Customer Analysis

Problem Statement: A well known company with numerous products needs to analyze their customer behaviour and classify them to know whether they will accept the campaigns held by the company.

In this project I will be doing an unsupervised clustering of data on customer records from the company's database. Customer clustering/segmentation is the practice of separating customers into groups that reflect similarities among customers in each cluster. It helps to modify products according to distinct needs and behaviours of the customers.

Importing libraries

In [1]:
```
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
```

In [2]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
from matplotlib.colors import ListedColormap
from sklearn import metrics
```

Loading data

In [3]:
```
data = pd.read_csv("market_train.csv")
data.head()
```

Out[3]:

	Unnamed: 0.1	Unnamed: 0	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt
0	0	0	5524	1957	S1	Lajang	58138000.0	0	0	
1	1	1	2174	1954	S1	Lajang	46344000.0	1	1	
2	2	2	4141	1965	S1	Bertunangan	71613000.0	0	0	
3	3	3	6182	1984	S1	Bertunangan	26646000.0	1	0	
4	4	4	5324	1981	S3	Menikah	58293000.0	1	0	

5 rows × 31 columns

In [4]:
```
test = pd.read_csv("market_test.csv")
test.shape
```

Out[4]: (559, 31)

In [5]:
```
data.shape
```
Out[5]:
```
(1680, 31)
```

Data Cleaning

In [6]:
```
# information about the data and features

data.info()
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1680 entries, 0 to 1679
Data columns (total 31 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Unnamed: 0.1         1680 non-null   int64  
 1   Unnamed: 0           1680 non-null   int64  
 2   ID                   1680 non-null   int64  
 3   Year_Birth           1680 non-null   int64  
 4   Education            1680 non-null   object 
 5   Marital_Status       1680 non-null   object 
 6   Income               1663 non-null   float64
 7   Kidhome              1680 non-null   int64  
 8   Teenhome             1680 non-null   int64  
 9   Dt_Customer          1680 non-null   object 
 10  Recency              1680 non-null   int64  
 11  MntCoke              1680 non-null   int64  
 12  MntFruits            1680 non-null   int64  
 13  MntMeatProducts      1680 non-null   int64  
 14  MntFishProducts      1680 non-null   int64  
 15  MntSweetProducts     1680 non-null   int64  
 16  MntGoldProds         1680 non-null   int64  
 17  NumDealsPurchases    1680 non-null   int64  
 18  NumWebPurchases      1680 non-null   int64  
 19  NumCatalogPurchases  1680 non-null   int64  
 20  NumStorePurchases    1680 non-null   int64  
 21  NumWebVisitsMonth    1680 non-null   int64  
 22  AcceptedCmp3         1680 non-null   int64  
 23  AcceptedCmp4         1680 non-null   int64  
 24  AcceptedCmp5         1680 non-null   int64  
 25  AcceptedCmp1         1680 non-null   int64  
 26  AcceptedCmp2         1680 non-null   int64  
 27  Complain             1680 non-null   int64  
 28  Z_CostContact        1680 non-null   int64  
 29  Z_Revenue            1680 non-null   int64  
 30  Response             1680 non-null   int64  
dtypes: float64(1), int64(27), object(3)
memory usage: 407.0+ KB
```

In [7]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 559 entries, 0 to 558
Data columns (total 31 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Unnamed: 0.1         559 non-null    int64  
 1   Unnamed: 0           559 non-null    int64  
 2   ID                   559 non-null    int64  
 3   Year_Birth           559 non-null    int64  
 4   Education            559 non-null    object 
 5   Marital_Status       559 non-null    object 
 6   Income               552 non-null    float64
 7   Kidhome              559 non-null    int64  
 8   Teenhome             559 non-null    int64  
 9   Dt_Customer          559 non-null    object 
 10  Recency              559 non-null    int64  
 11  MntCoke              559 non-null    int64  
 12  MntFruits            559 non-null    int64  
 13  MntMeatProducts      559 non-null    int64  
 14  MntFishProducts      559 non-null    int64  
 15  MntSweetProducts     559 non-null    int64  
 16  MntGoldProds         559 non-null    int64  
 17  NumDealsPurchases    559 non-null    int64  
 18  NumWebPurchases      559 non-null    int64  
 19  NumCatalogPurchases  559 non-null    int64  
 20  NumStorePurchases    559 non-null    int64  
 21  NumWebVisitsMonth    559 non-null    int64  
 22  AcceptedCmp3         559 non-null    int64  
 23  AcceptedCmp4         559 non-null    int64  
 24  AcceptedCmp5         559 non-null    int64  
 25  AcceptedCmp1         559 non-null    int64  
 26  AcceptedCmp2         559 non-null    int64  
 27  Complain             559 non-null    int64  
 28  Z_CostContact        559 non-null    int64  
 29  Z_Revenue            559 non-null    int64  
 30  Response             559 non-null    int64  
dtypes: float64(1), int64(27), object(3)
memory usage: 135.5+ KB
```

There are unwanted coulmns, missing values, values that are objects. These need to be addressed

In [8]:
```
# dropping 2 unwanted columns
data = data.drop(["Unnamed: 0.1","Unnamed: 0"], axis='columns')
```

In [9]:
```
test = test.drop(["Unnamed: 0.1","Unnamed: 0"], axis='columns')
```

In [10]:
```
# replacing null values
income_mean = data['Income'].mean()
data['Income'].fillna(value=income_mean, inplace = True)
```

In [11]:
```
income_mean = test['Income'].mean()
test['Income'].fillna(value=income_mean, inplace = True)
```

In [12]:

```
# converting Dt_Customer to datetime
data["Dt_Customer"] = pd.to_datetime(data["Dt_Customer"])
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1680 entries, 0 to 1679
Data columns (total 29 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   1680 non-null   int64
 1   Year_Birth           1680 non-null   int64
 2   Education            1680 non-null   object
 3   Marital_Status       1680 non-null   object
 4   Income               1680 non-null   float64
 5   Kidhome              1680 non-null   int64
 6   Teenhome             1680 non-null   int64
 7   Dt_Customer          1680 non-null   datetime64[ns]
 8   Recency              1680 non-null   int64
 9   MntCoke              1680 non-null   int64
 10  MntFruits            1680 non-null   int64
 11  MntMeatProducts      1680 non-null   int64
 12  MntFishProducts      1680 non-null   int64
 13  MntSweetProducts     1680 non-null   int64
```

In [ ]:

```
test["Dt_Customer"] = pd.to_datetime(test["Dt_Customer"])
```

In [14]:

```
# handling categorical variables
print("Total categories in the feature Marital_Status:\n", data["Marital_Status"].valu
print("Total categories in the feature Education:\n", data["Education"].value_counts()
```

```
Total categories in the feature Marital_Status:
 Menikah        650
Bertunangan    438
Lajang         360
Cerai          177
Janda           52
Duda             3
Name: Marital_Status, dtype: int64

Total categories in the feature Education:
 S1     834
S3     373
S2     279
D3     159
SMA     35
Name: Education, dtype: int64
```

In [15]:
```
# Replacing marital status with english synonyms
data["Marital_Status"] = data['Marital_Status'].replace('Menikah','Married')
data["Marital_Status"] = data['Marital_Status'].replace('Bertunangan','Engaged')
data["Marital_Status"] = data['Marital_Status'].replace('Lajang','Bachelor')
data["Marital_Status"] = data['Marital_Status'].replace('Cerai','Divorced')

test["Marital_Status"] = test['Marital_Status'].replace('Menikah','Married')
test["Marital_Status"] = test['Marital_Status'].replace('Bertunangan','Engaged')
test["Marital_Status"] = test['Marital_Status'].replace('Lajang','Bachelor')
test["Marital_Status"] = test['Marital_Status'].replace('Cerai','Divorced')

data["Marital_Status"].value_counts()
```

Out[15]:
```
Married     650
Engaged     438
Bachelor    360
Divorced    177
Janda        52
Duda          3
Name: Marital_Status, dtype: int64
```

Feature Engineering

In [16]:
```
#Creating some new features for grouping

#Age of customer today
data["Age"] = 2023-data["Year_Birth"]
test["Age"] = 2023-test["Year_Birth"]

#Total spendings on various items
data["Spent"] = data["MntCoke"]+ data["MntFruits"]+ data["MntMeatProducts"]+ data["Mnt
test["Spent"] = test["MntCoke"]+ test["MntFruits"]+ test["MntMeatProducts"]+ test["Mnt

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

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

#Segmenting education levels in three groups
data["Education"]=data["Education"].replace({"SMA":"Undergraduate","S1":"Graduate","D3
test["Education"]=test["Education"].replace({"SMA":"Undergraduate","S1":"Graduate","D3
```

In [17]:
```
#Dropping some of the redundant features
to_drop = ["Z_CostContact", "Z_Revenue", "Year_Birth", "ID"]
data = data.drop(to_drop, axis=1)
test = test.drop(to_drop, axis=1)
```

In [18]:

```
# converting income to thousands
data["Income"] = data["Income"]/1000
test["Income"] = test["Income"]/1000

data.describe()
```

Out[18]:

