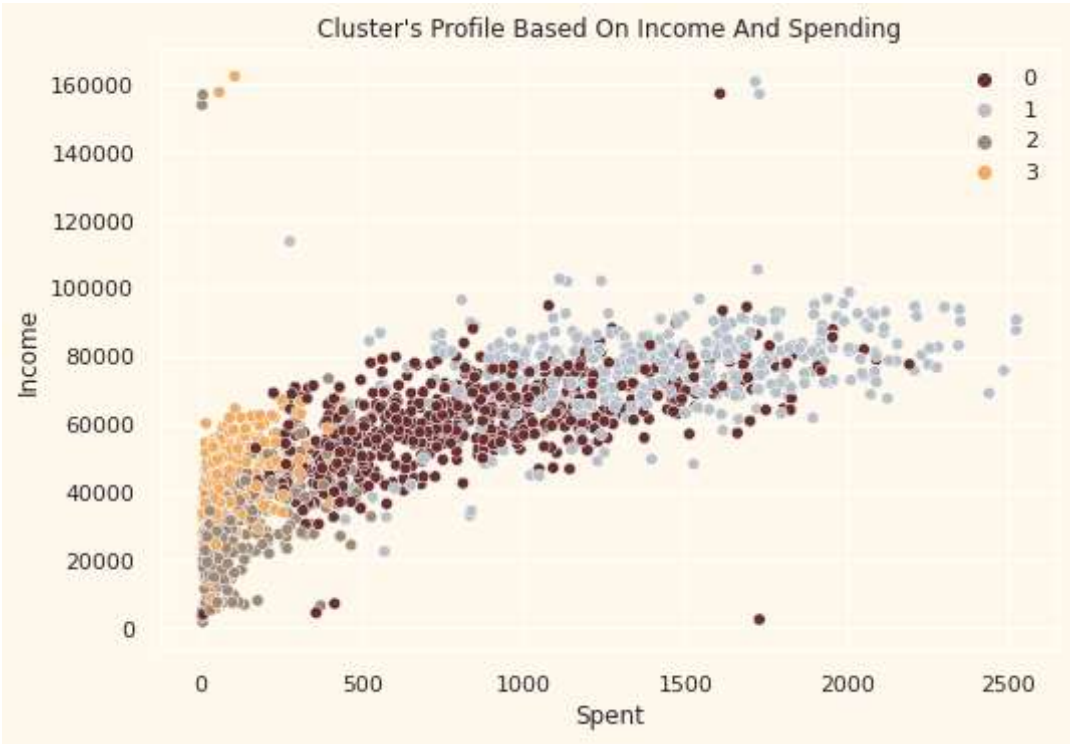
Firstly, let's examine the distribution within the clusters.

In [23]:
```
#Plotting countplot of clusters
pal = ["#682F2F","#B9C0C9", "#9F8A78","#F3AB60"]
pl = sns.countplot(x=data["Clusters"], palette= pal)
pl.set_title("Distribution Of The Clusters")
plt.show()
```




The clusters seem to be fairly distributed.

In [24]:
```
pl = sns.scatterplot(data = data,x=data["Spent"], y=data["Income"],hue=data["Clusters"], palette= pal)
pl.set_title("Cluster's Profile Based On Income And Spending")
plt.legend()
plt.show()
```




**The plot of Income versus Spending exhibits distinct patterns for each cluster:**

Group 0: High spending with average income

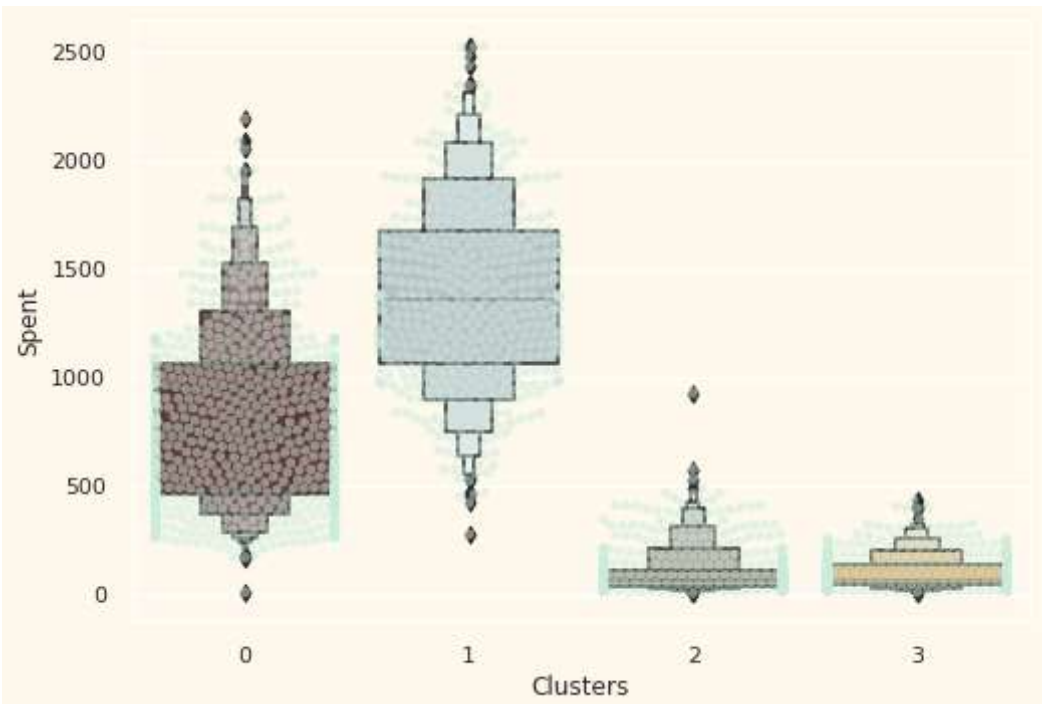
Group 1: High spending with high income

Group 2: Low spending and low income

Group 3: High spending but low income

Next, I will explore the detailed distribution of clusters concerning various products in the dataset, namely: Wines, Fruits, Meat, Fish, Sweets, and Gold.

