Customer Segmentation : Dimensionality reduction & Clustering with KMeans and Hierarchical (Agglomerative) using Python

Yassine HALLAL

Contents

I. Project objective . . . 2
II. About the dataset . . . 2
Attributes . . . 2
Acknowledgement . . . 3
III. Importing libraries . . . 3
IV. Importing data . . . 3
V. Data cleaning . . . 4
1. Dealing with missing values . . . 5
2. Feature engineering . . . 5
3. Dealing with outliers . . . 7
VI. A Brief Exploratory Data Analysis . . . 9
1. Univariate analysis . . . 9
2. Bivariate analysis . . . 14
VII. Data Preprocessing . . . 18
1. Encoding categorical features . . . 18
2. Scaling features . . . 19
VIII. Dimensionality Reduction . . . 19
Explained Variance Ratios of PCs . . . 20
Cumulative Variance explained by PCs . . . 21
IX. Clustering . . . 23
1. K-Means Clustering . . . 23
2. Hierarchical Clustering . . . 25
X. Evaluation . . . 27
XI. Profiling . . . 31
About the clusters 0 and 1 . . . 42
Conclusion . . . 42

I. Project objective

This project involves conducting unsupervised clustering on customer data extracted from a groceries firm's database. Customer segmentation aims to group customers based on similarities within clusters. By segmenting customers, I'll create distinct groups to maximize their relevance to the business. This segmentation will enable tailoring products to suit the specific needs and behaviors of each customer group. Additionally, it empowers the business to address the diverse concerns and preferences of different customer types effectively.

II. About the dataset

Attributes

1. People

- ID: Customer's unique identifier
- Year_Birth: Customer's birth year
- Education: Customer's education level
- Marital_Status: Customer's marital status
- Income: Customer's yearly household income
- Kidhome: Number of children in customer's household
- Teenhome: Number of teenagers in customer's household
- Dt_Customer: Date of customer's enrollment with the company
- Recency: Number of days since customer's last purchase
- Complain: 1 if the customer complained in the last 2 years, 0 otherwise

2. Products

- MntWines: Amount spent on wine in last 2 years
- MntFruits: Amount spent on fruits in last 2 years
- MntMeatProducts: Amount spent on meat in last 2 years
- MntFishProducts: Amount spent on fish in last 2 years
- MntSweetProducts: Amount spent on sweets in last 2 years
- MntGoldProds: Amount spent on gold in last 2 years

3. Promotion

- NumDealsPurchases: Number of purchases made with a discount
- AcceptedCmp1: 1 if customer accepted the offer in the 1st campaign, 0 otherwise
- AcceptedCmp2: 1 if customer accepted the offer in the 2nd campaign, 0 otherwise
- AcceptedCmp3: 1 if customer accepted the offer in the 3rd campaign, 0 otherwise
- AcceptedCmp4: 1 if customer accepted the offer in the 4th campaign, 0 otherwise
- AcceptedCmp5: 1 if customer accepted the offer in the 5th campaign, 0 otherwise
- Response: 1 if customer accepted the offer in the last campaign, 0 otherwise

4. Place

- NumWebPurchases: Number of purchases made through the company's website
- NumCatalogPurchases: Number of purchases made using a catalogue
- NumStorePurchases: Number of purchases made directly in stores
- NumWebVisitsMonth: Number of visits to company's website in the last month

5. Target

- Need to perform clustering to summarize customer segments.

Acknowledgement

- The dataset for this project is provided by Dr. Omar Romero-Hernandez.

For more information about the dataset, please visite here

III. Importing libraries

```
import numpy as np
import pandas as pd
import matplotlib
import scipy
import matplotlib.pyplot as plt
from matplotlib import colors
import seaborn as sns
import scipy.cluster.hierarchy as sch
import datetime
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
from sklearn.cluster import AgglomerativeClustering
from matplotlib.colors import ListedColormap
from sklearn import metrics
import sys
import warnings
warnings.filterwarnings("ignore")
```

IV. Importing data

```
data = pd.read_csv("C:/Users/Yassine/Desktop/customers.csv" , sep=";")
print("Number of datapoints:", len(data))
```

```
## Number of datapoints: 2240
```

```
data.head()
```

```
##      ID  Year_Birth   Education  ... Z_CostContact  Z_Revenue  Response
## 0  5524        1957  Graduation  ...             3         11         1
## 1  2174        1954  Graduation  ...             3         11         0
## 2  4141        1965  Graduation  ...             3         11         0
## 3  6182        1984  Graduation  ...             3         11         0
## 4  5324        1981         PhD  ...             3         11         0
##
## [5 rows x 29 columns]
```

V. Data cleaning

In this segment, I'll cover two key aspects:

- Dealing with missing values.
- Feature Engineering.
- Dealing with outliers.

These steps are essential for understanding the necessary actions to tidy up the dataset comprehensively. To gain a complete understanding of the dataset and the steps required for its cleansing, let's delve into the data information.

```
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
```

The above output reveals several important points :

- There are missing values within the 'income' column.
- The 'Dt_Customer' column, representing the customer joining date, isn't formatted as a DateTime data type.
- Our dataset contains categorical features denoted by 'object' data types, requiring encoding into numerical formats at a later stage.

1. Dealing with missing values

Given the scarcity of **missing values**, we'll proceed by simply **dropping** these rows. This action is not anticipated to significantly impact the overall dataset.

```
data = data.dropna()
print("The number of observations after dropping rows with missing values is: ", len(data))
```

```
## The number of observations after dropping rows with missing values is:  2216
```

2. Feature engineering

In the upcoming phase, I'll generate a new feature using the 'Dt_Customer' column to signify the duration each customer has been registered in the firm's database. To simplify this process, I'll calculate this duration relative to the most recent customer in the dataset. Therefore, I need to identify both the oldest and newest recorded dates to derive these values.

I'll create a new feature named 'Customer_For,' denoting the duration in days since each customer commenced shopping at the store, relative to the most recent recorded date in our records.

```
data['Dt_Customer'] = pd.to_datetime(data['Dt_Customer'])

newest_date = data['Dt_Customer'].max()
oldest_date = data['Dt_Customer'].min()

data['Customer_For'] = (newest_date - data['Dt_Customer']).dt.days

print("Newest Date:", newest_date)
```

```
## Newest Date: 2014-06-29 00:00:00
```

```
print("Oldest Date:", oldest_date)
```

```
## Oldest Date: 2012-07-30 00:00:00
```

```
data["Customer_For"] = pd.to_numeric(data["Customer_For"], errors="coerce")
print(data[['Dt_Customer', 'Customer_For']].head())
```

```
##   Dt_Customer  Customer_For
## 0  2012-09-04           663
## 1  2014-03-08           113
## 2  2013-08-21           312
## 3  2014-02-10           139
## 4  2014-01-19           161
```

To get a clear understanding of the categorical features, mainly *Marital_Status* and *Education*, we will explore their unique values

```
print("Categories in Marital_Status:\n", data["Marital_Status"].value_counts(), "\n")
```

```
## Categories in Marital_Status:
##  Marital_Status
## Married     857
## Together    573
## Single      471
## Divorced    232
## Widow        76
## Alone         3
## Absurd        2
## YOLO          2
## Name: count, dtype: int64
```

```
print("Categories in Education:\n", data["Education"].value_counts())
```

```
## Categories in Education:
##  Education
## Graduation    1116
## PhD            481
## Master         365
## 2n Cycle       200
## Basic           54
## Name: count, dtype: int64
```

In the upcoming phase, I'll be implementing the following steps to engineer new features:

- Deriving the **'Age'** of each customer using the **'Year_Birth**,' signifying their birth year.
- Introducing a new feature named '**Spending**' indicating the total expenditure made by customers across various categories over a two-year span.
- Creating the '**Living_With**' feature based on '**Marital_Status**' to extract information about couples' living arrangements.
- Constructing a '**Children**' feature to represent the total number of children, encompassing kids and teenagers, within a household.
- To gain a deeper understanding of households, generating a '**Family_Size**' feature.
- Establishing an '**Is_Parent**' feature to denote the parenthood status of customers.
- Simplifying the '**Education**' feature into three categories by consolidating its value counts.
- Finally, eliminating redundant features from the dataset.

```
#Age was calculated based on the last purchase date (2014)
data['Age'] = 2014 - data['Year_Birth']

data['Spending'] = data['MntWines'] + data['MntFruits'] + data['MntMeatProducts'] + data['MntFishProduc
```

```
data["Living_With"] = data["Marital_Status"].replace({"Married":"Partner", "Together":"Partner", "Absurd

data["Children"] = data["Kidhome"] + data["Teenhome"]

data["Family_Size"] = data["Living_With"].replace({"Alone": 1, "Partner":2})+ data["Children"]

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

data["Education"] = data["Education"].replace({"Basic":"Undergraduate","2n Cycle":"Undergraduate", "Grad


print(data[['Spending', 'Age', 'Children', 'Family_Size', 'Is_Parent', 'Education']].head())
```

```
##    Spending  Age  Children  Family_Size  Is_Parent     Education
## 0      1617   57         0            1          0      Graduate
## 1        27   60         2            3          1      Graduate
## 2       776   49         0            2          0      Graduate
## 3        53   30         1            3          1      Graduate
## 4       422   33         1            3          1  Postgraduate
```

```
#Changing colnames to make them easier to deal with

data = data.rename(columns={"MntWines": "Wines","MntFruits":"Fruits","MntMeatProducts":"Meat","MntFishP

#Dropping some of the redundant features

dropped = ["Marital_Status", "Dt_Customer", "Z_CostContact", "Z_Revenue", "Year_Birth", "ID"]
data = data.drop(dropped, axis=1)
```

3. Dealing with outliers

```
data.describe().loc[['count', 'mean', 'std', 'max', 'min']].T
```

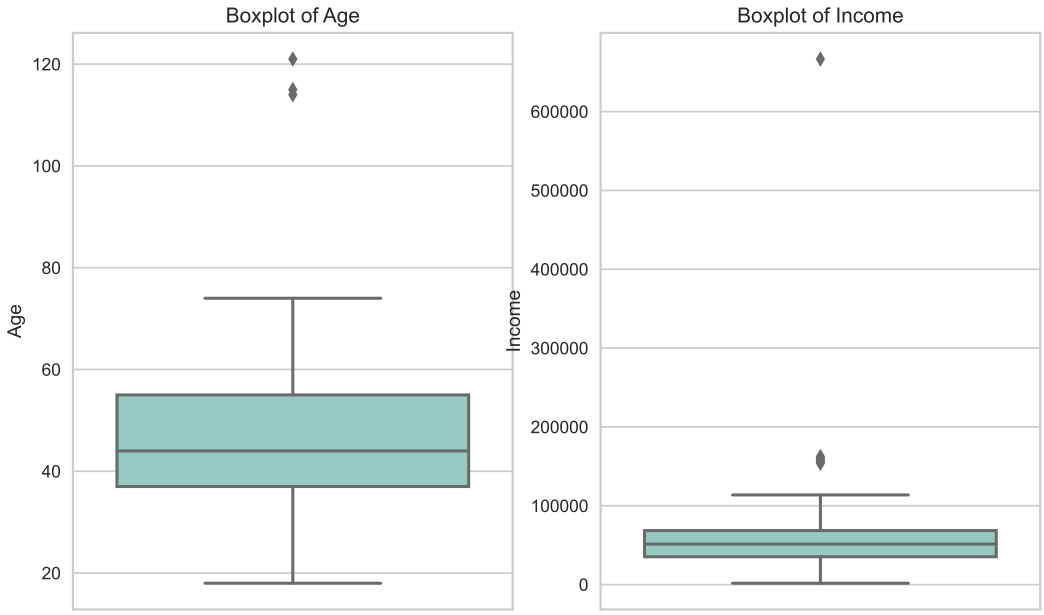
```
##                       count          mean           std       max     min
## Income               2216.0  52247.251354  25173.076661  666666.0  1730.0
## Kidhome              2216.0      0.441787      0.536896       2.0     0.0
## Teenhome             2216.0      0.505415      0.544181       2.0     0.0
## Recency              2216.0     49.012635     28.948352      99.0     0.0
## Wines                2216.0    305.091606    337.327920    1493.0     0.0
## Fruits               2216.0     26.356047     39.793917     199.0     0.0
## Meat                 2216.0    166.995939    224.283273    1725.0     0.0
## Fish                 2216.0     37.637635     54.752082     259.0     0.0
## Sweets               2216.0     27.028881     41.072046     262.0     0.0
## Gold                 2216.0     43.965253     51.815414     321.0     0.0
## NumDealsPurchases    2216.0      2.323556      1.923716      15.0     0.0
## NumWebPurchases      2216.0      4.085289      2.740951      27.0     0.0
## NumCatalogPurchases  2216.0      2.671029      2.926734      28.0     0.0
## NumStorePurchases    2216.0      5.800993      3.250785      13.0     0.0
## NumWebVisitsMonth    2216.0      5.319043      2.425359      20.0     0.0
## AcceptedCmp3         2216.0      0.073556      0.261106       1.0     0.0
```

```
## AcceptedCmp4         2216.0      0.074007      0.261842       1.0      0.0
## AcceptedCmp5         2216.0      0.073105      0.260367       1.0      0.0
## AcceptedCmp1         2216.0      0.064079      0.244950       1.0      0.0
## AcceptedCmp2         2216.0      0.013538      0.115588       1.0      0.0
## Complain             2216.0      0.009477      0.096907       1.0      0.0
## Response             2216.0      0.150271      0.357417       1.0      0.0
## Customer_For         2216.0    353.521209    202.434667     699.0      0.0
## Age                  2216.0     45.179603     11.985554     121.0     18.0
## Spending             2216.0    607.075361    602.900476    2525.0      5.0
## Children             2216.0      0.947202      0.749062       3.0      0.0
## Family_Size          2216.0      2.592509      0.905722       5.0      1.0
## Is_Parent            2216.0      0.714350      0.451825       1.0      0.0
```

From the descriptive statistics, we can see that there are outliers in the '**Income**' (666666) and '**Age**' (121) features.