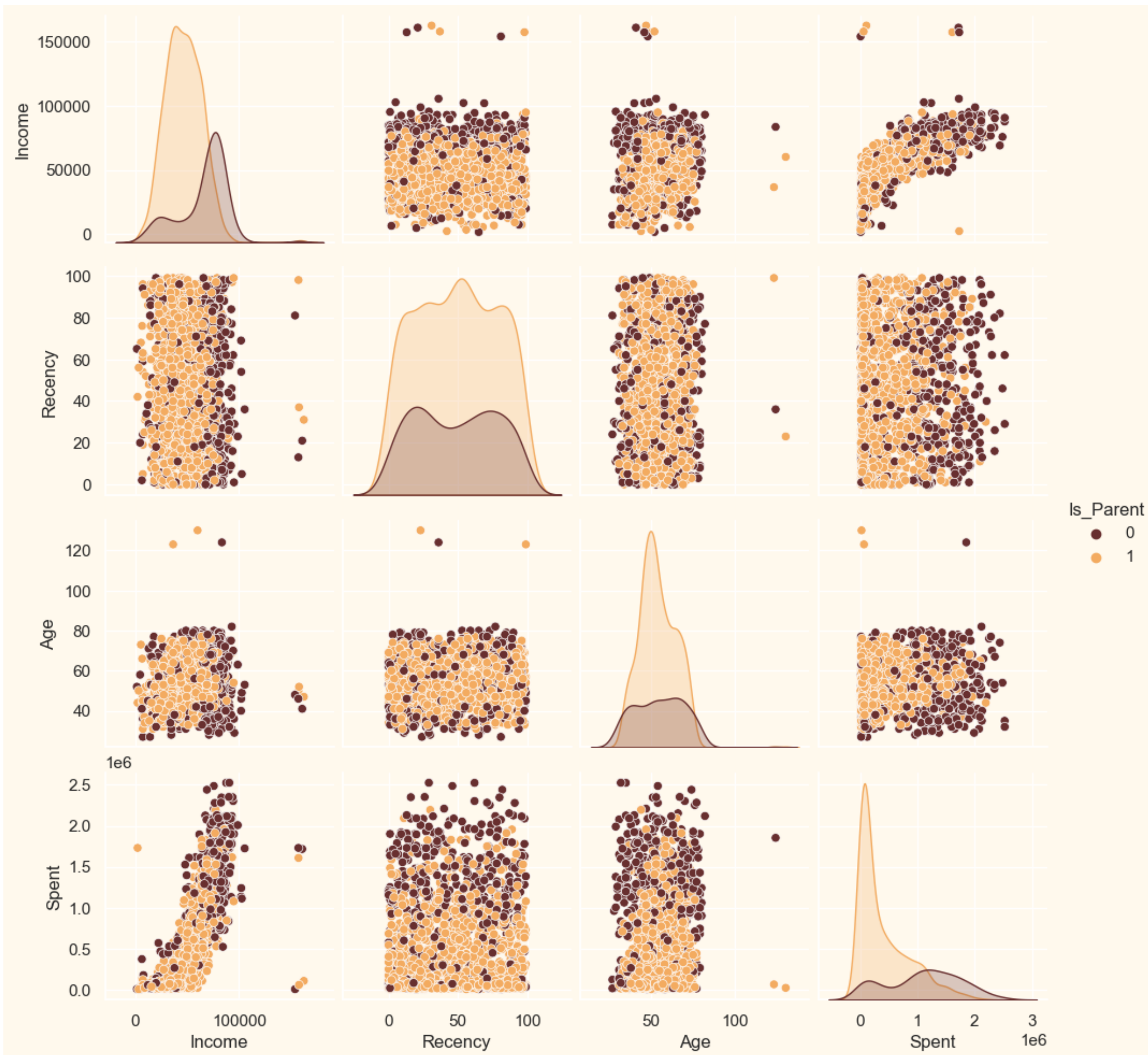
	Income	Kidhome	Teenhome	Recency	MntCoke	MntFruits	MntMeatProduc
count	1680.000000	1680.000000	1680.000000	1680.000000	1.680000e+03	1680.000000	1.680000e+
mean	52014.343355	0.452381	0.500000	49.083333	3.048994e+05	25918.452381	1.657738e+
std	21373.445420	0.546901	0.550055	28.930637	3.387051e+05	39532.059109	2.242424e+
min	1730.000000	0.000000	0.000000	0.000000	0.000000e+00	0.000000	1.000000e+
25%	35790.750000	0.000000	0.000000	24.000000	2.400000e+04	1000.000000	1.600000e+
50%	51445.500000	0.000000	0.000000	50.000000	1.730000e+05	8000.000000	6.800000e+
75%	67897.500000	1.000000	1.000000	74.000000	4.942500e+05	32000.000000	2.322500e+
max	162397.000000	2.000000	2.000000	99.000000	1.492000e+06	199000.000000	1.725000e+

8 rows × 26 columns

In [20]:

```
#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", "
#Plotting following features
To_Plot = [ "Income", "Recency", "Age", "Spent", "Is_Parent"]
print("Reletiving Plot Of Some Selected Features")
plt.figure()
sns.pairplot(data[To_Plot], hue= "Is_Parent",palette= (["#682F2F","#F3AB60"]))
#Taking hue
plt.show()
```

```
Reletiving Plot Of Some Selected Features

<Figure size 640x480 with 0 Axes>
```



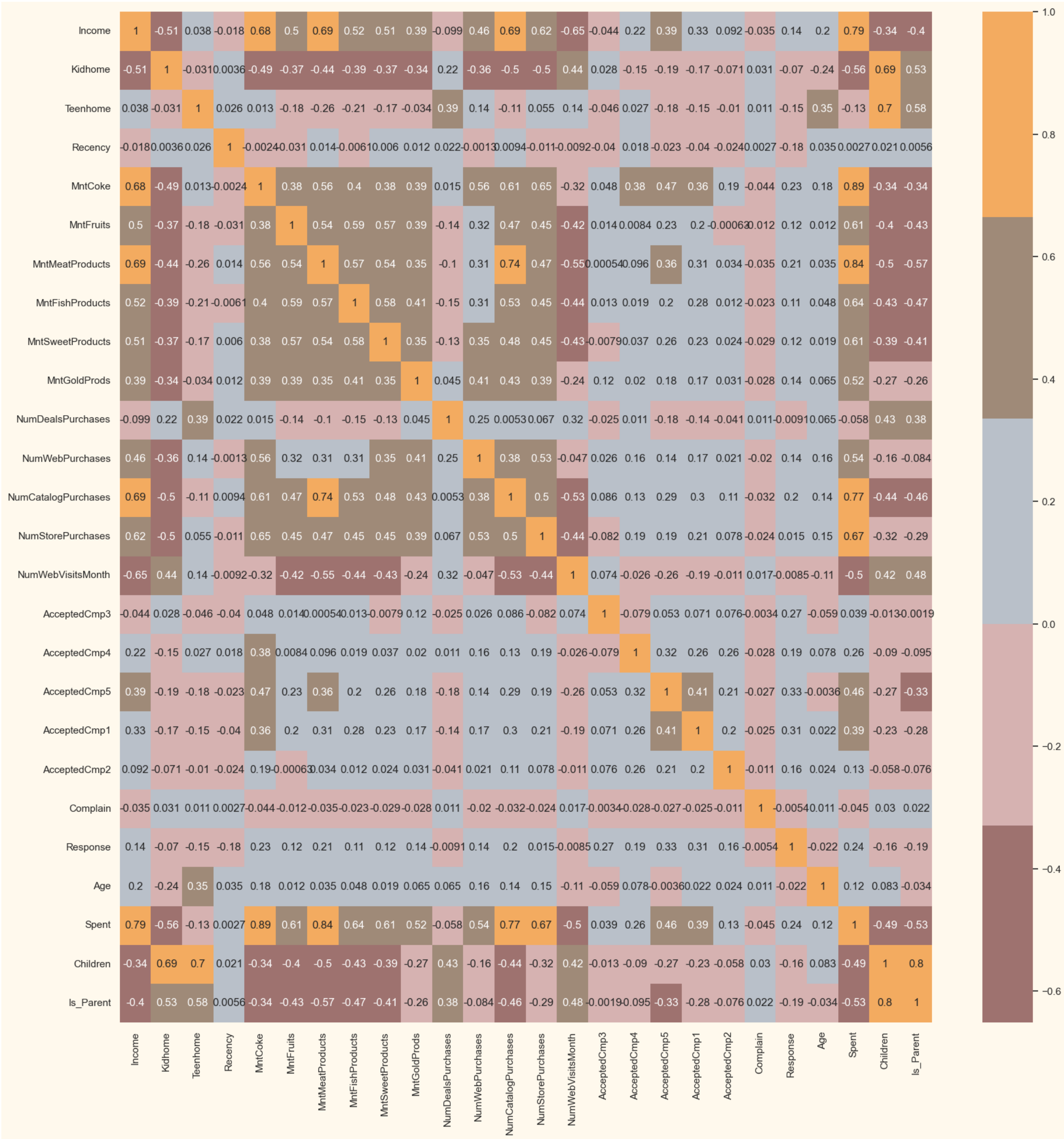
In [21]:

```
#Dropping the outliers on Age and income.
data = data[(data["Age"]<90)]
data = data[(data["Income"]<600000)]

len(data)
```

Out[21]: 1677

In [22]:
```python
#correlation matrix
corrmat= data.corr()
plt.figure(figsize=(20,20))
sns.heatmap(corrmat,annot=True, cmap=cmap, center=0)
```

Out[22]: <AxesSubplot:>



Data Preprocessing

In [23]:
```python
#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', 'Marital_Status']
```

In [24]:
```
#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 [25]:
```
LE=LabelEncoder()
for i in object_cols:
    test[i]=test[[i]].apply(LE.fit_transform)
```

In [26]:
```
#Creating a copy of data
ds = data.copy()
# creating a subset of dataframe by dropping the features on deals accepted and promot
cols_del = ['AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1','AcceptedCm
ds = ds.drop(cols_del, axis=1)
```

In [27]:
```
ts = test.copy()
ts = ts.drop(cols_del, axis=1)
```

In [28]:
```
#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 [29]:
```
scaler = StandardScaler()
scaler.fit(ts)
scaled_ts = pd.DataFrame(scaler.transform(ts),columns= ts.columns )
```

In [30]:
```
#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[30]:

	Education	Marital_Status	Income	Kidhome	Teenhome	Recency	MntCoke	MntFruits	MntMeatProd
0	-0.802765	-1.489737	0.287258	-0.827652	-0.909671	0.308614	0.974724	1.575077	1.69
1	-0.802765	-1.489737	-0.264686	1.001034	0.908587	-0.383107	-0.868284	-0.630678	-0.71
2	-0.802765	0.024072	0.917871	-0.827652	-0.909671	-0.798139	0.357434	0.586290	-0.17
3	-0.802765	0.024072	-1.186528	1.001034	-0.909671	-0.798139	-0.868284	-0.554617	-0.65
4	1.064415	1.033277	0.294512	1.001034	-0.909671	1.553710	-0.389811	0.434169	-0.21

5 rows × 21 columns

Dimensionality Reduction

Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables. There are a lot of features which classify the

customers, but many of them are redundant or correlated. So reducing the dimensionality helps in easy working with features.

Principal component analysis (PCA) is a technique for reducing the dimensionality of such datasets, increasing interpretability but at the same time minimizing information loss. We will first reduce the dimensionality and then try plotting the reduced dataframe

In [31]:

```
#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[31]:

	count	mean	std	min	25%	50%	75%	max
col1	1677.0	3.495514e-17	2.810285	-4.990753	-2.548165	-0.825075	2.371183	7.473642
col2	1677.0	-4.753370e-17	1.601270	-3.481987	-1.315620	-0.199698	1.155824	5.585791
col3	1677.0	-1.224754e-17	1.163446	-3.202030	-0.804723	-0.004204	0.847023	5.064139

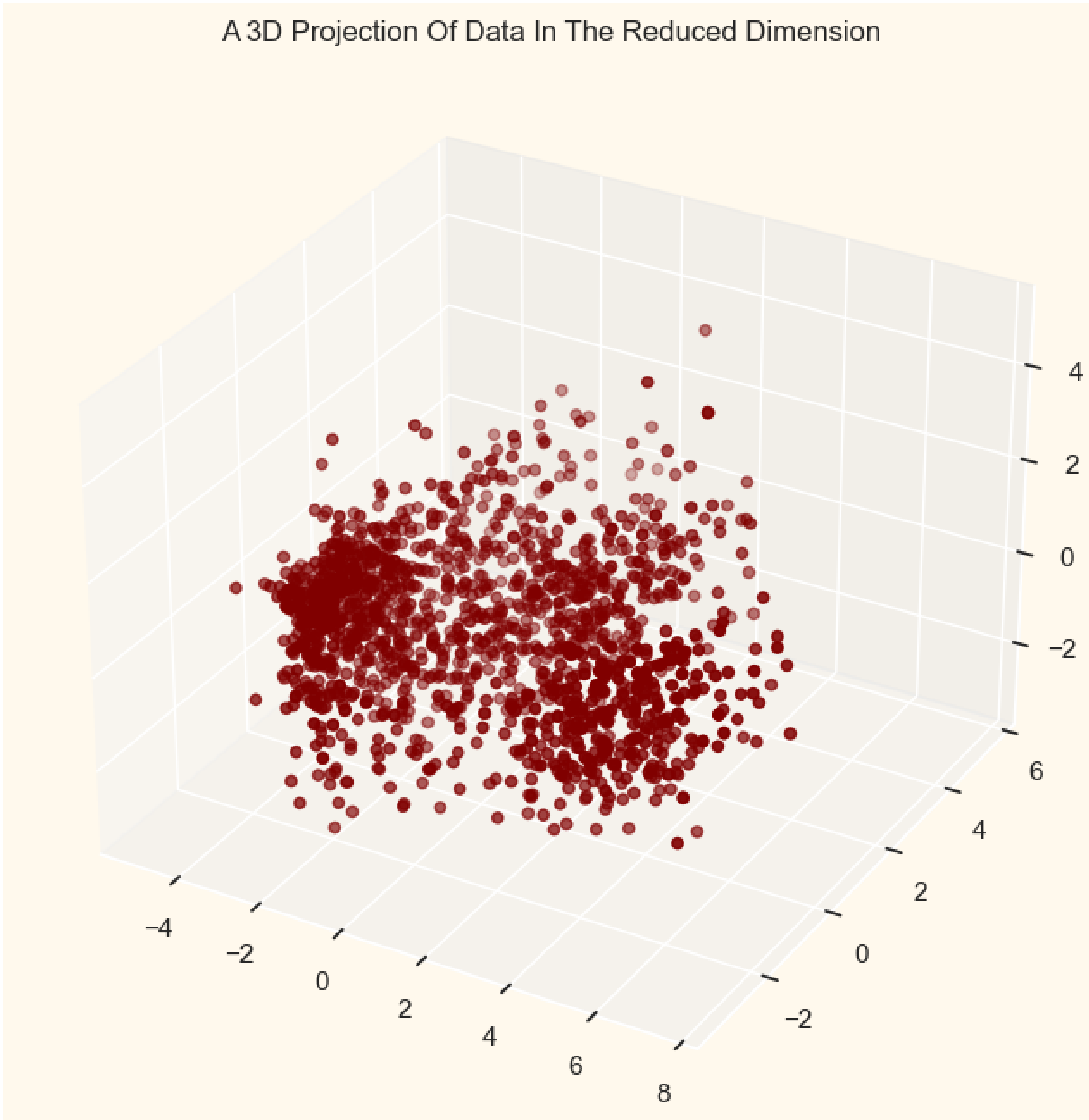
In [32]:

```
pca.fit(scaled_ts)
PCA_ts = pd.DataFrame(pca.transform(scaled_ts), columns=(["col1","col2", "col3"]))
PCA_ts.describe().T
```

Out[32]:

	count	mean	std	min	25%	50%	75%	max
col1	559.0	-1.191653e-17	2.736717	-5.565964	-2.531598	-0.754421	2.294045	6.959592
col2	559.0	1.124126e-16	1.611310	-3.742891	-1.357901	-0.061365	1.330641	5.220100
col3	559.0	1.827201e-17	1.154425	-3.217005	-0.767433	0.021124	0.738971	4.527263

In [33]:
```
#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

The clustering will be performed by Agglomerative clustering. Agglomerative clustering is a hierarchical clustering method. It involves merging examples until the desired number of clusters is achieved.

Steps involved in the Clustering:

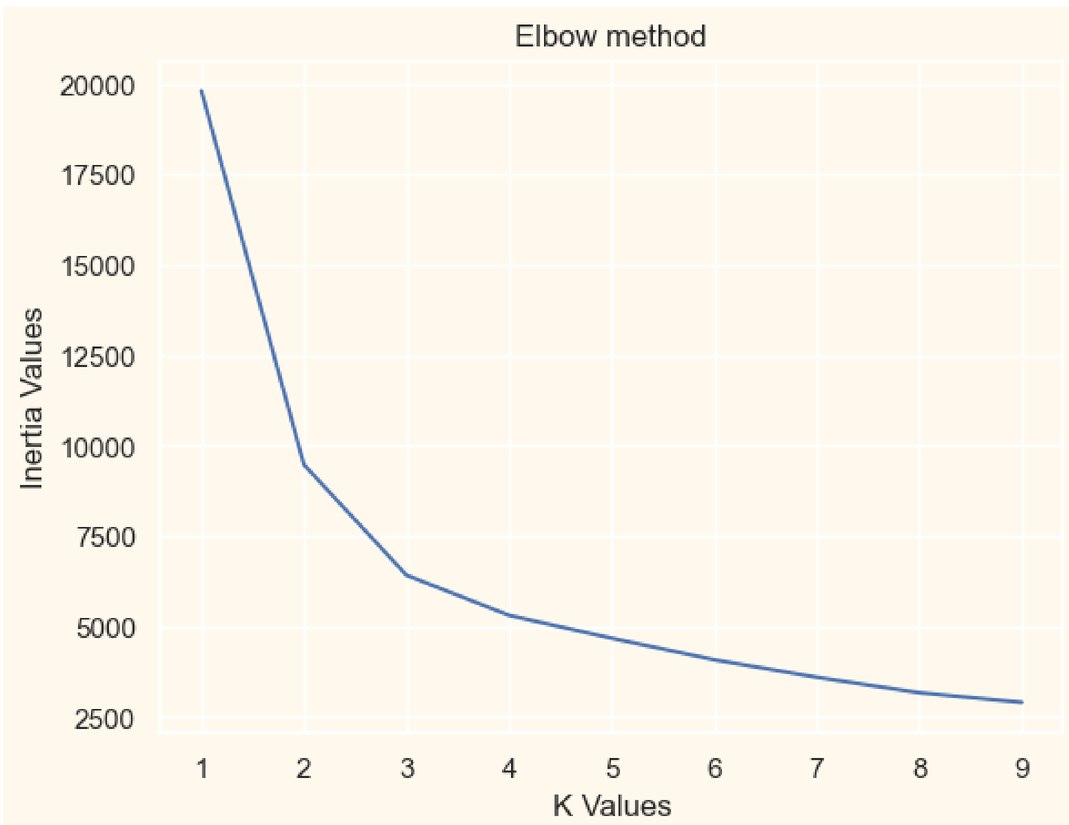
Elbow Method to determine the number of clusters to be formed

Clustering via Agglomerative Clustering

In [34]:
```
inertia_values=[]
k = range(1,10)
for i in k:
    model = KMeans(n_clusters=i)
    model.fit(PCA_ds)
    inertia_values.append(model.inertia_)
```