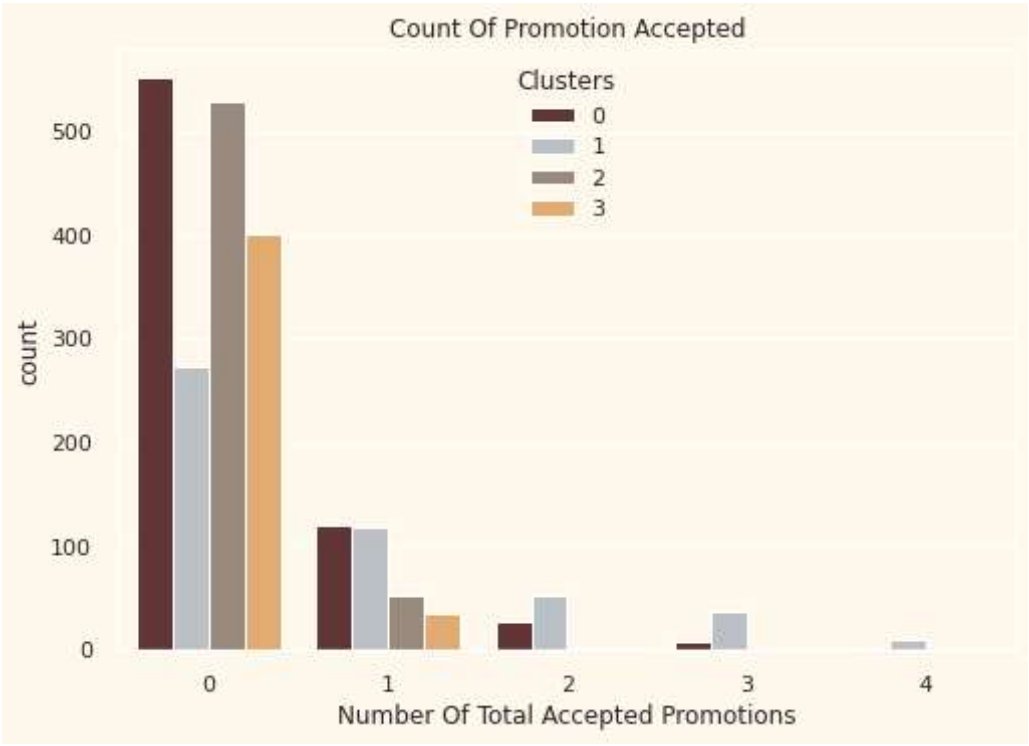
In [25]:
```
plt.figure()
pl=sns.swarmplot(x=data["Clusters"], y=data["Spent"], color= "#CBEDDD", alpha=0.5 )
pl=sns.boxenplot(x=data["Clusters"], y=data["Spent"], palette=pal)
plt.show()
```




The above plot indicates that Cluster 1 comprises the largest set of customers, closely followed by Cluster 0. This observation prompts us to delve into the specific spending behaviors within each cluster, aiding in the formulation of targeted marketing strategies.

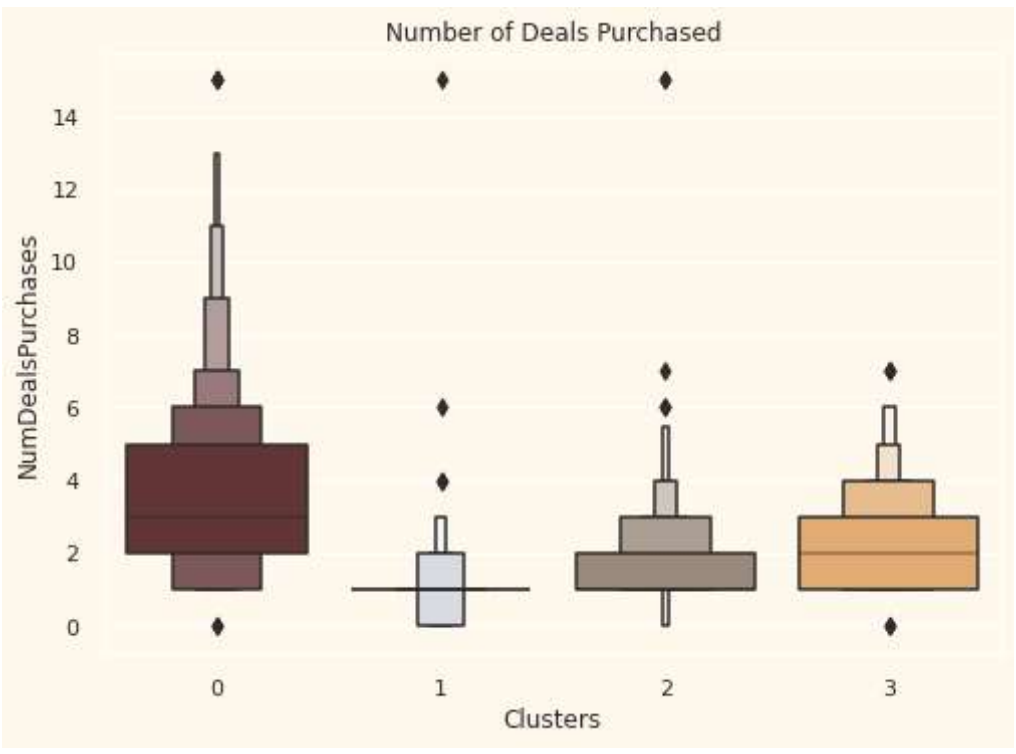
Next, let's examine the performance of our past marketing campaigns.