The boxplot will help us detect potential outliers (Extreme values) in the two features.

```
plt.figure(figsize=(10, 6))

plt.subplot(1, 2, 1)
sns.boxplot(data=data, y='Age', palette='Set3')
plt.title('Boxplot of Age')
plt.ylabel('Age')

plt.subplot(1, 2, 2)
sns.boxplot(data=data, y='Income', palette='Set3')
plt.title('Boxplot of Income')
plt.ylabel('Income')

plt.show()
```



We can clearly see that the **Age** feature has 3 outliers, while the **Income** feature has 1 outlier. We will drop these observations (Total of 4).

```
data = data[(data["Age"] < 90)]
data = data[(data["Income"] < 600000)]
print("Number of observations after removing the outliers is:", len(data))
```

```
## Number of observations after removing the outliers is: 2212
```

Now the data is clean, new features has been added, and outliers has been deleted. Now we will proceed to data preprocessing.

VI. A Brief Exploratory Data Analysis

This section is devised into 2 main categories :

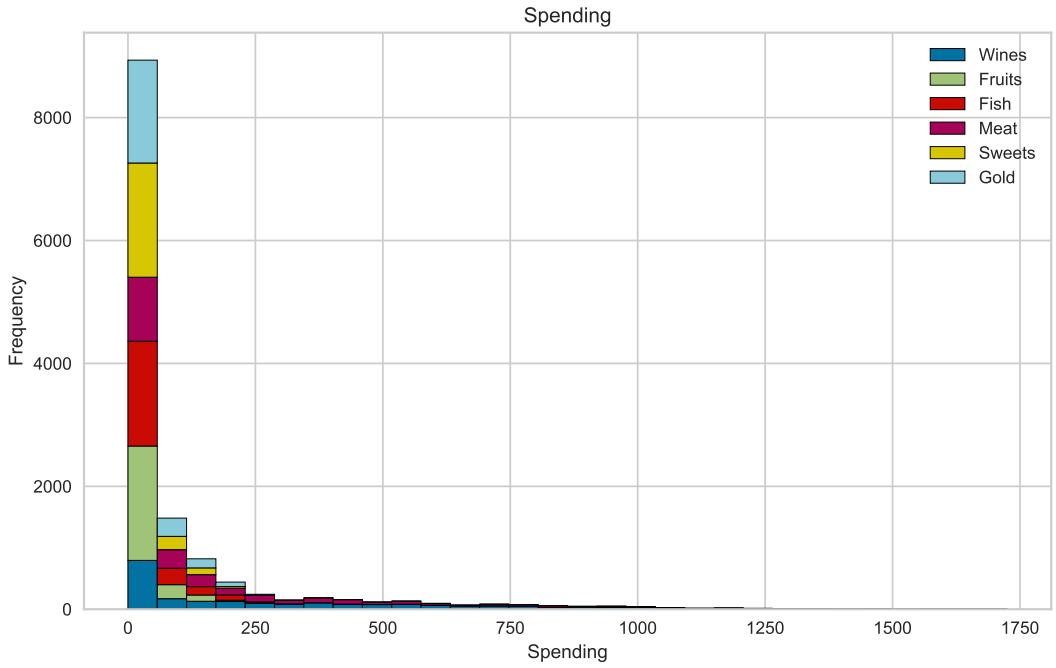
- Univariate analysis
- Bivariate analysis

1. Univariate analysis

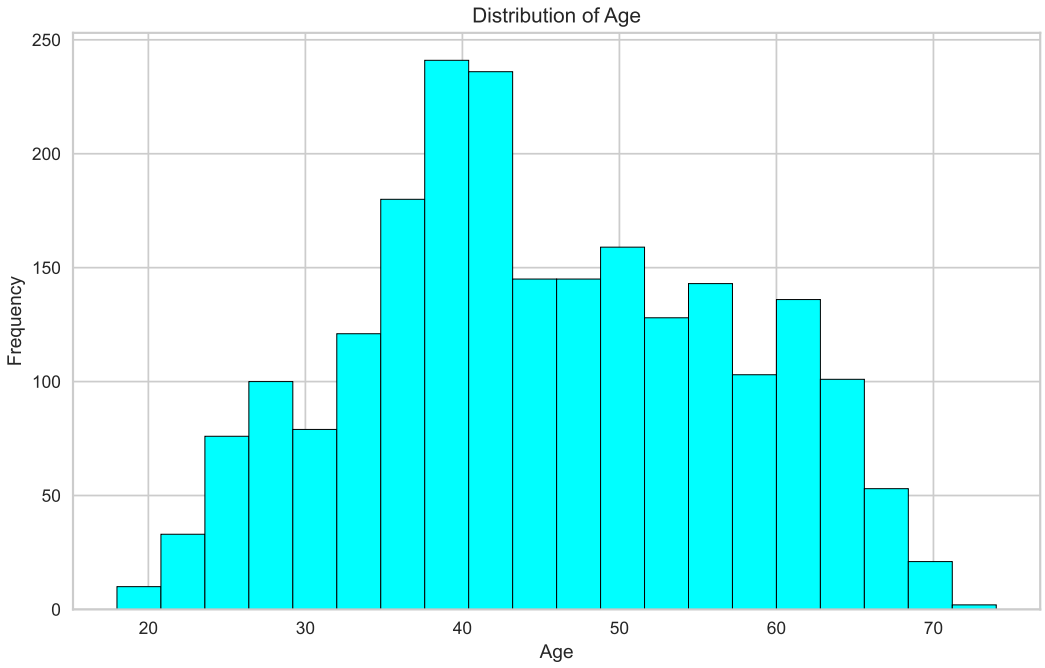
This involves examining individual variables in a dataset to understand their distributions, patterns, and central tendencies. The main variables we are going to explore are : Spending, Age, Education and Marital Status.

```
cmap = colors.ListedColormap(["#683F2F", "#9E826F", "#D5B2B1", "#B4C0C9", "#9F8A88", "#F3AB61"])


spending_cols = ['Wines', 'Fruits', 'Fish', 'Meat', 'Sweets', 'Gold']

# Create a stacked histogram showing contribution of each variable to spending
plt.figure(figsize=(10, 6))
plt.hist(data[spending_cols], bins=30, stacked=True, edgecolor='black')
```

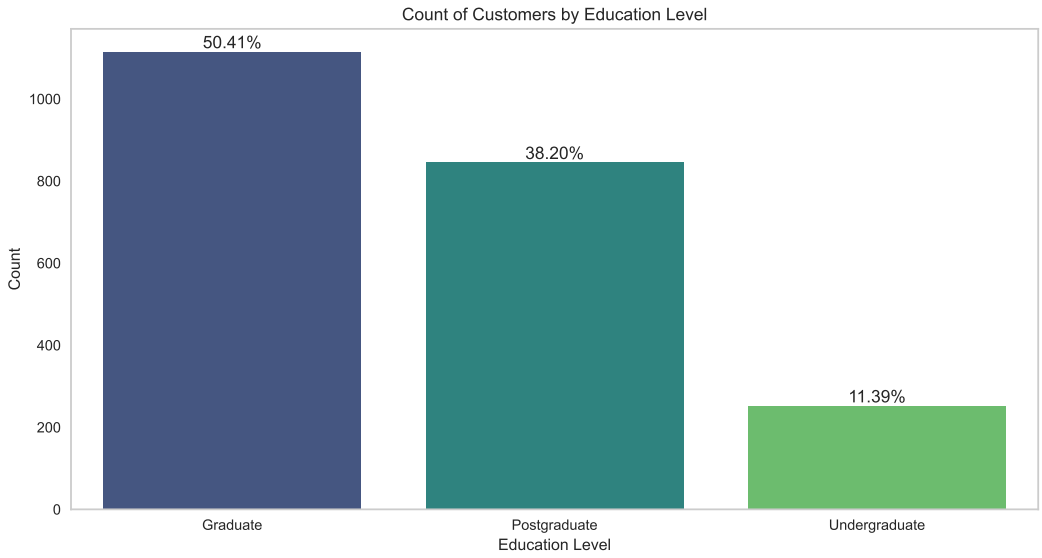
```
plt.title('Spending')
plt.xlabel('Spending')
plt.ylabel('Frequency')
plt.legend(spending_cols)
plt.grid(True)
plt.show()
```



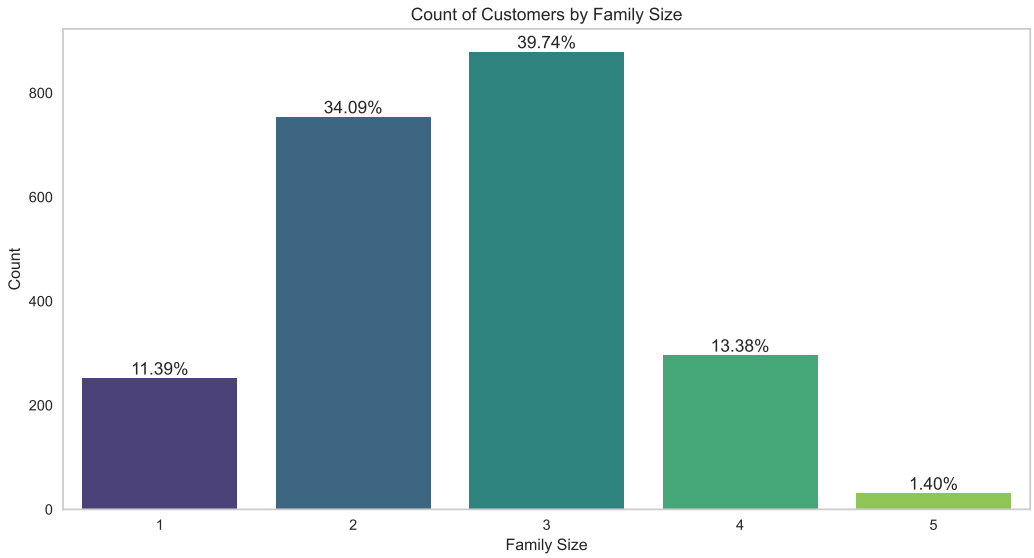

```
plt.figure(figsize=(10, 6))
plt.hist(data["Age"], bins=20, color='cyan', edgecolor='black')
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



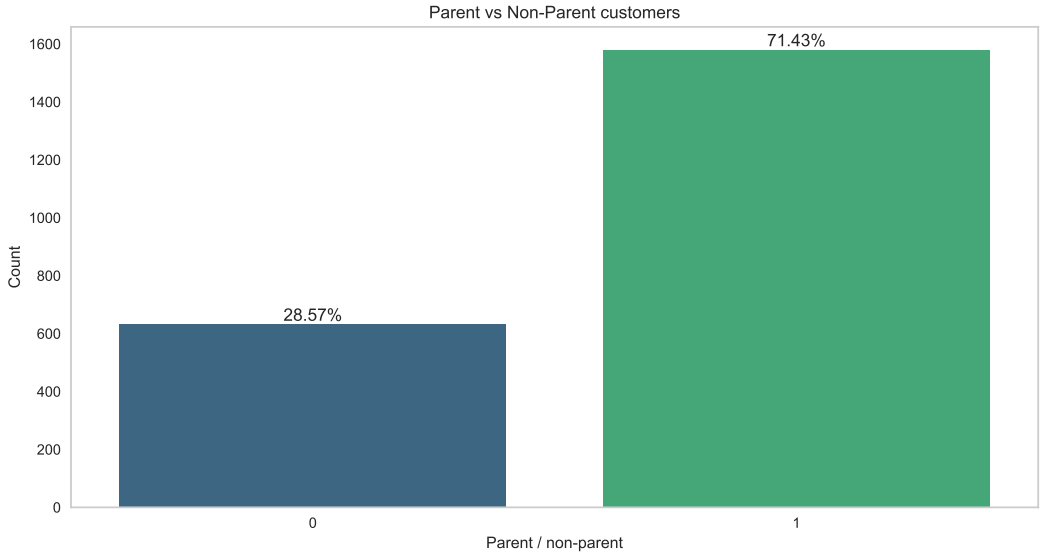

```
plt.figure(figsize=(12, 6))
total = float(len(data['Education']))
ax = sns.countplot(data=data, x='Education', palette='viridis')

for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x() + p.get_width() / 2., height + 0.3, '{:.2f}%'.format((height/total) * 100),
            ha='center', va='bottom')

plt.title('Count of Customers by Education Level')
plt.xlabel('Education Level')
plt.ylabel('Count')
plt.grid(axis='y')
plt.show()
```




```
plt.figure(figsize=(12, 6))
total = float(len(data['Family_Size']))
ax = sns.countplot(data=data, x='Family_Size', palette='viridis')

for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x() + p.get_width() / 2., height + 0.3, '{:.2f}%'.format((height/total) * 100),
            ha='center', va='bottom')

plt.title('Count of Customers by Family Size')
plt.xlabel('Family Size')
plt.ylabel('Count')
plt.grid(axis='y')
plt.show()
```



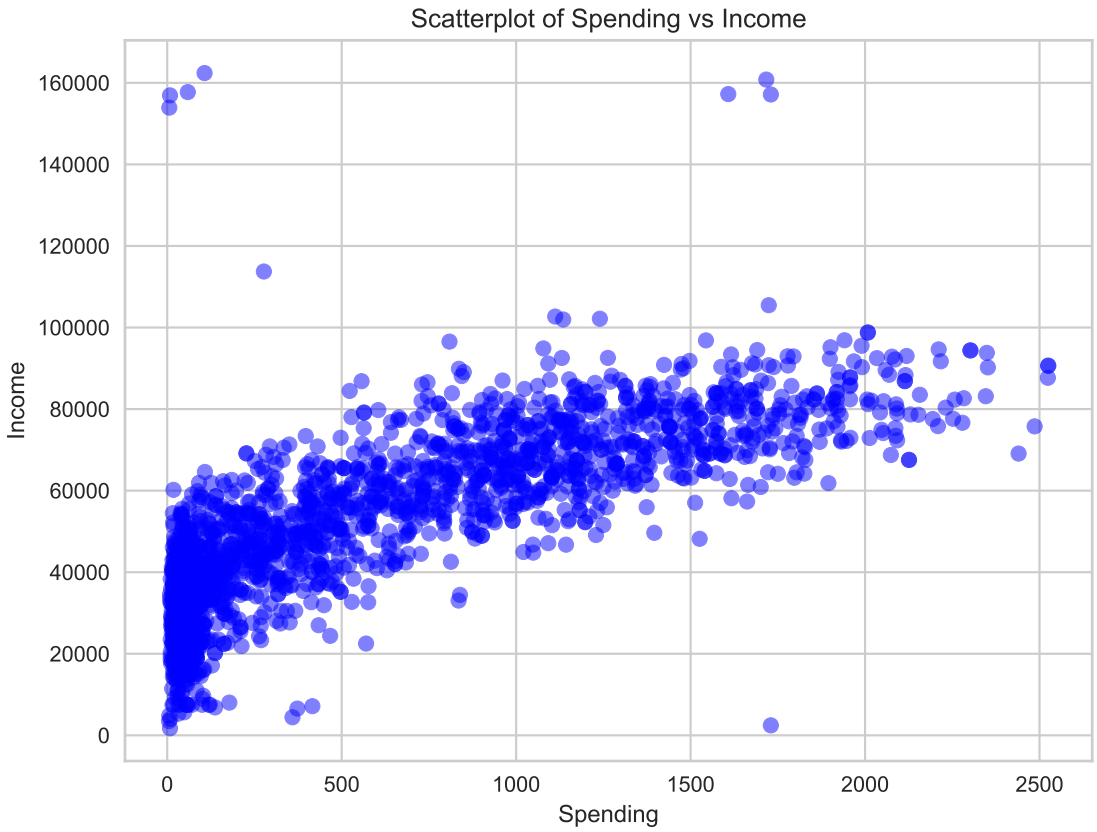