```
C:\Users\HP\anaconda3\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=7.
  warnings.warn(
```

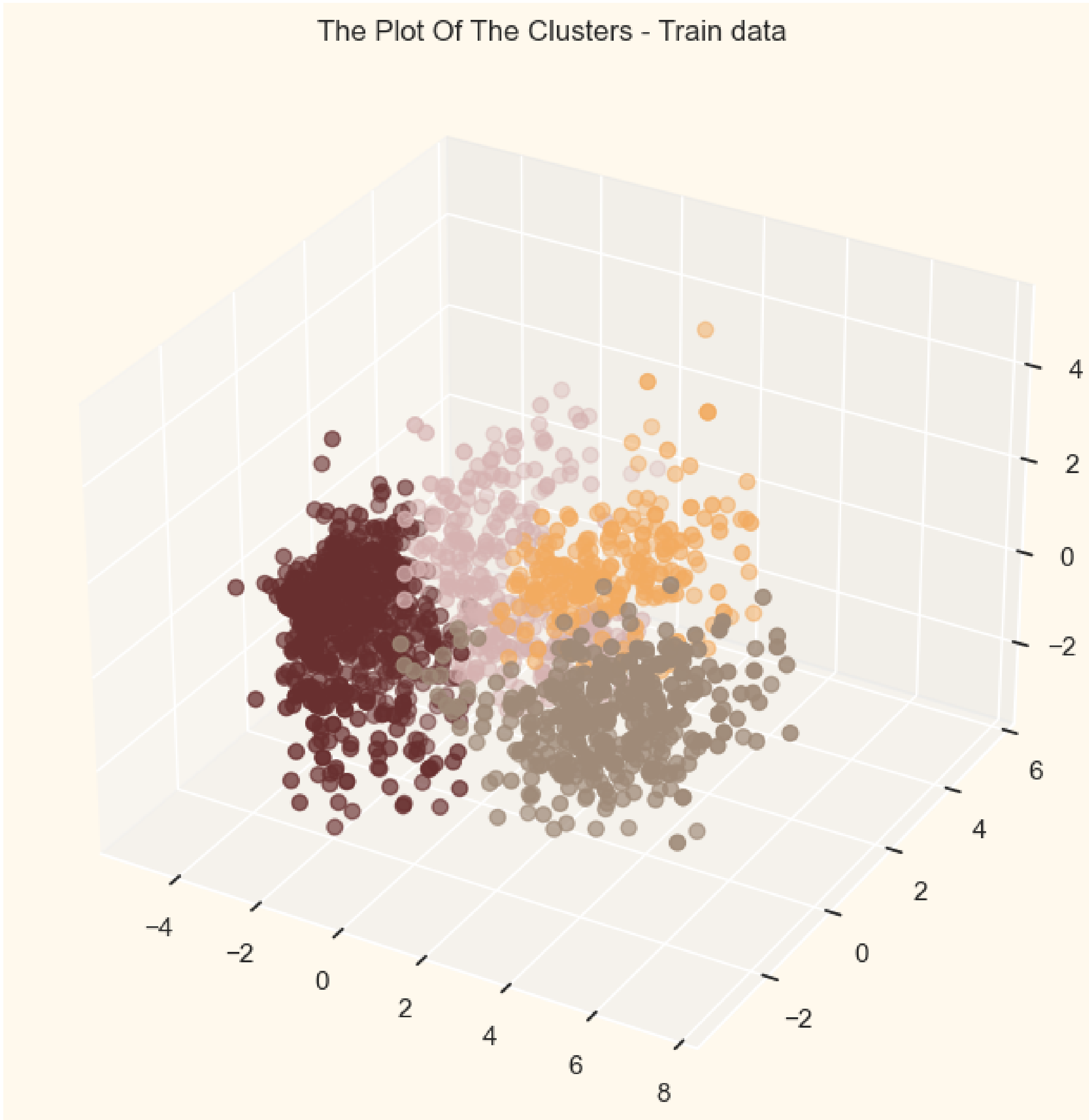
In [35]:
```
plt.plot(k,inertia_values)
plt.title("Elbow method")
plt.xlabel("K Values")
plt.ylabel("Inertia Values")

plt.show()
```



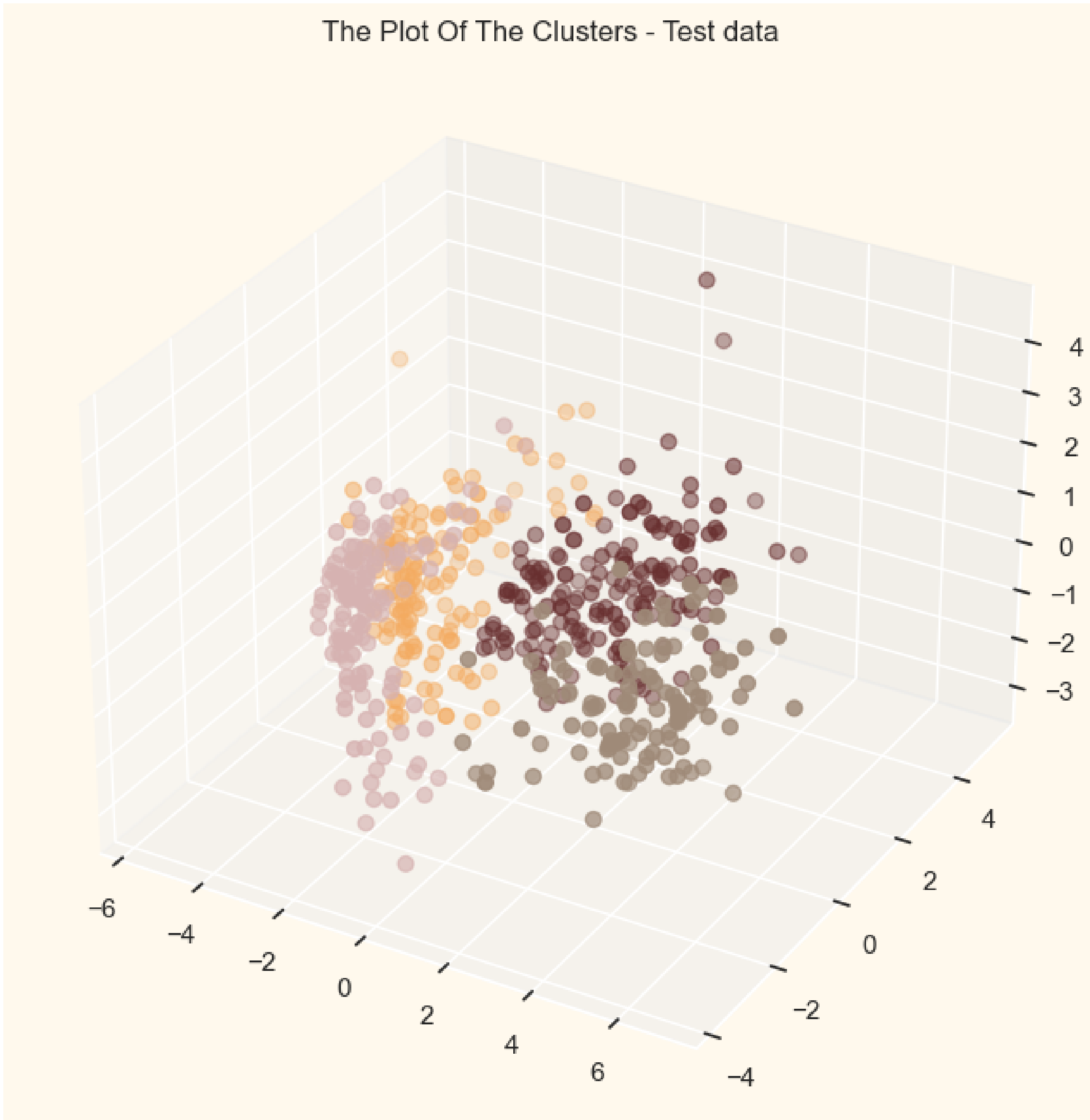
In [36]:
```
#Initiating the Agglomerative Clustering model with k=4
AC = AgglomerativeClustering(n_clusters=4)
# fit model and predict clusters
yhat_AC = AC.fit_predict(PCA_ds)
PCA_ds["Clusters"] = yhat_AC
#Adding the Clusters feature to the orignal dataframe.
data["Clusters"]= yhat_AC
```

In [37]:
```
test_AC = AC.fit_predict(PCA_ts)
PCA_ts["Clusters"] = test_AC
test["Clusters"]= test_AC
```

In [39]:
```
#Plotting the clusters for train data
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 - Train data")
plt.show()
```



In [40]:
```
#Plotting the clusters for test data
t =PCA_ts["col1"]
u =PCA_ts["col2"]
v =PCA_ts["col3"]