In [26]:
```
#Creating a feature to get a sum of accepted promotions
data["Total_Promos"] = data["AcceptedCmp1"]+ data["AcceptedCmp2"]+ data["AcceptedCmp3"]+ data["AcceptedCmp4"]+ data["Ac
#Plotting count of total campaign accepted.
plt.figure()
pl = sns.countplot(x=data["Total_Promos"],hue=data["Clusters"], palette= pal)
pl.set_title("Count Of Promotion Accepted")
pl.set_xlabel("Number Of Total Accepted Promotions")
plt.show()
```




The response to the campaigns thus far has been underwhelming, with very few participants overall. Additionally, no one has participated in all five campaigns. This suggests the need for more targeted and well-planned campaigns to enhance sales.

In [27]:
```
#Plotting the number of deals purchased
plt.figure()
pl=sns.boxenplot(y=data["NumDealsPurchases"],x=data["Clusters"], palette= pal)
pl.set_title("Number of Deals Purchased")
plt.show()
```

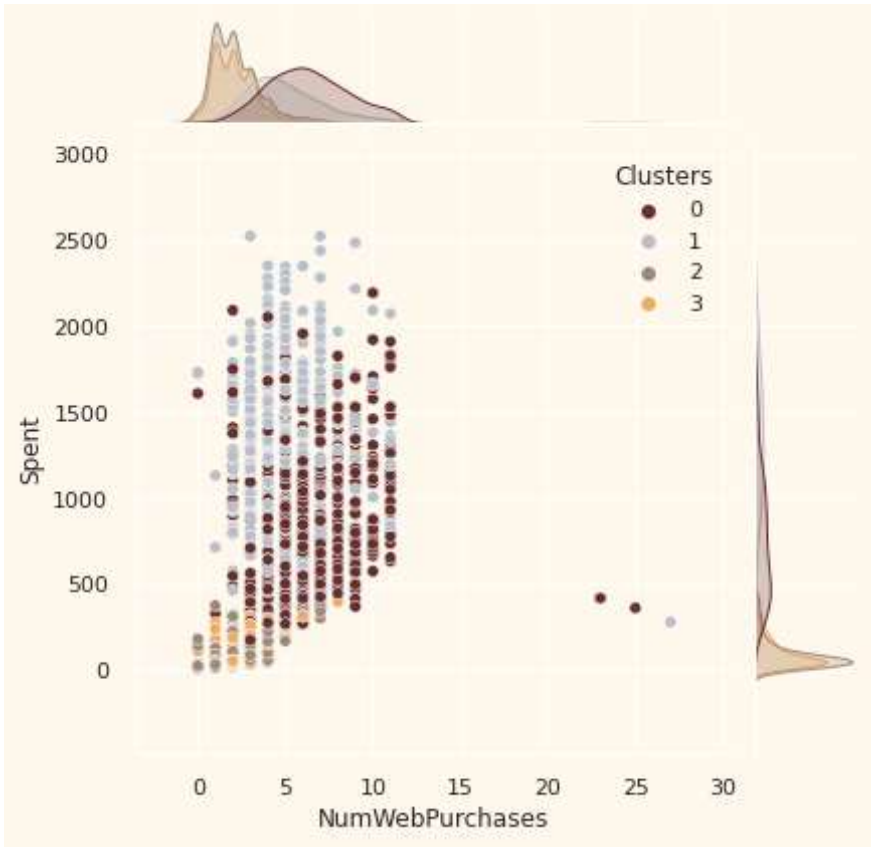



Contrary to the campaigns, the offered deals performed well, particularly with Cluster 0 and Cluster 3. Interestingly, our prominent customers in Cluster 1 showed less interest in the deals. Unfortunately, there doesn't appear to be any significant attraction for Cluster 2.

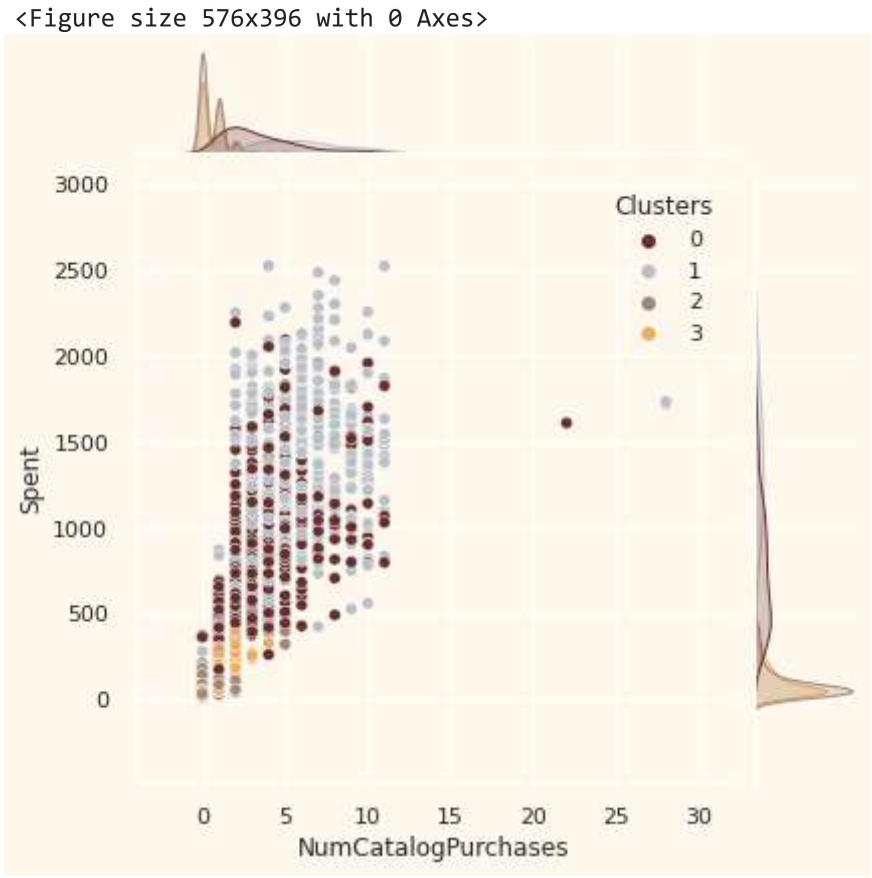
In [28]:
```
#for more details on the purchasing style
Places =["NumWebPurchases", "NumCatalogPurchases", "NumStorePurchases",  "NumWebVisitsMonth"]


for i in Places:
    plt.figure()
    sns.jointplot(x=data[i],y = data["Spent"],hue=data["Clusters"], palette= pal)
    plt.show()
```