```
plt.figure(figsize=(12, 6))
total = float(len(data['Is_Parent']))
ax = sns.countplot(data=data, x='Is_Parent', palette='viridis')

for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x() + p.get_width() / 2., height + 0.3, '{:.2f}%'.format((height/total) * 100),
            ha='center', va='bottom')

plt.title('Parent vs Non-Parent customers')
plt.xlabel('Parent / non-parent')
plt.ylabel('Count')
plt.grid(axis='y')
plt.show()
```



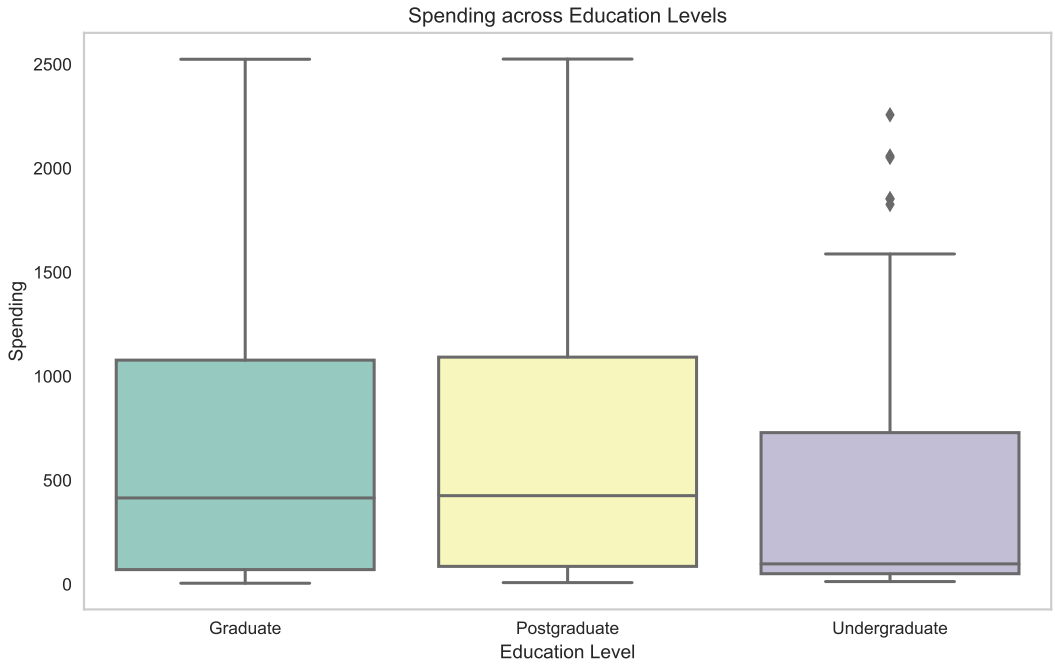
2. Bivariate analysis

This involves exploring the relationship between two variables in a dataset. It helps in understanding correlations, dependencies, or associations between pairs of variables.

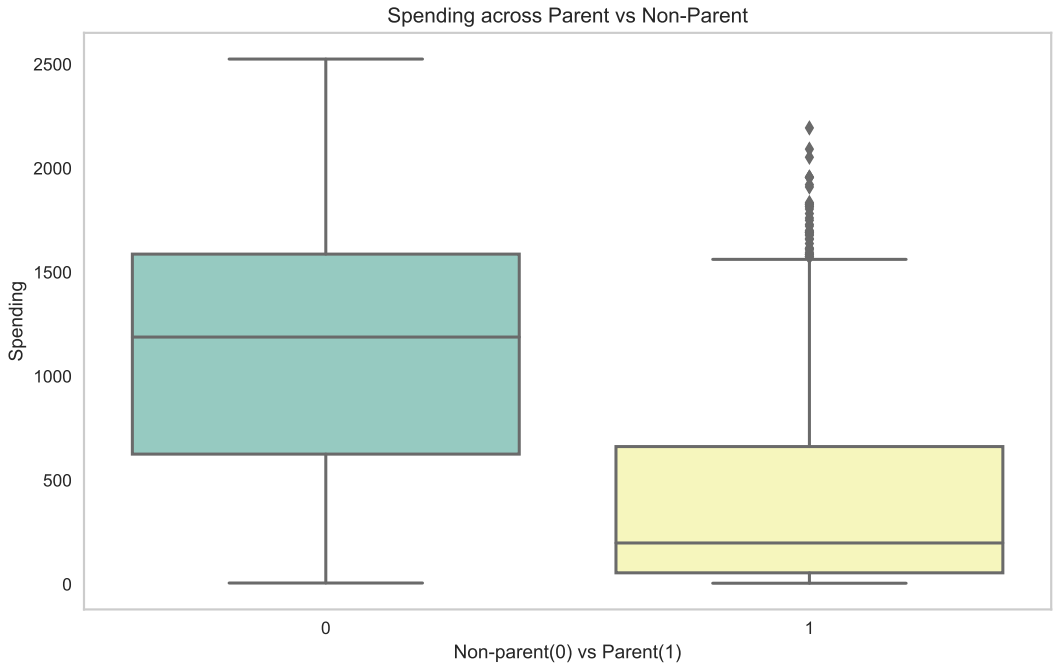
```
plt.figure(figsize=(8, 6))
plt.scatter(data['Spending'], data['Income'], alpha=0.5, color='blue')
plt.title('Scatterplot of Spending vs Income')
plt.xlabel('Spending')
plt.ylabel('Income')
plt.grid(True)
plt.show()
```



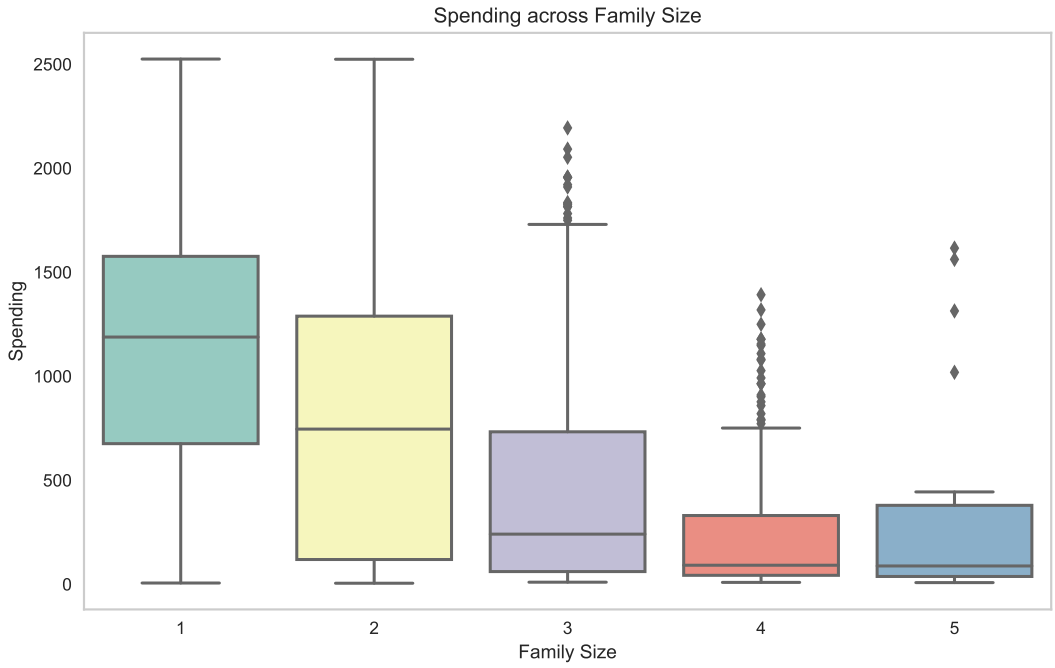

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=data, x='Education', y='Spending', palette='Set3')
plt.title('Spending across Education Levels')
plt.xlabel('Education Level')
plt.ylabel('Spending')
plt.grid(axis='y')
plt.show()
```




```
plt.figure(figsize=(10, 6))
sns.boxplot(data=data, x='Is_Parent', y='Spending', palette='Set3')
plt.title('Spending across Parent vs Non-Parent')
plt.xlabel('Non-parent(0) vs Parent(1)')
plt.ylabel('Spending')
plt.grid(axis='y')
plt.show()
```




```
plt.figure(figsize=(10, 6))
sns.boxplot(data=data, x='Family_Size', y='Spending', palette='Set3')
plt.title('Spending across Family Size')
plt.xlabel('Family Size')
plt.ylabel('Spending')
plt.grid(axis='y')
plt.show()
```



VII. Data Preprocessing

In this stage, I'll prepare the data for clustering operations through several preprocessing steps:

- Encoding categorical features using label encoding.
- Scaling the features utilizing the standard scaler method to ensure uniformity in their scales.
- Creating a subset dataframe to reduce dimensionality, focusing on specific features.

1. Encoding categorical features

```
cat_cols = data.select_dtypes(include=['object']).columns.tolist()

label_encoder = LabelEncoder()

# Iterate through each categorical column and encode it
for col in cat_cols:
    if col in data.columns:
        data[col] = label_encoder.fit_transform(data[col])

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

```
## All features are now numerical
```

2. Scaling features

```
#Make copy of "data" to "df"
df = data.copy()

cols_del = ['AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1','AcceptedCmp2', 'Complain',

df = df.drop(cols_del, axis = 1)

#Scaling
scaler = StandardScaler()
scaler.fit(df)
```

```
## StandardScaler()
```

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

```
## All features are now scaled
```

```
print("The scaled DataFrame prepared for additional modeling tasks:")
```

```
## The scaled DataFrame prepared for additional modeling tasks:
```

```
print("")
```

```
scaled_data.head()
```

```
##    Education    Income   Kidhome  ...  Children  Family_Size  Is_Parent
## 0  -0.893586  0.287105 -0.822754  ... -1.264598    -1.758359  -1.581139
## 1  -0.893586 -0.260882  1.040021  ...  1.404572     0.449070   0.632456
## 2  -0.893586  0.913196 -0.822754  ... -1.264598    -0.654644  -1.581139
## 3  -0.893586 -1.176114  1.040021  ...  0.069987     0.449070   0.632456
## 4   0.571657  0.294307  1.040021  ...  0.069987     0.449070   0.632456
##
## [5 rows x 23 columns]
```

VIII. Dimensionality Reduction

In this scenario, clustering relies on multiple factors termed as attributes or features. Managing a high number of features can be challenging, especially when some are correlated and redundant. To address this, I'll conduct dimensionality reduction on these selected features before initiating the clustering process.

Dimensionality reduction involves streamlining the number of considered variables, obtaining a concise set of essential variables while retaining most of the information.

Principal Component Analysis (PCA) serves as a method to achieve this reduction, aiming to improve interpretability while minimizing information loss. The steps in this phase encompass:

- Implementing PCA for dimensionality reduction.

- The number of dimensions will be determined by the retained variance, crucial for enhancing clustering accuracy.
- Visualizing the reduced dataset through plots.

```
pc = PCA(svd_solver='auto')
pc.fit(scaled_data)
```

```
## PCA()
```

```
print('Total no. of principal components =',pc.n_components_)
```

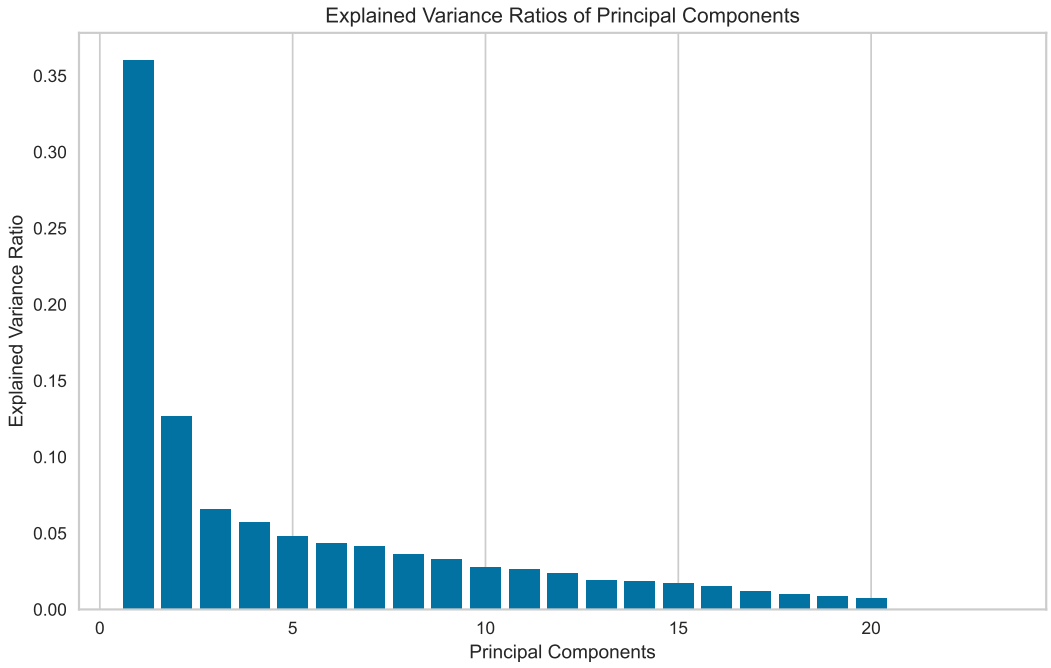
```
## Total no. of principal components = 23
```

We'll investigate the proportion of variance clarified by each principal component. These components will be sorted in descending order based on their respective explained variance ratios. This exploration allows us to understand how much information each principal component retains from the original dataset, aiding in selecting the most informative components for further analysis.

Explained Variance Ratios of PCs