fig = plt.figure(figsize=(10,8))
ax = plt.subplot(111, projection='3d', label="bla")
ax.scatter(t, u, v, s=40, c=PCA_ts["Clusters"], marker='o', cmap = cmap )
ax.set_title("The Plot Of The Clusters - Test data")
plt.show()
```



Evaluating Models

The purpose of this section is to study the patterns in the clusters formed and determine the nature of the clusters' patterns.

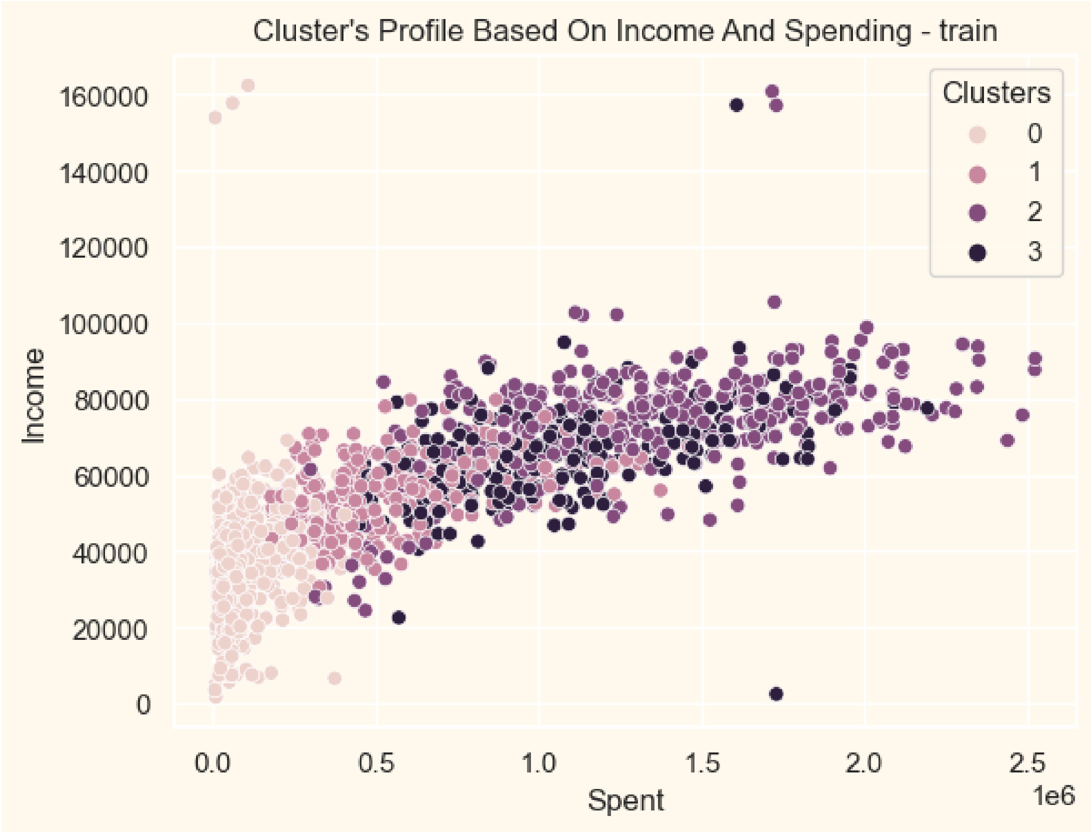
In [41]:
```
pl = sns.countplot(x=test["Clusters"])
pl.set_title("Distribution Of The Clusters for test data")
```

Out[41]:
```
Text(0.5, 1.0, 'Distribution Of The Clusters for test data')
```

The clusters seem to be fairly distributed

In [42]:
```
pl = sns.scatterplot(data = data,x=data["Spent"], y=data["Income"],hue=data["Clusters"
pl.set_title("Cluster's Profile Based On Income And Spending - train")
```

Out[42]:
```
Text(0.5, 1.0, "Cluster's Profile Based On Income And Spending - train")
```

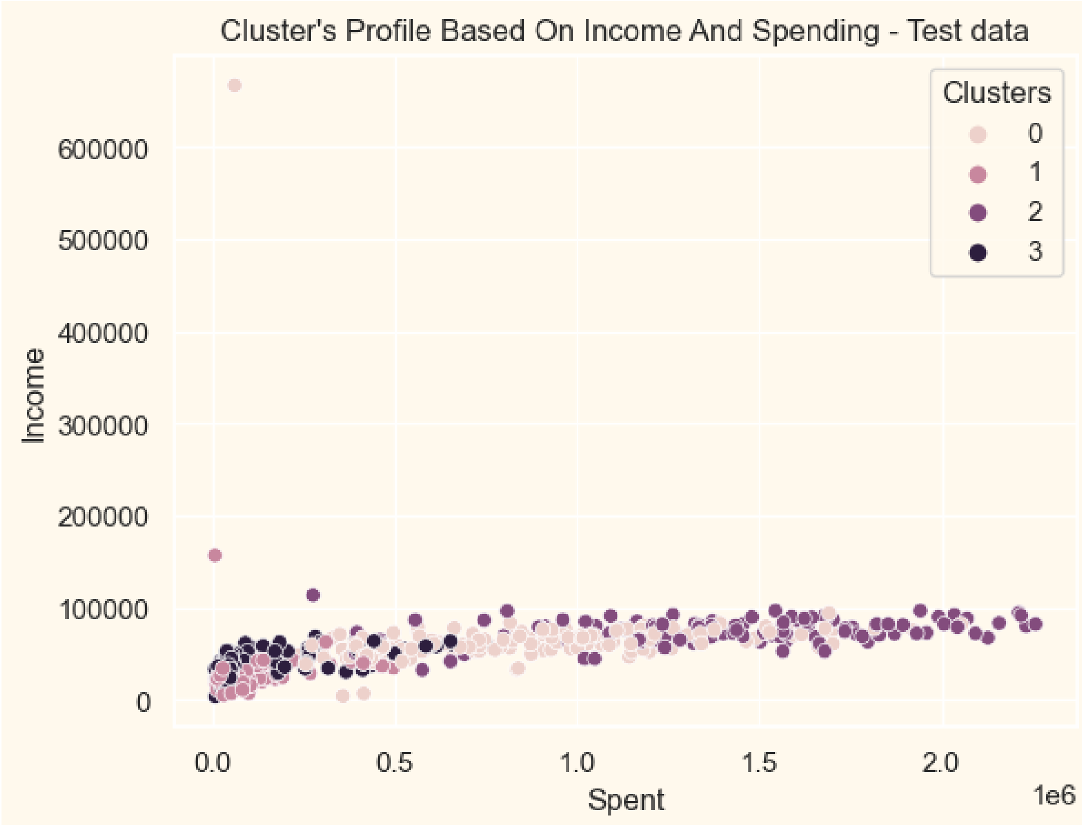


Classification

```
Group 0: Low income and low spending
Group 1: High income and high spending
Group 2: High income and low spending
Group 3: Average income and average spending
```

In [43]:
```
pl = sns.scatterplot(data = test,x=test["Spent"], y=test["Income"],hue=test["Clusters"
pl.set_title("Cluster's Profile Based On Income And Spending - Test data")
```

Out[43]:
```
Text(0.5, 1.0, "Cluster's Profile Based On Income And Spending - Test data")
```

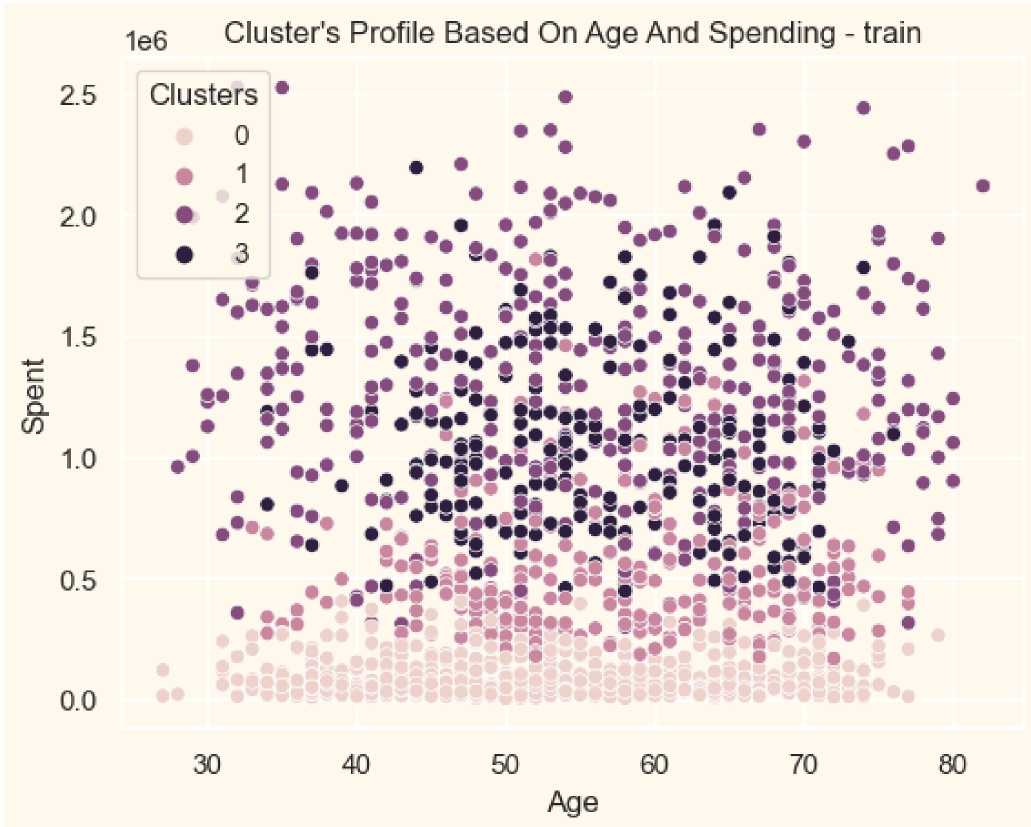


Classification

```
Group 0: High income and average spending
Group 1: Low income and low spending
Group 2: Low income high spending
Group 3: Average income and low spending
```

In [45]:
```
pl = sns.scatterplot(data = data,x=data["Age"], y=data["Spent"],hue=data["Clusters"])
pl.set_title("Cluster's Profile Based On Age And Spending - train")
```

Out[45]: Text(0.5, 1.0, "Cluster's Profile Based On Age And Spending - train")

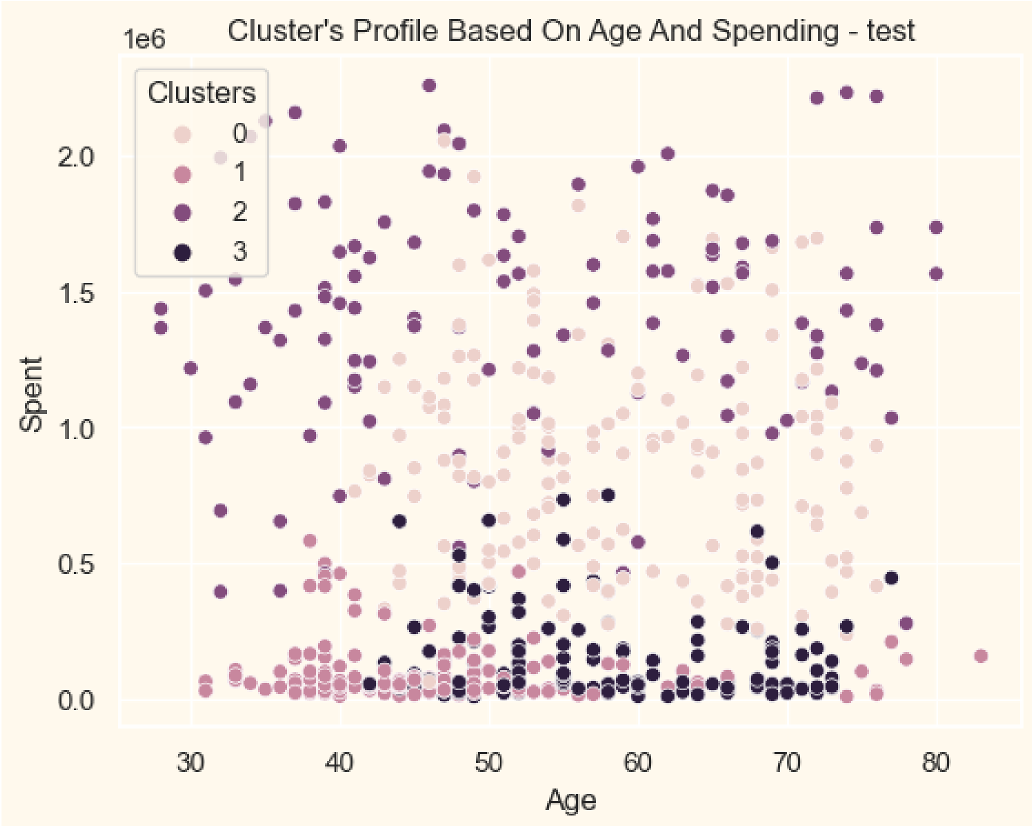


In [46]:

```
pl = sns.scatterplot(data = test,x=test["Age"], y=test["Spent"],hue=test["Clusters"])
pl.set_title("Cluster's Profile Based On Age And Spending - test")
```

Out[46]:

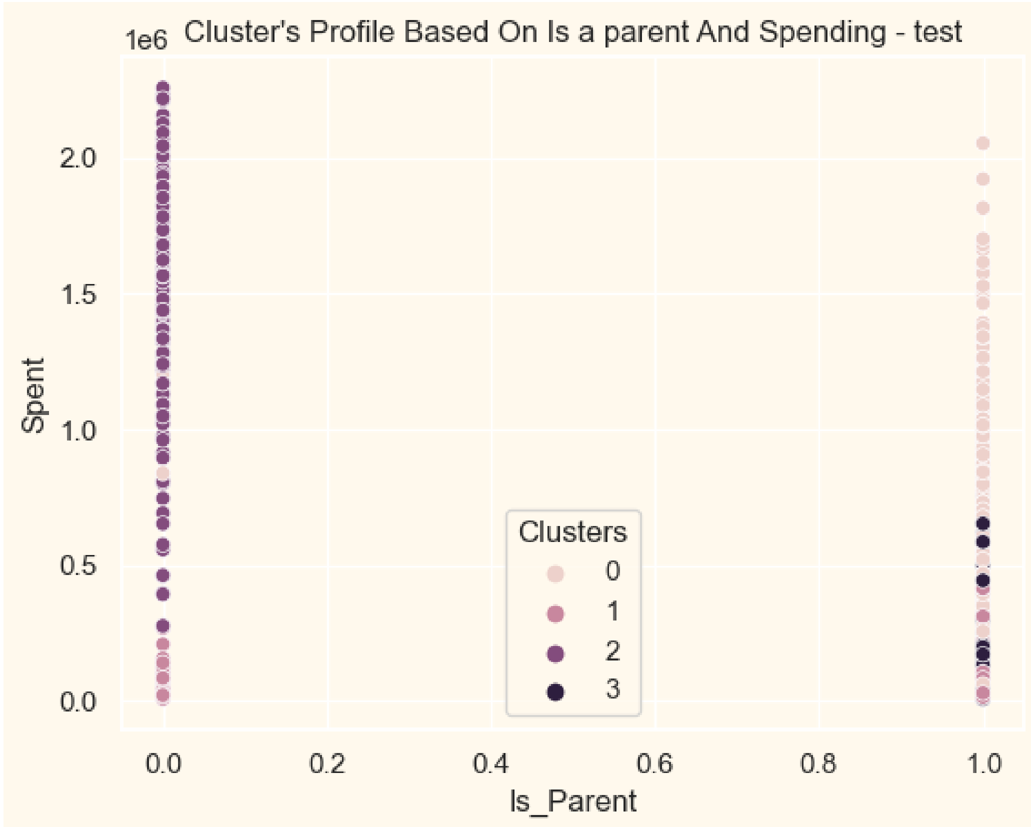
```
Text(0.5, 1.0, "Cluster's Profile Based On Age And Spending - test")
```




```
Group 0: Age between 40 & 80 with average to high spending
Group 1: Age between 30 & 80 with average spending
Group 2: Age between 25 & 80 with high spending
Group 3: Age between 40 & 70 low spending
```

In [47]:
```
pl = sns.scatterplot(data = test,x=test["Is_Parent"], y=test["Spent"],hue=test["Cluste
pl.set_title("Cluster's Profile Based On Is a parent And Spending - test")
```

Out[47]:
```
Text(0.5, 1.0, "Cluster's Profile Based On Is a parent And Spending - test")
```



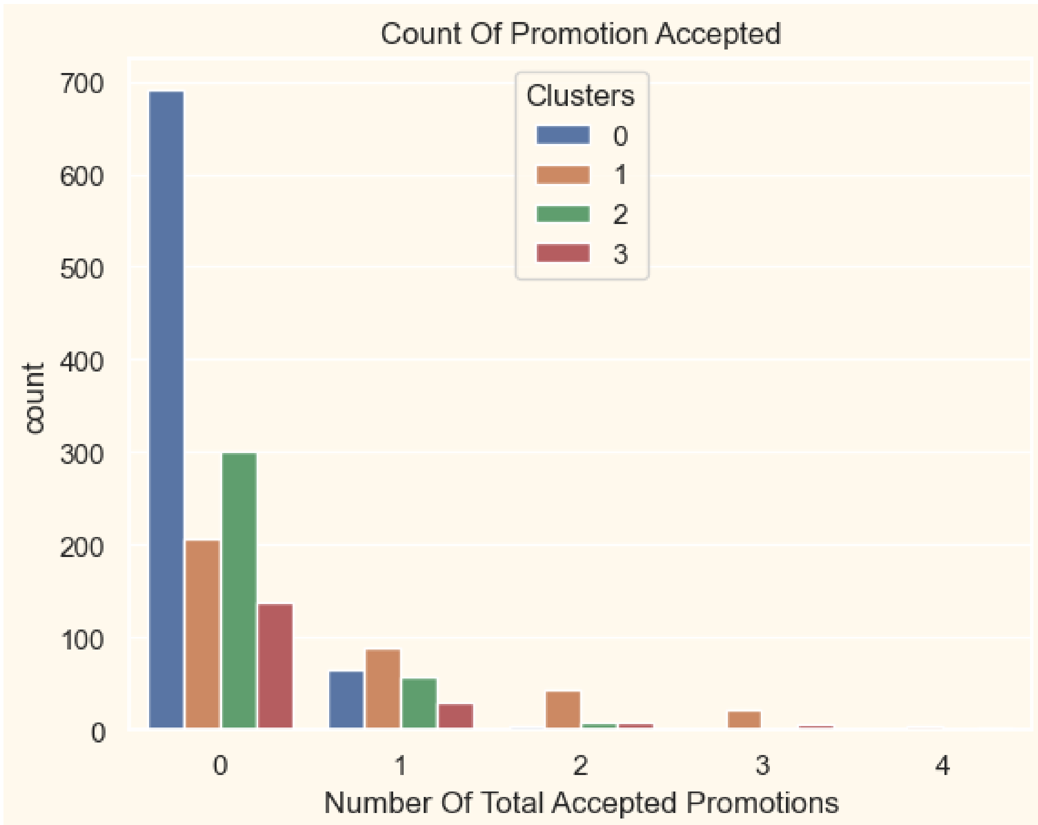
Most spending comes from customers who are not parents

In [97]:

```
#Creating a feature to get a sum of accepted promotions for train data
data["Total_Promos"] = data["AcceptedCmp1"]+ data["AcceptedCmp2"]+ data["AcceptedCmp3"
#Plotting count of total campaign accepted.
pl = sns.countplot(x=data["Total_Promos"],hue=data["Clusters"])
pl.set_title("Count Of Promotion Accepted")
pl.set_xlabel("Number Of Total Accepted Promotions")
```

Out[97]:

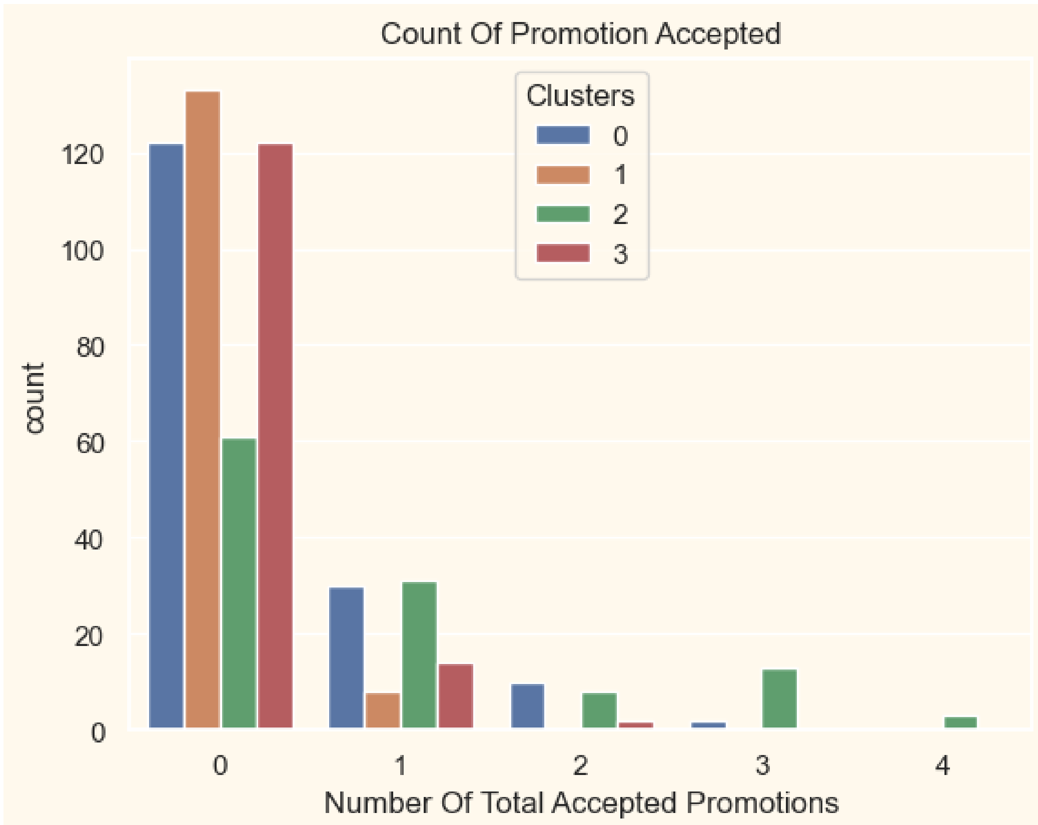
```
Text(0.5, 0, 'Number Of Total Accepted Promotions')
```



In [98]:

```
#Creating a feature to get a sum of accepted promotions for test data
test["Total_Promos"] = test["AcceptedCmp1"]+ test["AcceptedCmp2"]+ test["AcceptedCmp3"
#Plotting count of total campaign accepted.
pl = sns.countplot(x=test["Total_Promos"],hue=test["Clusters"])
pl.set_title("Count Of Promotion Accepted")
pl.set_xlabel("Number Of Total Accepted Promotions")
```

Out[98]: Text(0.5, 0, 'Number Of Total Accepted Promotions')



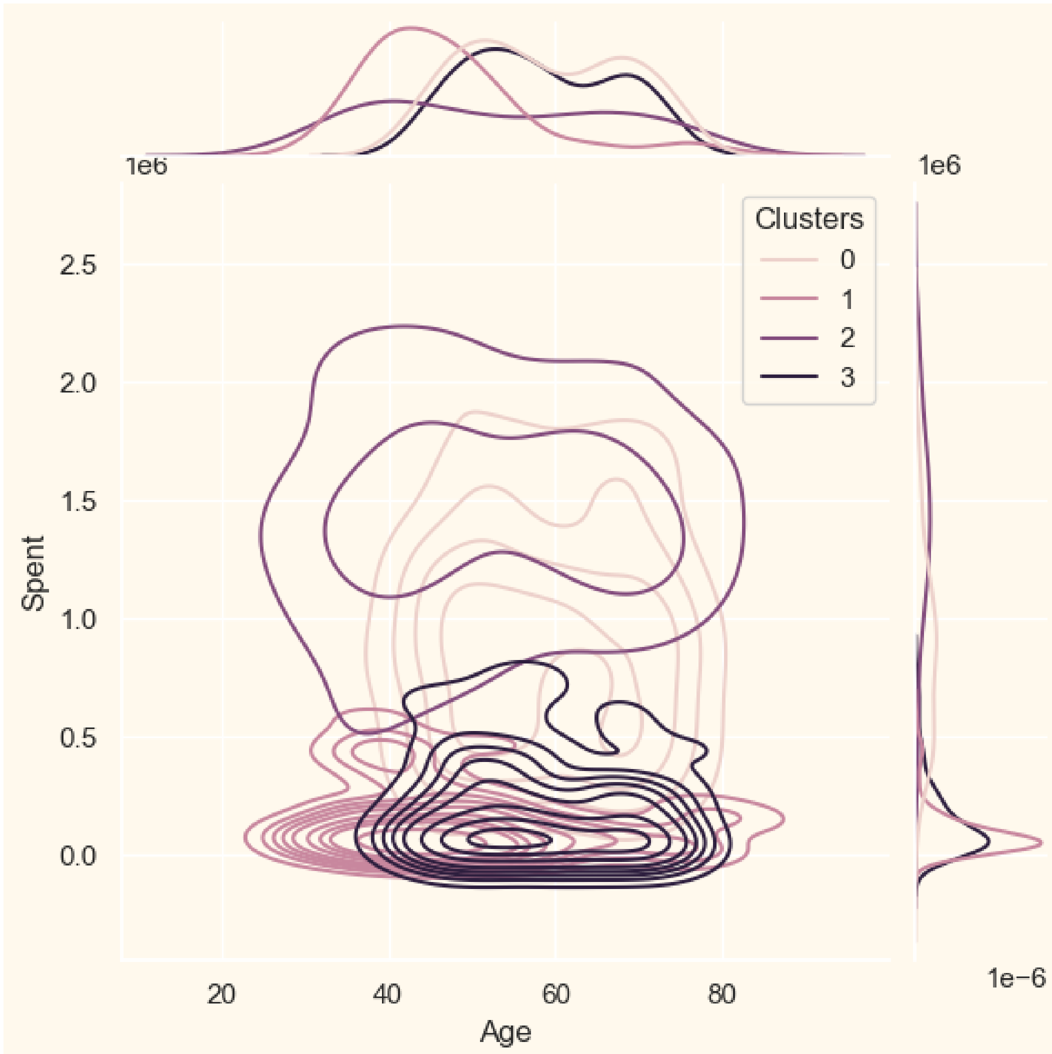
There isnt any overwhelming response to the campaigns so far. There are only a few participants overall. Moreover, no one has taken part in all 5 of them. Perhaps better-targeted and well-planned campaigns are required to boost sales.

Profiling the clusters

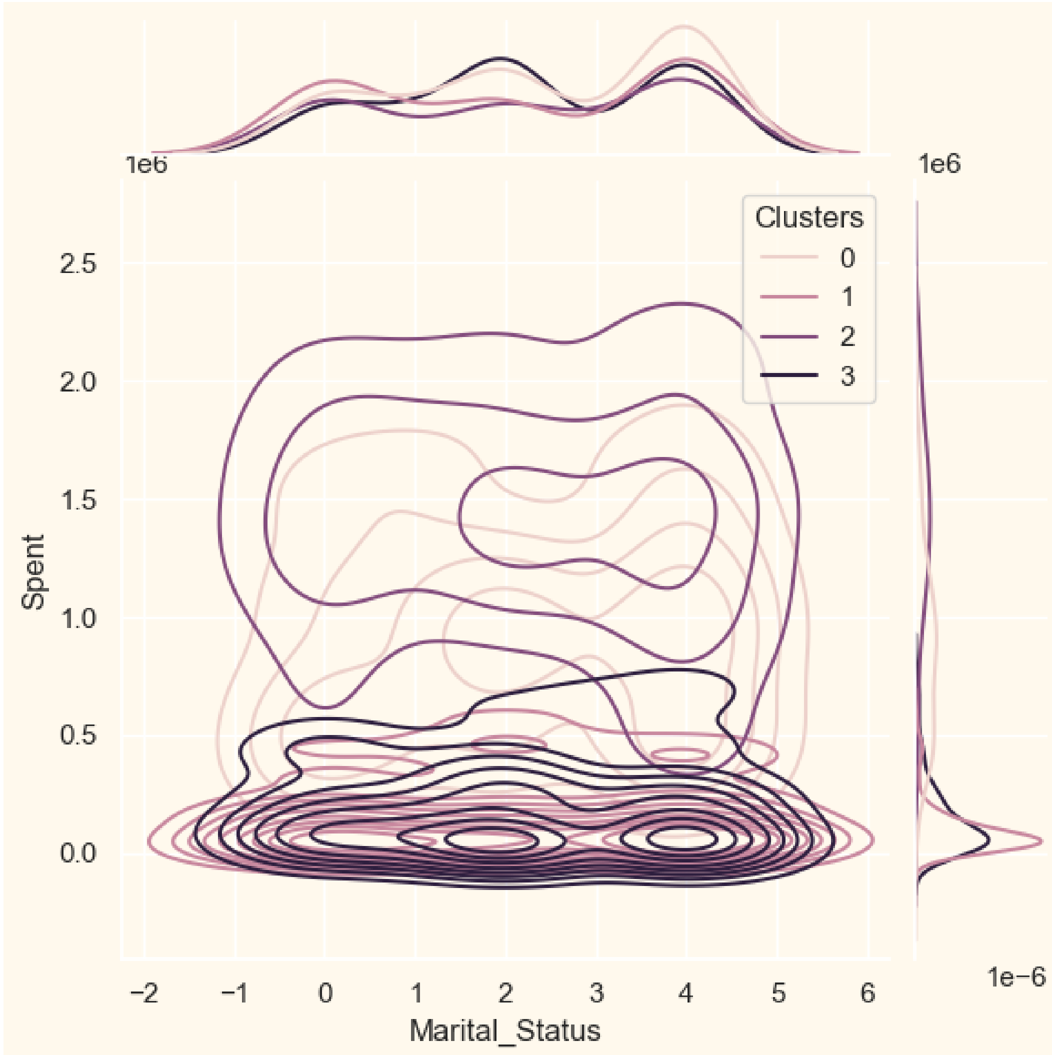
This is done to get an idea about who are valuable customer and who needs more attention from the retail store's marketing team. Plotting some of the features that indicates customer's personal traits based on the cluster they are in.

In [52]:
```
Features = ["Age", "Marital_Status", "Teenhome", "Is_Parent", "Education"]