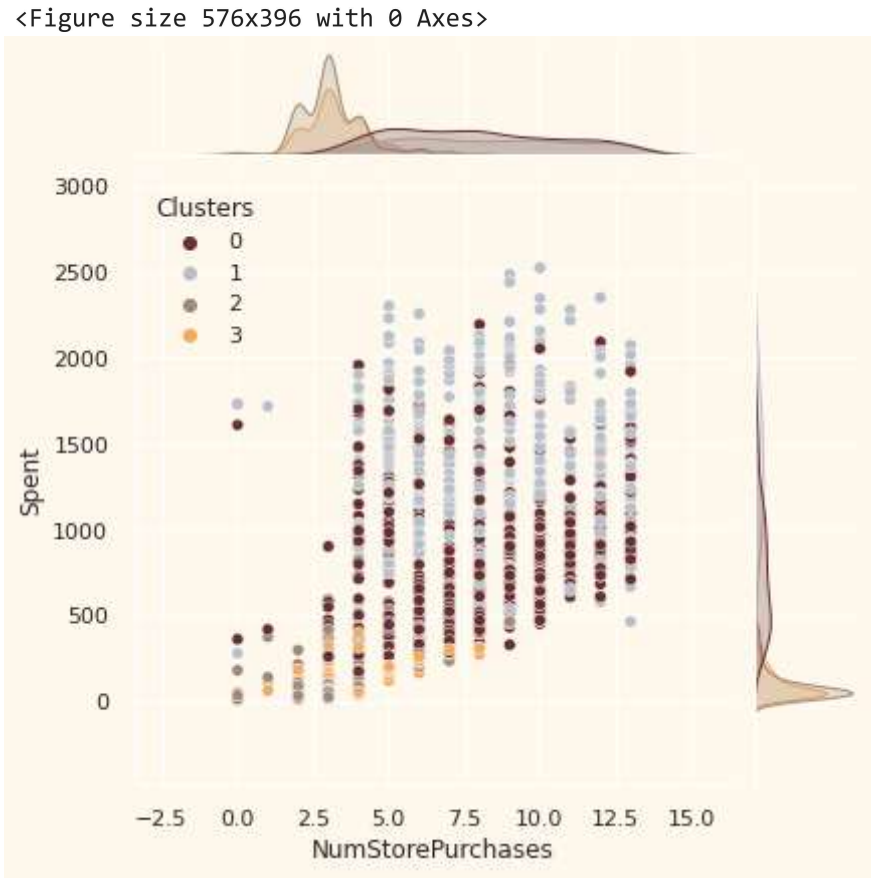
```
<Figure size 576x396 with 0 Axes>
```



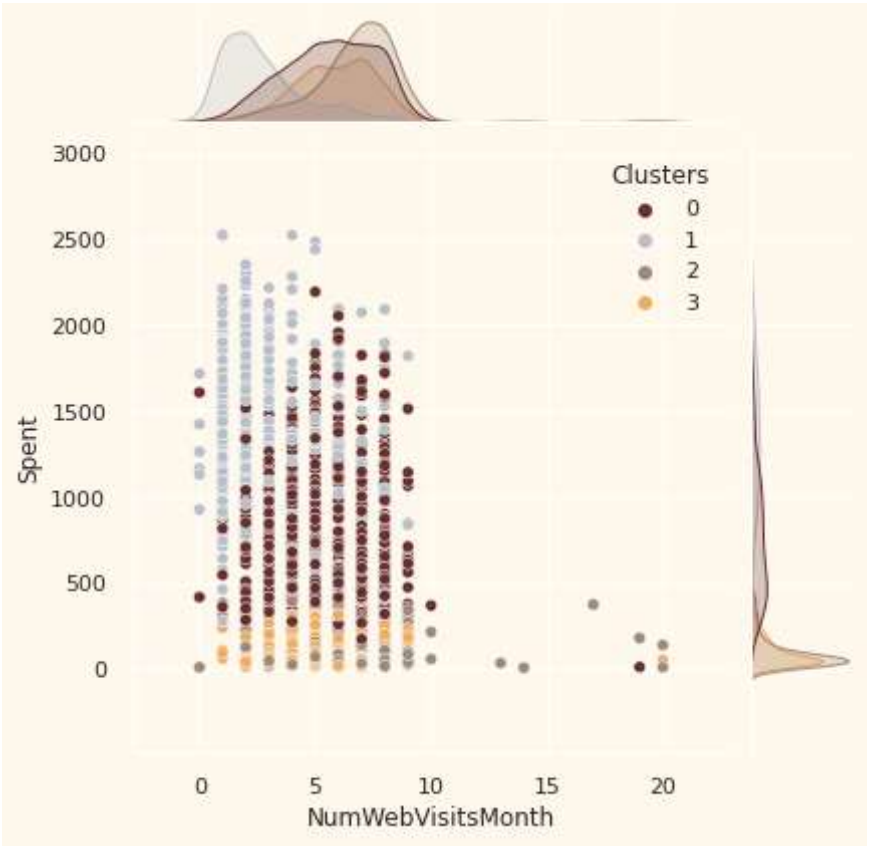

<Figure size 576x396 with 0 Axes>




<Figure size 576x396 with 0 Axes>




<Figure size 576x396 with 0 Axes>




## Profiling

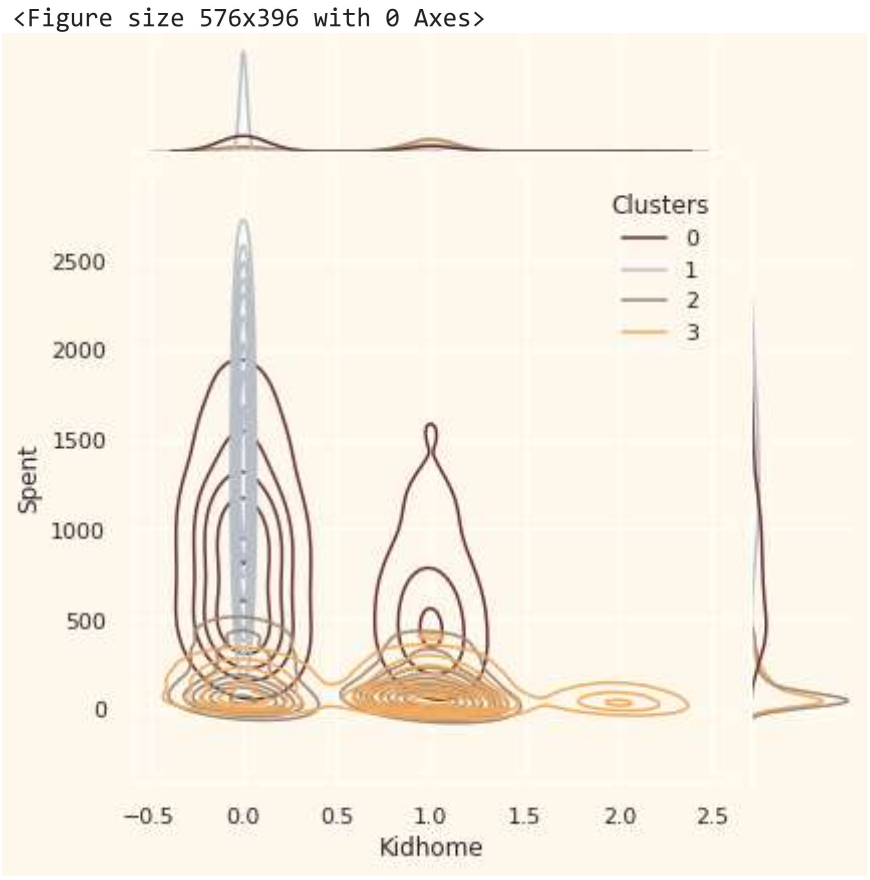
Now that the clusters have been formed and their purchasing habits examined, we aim to profile these clusters to determine our star customers and those requiring more attention from the retail store's marketing team.

To achieve this, I will plot features indicative of the customers' personal traits concerning the cluster they belong to. Based on the outcomes observed, I will draw conclusions to identify star customers and areas requiring more focus.

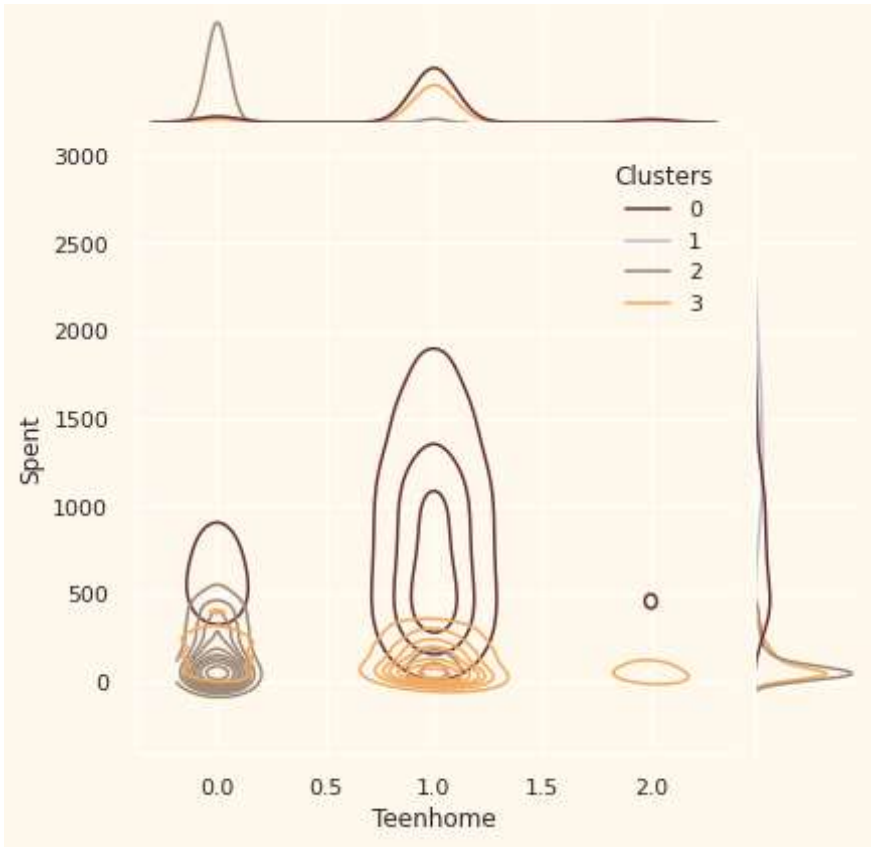
In [29]:
```
Personal = [ "Kidhome","Teenhome","Customer_For", "Age", "Children", "Family_Size", "Is_Parent", "Education","Living_Wi

for i in Personal:
    plt.figure()
    sns.jointplot(x=data[i], y=data["Spent"], hue =data["Clusters"], kind="kde", palette=pal)
    plt.show()
```

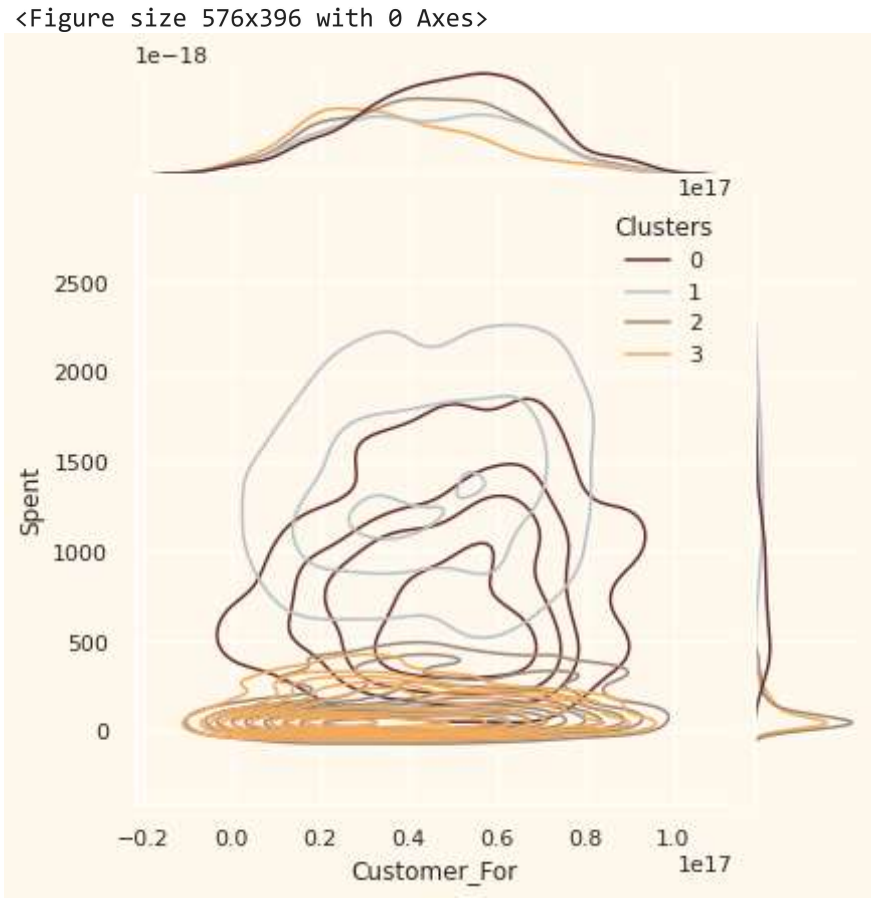
```
<Figure size 576x396 with 0 Axes>
```



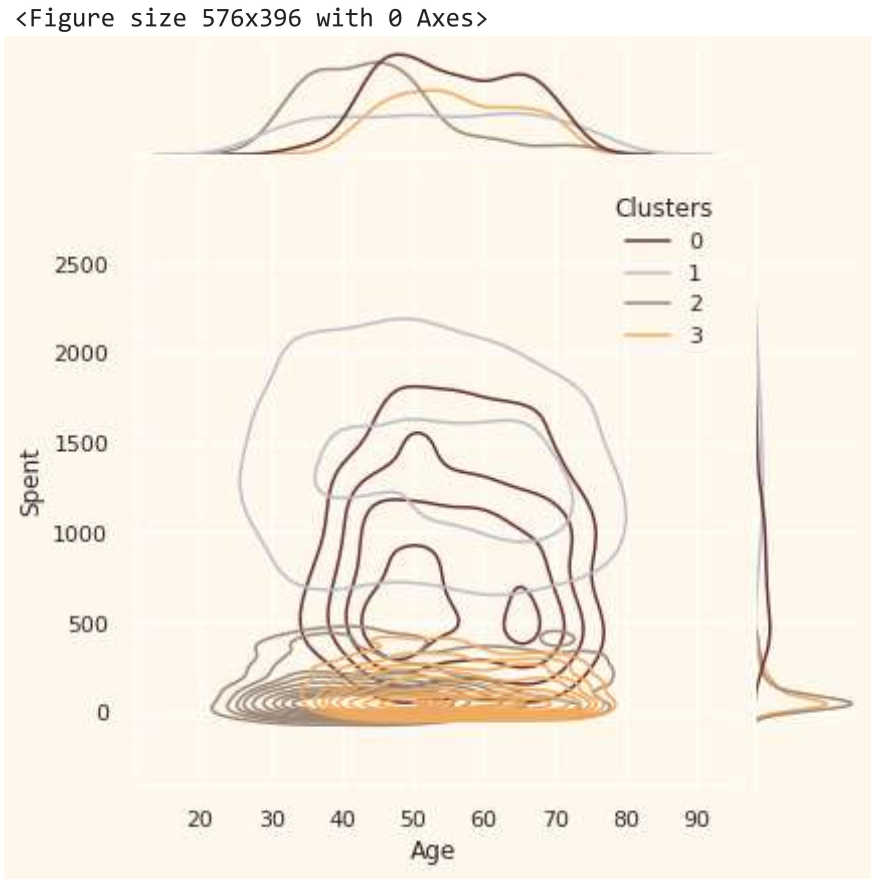

```
<Figure size 576x396 with 0 Axes>
```



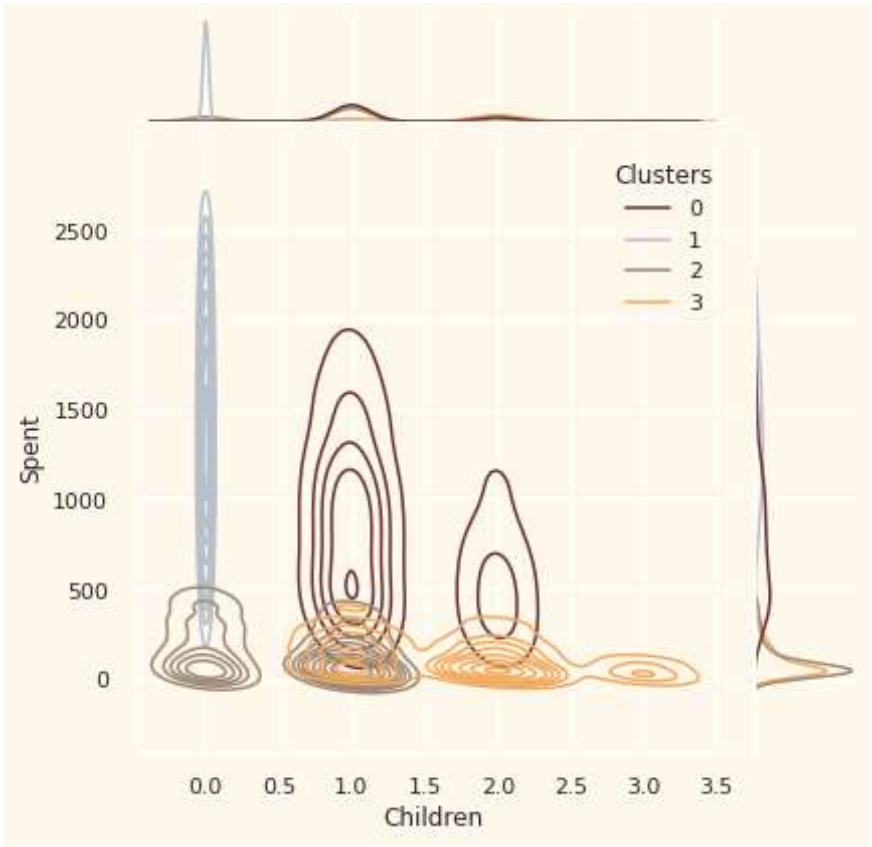

<Figure size 576x396 with 0 Axes>



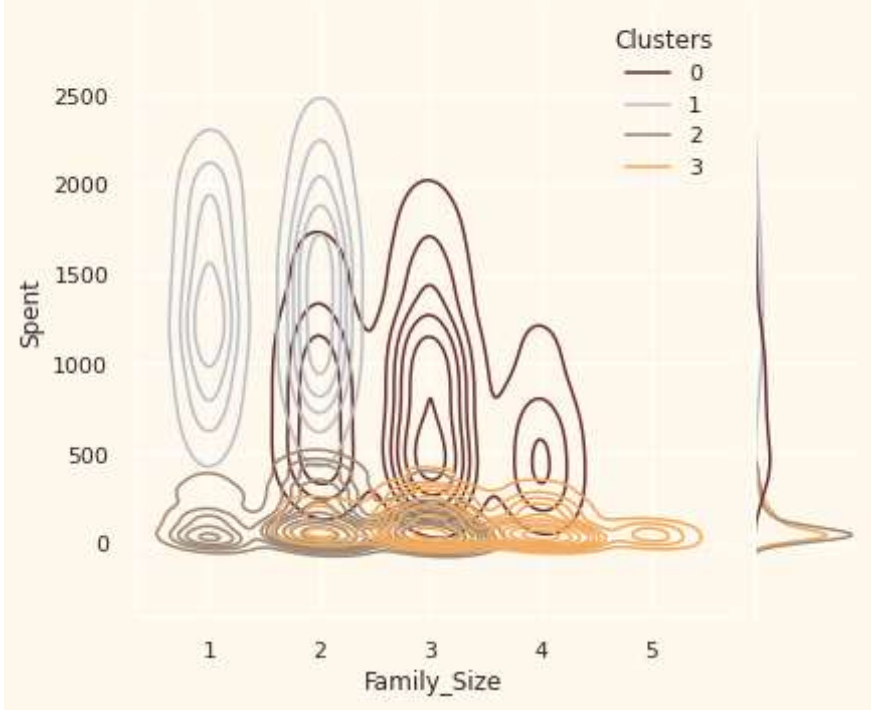

<Figure size 576x396 with 0 Axes>



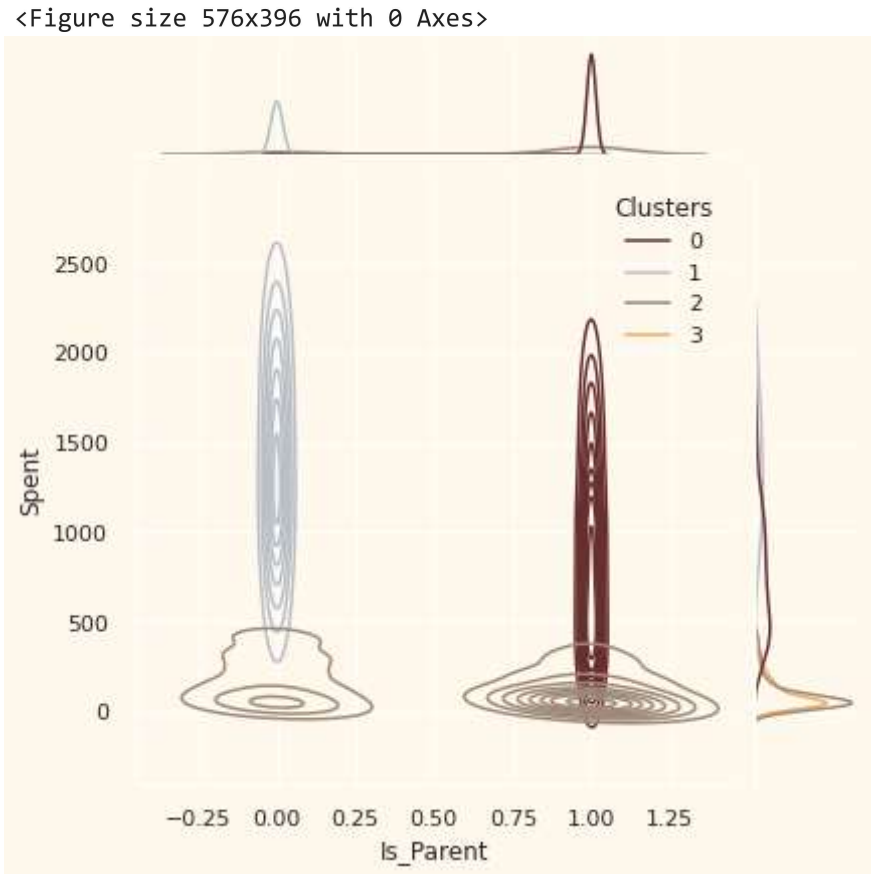

<Figure size 576x396 with 0 Axes>



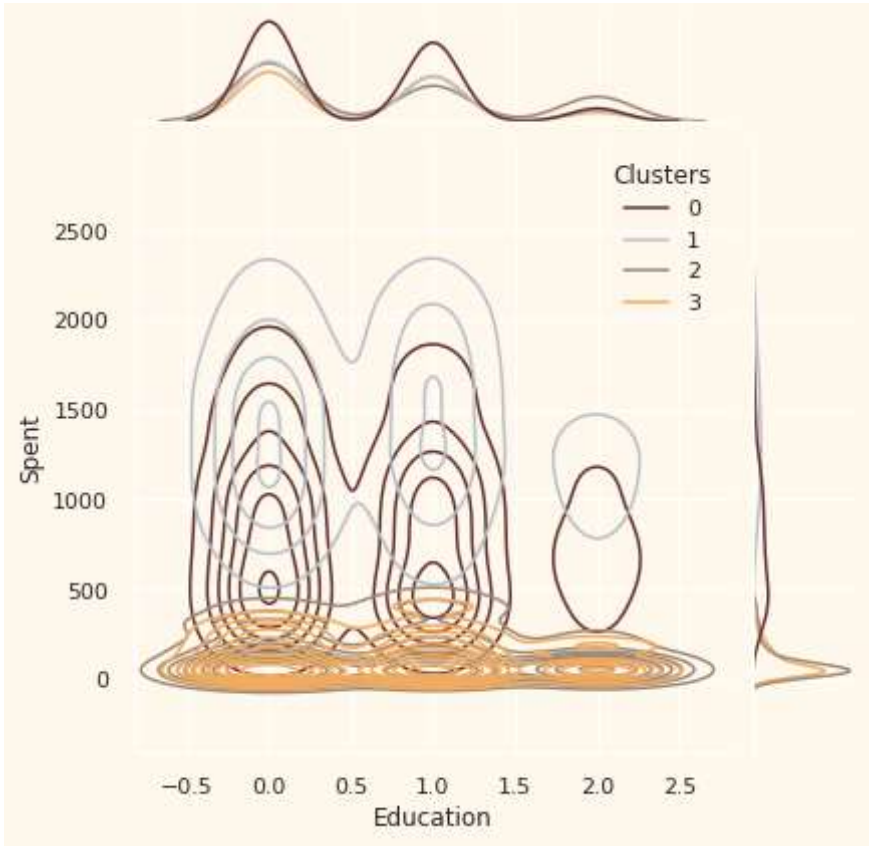

<Figure size 576x396 with 0 Axes>



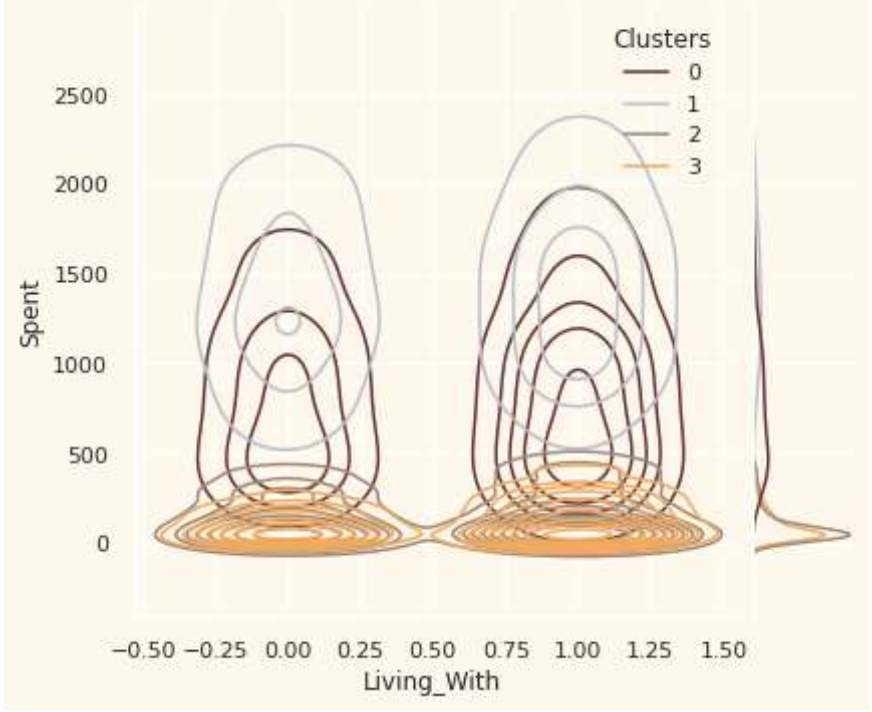

<Figure size 576x396 with 0 Axes>




<Figure size 576x396 with 0 Axes>




<Figure size 576x396 with 0 Axes>




**Profiling the Clusters**

The following information can be deduced about the customers in different clusters:

About Cluster Number 0:

They are definitely parents, with a maximum of 4 family members and a minimum of 2. Single parents are a subset of this group. Most have a teenager at home, and they are relatively older.

About Cluster Number 1:

They are definitely not parents. The maximum family size is only 2 members. There is a slight majority of couples over single individuals. This group spans all ages and represents a high-income segment.

About Cluster Number 2:

The majority of these people are parents, with a maximum of 3 family members. They mainly have one child (usually not teenagers) and are relatively younger.

About Cluster Number 3:

They are definitely parents, with a maximum of 5 family members and a minimum of 2. The majority have a teenager at home, and they are relatively older. This group represents a lower-income segment.

## Conclusion

In this project, I conducted unsupervised clustering using dimensionality reduction followed by agglomerative clustering, resulting in 4 distinct clusters. These clusters were then utilized to profile customers based on their family structures, income, and spending patterns. The insights gained can be valuable for devising improved marketing strategies.