```
var = pc.explained_variance_ratio_
print(var)
```

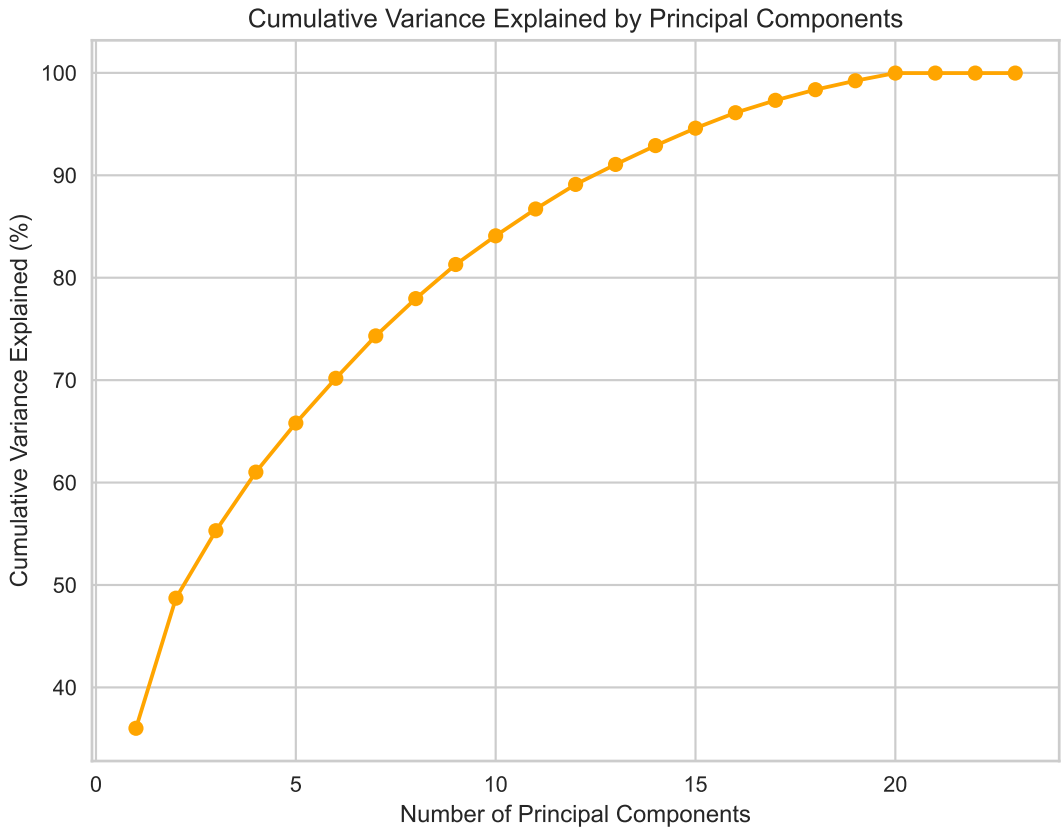
```
plt.figure(figsize=(10, 6))
plt.bar(range(1, len(var) + 1), var)
plt.title('Explained Variance Ratios of Principal Components')
plt.xlabel('Principal Components')
plt.ylabel('Explained Variance Ratio')
plt.grid(axis='y')
plt.show()
```



The first principal component explains roughly 36% of the variance, while the second principal component accounts for approximately 14%. By summing these, we can determine the cumulative variance explained by these components. To simplify comprehension, we're converting these values into percentages for easier observation.

Cumulative Variance explained by PCs

```
cum_var = np.cumsum(np.round(pc.explained_variance_ratio_, decimals=4)*100)

plt.figure(figsize=(8, 6))
plt.plot(range(1, len(cum_var) + 1), cum_var, marker='o', linestyle='-', color='orange')
plt.title('Cumulative Variance Explained by Principal Components')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Variance Explained (%)')
plt.grid(True)
plt.show()
```

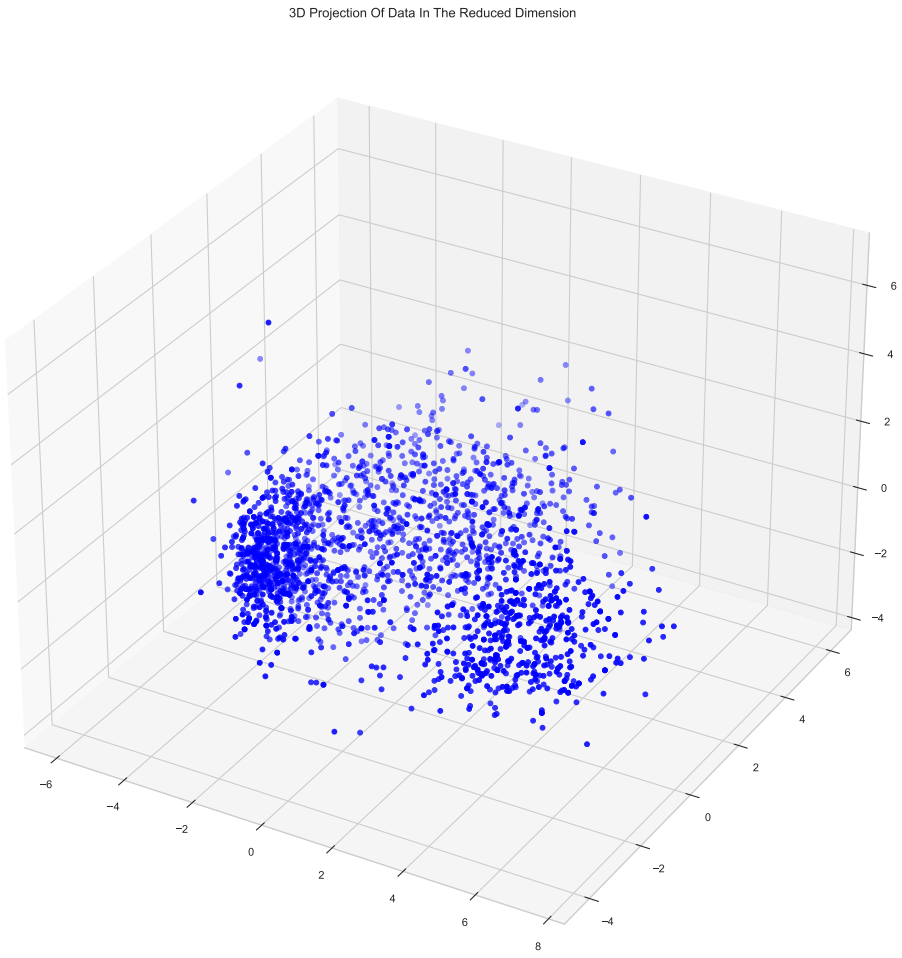


The cumulative variance plot indicates the 90% threshold is getting crossed at PC = 13, which is quite complicated in many aspects. For the sake of simplicity, we will retain PC = 3 and proceed to other clustering algorithms.

```
pca = PCA(n_components=3)
pca.fit(scaled_data)
```

```
PCA_ds = pd.DataFrame(pca.transform(scaled_data), columns=(["PC1","PC2", "PC3"]))
PCA_ds.describe()
```

```
x =PCA_ds["PC1"]
y =PCA_ds["PC2"]
z =PCA_ds["PC3"]
#To plot
fig = plt.figure(figsize=(16,16))
ax = fig.add_subplot(111, projection="3d")
ax.scatter(x,y,z, c="blue", marker="o" )
ax.set_title("3D Projection Of Data In The Reduced Dimension")
plt.show()
```



IX. Clustering

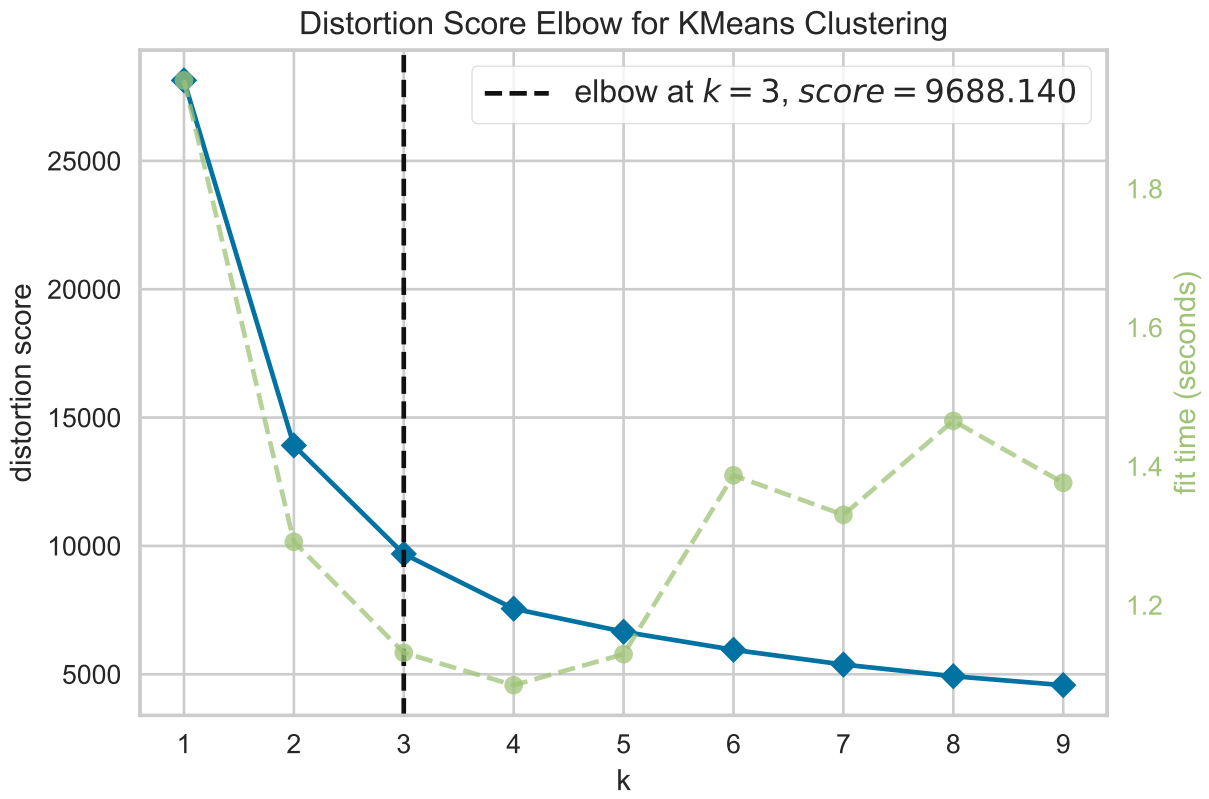
1. K-Means Clustering

K-Means clustering stands as one of the most commonly employed clustering methods. It operates by measuring the Euclidean distance between clusters during each iteration to assign data points to specific clusters. Determining the appropriate number of clusters often involves employing various methods, and one prevalent approach is the ***Elbow Curve***.

The Elbow Curve illustrates a graphical representation where the 'knee'-like bend signifies a potential optimal number of clusters for the K-Means algorithm.

```
m = KMeans(random_state=123)
v = KElbowVisualizer(m, k=(1, 10))

# Fit the data to the visualizer
v.fit(PCA_ds)
```

```
# Finalize and display the Elbow Curve
v.poof()
```

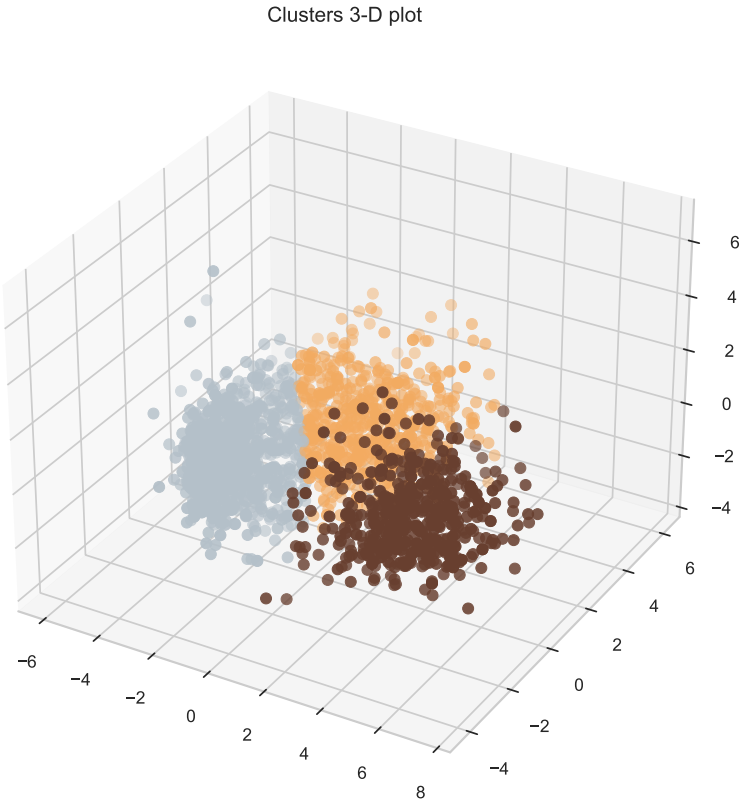


The above plot indicates that 3 will be an optimal number of clusters for this data. Next, we will be fitting the K-Means Clustering Algorithm to get the final clusters.

```
kmeans = KMeans(n_clusters=3, random_state=4234)
yhat_kmeans = kmeans.fit_predict(PCA_ds)

# Add the 'Clusters' column to the PCA dataset
PCA_ds['Clusters'] = yhat_kmeans

PCA_ds.head()
```

```
##         PC1       PC2       PC3  Clusters
## 0  4.986336 -0.161545  2.444084         0
## 1 -2.874168  0.022724 -1.529308         1
## 2  2.615763 -0.731407 -0.265223         0
## 3 -2.654568 -1.455868 -0.398179         1
## 4 -0.656015  0.177807 -0.142205         1
```

```
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("Clusters 3-D plot")
plt.show()
```



2. Hierarchical Clustering

In hierarchical clustering, two primary methods exist: **Divisive** and **Agglomerative**. Divisive, also known as top-down clustering, starts with all observations grouped into a single cluster. This cluster is successively divided into smaller clusters based on their dissimilarity until each observation forms its own cluster.

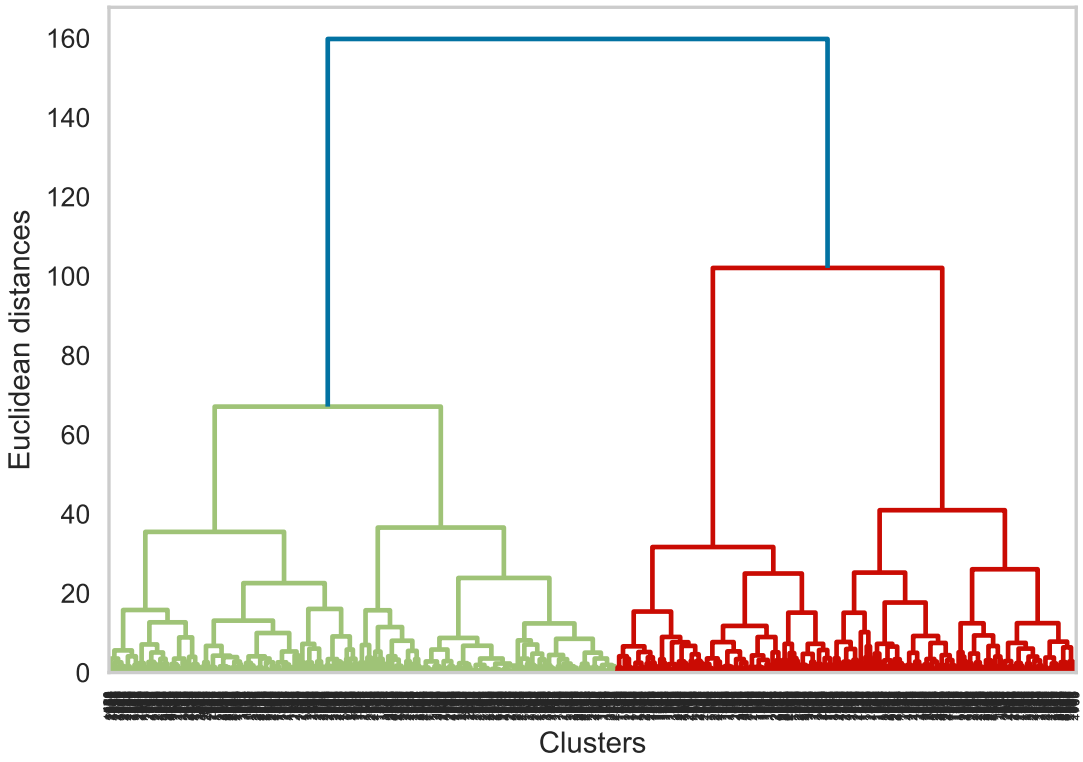
Contrarily, Agglomerative clustering, also called bottom-up clustering, begins with each observation as a separate cluster. Similar clusters are progressively merged together based on their similarity until reaching a single cluster containing all observations.

In the Agglomerative method, determining the optimal number of clusters often involves inspecting a dendrogram, a tree-like diagram illustrating the merging process and distances between clusters. The ideal number of clusters can be inferred from the **dendrogram**.

- **Dendrogram Plotting using Ward's method**

```
#Previously we added a column to PCA_ds called "clusters"
#We should retain and delete it before passing to other clustering algos

km_clusters = PCA_ds['Clusters']
clus_del = ['Clusters']

PCA_ds = PCA_ds.drop(clus_del, axis = 1)
```

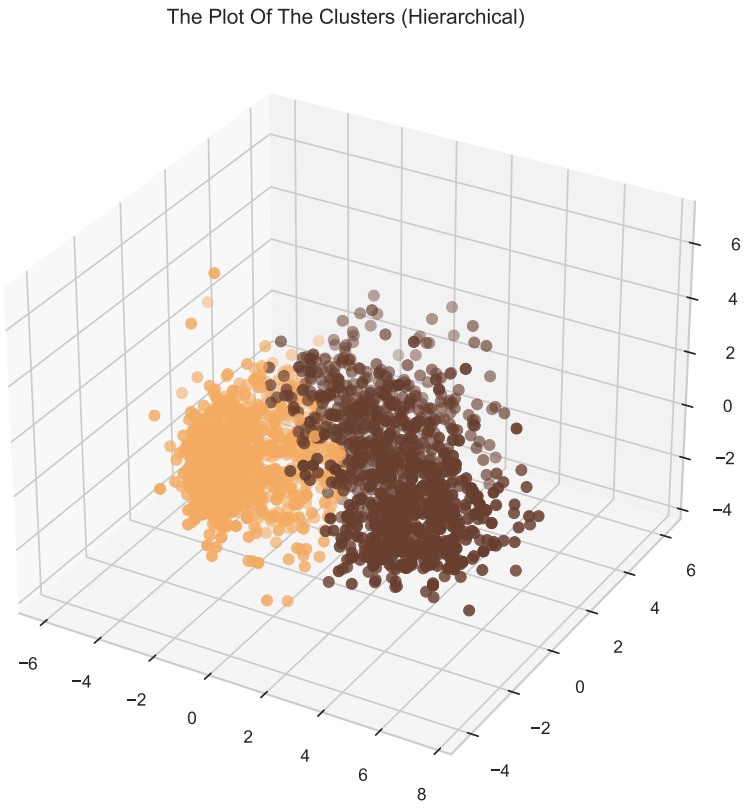
```
plt.rcParams['axes.facecolor'] = 'white'
plt.rcParams['axes.grid'] = False
dendrogram = sch.dendrogram(sch.linkage(PCA_ds, method='ward'))
plt.xlabel('Clusters')
plt.ylabel('Euclidean distances')
plt.rcParams['axes.facecolor'] = 'white'
plt.rcParams['axes.grid'] = False
plt.show()
```



We can see 2 prominent clusters here (green, red)

```
AC = AgglomerativeClustering(n_clusters=2)
# fit model and predict clusters
yhat_AC = AC.fit_predict(PCA_ds)
PCA_ds["H_clus"] = yhat_AC
#Adding the Clusters feature to the orignal dataframe.
data["H_clus"]= yhat_AC
```

```
fig = plt.figure(figsize=(10,8))
ax = plt.subplot(111, projection='3d', label="bla")
ax.scatter(x, y, z, s=40, c=PCA_ds["H_clus"], marker='o', cmap = cmap )
ax.set_title("The Plot Of The Clusters (Hierarchical)")
plt.show()
```



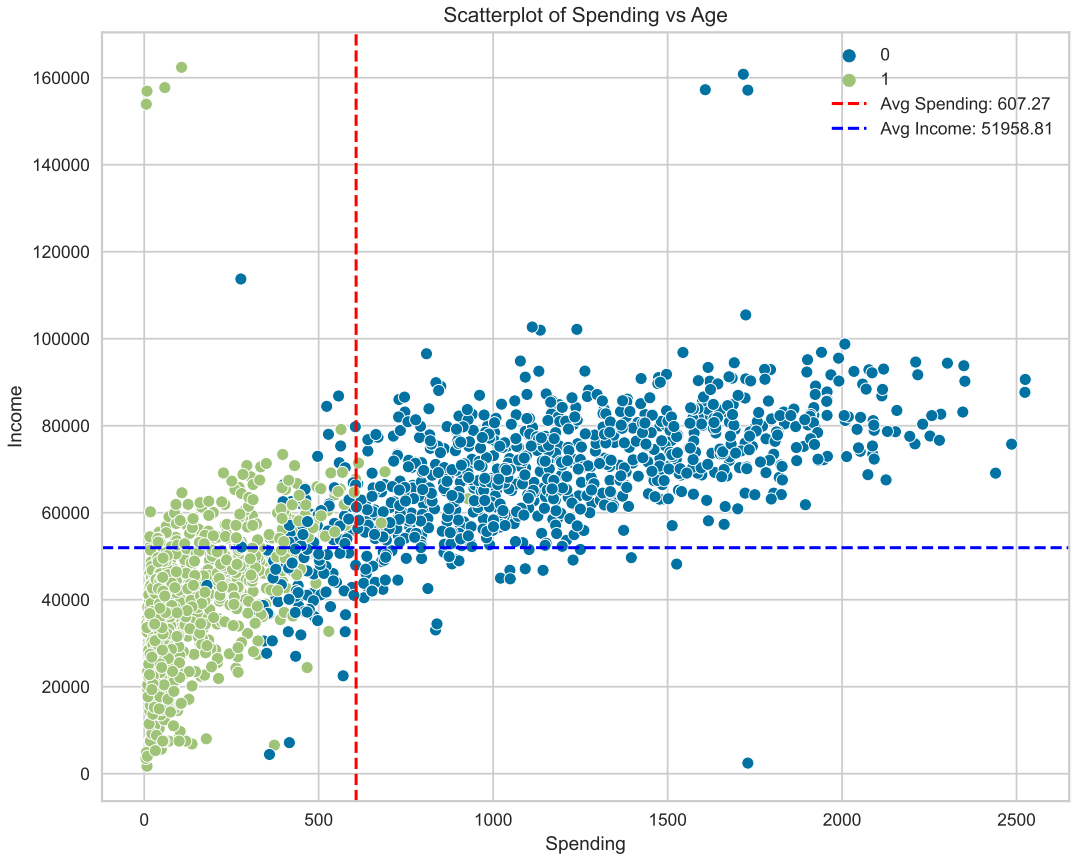
From the results above, we showed that the **K-Means** algorithm reveals **three distinct clusters** based on centroid positions, while **Agglomerative Clustering** identifies **two clusters** by progressively merging similar points.

X. Evaluation

For evaluation, and for the sake of simplicity, we will choose only 2 clusters based on Agglomerative Clustering algorithm results.

```
plt.figure(figsize=(10, 8))
sns.scatterplot(data = data,x=data["Spending"], y=data["Income"],hue=data["H_clus"])
plt.title('Scatterplot of Spending vs Age')
plt.xlabel('Spending')
plt.ylabel('Income')
```

```
plt.grid(True)

avg_income = data["Income"].mean()
avg_spending = data["Spending"].mean()

plt.axvline(x=avg_spending, color='red', linestyle='--', label=f'Avg Spending: {avg_spending:.2f}')
plt.axhline(y=avg_income, color='blue', linestyle='--', label=f'Avg Income: {avg_income:.2f}')
plt.legend()

plt.show()
```

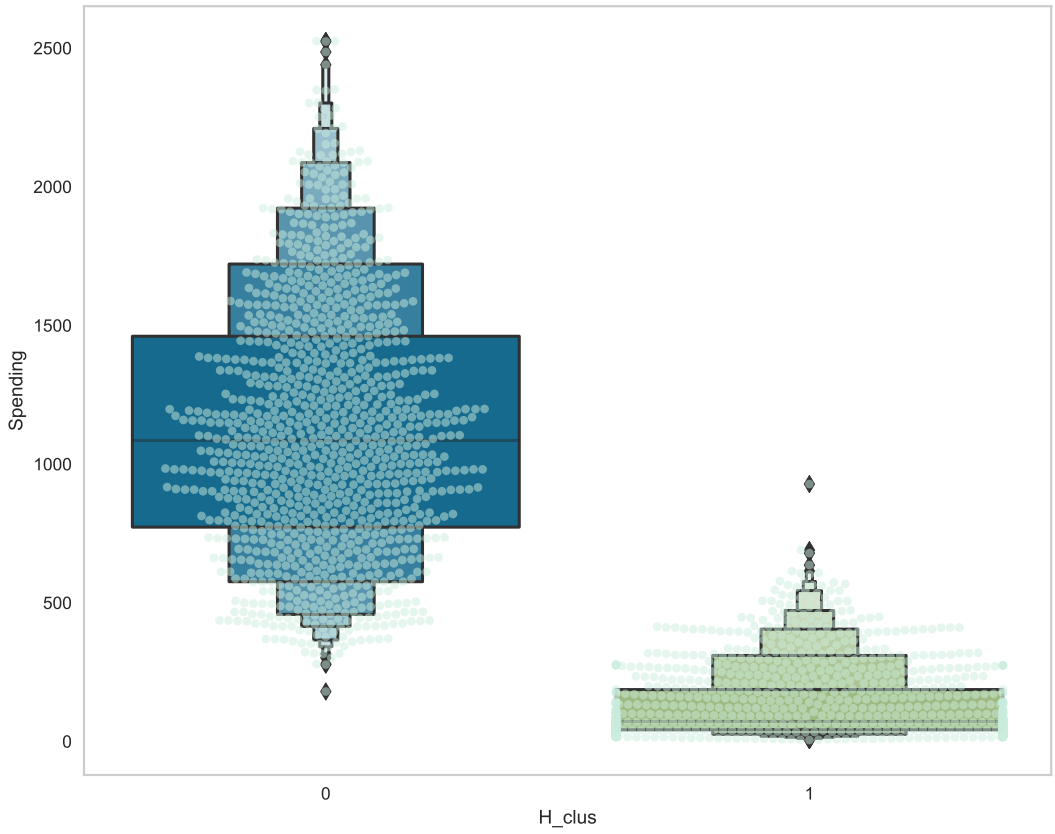


We can see from the plot, that the clusters are divised into two distinct groups :

- 1st group (green) : relatively has spending and income bellow their average respectively. (Low spending low income)
- 2nd group (blue) : relatively has Spending and Income above their average respectively. (high spending high income)

I'll delve into the specific breakdown of clusters concerning product preferences within the dataset. I'll analyze the distribution across various product categories, such as Wines, Fruits, Meat, Fish, Sweets, and Gold, to gain detailed insights into cluster preferences.

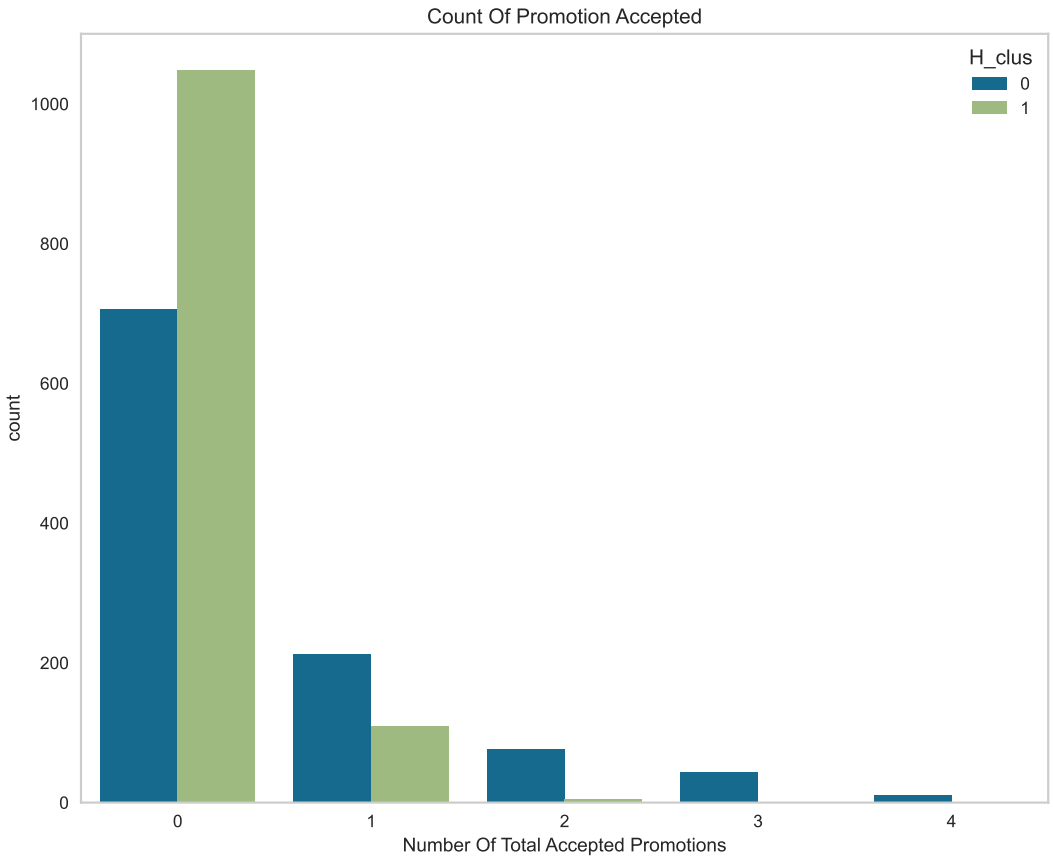
```
plt.figure(figsize = (10,8))
pl=sns.swarmplot(x=data["H_clus"], y=data["Spending"], color= "#CBEDDD", alpha=0.5 )
pl=sns.boxenplot(x=data["H_clus"], y=data["Spending"])
plt.show()
```



From the above plot, it can be clearly seen that cluster 0 is our biggest set of customers closely followed by cluster 1. We can explore what each cluster is spending on for the targeted marketing strategies.

Let us next explore how did our promotions campaign do in the past.

```
data["Total_Promos"] = data["AcceptedCmp1"]+ data["AcceptedCmp2"]+ data["AcceptedCmp3"]+ data["Accepted

#Plotting count of total campaign accepted.
plt.figure(figsize = (10,8))
pl = sns.countplot(x=data["Total_Promos"],hue=data["H_clus"])
pl.set_title("Count Of Promotion Accepted")
pl.set_xlabel("Number Of Total Accepted Promotions")
plt.show()
```

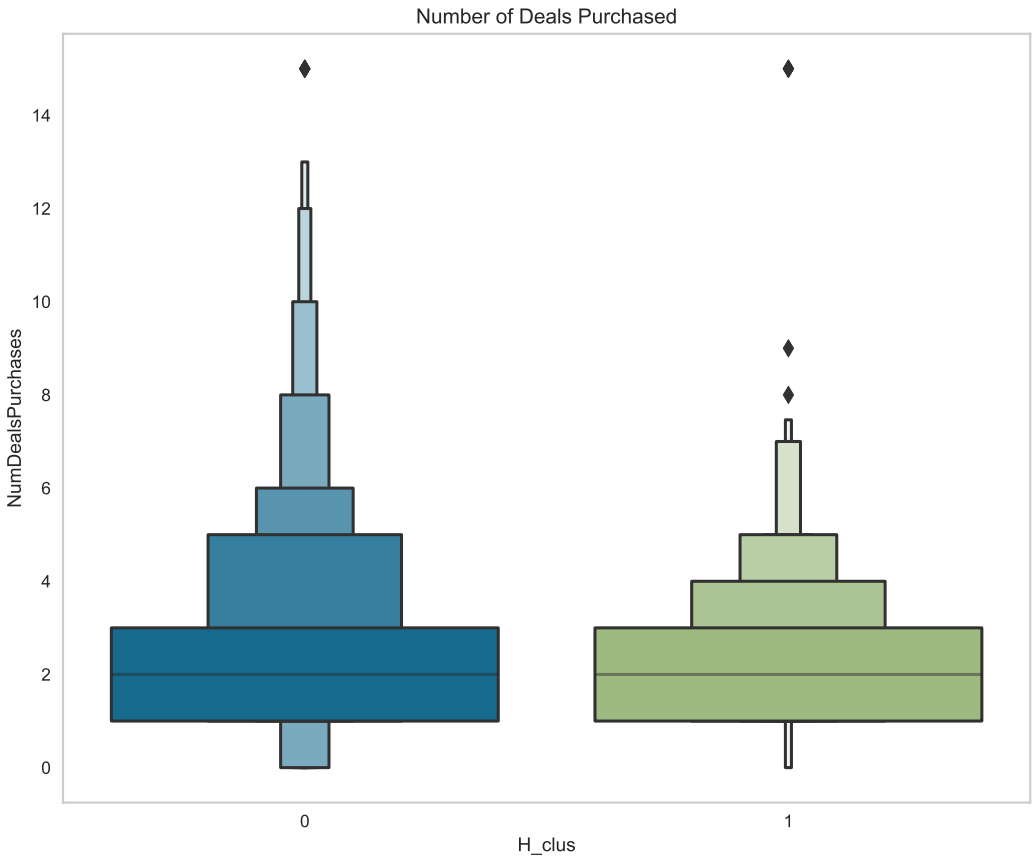


The analysis reveals a general declining trend in the count of accepted promotions as the number of promotions increases. However, Cluster 0 stands out by actively engaging in all promotion types, showcasing a consistent and noteworthy response to various promotional campaigns compared to other clusters.

It is suggested that marketing campaigns should maintain cluster 0 and encourage cluster 1 in order to increase sales.

Let's see how these clusters do with the number of deals purchased :

```
plt.figure(figsize = (10, 8))
pl=sns.boxenplot(y=data["NumDealsPurchases"],x=data["H_clus"])
pl.set_title("Number of Deals Purchased")
plt.show()
```



Clusters 0 and 1 showcase a relatively comparable distribution in terms of the number of deals purchased. This similarity suggests that these clusters share analogous purchasing patterns concerning the acquisition of deals. Despite potentially belonging to distinct segments, they demonstrate a resemblance in their engagement with deals, indicating overlapping behaviors in deal-based purchasing activities.

XI. Profiling

With the established clusters and insights into their purchasing behaviors, it's time to profile the individuals within these clusters. This profiling will aid in identifying our star customers and those who require more attention from the retail store's marketing team.

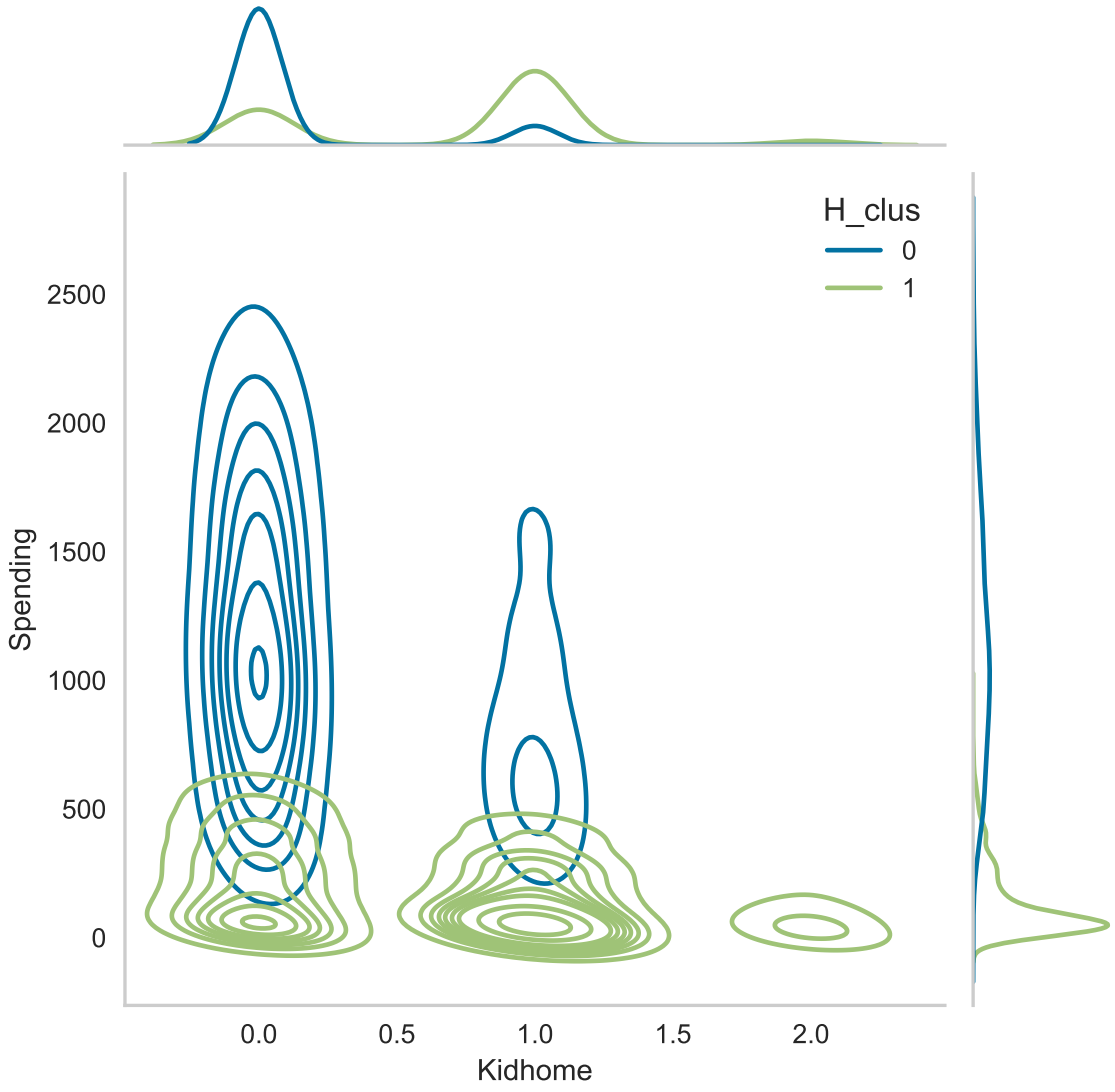
To achieve this, I'll visualize key customer attributes relative to their respective clusters. By analyzing these outcomes, I'll draw conclusions regarding the prominence of specific customer segments.

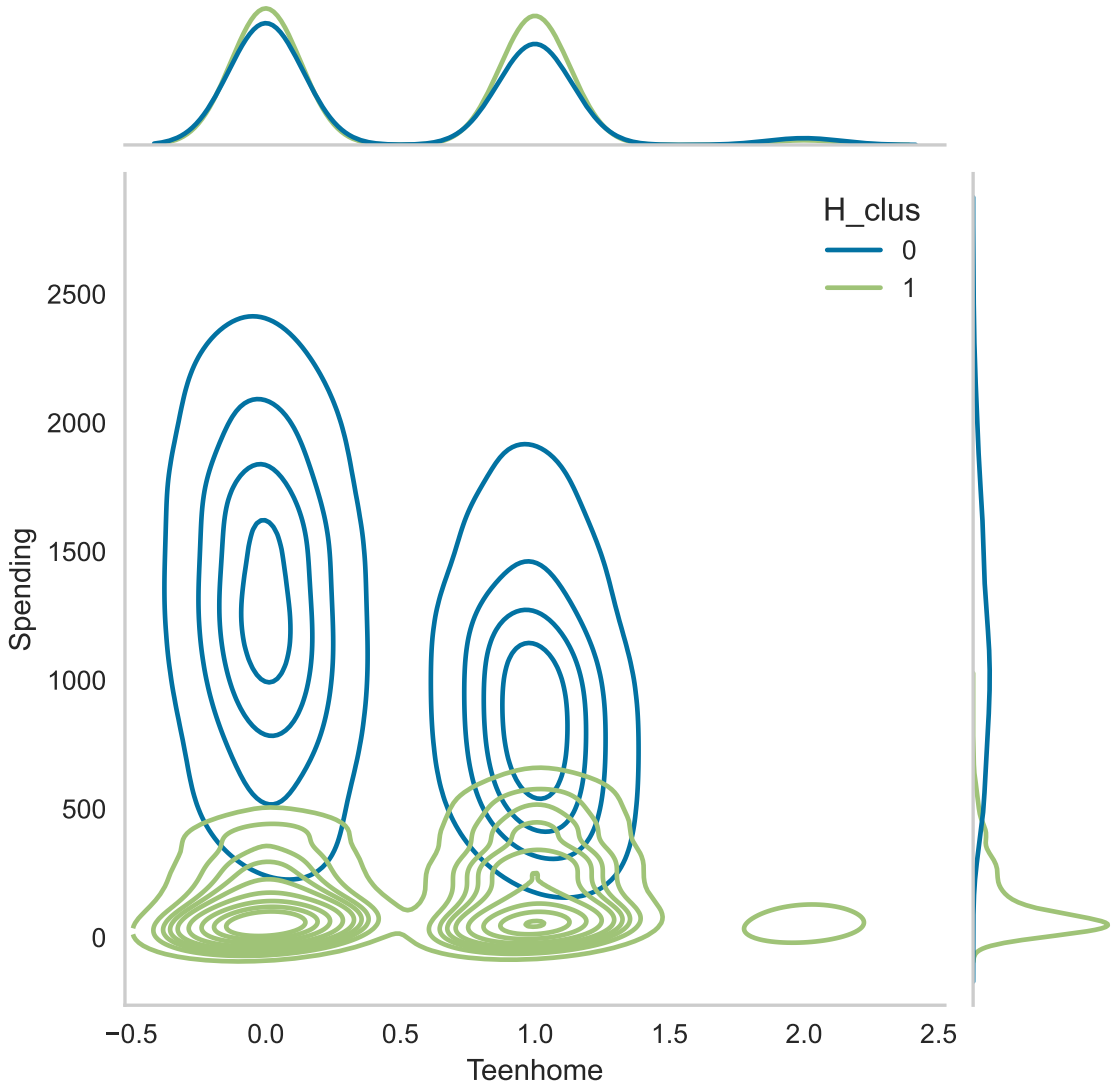
Other methods used in Profiling are :

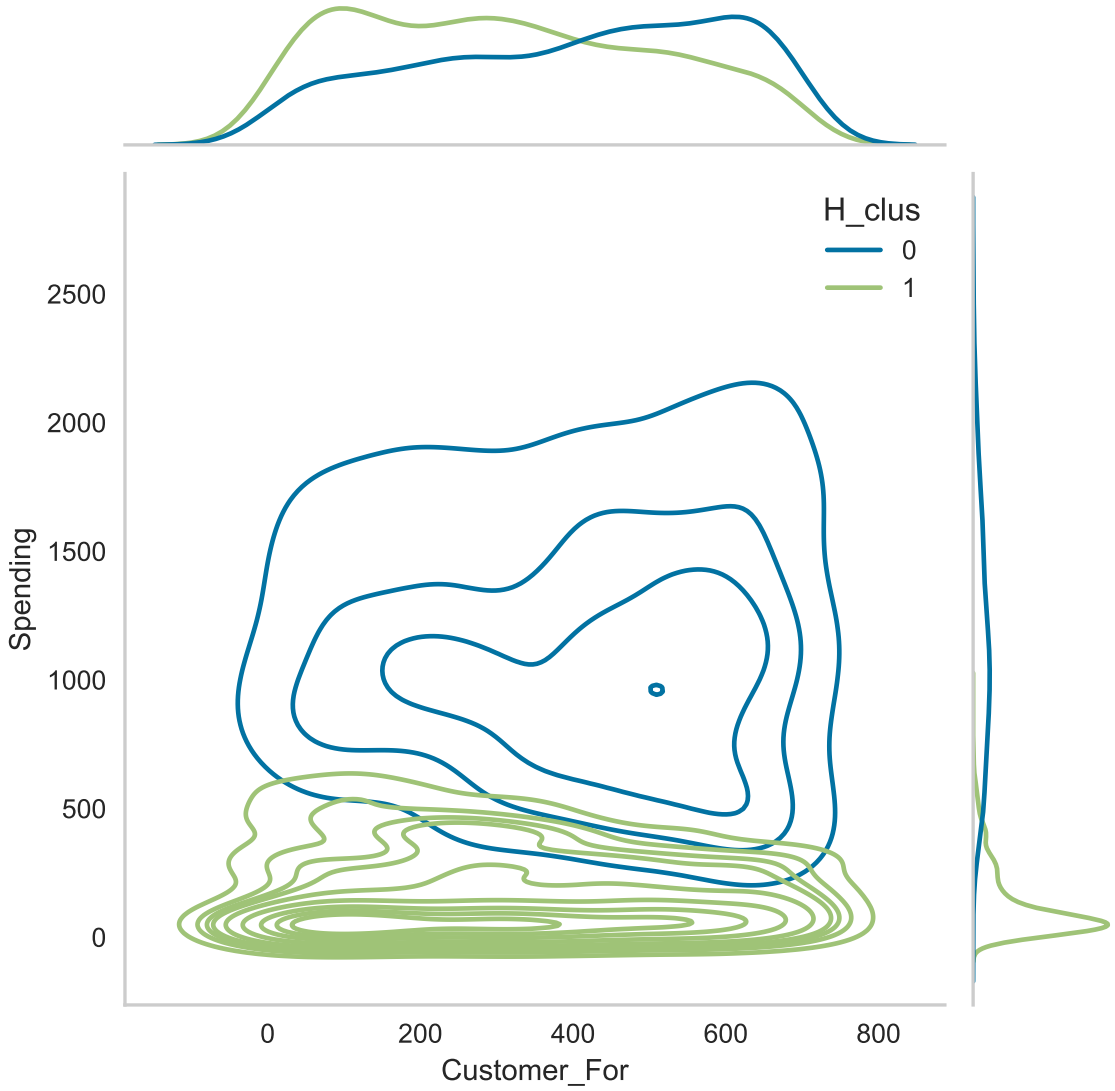
- **ANOVA** : Used to compare means between clusters for numerical features. This helps identify statistically significant differences between clusters regarding specific attributes.
- **Chi-square test** : Used to assess the association between categorical variables and clusters. It determines whether there is a significant relationship between categorical variables and cluster membership.

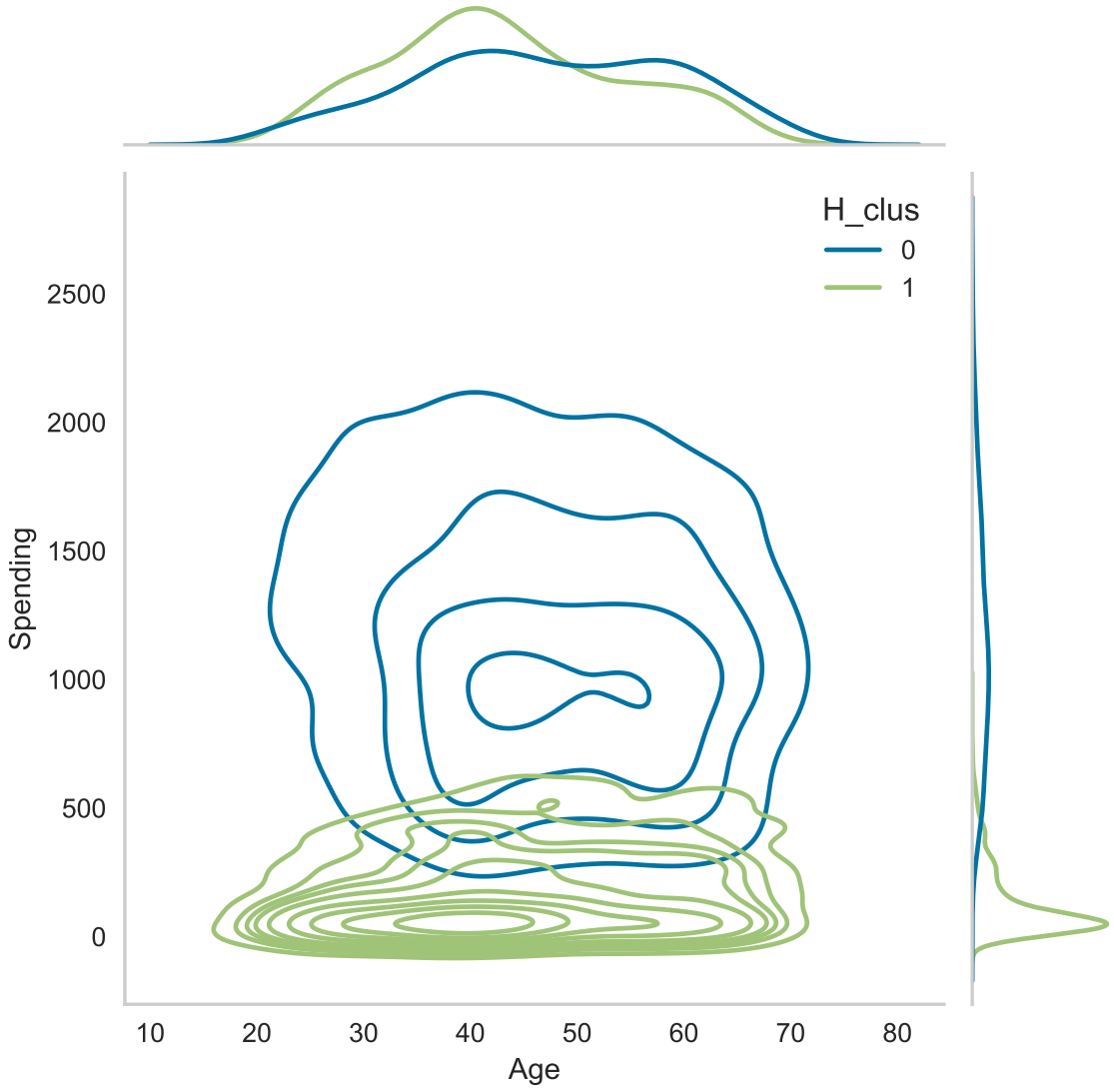
in this project, we will not dive into these statistical methods.

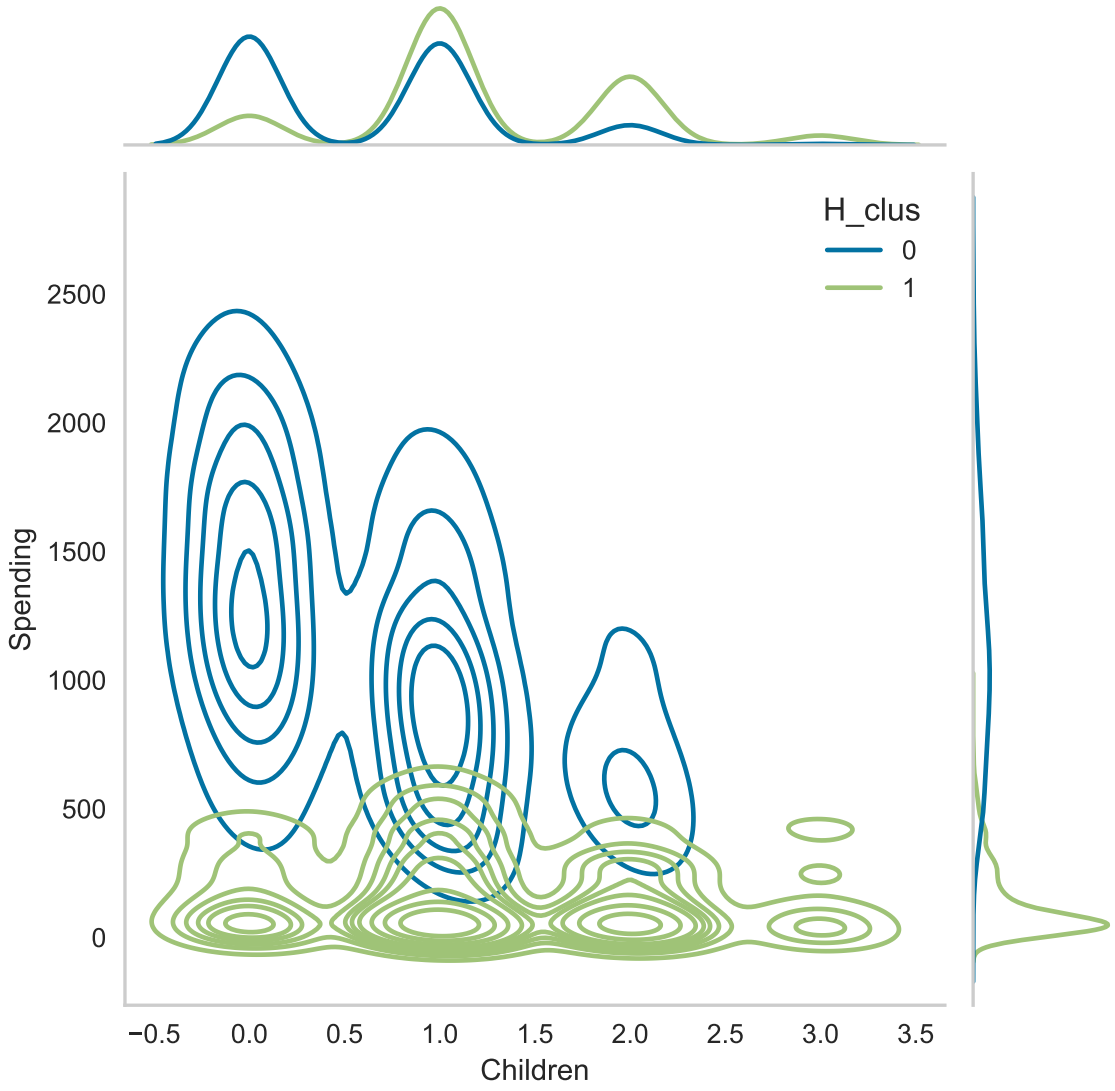
```
Personal = [ "Kidhome","Teenhome","Customer_For", "Age", "Children", "Family_Size", "Is_Parent", "Educa

for i in Personal:
    plt.figure(figsize = (10,8))
    sns.jointplot(x=data[i], y=data["Spending"], hue =data["H_clus"], kind="kde")
    plt.show()
```

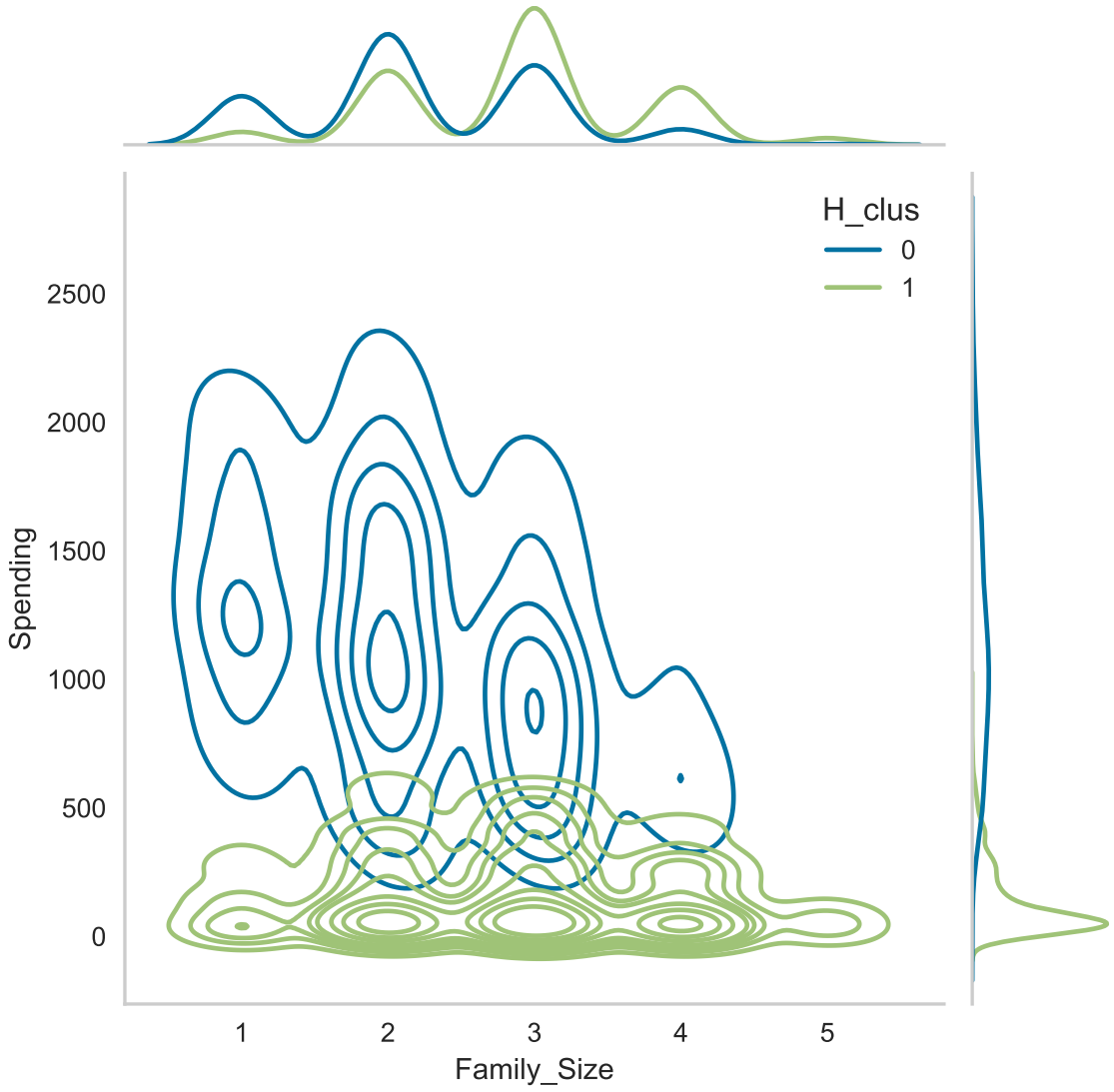


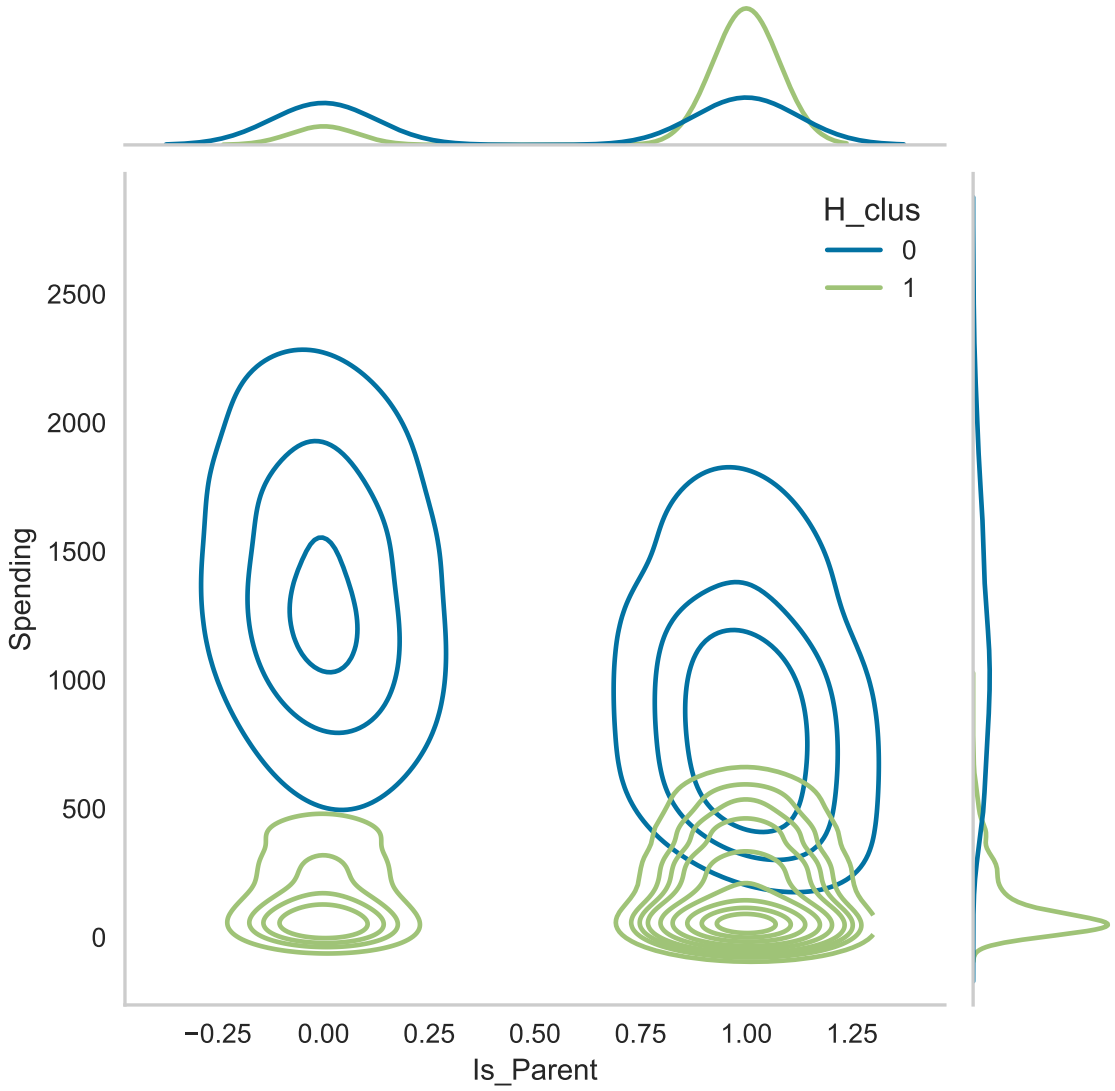

H_clus
0
1
2500
2000
1500
1000
500
0
Spending
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
Teenhome

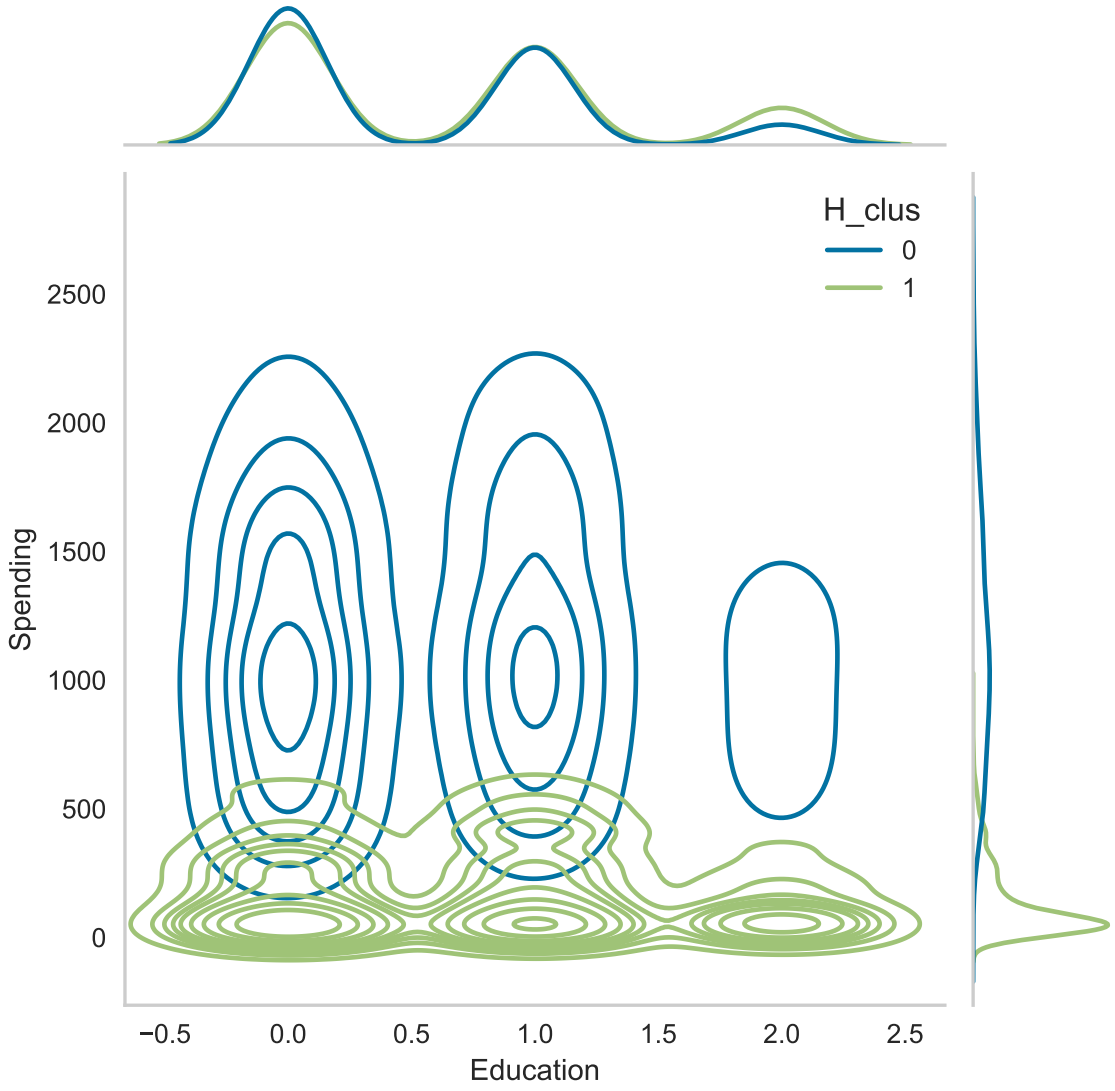

H_clus
0
1
2500
2000
1500
1000
500
0
Spending
0
200
400
600
800
Customer_For

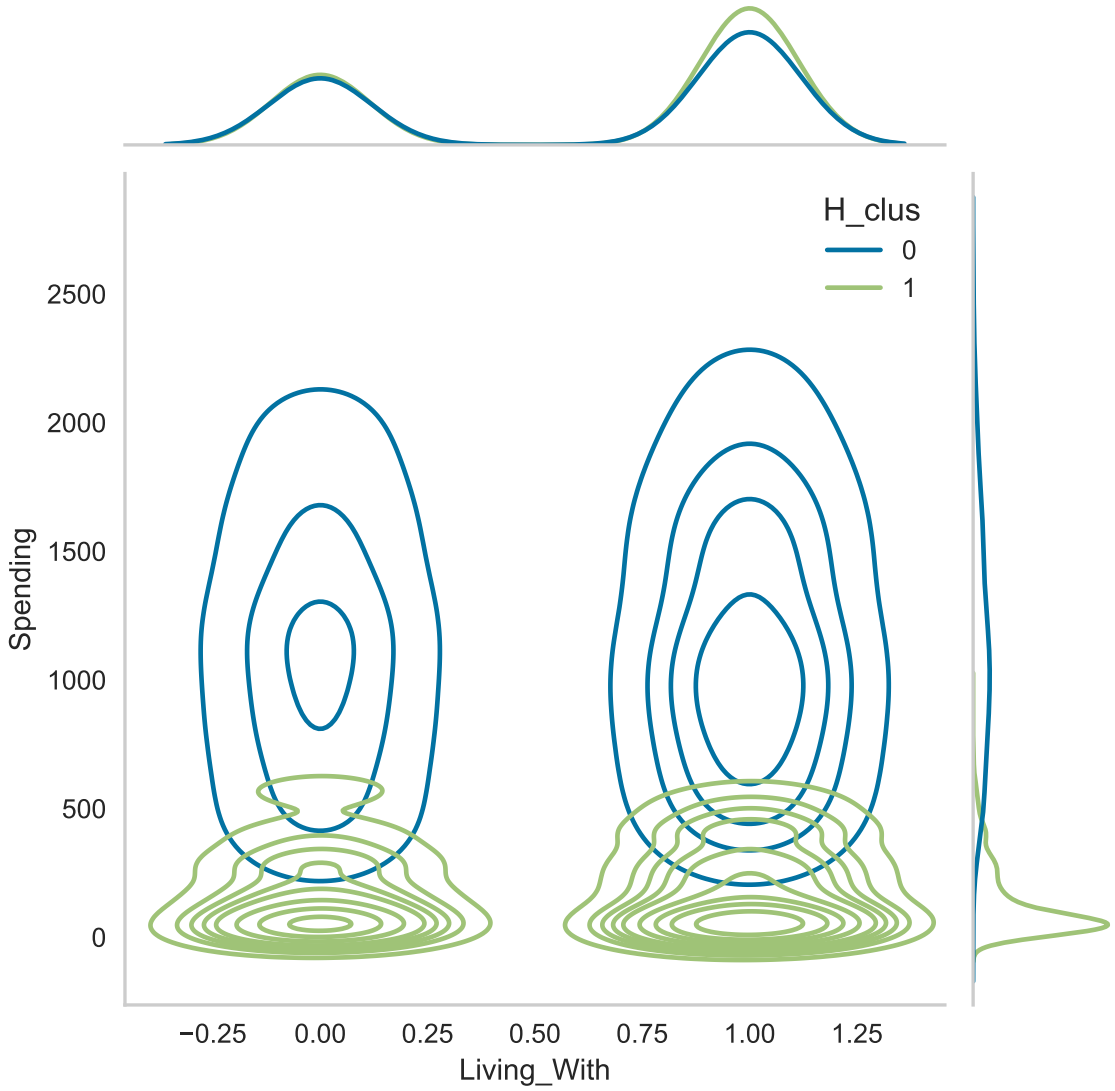

H_clus
0
1
Spending
2500
2000
1500
1000
500
0
10
20
30
40
50
60
70
80
Age


H_clus
0
1
2500
2000
1500
1000
500
0
Spending
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Children


H_clus
0
1
2500
2000
1500
1000
500
0
Spending
1
2
3
4
5
Family_Size


H_clus
0
1
2500
2000
1500
1000
500
0
Spending
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
Is_Parent


H_clus
0
1
2500
2000
1500
1000
500
0
Spending
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
Education


H_clus
0
1
2500
2000
1500
1000
500
0
Spending
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
Living_With

About the clusters 0 and 1

Cluster 0

- At max has 4 members of the family and at least 1. Majority has 2 members of the family
- Have at most 2 kids.
- Most of them has 0 kidhome (most spending) and very few has 1, none has 2 kidhome.
- At most has 1 teenager at home.
- Relatively older.
- Low income

Cluster 1

- Definitely a parent.
- Most has 3 family members.
- Have at most 3 kids.
- Relatively younger.
- Span all ages
- High income

Conclusion

In this project, I applied unsupervised clustering techniques involving dimensionality reduction, K-means, and agglomerative clustering methods. I identified two distinct customer clusters and utilized these clusters to profile customers based on their family compositions, income, and spending behaviors. These insights can significantly enhance the development of more targeted and effective marketing strategies.