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

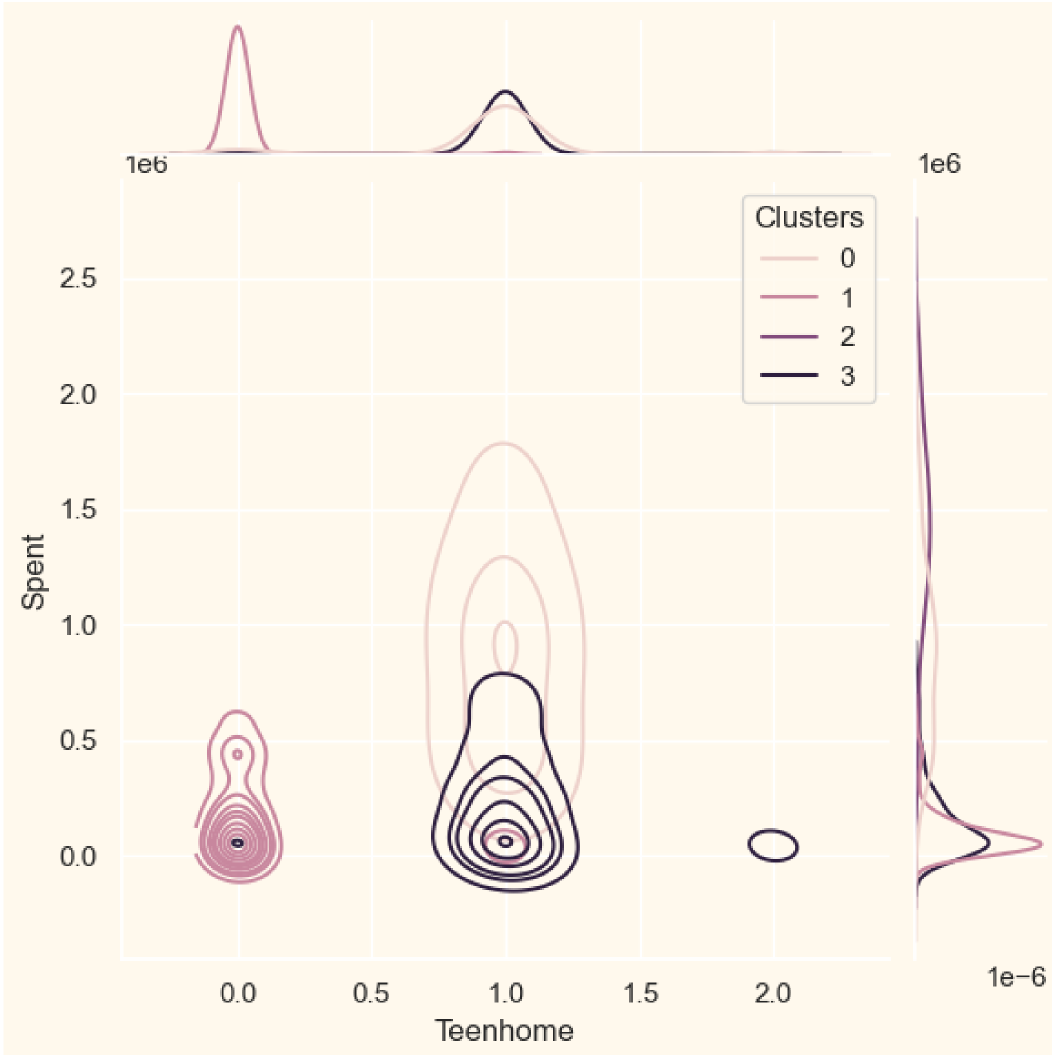
<Figure size 640x480 with 0 Axes>



<Figure size 640x480 with 0 Axes>

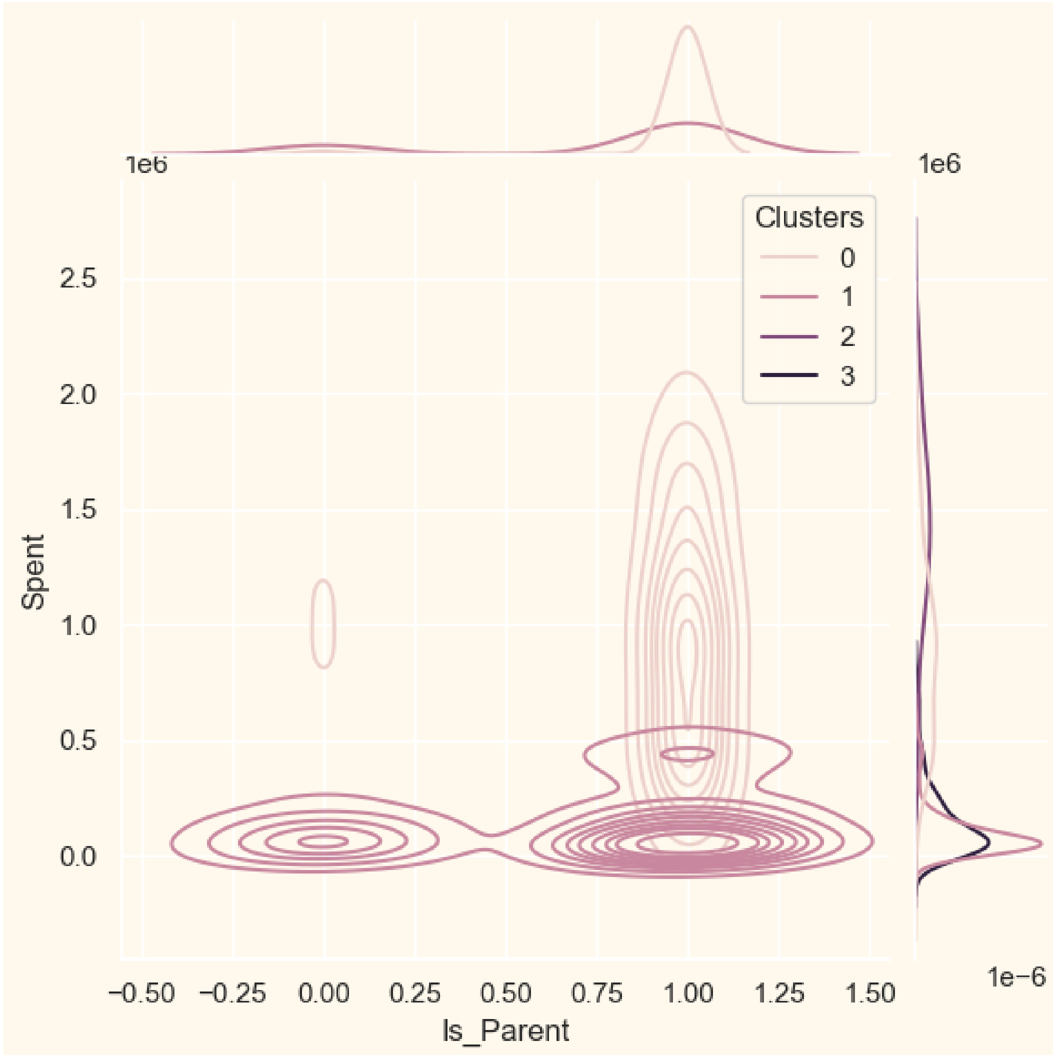



```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:316: UserWarning: Da
taset has 0 variance; skipping density estimate. Pass `warn_singular=False` to disabl
e this warning.
  warnings.warn(msg, UserWarning)

<Figure size 640x480 with 0 Axes>
```

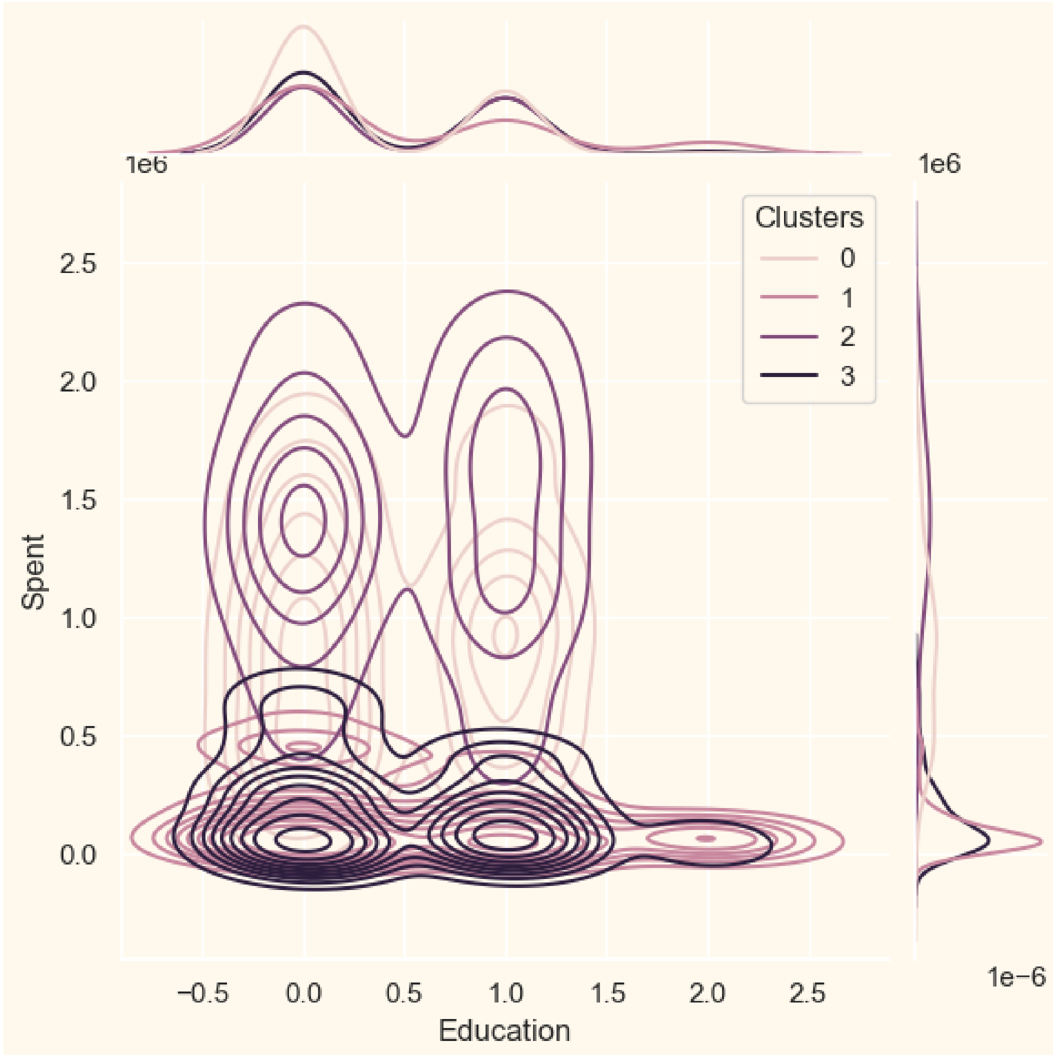



```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:316: UserWarning: Da
taset has 0 variance; skipping density estimate. Pass `warn_singular=False` to disabl
e this warning.
  warnings.warn(msg, UserWarning)
```

```
<Figure size 640x480 with 0 Axes>
```




```
&lt;Figure size 640x480 with 0 Axes&gt;
```



Cluster 0:

    Between age 40 & 80 with average spending. Mostly parents with teens at home. Mos
    tly graduates

Cluster 1:

    Between age 20 & 90 with low spending. Mostly parents with all education types

Cluster 2:

    Between age 40 & 80 with high spending. Includes people in all marital status and
    all education types

Cluster 3:

    Between age 40 & 80 low spending. There are mostly parents with teens.

In [ ]: