Unsupervised Machine Learning Algorithms Final Project

Customer Segmentation using Clustering

Carmen Paz

The purpose of this project is to perform unsupervised clustering of customers' data for customer segmentation. The customers are divided into distinct groups or clusters based on their shared characteristics so that the company can better facilitate targeted campaigns.
The data set customer-personality-analysis contains various information about customer demographics and past purchases.

1. Main Objective of the Analysis

The main objective of this analysis is to segment customers based on their purchasing behaviors and demographic features using clustering techniques. The goal is to create meaningful customer segments that can be targeted more effectively with personalized marketing strategies. By grouping similar customers together, businesses can improve customer retention, optimize marketing spend, and provide more relevant offers to each group. The model will focus on clustering and dimensionality reduction to simplify complex datasets and derive actionable insights for the business.

2. Brief Description of the Dataset

The dataset used for this analysis comes from an e-commerce company that tracks various attributes of their customers. It contains information such as customer demographics (e.g., age, income, marital status) and purchasing behaviors (e.g., frequency of purchases, types of products purchased).
Below is a summary of some key attributes:
Age: The age of the customer.
Income: The income level of the customer.
Marital_Status: The marital status of the customer.
Kidhome: Number of children at home.
Teenhome: Number of teenagers at home.
Recency: The recency of the customer’s last purchase.
MntWines, MntFruits, etc.: Money spent on specific product categories.

The analysis aims to identify customer segments that can be targeted for different marketing strategies.
The data will be used to group customers based on their similarities, with the ultimate goal of driving business value.

3. Data Exploration and Cleaning

The dataset underwent initial exploration and cleaning to ensure that it was ready for analysis.
The steps taken included:
Handling Missing Values: Missing values were either imputed with the median or mean (for numerical variables) or dropped (if the percentage of missing data was too high).
Outlier Detection: Outliers were detected using Z-scores or Interquartile Range (IQR), and extreme values were either capped or removed based on their impact on the analysis.
Feature Engineering: Categorical variables such as marital status and education level were encoded using one-hot encoding or label encoding, depending on the feature's nature.
Normalization: To ensure optimal performance from the K-Means algorithm, all numerical features were scaled using standardization or Min-Max scaling.
The dataset was thoroughly cleaned and preprocessed, ensuring that it was ready for modeling.

4. Training Unsupervised Model

Training the K-Means Clustering Model K-Means Clustering
Model Setup: We applied K-Means clustering with the goal of dividing the customers into three distinct groups based on their demographic and purchasing behavior features.
Parameters:
n_clusters=3: We selected 3 clusters based on business requirements and exploratory analysis.
n_init=100: The algorithm was run 100 times with different initial centroids to ensure the solution was stable and less dependent on the initial starting points.
random_state=10: This parameter was set to ensure reproducibility of the results.
Model Training: The K-Means algorithm was trained using the scaled dataset, and it identified the clusters based on the customer features.
Evaluation and Results:
Inertia: The inertia (sum of squared distances of samples to their closest cluster center) was used to evaluate the model. A lower inertia indicates better clustering.
Silhouette Score: The silhouette score was computed to assess the quality of the clusters. A higher silhouette score suggests well-defined clusters.
The K-Means model successfully identified 3 customer segments that are distinct and represent different customer behavior patterns.

5. Recommended Final Model

After training the K-Means model, it was clear that the model provided meaningful and interpretable customer segments. Given its simplicity, scalability, and the clear distinctions between clusters, K-Means was chosen as the final model. The results were easy to interpret, with each of the three clusters showing distinct patterns of customer spending and demographics, making it suitable for targeted marketing.

6. Key Findings and Insights

Cluster 1: High-Spending, High-Income Customers: This group consists of customers with high income and high spending across multiple product categories. They are likely loyal and respond well to premium offers or new product launches.
Cluster 2: Moderate-Spending, Mid-Income Customers: Customers in this cluster have moderate income and spending behavior. They might be ideal candidates for loyalty programs, special discounts, or cross-selling offers to encourage higher spending.
Cluster 3: Low-Spending, Low-Income Customers: This group represents customers with lower income and spending habits. They may be price-sensitive, and targeted promotional offers or product bundles could help increase engagement.
These segments provide actionable insights that can be used to tailor marketing strategies to the specific needs and characteristics of each group.

7. Suggestions for Next Steps

Further Feature Engineering: Consider adding additional features such as customer engagement (website visits, time spent on the website, etc.) or customer feedback to improve clustering results and gain deeper insights.
Revisit Clustering with Different Numbers of Clusters: Although 3 clusters were chosen based on the initial analysis, experimenting with different numbers of clusters could refine the segmentation further. Methods like the Elbow Method or Silhouette Analysis could assist in determining the optimal number of clusters.
Real-Time Segmentation: Implementing real-time customer segmentation could enable businesses to adapt their marketing strategies based on the evolving behaviors of customers.
These steps can help enhance the accuracy and utility of the segmentation model, ultimately leading to more effective marketing and business strategies.

Data Source

Data source citation: Patel, A. (2021, August 22). Customer personality analysis. Kaggle. https://www.kaggle.com/datasets/imakash3011/customer-personality-analysis/data
marketing_campaign.csv

The features in the data are described as follows:

1. ID: Customer's unique identifier
2. Year_Birth: Customer's birth year
3. Education: Customer's education level
4. Marital_Status: Customer's marital status
5. Income: Customer's yearly household income
6. Kidhome: Number of children in customer's household
7. Teenhome: Number of teenagers in customer's household
8. Dt_Customer: Date of customer's enrollment with the company
9. Recency: Number of days since customer's last purchase
10. Complain: 1 if the customer complained in the last 2 years, 0 otherwise
11. MntWines: Amount spent on wine in last 2 years
12. MntFruits: Amount spent on fruits in last 2 years
13. MntMeatProducts: Amount spent on meat in last 2 years
14. MntFishProducts: Amount spent on fish in last 2 years
15. MntSweetProducts: Amount spent on sweets in last 2 years
16. MntGoldProds: Amount spent on gold in last 2 years
17. NumDealsPurchases: Number of purchases made with a discount
18. AcceptedCmp1: 1 if customer accepted the offer in the 1st campaign, 0 otherwise
19. AcceptedCmp2: 1 if customer accepted the offer in the 2nd campaign, 0 otherwise
20. AcceptedCmp3: 1 if customer accepted the offer in the 3rd campaign, 0 otherwise
21. AcceptedCmp4: 1 if customer accepted the offer in the 4th campaign, 0 otherwise
22. AcceptedCmp5: 1 if customer accepted the offer in the 5th campaign, 0 otherwise
23. Response: 1 if customer accepted the offer in the last campaign, 0 otherwise
24. NumWebPurchases: Number of purchases made through the company’s website
25. NumCatalogPurchases: Number of purchases made using a catalogue
26. NumStorePurchases: Number of purchases made directly in stores
27. NumWebVisitsMonth: Number of visits to company’s website in the last month
28. Z_CostContact: same for all rows
29. Z_Revenue: same for all rows

Approach

Data Preprocessing: Clean and standardize the dataset, ensuring numerical consistency (e.g., scaling the features).
K-means Clustering: Implement the K-means algorithm with varying values of k (number of clusters), and use the elbow method to select the best value of k.
Clustering Visualization: Generate a 2D plot of customer segments with respect to the principal components to visually inspect cluster distribution.
Analysis: Draw conclusions about customer behavior and identify actionable segments.

Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import datetime
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.axes as axes
from matplotlib import cm
import missingno.missingno as msno
import plotly.express as px
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_samples, silhouette_score
from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram
from datetime import datetime
from warnings import filterwarnings
filterwarnings(action='ignore')
```

Exploratory Data Analysis

We will start by loading in the data and checking the data size.

In [2]:
```
# Load data and check shape
df = pd.read_csv("marketing_campaign.csv", sep="\t")
df.shape
```

Out[2]:
```
(2240, 29)
```

Data size: data has 2240 rows and 29 columns

In [3]:
```
df.head()
```

Out[3]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Custo
0	5524	1957	Graduation	Single	58138.0	0	0	04-09-2
1	2174	1954	Graduation	Single	46344.0	1	1	08-03-2
2	4141	1965	Graduation	Together	71613.0	0	0	21-08-2
3	6182	1984	Graduation	Together	26646.0	1	0	10-02-2
4	5324	1981	PhD	Married	58293.0	1	0	19-01-2

5 rows × 29 columns

Data Cleaning

In this notebook, we will clean up the dataset and create a new dataframe with the data we're going to use.

We're going to:

- Find any missing values and decide what to do with rows with missing values, if any
- Simplify categorical data by reducing the number of unique values
- Create new columns based on values from old columns (e.g. `Age` column using the `Year_Birth` column)
- Choose certain columns to keep to reduce number of dimensions

Find missing values

When we check the info of the data we can see that 24 values in the Income column are missing, there are no other non-null values.

In [4]:
```
df.info()
```

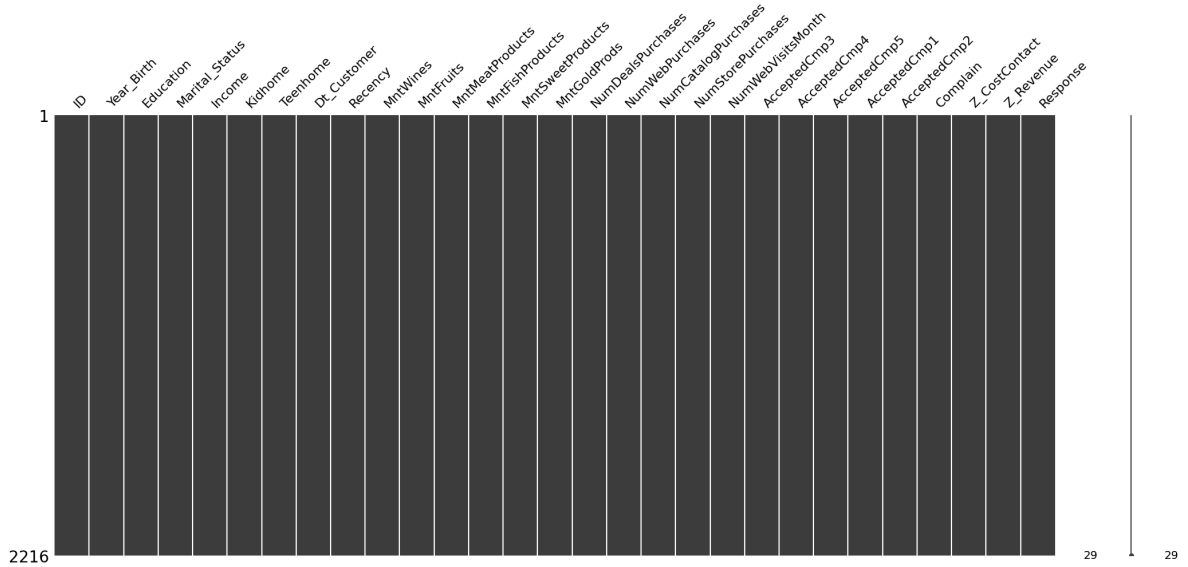
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 29 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   2240 non-null   int64  
 1   Year_Birth           2240 non-null   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
```

We will drop the NA values since there are only 24 of them:

In [5]:
```
df = df.dropna()
df.shape
```

Out[5]:
```
(2216, 29)
```

In [6]:
```
msno.matrix(df)
plt.show()
```



In [7]:
```
missing_rows = df.Income.isna().sum()
total_rows = df.shape[0]

print(f"""
There are {missing_rows} missing data points in the "Income" column.
Those rows make up {round((missing_rows/total_rows)*100, 2)}% of the total datas
""")
```

```
There are 0 missing data points in the "Income" column.
Those rows make up 0.0% of the total dataset.
```

Customer age

In [8]:
```
df.head()
```

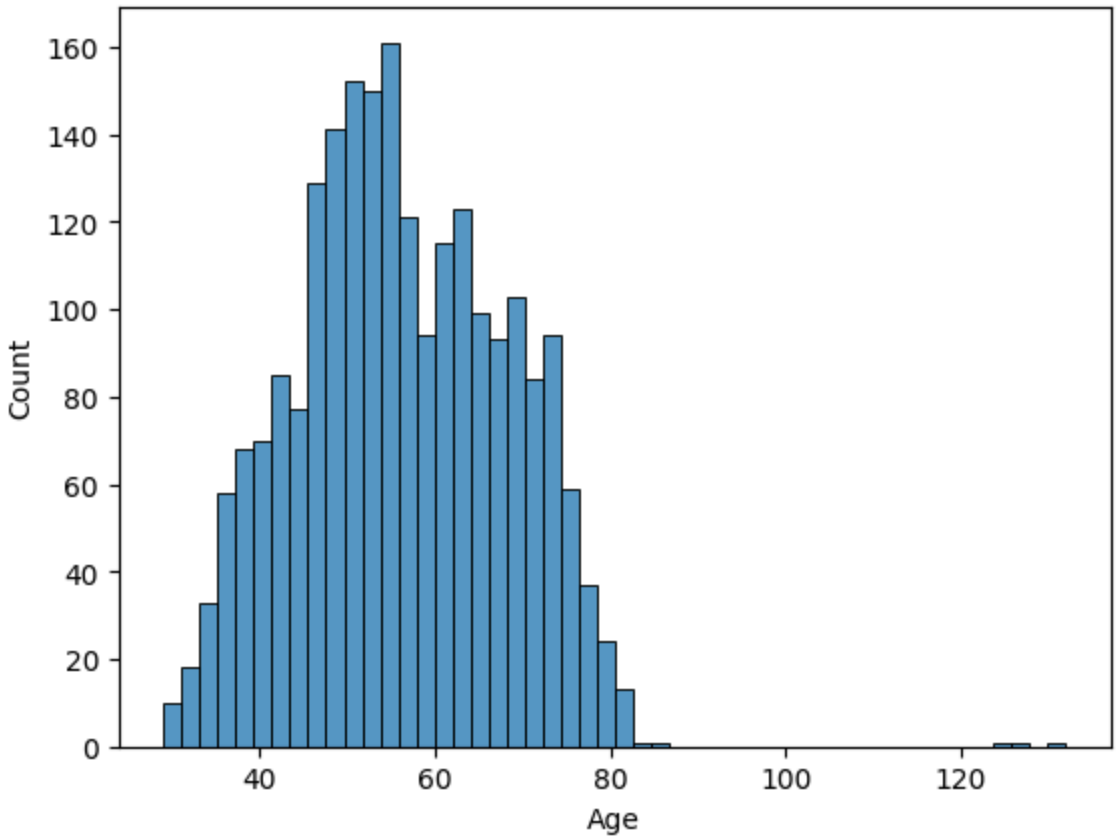
Out[8]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Custo
0	5524	1957	Graduation	Single	58138.0	0	0	04-09-2
1	2174	1954	Graduation	Single	46344.0	1	1	08-03-2
2	4141	1965	Graduation	Together	71613.0	0	0	21-08-2
3	6182	1984	Graduation	Together	26646.0	1	0	10-02-2
4	5324	1981	PhD	Married	58293.0	1	0	19-01-2

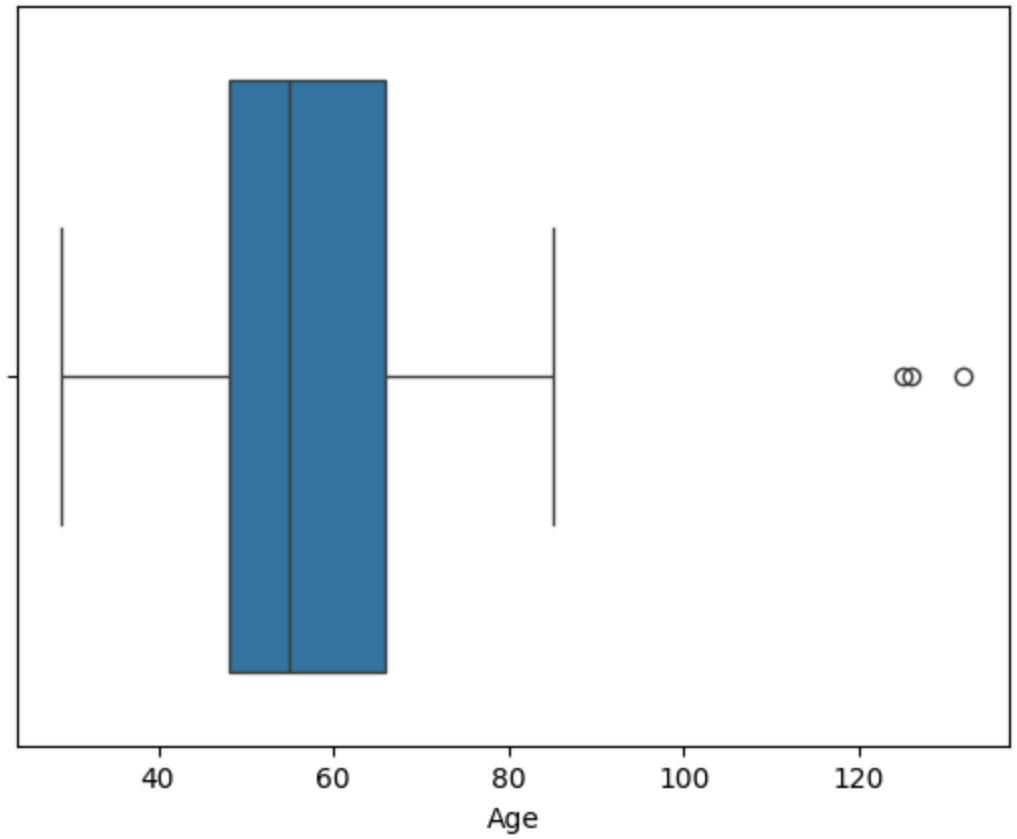
5 rows × 29 columns

In [9]:
```
df['Age'] = datetime.now().year - df.Year_Birth
```

In [10]:
```
sns.histplot(x="Age", data=df, bins=50)
plt.show()
```



In [11]:
```
sns.boxplot(x="Age", data=df)
plt.show()
```



Clean up Education column

We will simplify the `Education` column into "Undergraduate" and "Postgraduate"

Let's check the unique values of feature "Education":

In [12]:
```
df["Education"].unique()
```

Out[12]:
```
array(['Graduation', 'PhD', 'Master', 'Basic', '2n Cycle'], dtype=object)
```

In [13]:
```
df.Education.value_counts()
```

Out[13]:
```
Education
Graduation    1116
PhD            481
Master         365
2n Cycle       200
Basic           54
Name: count, dtype: int64
```

In [14]:
```
def edu(x):
    education = {
        'Basic'     :'Undergraduate',
        '2n Cycle'  :'Undergraduate',
        'Graduation':'Postgraduate',
        'Master'    :'Postgraduate',
        'PhD'       :'Postgraduate'
    }
    return education[x]
```

In [15]:
```
df['Simplified_Education'] = df.Education.apply(edu)
```

In [16]:
```
df[['Education', 'Simplified_Education']].head()
```

Out[16]:

	Education	Simplified_Education
0	Graduation	Postgraduate
1	Graduation	Postgraduate
2	Graduation	Postgraduate
3	Graduation	Postgraduate
4	PhD	Postgraduate

The unique values for Education are 'Graduation', 'PhD', 'Master', 'Basic', '2n Cycle'. Based on information online, 2n Cycle corresponds to master's degree in some european countries but it is not clear if Graduation and Basic correspond to Undergraduate degree, so we will leave the values as they are.

Clean up Marital Status column

We will simplify the `Marital_Status` column values into either "Relationship" or "Alone"

For Marital_Status feature we can see that there are two invalid entries: Absurd and YOLO. We will drop these.

In [17]:
```
df = df.drop(df[(df["Marital_Status"] == "YOLO") | (df["Marital_Status"] == "Abs
# double check the unique values
df["Marital_Status"].unique()
```

Out[17]:
```
array(['Single', 'Together', 'Married', 'Divorced', 'Widow', 'Alone'],
      dtype=object)
```

In [18]:
```
df.Marital_Status.value_counts()
```

Out[18]:
```
Marital_Status
Married     857
Together    573
Single      471
Divorced    232
Widow        76
Alone         3
Name: count, dtype: int64
```

In [19]:
```
print(df[df['Marital_Status'].str.contains('YOLO', case=False, na=False)])
```

```
Empty DataFrame
Columns: [ID, Year_Birth, Education, Marital_Status, Income, Kidhome, Teenhome, D
t_Customer, Recency, MntWines, MntFruits, MntMeatProducts, MntFishProducts, MntSw
eetProducts, MntGoldProds, NumDealsPurchases, NumWebPurchases, NumCatalogPurchase
s, NumStorePurchases, NumWebVisitsMonth, AcceptedCmp3, AcceptedCmp4, AcceptedCmp
5, AcceptedCmp1, AcceptedCmp2, Complain, Z_CostContact, Z_Revenue, Response, Age,
Simplified_Education]
Index: []

[0 rows x 31 columns]
```

In [20]:
```
df[df['Marital_Status'] == 'YOLO']
```

Out[20]:

ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer

0 rows × 31 columns

In [21]:
```
df[df['Marital_Status'] == 'Absurd']
```

Out[21]:

ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer

0 rows × 31 columns

Dropping the "YOLO" and "Absurd" marital status because we are not sure if that means "Single" or "In a relationship". We do not want to make assumptions that may be incorrect.

In [22]:
```
df = df[(df.Marital_Status != 'YOLO') & (df.Marital_Status != 'Absurd')]
```

In [23]:
```
df.Marital_Status.value_counts()
```

Out[23]:
```
Marital_Status
Married     857
Together    573
Single      471
Divorced    232
Widow        76
Alone         3
Name: count, dtype: int64
```

In [24]:
```
def married_single(x):
    relationship = {
        'Married' :'Relationship',
        'Together':'Relationship',
        'Single'  :'Alone',
        'Divorced':'Alone',
        'Widow'   :'Alone',
        'Alone'   :'Alone'
    }

    return relationship[x]
```

In [25]:
```
df['Relationship_Status'] = df.Marital_Status.apply(married_single)
```

In [26]:
```
df[['Marital_Status', 'Relationship_Status']].head(10)
```

Out[26]:

	Marital_Status	Relationship_Status
0	Single	Alone
1	Single	Alone
2	Together	Relationship
3	Together	Relationship
4	Married	Relationship
5	Together	Relationship
6	Divorced	Alone
7	Married	Relationship
8	Together	Relationship
9	Together	Relationship

Clean up the children column

We will calculate how many children are in each household and create a new column containing boolean values on whether or not the household has a child.

In [27]:
```
# Total number of children
df['Num_children'] = df.Kidhome + df.Teenhome

# Has a child (True/False)
df['Has_child'] = df['Num_children'].apply(lambda x: x != 0)
```

In [28]:
```
df[['Num_children', 'Has_child']].head()
```

Out[28]:

	Num_children	Has_child
0	0	False
1	2	True
2	0	False
3	1	True
4	1	True

Rename the products columns

We will rename the products columns to something simpler and easier to understand for our analysis. Then we will add the total amount of products purchased.

In [29]:
```
df = df.rename(columns={
    'MntWines'        :'Wines',
    'MntFruits'       :'Fruits',
    'MntMeatProducts' :'Meat',
    'MntFishProducts' :'Fish',
    'MntSweetProducts':'Sweets',
    'MntGoldProds'    :'Gold'
})
```

In [30]:
```
df.iloc[:,9:15].head()
```

Out[30]:

	Wines	Fruits	Meat	Fish	Sweets	Gold
0	635	88	546	172	88	88
1	11	1	6	2	1	6
2	426	49	127	111	21	42
3	11	4	20	10	3	5
4	173	43	118	46	27	15

In [31]:
```
# Sum of products
df['Total'] = df.iloc[:,9:15].sum(axis=1)
```

In [32]:
```
df[['Wines', 'Fruits', 'Meat', 'Fish', 'Sweets', 'Gold', 'Total']].head()
```

Out[32]:

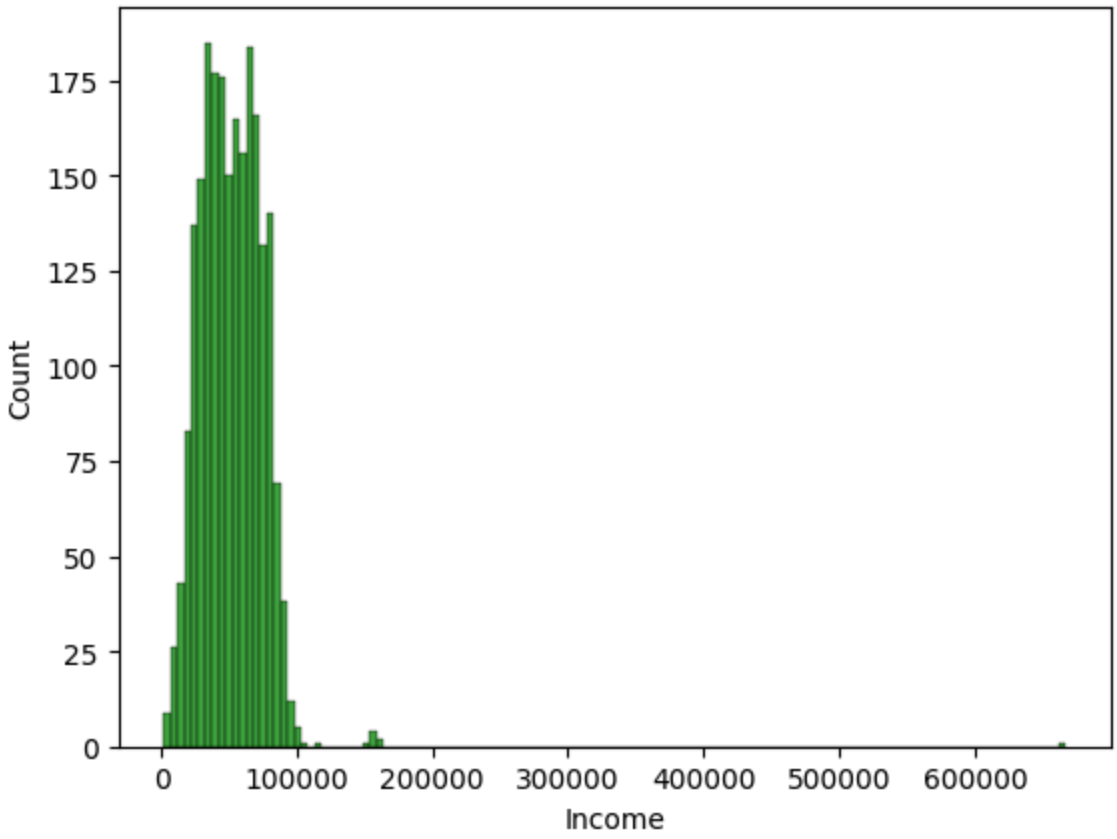
	Wines	Fruits	Meat	Fish	Sweets	Gold	Total
0	635	88	546	172	88	88	1617
1	11	1	6	2	1	6	27
2	426	49	127	111	21	42	776
3	11	4	20	10	3	5	53
4	173	43	118	46	27	15	422

Income column

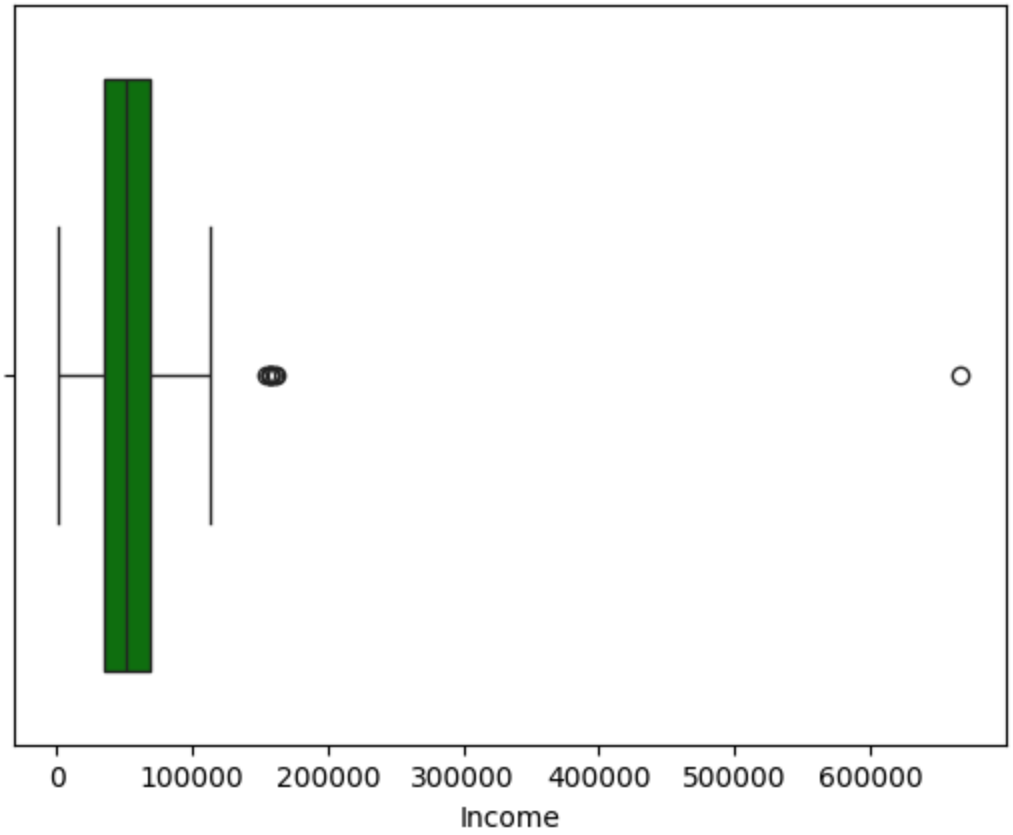
In [33]:
```
df.Income.describe()
```

Out[33]:
```
count      2212.000000
mean      52232.510850
std       25187.455359
min        1730.000000
25%       35233.500000
50%       51381.500000
75%       68522.000000
max      666666.000000
Name: Income, dtype: float64
```

In [34]:
```
sns.histplot(x="Income", data=df, color='green')

plt.show()
```



In [35]:
```
sns.boxplot(x="Income", data=df, color='green')

plt.show()
```



Let's check out the high income earners to see if there is any useful information

In [36]:
```
df[df['Income'] >= 150000]
```

Out[36]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_
164	8475	1973	PhD	Married	157243.0	0	1	0
617	1503	1976	PhD	Together	162397.0	1	1	0
655	5555	1975	Graduation	Divorced	153924.0	0	0	0
687	1501	1982	PhD	Married	160803.0	0	0	0
1300	5336	1971	Master	Together	157733.0	1	0	0
1653	4931	1977	Graduation	Together	157146.0	0	0	2
2132	11181	1949	PhD	Married	156924.0	0	0	2
2233	9432	1977	Graduation	Together	666666.0	1	0	0

8 rows × 35 columns

We will remove these because there are too few data points to get any meaningful information. These outliers could affect the model.

In [37]:
```
df = df[df['Income'] <= 150000]
```

In [38]:
```
df.Income.describe()
```

Out[38]:
```
count      2204.000000
mean      51617.730490
std       20721.492888
min        1730.000000
25%       35191.500000
50%       51301.000000
75%       68289.750000
max      113734.000000
Name: Income, dtype: float64
```

Customer seniority

In [39]:
```
# Convert string to datetime object
df['Dt_Customer'] = df['Dt_Customer'].apply(lambda x: datetime.strptime(x, '%d-%
```

In [40]:
```
# Calculate the number of days since customers joined
df['days_joined'] = datetime.now().date() - df.Dt_Customer
df['days_joined'] = df.days_joined.apply(lambda x: x.days)

# Convert number of days to approximate years
# Approximate because it does not take into account leap years
df['Seniority'] = round(df.days_joined / 365, 2)
```

In [41]:
```
df[['Dt_Customer', 'days_joined', 'Seniority']].head()
```

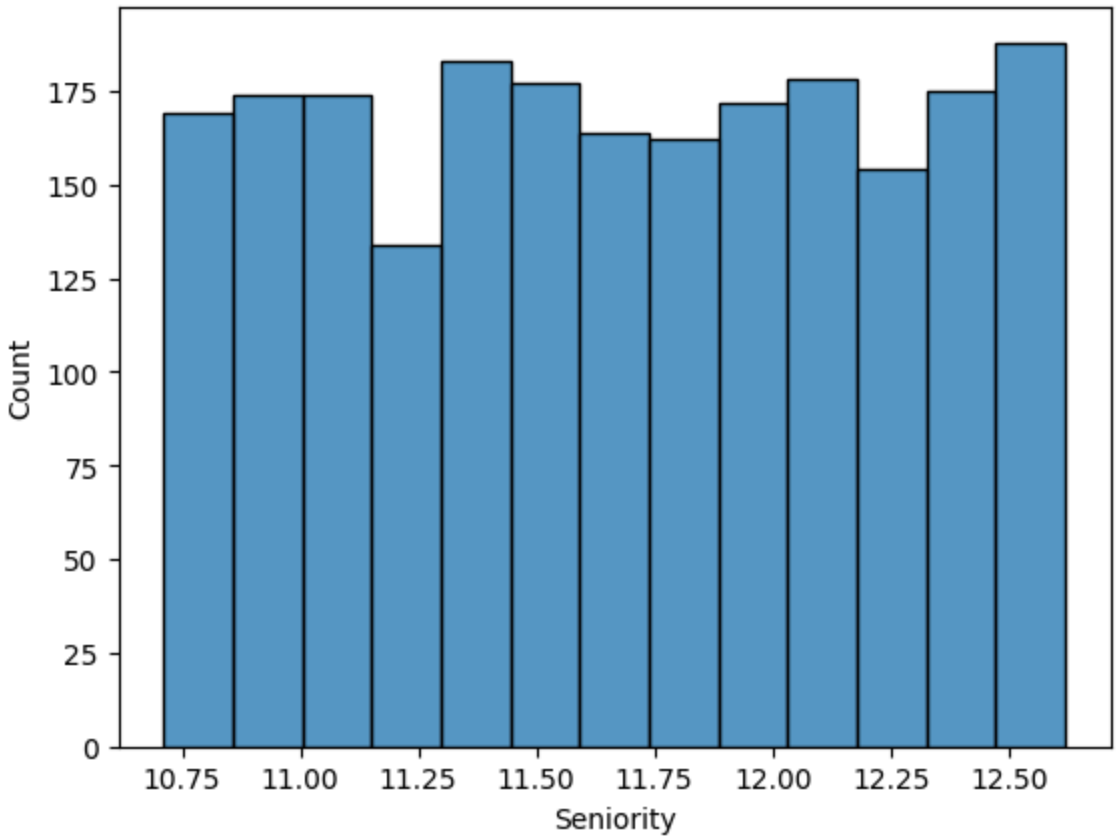
Out[41]:

	Dt_Customer	days_joined	Seniority
0	2012-09-04	4571	12.52
1	2014-03-08	4021	11.02
2	2013-08-21	4220	11.56
3	2014-02-10	4047	11.09
4	2014-01-19	4069	11.15

In [42]:
```
df.Seniority.describe()
```

Out[42]:
```
count    2204.000000
mean       11.674678
std         0.554673
min        10.710000
25%        11.200000
50%        11.680000
75%        12.160000
max        12.620000
Name: Seniority, dtype: float64
```

In [43]:
```
sns.histplot(x="Seniority", data=df)

plt.show()
```



Convert data types

Income column

Convert from float to integer

In [44]:
```
df.Income.dtype
```

Out[44]:
```
dtype('float64')
```

In [45]:
```
# Check if any data in the income column does not end with a ".0"
print(df.Income.shape[0] - df.Income.astype(str).str.endswith('.0').sum())
```

```
0
```

Since all values in the Income column end with `.0`, we will convert them to integers

In [46]:
```
df['Income'] = df.Income.astype(int)
```

Create new DataFrame with selected columns

Let's check the values of our data set:

In [47]:
```
df.describe()
```

Out[47]:

	ID	Year_Birth	Income	Kidhome	Teenhome	Recency
count	2204.000000	2204.000000	2204.000000	2204.000000	2204.000000	2204.000000
mean	5586.274501	1968.797641	51617.730490	0.442831	0.506352	49.052178
std	3247.672073	11.988415	20721.492888	0.537220	0.544402	28.931031
min	0.000000	1893.000000	1730.000000	0.000000	0.000000	0.000000
25%	2814.750000	1959.000000	35191.500000	0.000000	0.000000	24.000000
50%	5458.500000	1970.000000	51301.000000	0.000000	0.000000	49.000000
75%	8418.500000	1977.000000	68289.750000	1.000000	1.000000	74.000000
max	11191.000000	1996.000000	113734.000000	2.000000	2.000000	99.000000

8 rows × 31 columns

In [48]:
```
df.columns
```

Out[48]:
```
Index(['ID', 'Year_Birth', 'Education', 'Marital_Status', 'Income', 'Kidhome',
       'Teenhome', 'Dt_Customer', 'Recency', 'Wines', 'Fruits', 'Meat', 'Fish',
       'Sweets', 'Gold', 'NumDealsPurchases', 'NumWebPurchases',
       'NumCatalogPurchases', 'NumStorePurchases', 'NumWebVisitsMonth',
       'AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1',
       'AcceptedCmp2', 'Complain', 'Z_CostContact', 'Z_Revenue', 'Response',
       'Age', 'Simplified_Education', 'Relationship_Status', 'Num_children',
       'Has_child', 'Total', 'days_joined', 'Seniority'],
      dtype='object')
```

In [49]:
```
temp_df = df[['Age', 'Simplified_Education', 'Relationship_Status', 'Income',
              'Seniority', 'Num_children', 'Has_child', 'Wines', 'Fruits',
              'Meat', 'Fish', 'Sweets', 'Gold', 'Total']]
```

In [50]:
```
temp_df = temp_df.rename(columns={
    'Simplified_Education':'Education',
})
```

In [51]:
```
temp_df.head()
```

Out[51]:

	Age	Education	Relationship_Status	Income	Seniority	Num_children	Has_child
0	68	Postgraduate	Alone	58138	12.52	0	False
1	71	Postgraduate	Alone	46344	11.02	2	True
2	60	Postgraduate	Relationship	71613	11.56	0	False
3	41	Postgraduate	Relationship	26646	11.09	1	True
4	44	Postgraduate	Relationship	58293	11.15	1	True

In [52]:
```
temp_df.Education.value_counts(normalize=True)
```

Out[52]:
```
Education
Postgraduate     0.884755
Undergraduate    0.115245
Name: proportion, dtype: float64
```

In [53]:
```
temp_df.Relationship_Status.value_counts(normalize=True)
```

Out[53]:
```
Relationship_Status
Relationship    0.645644
Alone           0.354356
Name: proportion, dtype: float64
```

In [54]:
```
temp_df.Num_children.value_counts(normalize=True)
```

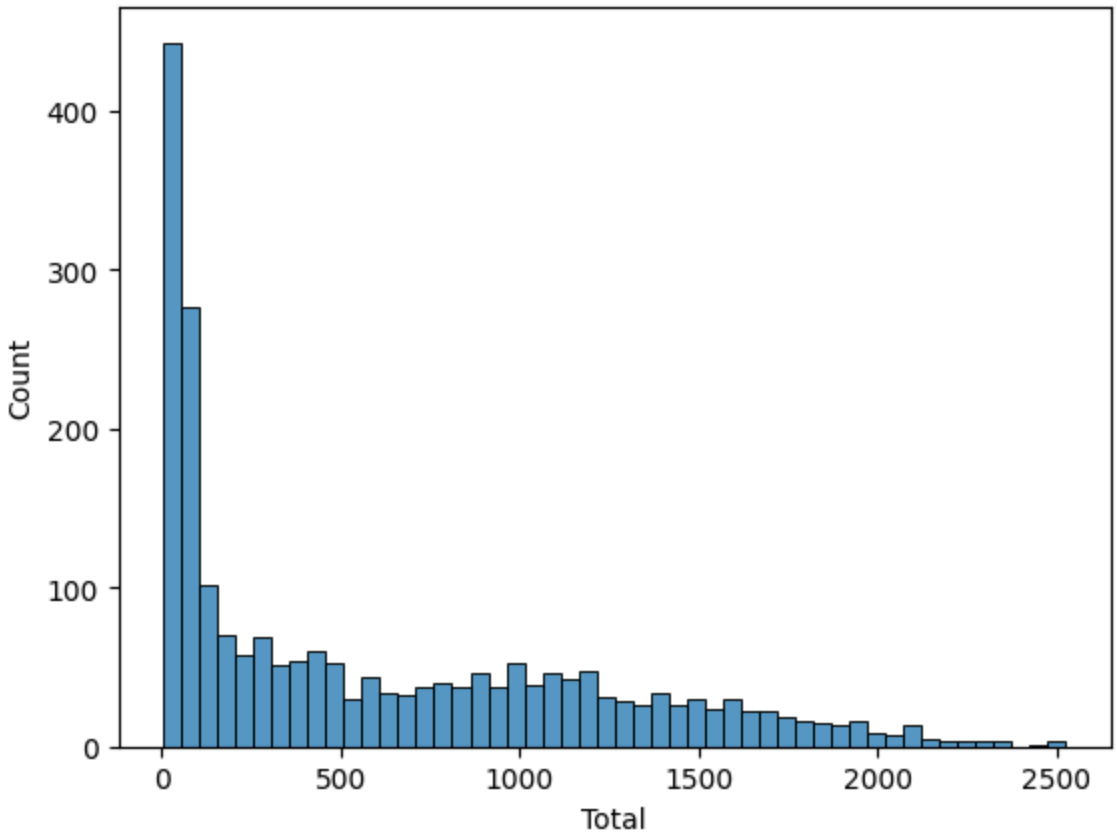
Out[54]:
```
Num_children
1    0.504537
0    0.284483
2    0.188294
3    0.022686
Name: proportion, dtype: float64
```

In [55]:
```
temp_df.describe()
```

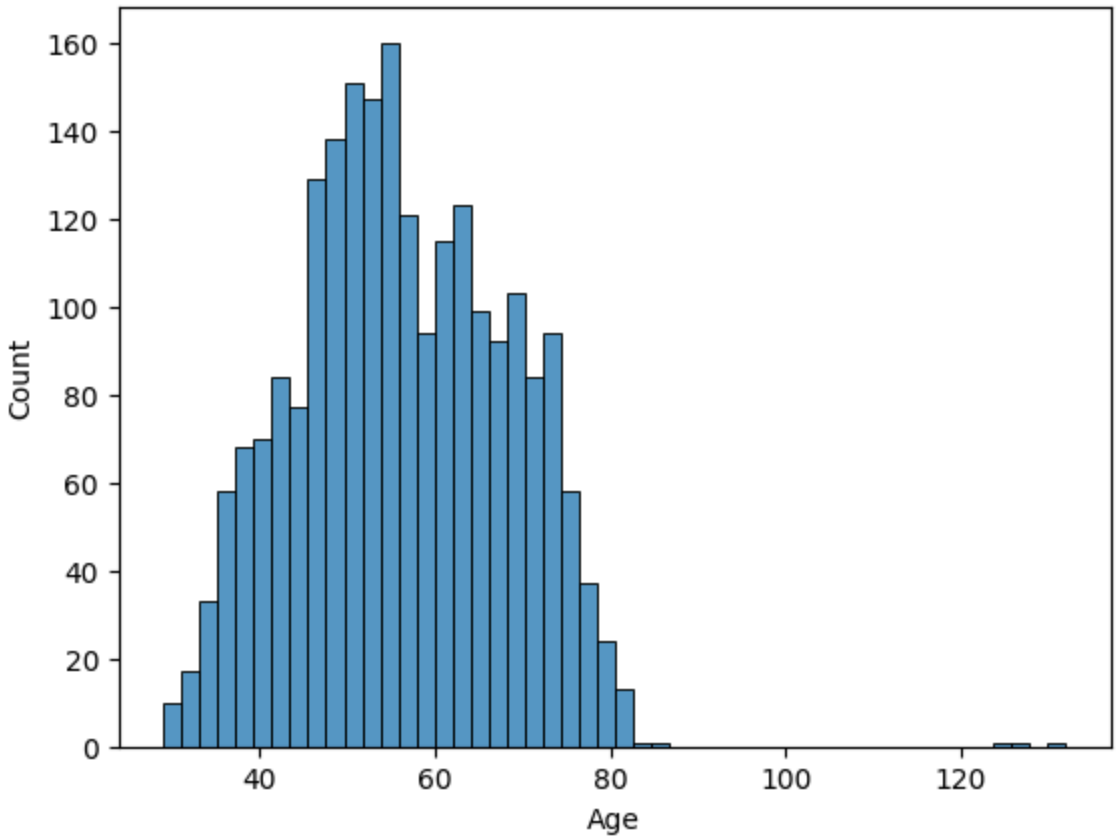
Out[55]:

	Age	Income	Seniority	Num_children	Wines	Fruit
count	2204.000000	2204.000000	2204.000000	2204.000000	2204.000000	2204.00000(
mean	56.202359	51617.730490	11.674678	0.949183	306.041742	26.40381
std	11.988415	20721.492888	0.554673	0.749128	337.800997	39.83047
min	29.000000	1730.000000	10.710000	0.000000	0.000000	0.00000(
25%	48.000000	35191.500000	11.200000	0.000000	24.000000	2.00000(
50%	55.000000	51301.000000	11.680000	1.000000	176.000000	8.00000(
75%	66.000000	68289.750000	12.160000	1.000000	507.250000	33.00000(
max	132.000000	113734.000000	12.620000	3.000000	1493.000000	199.00000(

In [56]:
```
sns.histplot(x="Total", data=temp_df, bins=50)
plt.show()
```



In [57]:
```
sns.histplot(x="Age", data=temp_df, bins=50)
plt.show()
```



We can see as the minimum value for Year_Birth 1893. That must be a mistake, so we will filter out birth years less than 1930.

In [58]:
```
df = df[df['Year_Birth'] >= 1930]
```

We can also see an outlier for Income, maximum value of 666666, we will filter out Income higher than 600000

In [59]:
```
df = df[df['Income'] < 600000]
```

In [60]:
```
temp_df.head()
```

Out[60]:

	Age	Education	Relationship_Status	Income	Seniority	Num_children	Has_child
0	68	Postgraduate	Alone	58138	12.52	0	False
1	71	Postgraduate	Alone	46344	11.02	2	True
2	60	Postgraduate	Relationship	71613	11.56	0	False
3	41	Postgraduate	Relationship	26646	11.09	1	True
4	44	Postgraduate	Relationship	58293	11.15	1	True

Feature Engineering

The unique values for Marital_Status are Single, Together, Married, Divorced, Widow, Alone. However we only care if the customer has a partner at home or not, so we will create a new feature Partner_Status and code Married, Partner and Together as 1 and Widow, Divorced, Single and Alone as 0.

In [61]:
```
df["Partner_Status"]=df["Marital_Status"].replace({"Married": 1, "Partner": 1, "
                                                   "Widow": 0, "Divorced": 0, "Single":
```

In [62]:
```
df.head()
```

Out[62]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Custo
0	5524	1957	Graduation	Single	58138	0	0	2012-09
1	2174	1954	Graduation	Single	46344	1	1	2014-03
2	4141	1965	Graduation	Together	71613	0	0	2013-08
3	6182	1984	Graduation	Together	26646	1	0	2014-02
4	5324	1981	PhD	Married	58293	1	0	2014-01

5 rows × 38 columns

In [63]:
```
df.describe()
```

Out[63]:

	ID	Year_Birth	Income	Kidhome	Teenhome	Recency
count	2201.000000	2201.000000	2201.000000	2201.000000	2201.000000	2201.000000
mean	5584.809632	1968.895048	51606.144480	0.442980	0.506588	49.047251
std	3246.102409	11.701763	20721.181896	0.537289	0.544457	28.924487
min	0.000000	1940.000000	1730.000000	0.000000	0.000000	0.000000
25%	2815.000000	1959.000000	35178.000000	0.000000	0.000000	24.000000
50%	5455.000000	1970.000000	51287.000000	0.000000	0.000000	49.000000
75%	8418.000000	1977.000000	68281.000000	1.000000	1.000000	74.000000
max	11191.000000	1996.000000	113734.000000	2.000000	2.000000	99.000000

8 rows × 32 columns

We will create feature "Age" by substracting Year_Birth from 2024.

In [64]:
```
df['Age'] = 2024 - df['Year_Birth']
```

We will drop the features we are not going to use:

In [65]:
```
cols_drop = ["Marital_Status", "Dt_Customer", "Z_CostContact", "Z_Revenue", "ID"
df = df.drop(cols_drop, axis=1)
```

In [66]:
```
df.head()
```

Out[66]:

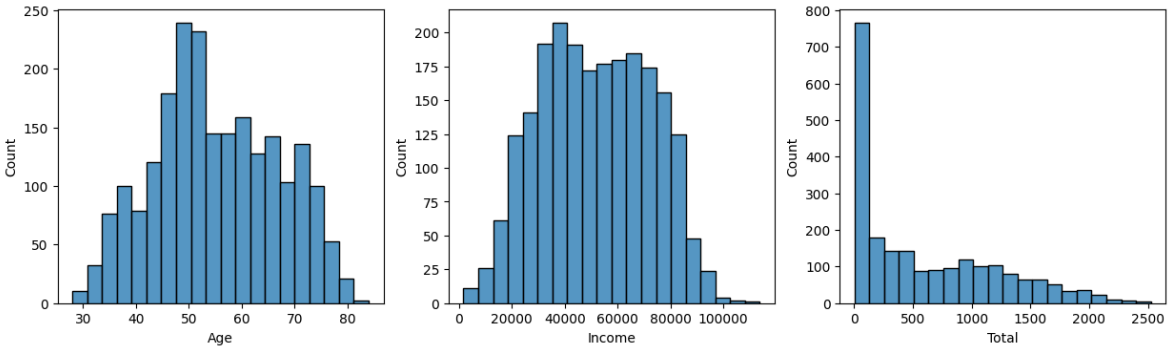
	Education	Income	Kidhome	Teenhome	Recency	Wines	Fruits	Meat	Fish	Swee
0	Graduation	58138	0	0	58	635	88	546	172	
1	Graduation	46344	1	1	38	11	1	6	2	
2	Graduation	71613	0	0	26	426	49	127	111	
3	Graduation	26646	1	0	26	11	4	20	10	
4	PhD	58293	1	0	94	173	43	118	46	

5 rows × 32 columns

Data Visualizations

Let's chack the distributions for age, income and total spent.

In [67]:
```
fig, axes = plt.subplots(1, 3, figsize=(15, 4))
axes = axes.flatten()
sns.histplot(data=df, x="Age", bins=20, ax=axes[0])
sns.histplot(data=df, x="Income", bins=20, ax=axes[1])
sns.histplot(data=df, x="Total", bins=20, ax=axes[2])
#sns.histplot(data=df, x="Total_Spent", bins=20, ax=axes[2])

plt.show()
```

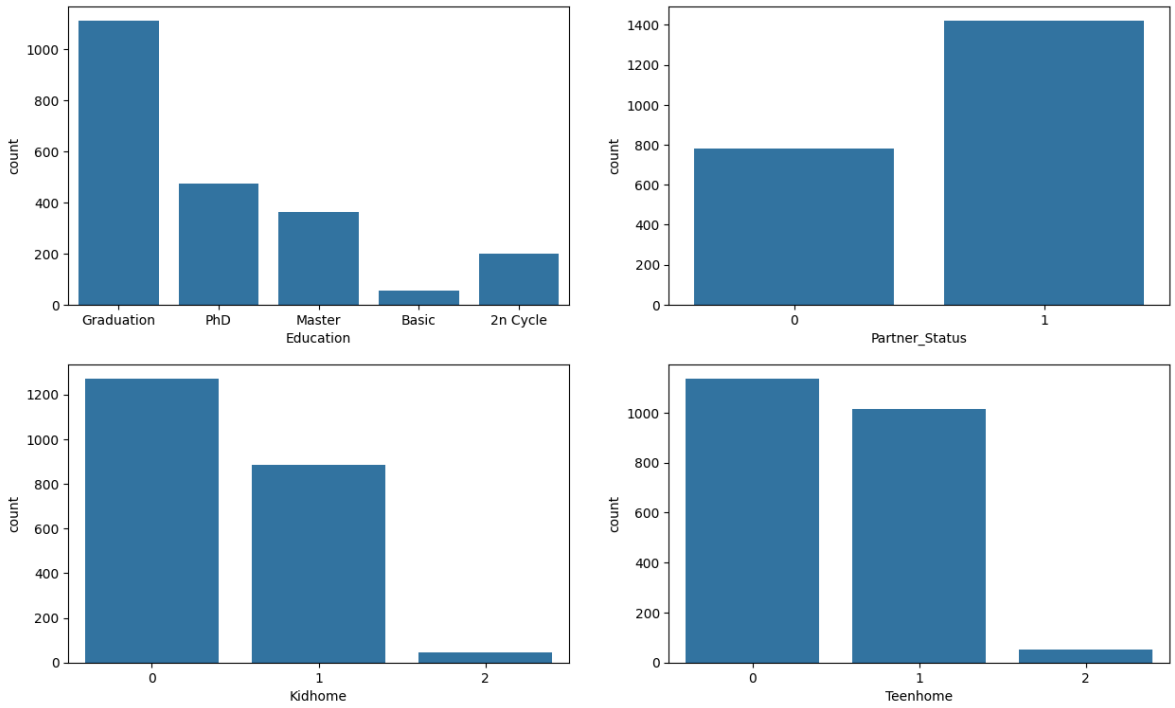


We can see that Age is roughly normally distributed, there are still some outliers in the high income group and there is a high number of customers with very low total spending.

Next we will check demographic information of the customers, their education, partner status, how many kids they have at home and how many teenagers they have at home.

In [68]:
```
fig, axes = plt.subplots(2, 2, figsize=(15, 9))
axes = axes.flatten()

sns.countplot(data=df, x="Education", ax=axes[0])
sns.countplot(data=df, x="Partner_Status", ax=axes[1])
sns.countplot(data=df, x="Kidhome", ax=axes[2])
sns.countplot(data=df, x="Teenhome", ax=axes[3])
```

Out[68]: <Axes: xlabel='Teenhome', ylabel='count'>

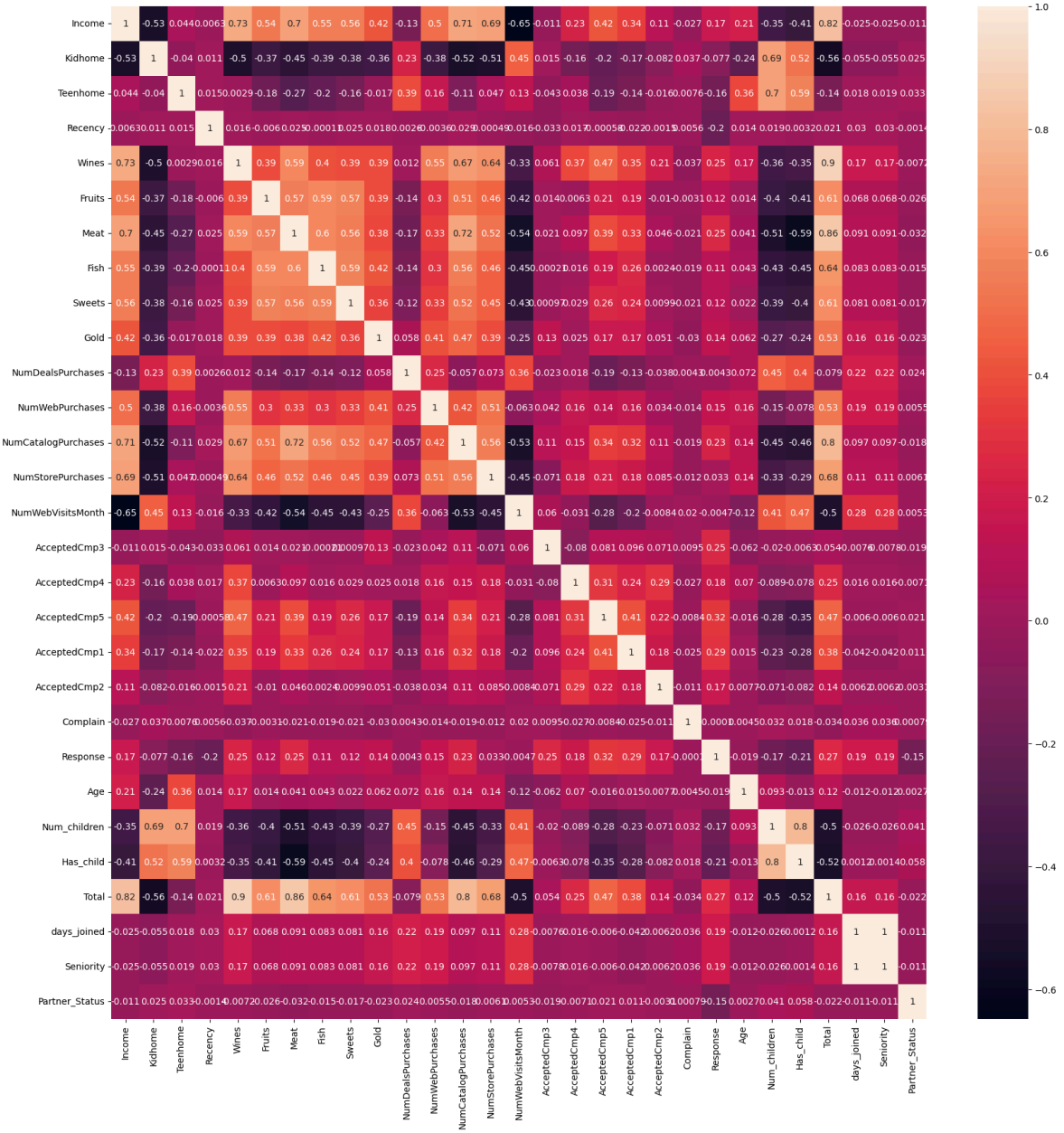


Graduation is the highest education category, most customers have a partner. Many customers do not have a kid or teenager at home, many have one kid or one teenager and very few have two.

Correlation between features

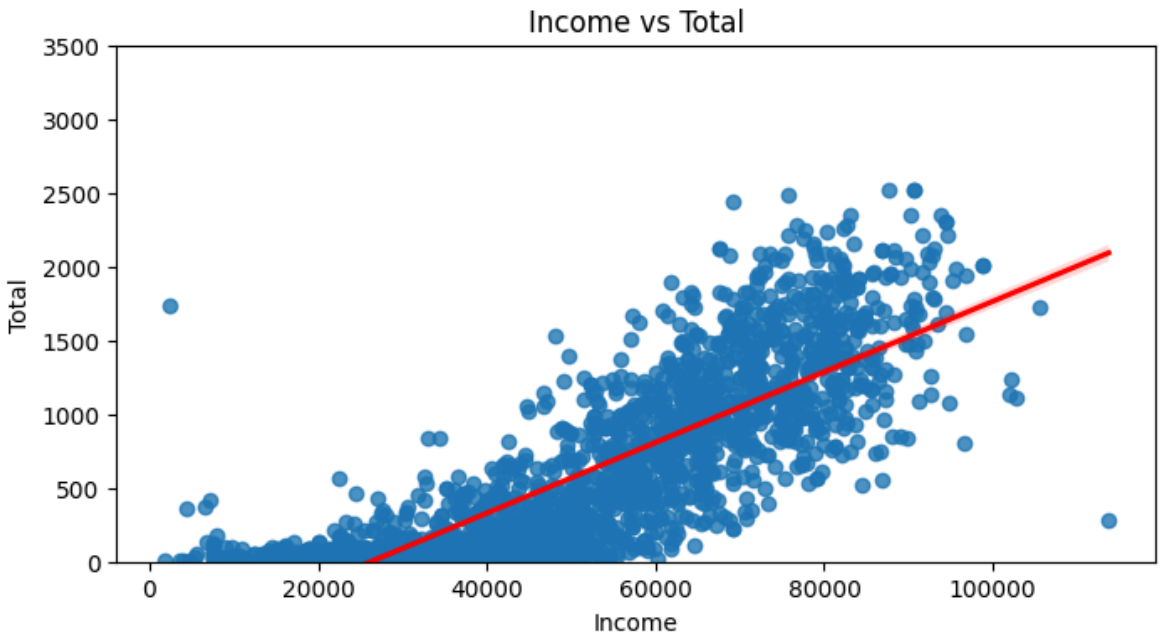
Now we are going to check the correlation between features with a correlation matrix heat map. We can see that there is correlation between many features.

In [69]:
```
cor = df[df.columns[df.dtypes != 'object']].corr()

plt.figure(figsize=(20,20))
sns.heatmap(cor, annot=True)
plt.savefig('corheat.png', dpi=300)
```



Let's check the relationship between income and total spent. We can see that there is a strong relationship.

In [70]:
```
# 'Total'='Total_Spent'
plt.figure(figsize=(8, 4))
sns.regplot(x='Income', y='Total', data=df, line_kws={'color': 'red'})

plt.title('Income vs Total')
axes = plt.gca()
axes.set_ylim(0, 3500)
plt.xlabel('Income')
plt.ylabel('Total')
#plt.ylabel('Total_Spent')
plt.show()
```



Exploratory Data Analysis

The following is a list of what we will be looking for:

- View the frequency counts of unique values of categorical data
- View the distributions of numerical data

Missing Values

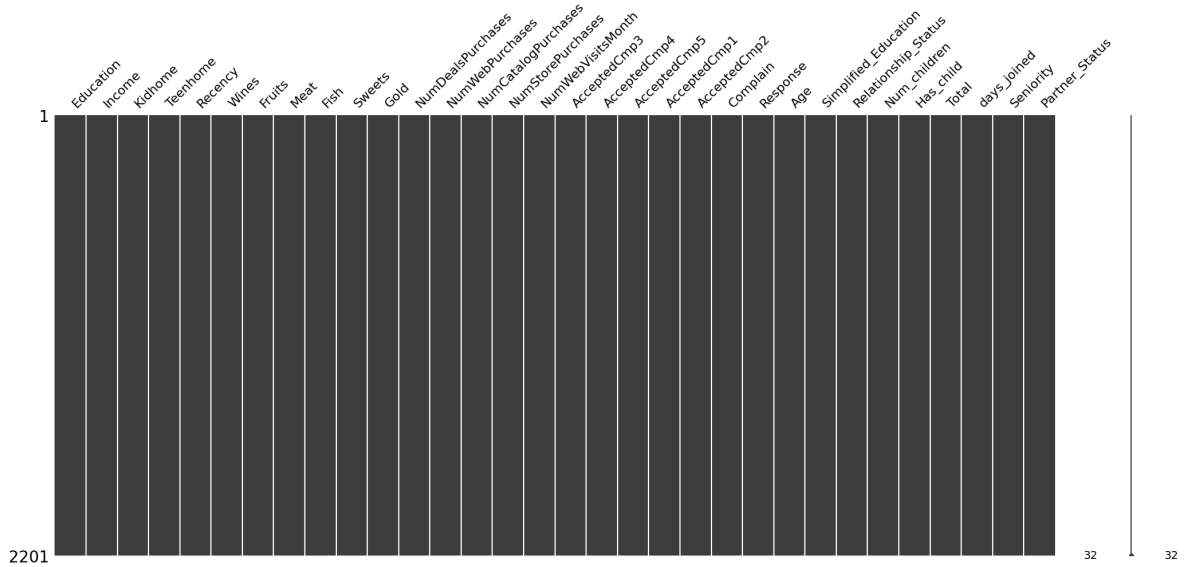
In [71]:
```
df.head()
```

Out[71]:

	Education	Income	Kidhome	Teenhome	Recency	Wines	Fruits	Meat	Fish	Swee
0	Graduation	58138	0	0	58	635	88	546	172	
1	Graduation	46344	1	1	38	11	1	6	2	
2	Graduation	71613	0	0	26	426	49	127	111	
3	Graduation	26646	1	0	26	11	4	20	10	
4	PhD	58293	1	0	94	173	43	118	46	

5 rows × 32 columns

In [72]:
```
msno.matrix(df)
plt.show()
```



Categorical Data

In [73]:
```
cat_df = df[['Education', 'Relationship_Status', 'Has_child']]

cat_df.head()
```

Out[73]:

	Education	Relationship_Status	Has_child
0	Graduation	Alone	False
1	Graduation	Alone	True
2	Graduation	Relationship	False
3	Graduation	Relationship	True
4	PhD	Relationship	True

Education

In [74]:
```
education_count = cat_df['Education'].value_counts(normalize=True)
pd.DataFrame(round(education_count*100, 2))
```

Out[74]:

Education	proportion
Graduation	50.52
PhD	21.54
Master	16.49
2n Cycle	9.00
Basic	2.45

In [75]:
```
px.bar(education_count)
```

Relationship Status

In [76]:
```
relationship_count = cat_df['Relationship_Status'].value_counts(normalize=True)
pd.DataFrame(round(relationship_count*100, 2))
```

Out[76]:

Relationship_Status	proportion
Relationship	64.61
Alone	35.39

In [77]:
```
px.bar(relationship_count)
```

Has Child

In [78]:
```
haschild_count = cat_df['Has_child'].value_counts(normalize=True)
pd.DataFrame(round(haschild_count*100, 2))
```

Out[78]:

Has_child	proportion
True	71.56
False	28.44

In [79]:
```
px.bar(haschild_count)
```

Numerical Data

In [80]:
```
num_df = df[['Age', 'Income', 'Seniority', 'Num_children', 'Wines',
             'Fruits', 'Meat', 'Fish', 'Sweets', 'Gold', 'Total']]

num_df.head()
```

Out[80]:

	Age	Income	Seniority	Num_children	Wines	Fruits	Meat	Fish	Sweets	Gold	To
0	67	58138	12.52	0	635	88	546	172	88	88	16
1	70	46344	11.02	2	11	1	6	2	1	6	
2	59	71613	11.56	0	426	49	127	111	21	42	7
3	40	26646	11.09	1	11	4	20	10	3	5	
4	43	58293	11.15	1	173	43	118	46	27	15	4

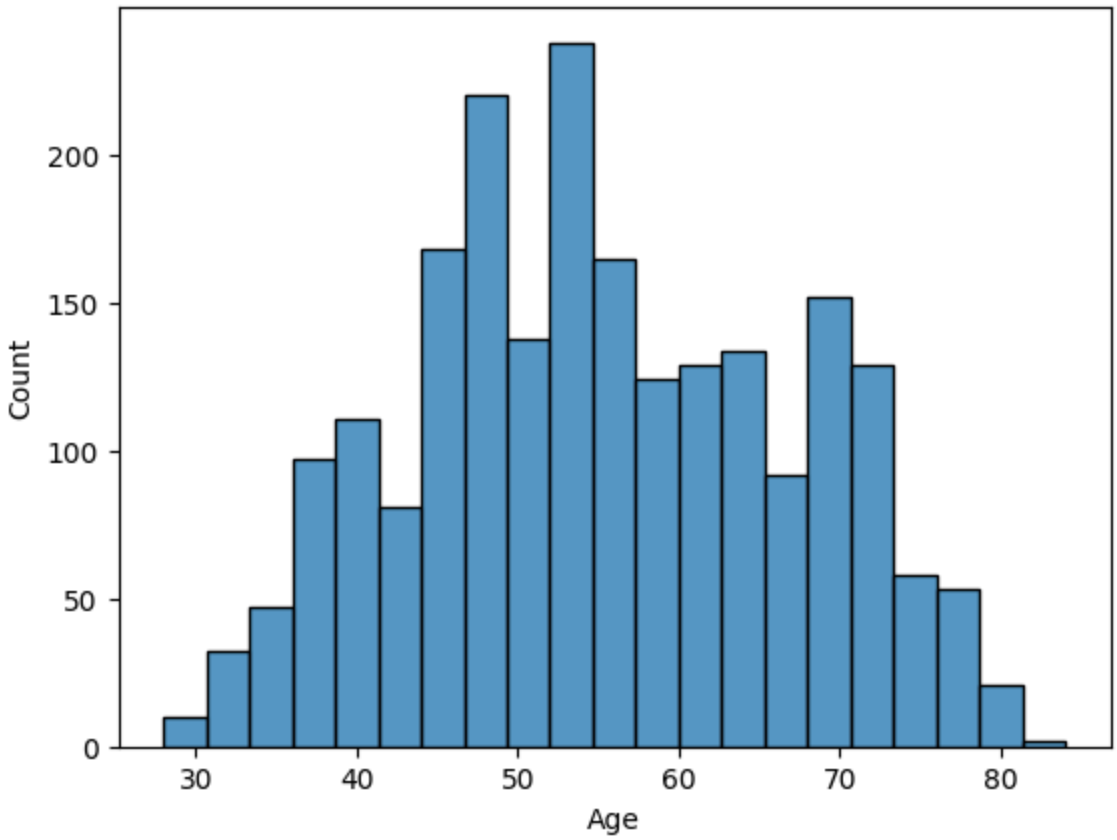
In [81]:
```
num_df.describe()
```

Out[81]:

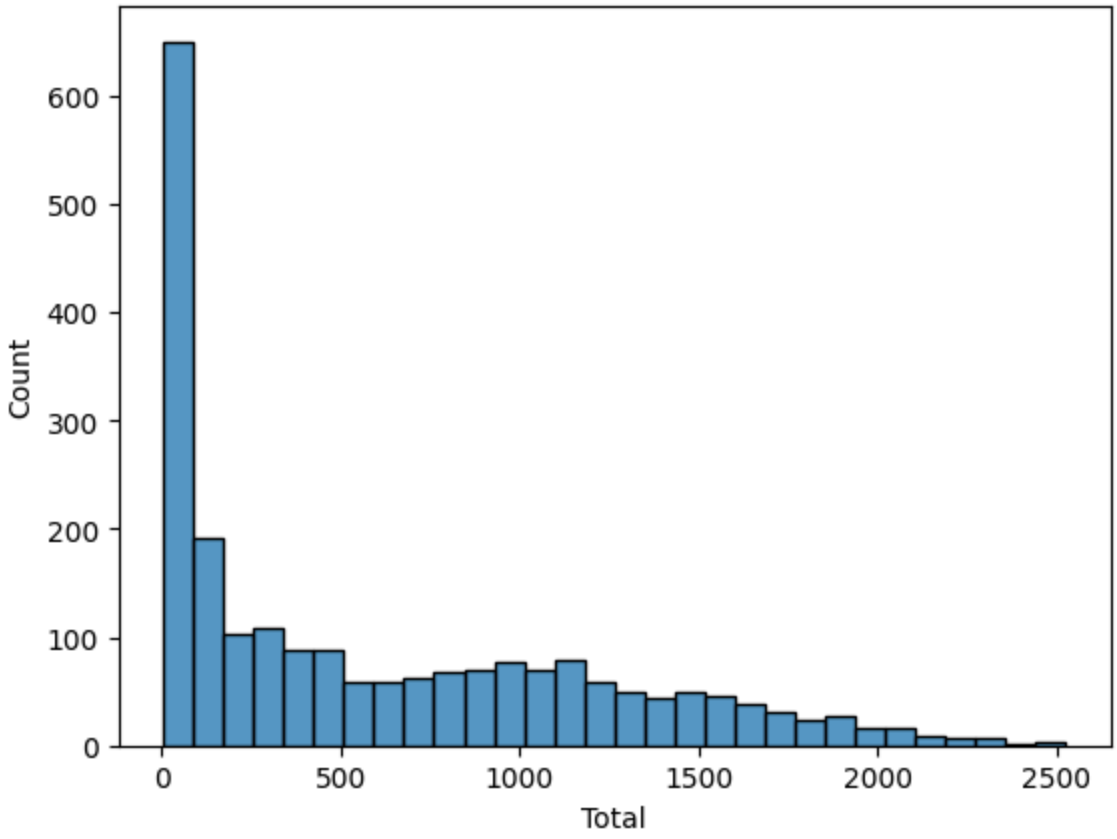
	Age	Income	Seniority	Num_children	Wines	Fruit
count	2201.000000	2201.000000	2201.000000	2201.000000	2201.000000	2201.00000(
mean	55.104952	51606.144480	11.675261	0.949568	306.105407	26.371645
std	11.701763	20721.181896	0.554713	0.749364	337.778943	39.772318
min	28.000000	1730.000000	10.710000	0.000000	0.000000	0.00000(
25%	47.000000	35178.000000	11.200000	0.000000	24.000000	2.00000(
50%	54.000000	51287.000000	11.680000	1.000000	176.000000	8.00000(
75%	65.000000	68281.000000	12.160000	1.000000	507.000000	33.00000(
max	84.000000	113734.000000	12.620000	3.000000	1493.000000	199.00000(

Age

In [82]:
```
sns.histplot(x="Age", data=num_df)
plt.show()
```



In [83]:
```
sns.histplot(x="Total", data=num_df, bins=30)
plt.show()
```



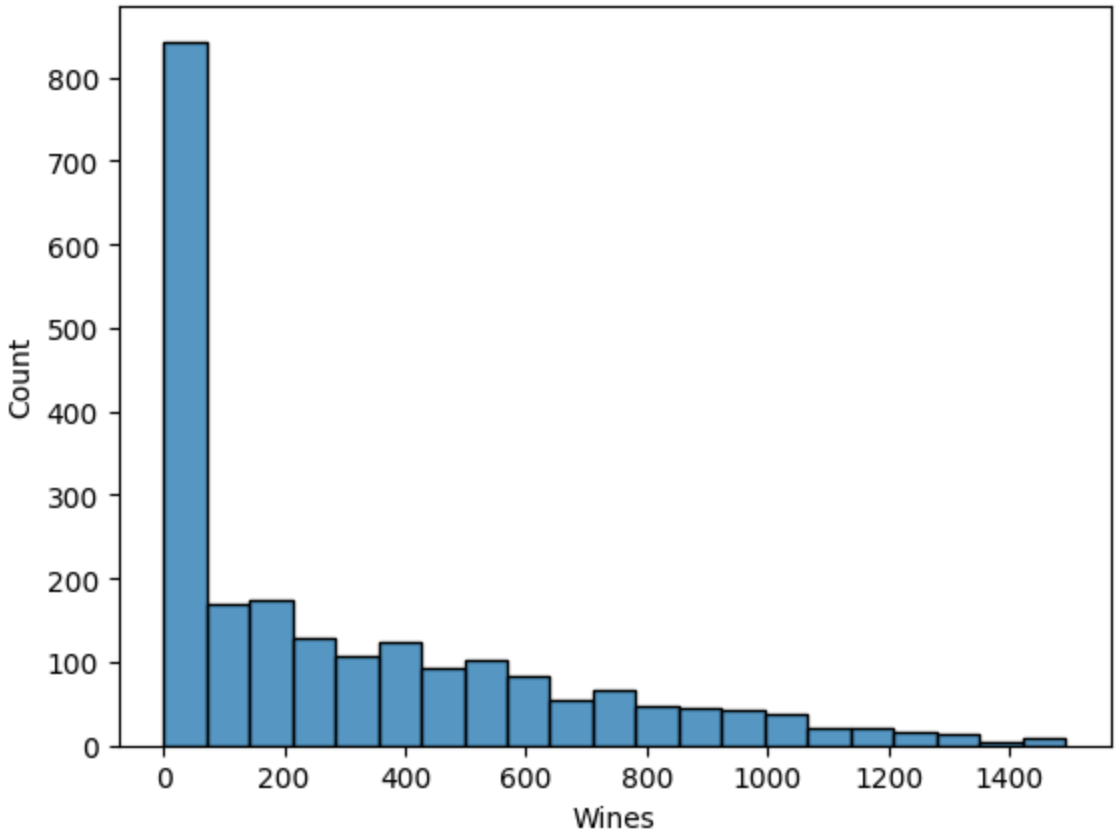
Items

In [84]:
```
item_sum = num_df.iloc[:,4:10].sum(axis=0)
pd.DataFrame(item_sum, columns=['# Sold'])
```

Out[84]:

	# Sold
Wines	673738
Fruits	58044
Meat	363788
Fish	82834
Sweets	59751
Gold	96654

In [85]:
```
sns.histplot(x="Wines", data=num_df)
plt.show()
```



In [86]:
```
item_df = pd.DataFrame({
    'Item': item_sum.index,
    'Amount Sold': item_sum.values
})

item_df['Percentage'] = item_df['Amount Sold'].apply(lambda x: round(((x/num_df[
```

```
# Sort by percentage (descending)
item_df.sort_values(['Percentage'], ascending=False, inplace=True)

item_df
```

Out[86]:

	Item	Amount Sold	Percentage
0	Wines	673738	50.47
2	Meat	363788	27.25
5	Gold	96654	7.24
3	Fish	82834	6.21
4	Sweets	59751	4.48
1	Fruits	58044	4.35

In [87]:
```
px.bar(item_df, x='Item', y='Amount Sold')
```

Modeling

In this notebook, we are going to use the K-Means algorithm to segment the customers into different clusters.

We will:

- Prepare the dataset for analysis by one hot encoding or label encoding the categorical data
- Choose the number of clusters by looking at the inertia and silhouette score
- Visualize the data from each cluster

In [88]:
```
df.head()
```

Out[88]:

	Education	Income	Kidhome	Teenhome	Recency	Wines	Fruits	Meat	Fish	Swee
0	Graduation	58138	0	0	58	635	88	546	172	
1	Graduation	46344	1	1	38	11	1	6	2	
2	Graduation	71613	0	0	26	426	49	127	111	
3	Graduation	26646	1	0	26	11	4	20	10	
4	PhD	58293	1	0	94	173	43	118	46	

5 rows × 32 columns

In [89]:
```
df.dtypes
```

Out[89]:
```
Education               object
Income                   int64
Kidhome                  int64
Teenhome                 int64
Recency                  int64
Wines                    int64
Fruits                   int64
Meat                     int64
Fish                     int64
Sweets                   int64
Gold                     int64
NumDealsPurchases        int64
NumWebPurchases          int64
NumCatalogPurchases      int64
NumStorePurchases        int64
NumWebVisitsMonth        int64
AcceptedCmp3             int64
AcceptedCmp4             int64
AcceptedCmp5             int64
AcceptedCmp1             int64
AcceptedCmp2             int64
Complain                 int64
Response                 int64
Age                      int64
Simplified_Education    object
Relationship_Status     object
Num_children             int64
Has_child                 bool
Total                    int64
days_joined              int64
Seniority              float64
Partner_Status           int64
dtype: object
```

Convert Categorical Data to Numerical

In [90]:
```
# One Hot Encode Education column
df['Edu'] = df['Education'].apply(lambda x: 1 if x == "Postgraduate" else 0)
df['Child'] = df['Has_child'].apply(lambda x: 0 if x == False else 1)

# Label Encode the Relationship Status column
df['Single'] = df['Relationship_Status'].apply(lambda x: 1 if x == 'Alone' else
df['Relationship'] = df['Relationship_Status'].apply(lambda x: 1 if x == 'Relati
```

Save new DataFrame with numerical columns

In [91]:
```
num_df = df[['Age', 'Edu', 'Single', 'Relationship', 'Income',
             'Seniority', 'Num_children', 'Child', 'Wines',
             'Fruits', 'Meat', 'Fish', 'Sweets', 'Gold', 'Total']]
```

In [92]:
```
num_df.head()
```

Out[92]:

	Age	Edu	Single	Relationship	Income	Seniority	Num_children	Child	Wines	Fru
0	67	0	1	0	58138	12.52	0	0	635	
1	70	0	1	0	46344	11.02	2	1	11	
2	59	0	0	1	71613	11.56	0	0	426	
3	40	0	0	1	26646	11.09	1	1	11	
4	43	0	0	1	58293	11.15	1	1	173	

In [93]:
```
# Reset index
num_df.reset_index(drop=True, inplace=True)
```

In [94]:
```
num_df.dtypes
```

Out[94]:
```
Age               int64
Edu               int64
Single            int64
Relationship      int64
Income            int64
Seniority       float64
Num_children      int64
Child             int64
Wines             int64
Fruits            int64
Meat              int64
Fish              int64
Sweets            int64
Gold              int64
Total             int64
dtype: object
```

In [95]:
```
correlation_df = num_df.corr()

#correlation_df
```

We want to group customers by how much they spend, so we will choose columns with high correlation values with `Total` column

In [96]:
```
# Strong positive correlation with 'Total'
pos_corr = correlation_df['Total'][(correlation_df['Total'] >= 0.5)]
pos_corr
```

Out[96]:
```
Income      0.823645
Wines       0.897616
Fruits      0.614031
Meat        0.857106
Fish        0.644740
Sweets      0.609521
Gold        0.529779
Total       1.000000
Name: Total, dtype: float64
```

In [97]:
```
# Strong negative correlation with 'Total'
neg_corr = correlation_df['Total'][(correlation_df['Total'] <= -0.5)]
neg_corr
```

Out[97]:
```
Num_children   -0.500347
Child          -0.522597
Name: Total, dtype: float64
```

NOTE

This is where I remove the `Child` column because it was heavily impacting my model during my initial attempt. `Num_children` is enough.

In [98]:
```
all_df = num_df[neg_corr.index.tolist() + pos_corr.index.tolist()].copy()

X = all_df.drop('Child', axis=1).copy()

X.head()
```

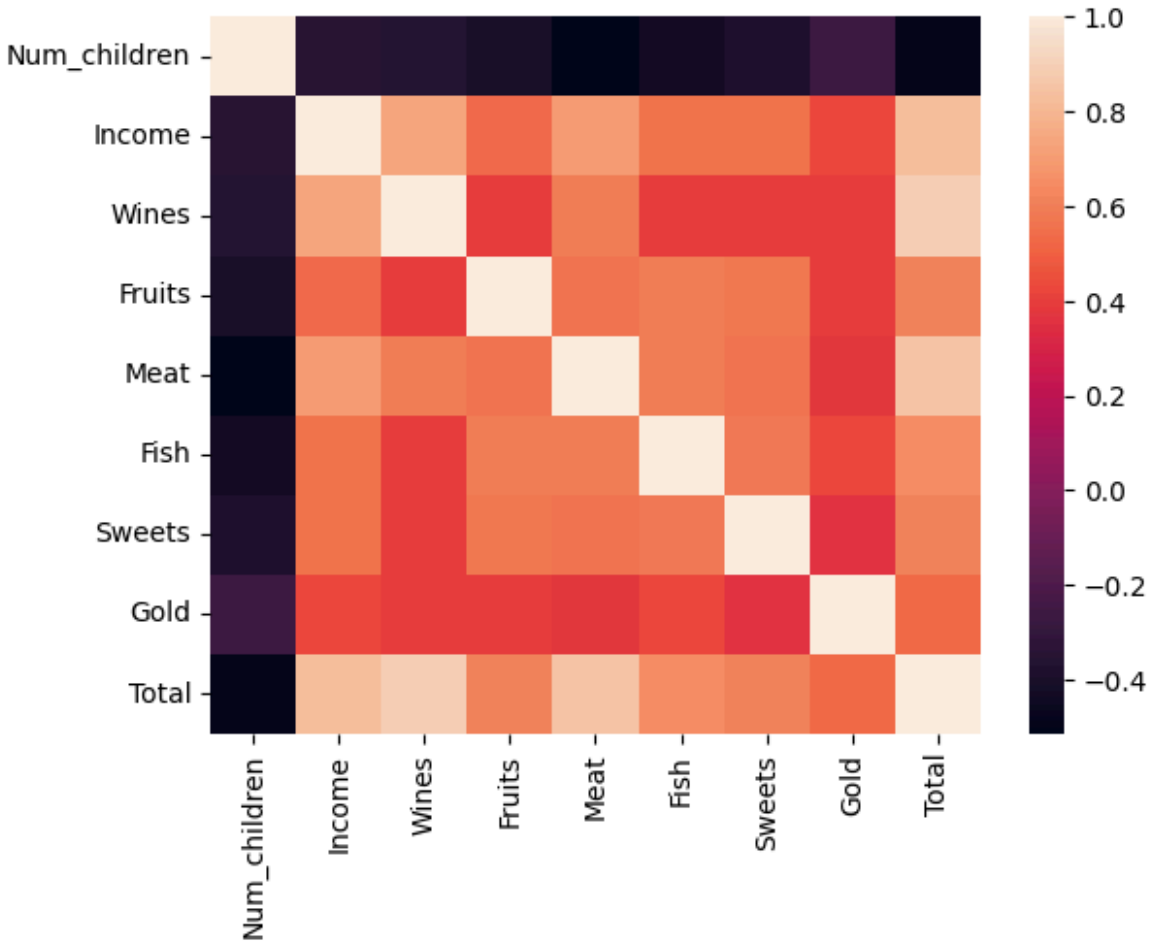
Out[98]:

	Num_children	Income	Wines	Fruits	Meat	Fish	Sweets	Gold	Total
0	0	58138	635	88	546	172	88	88	1617
1	2	46344	11	1	6	2	1	6	27
2	0	71613	426	49	127	111	21	42	776
3	1	26646	11	4	20	10	3	5	53
4	1	58293	173	43	118	46	27	15	422

Remove the singular items

In [99]:
```
# X = num_df[['Num_children', 'Child', 'Income', 'Total']]

# X.head()
```

In [100…
```
sns.heatmap(X.corr())
plt.show()
```



Scale the data between 0 and 1

In [101…
```
scaler = MinMaxScaler()
```

In [102…
```
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)

X_scaled.head()
```

Out[102…

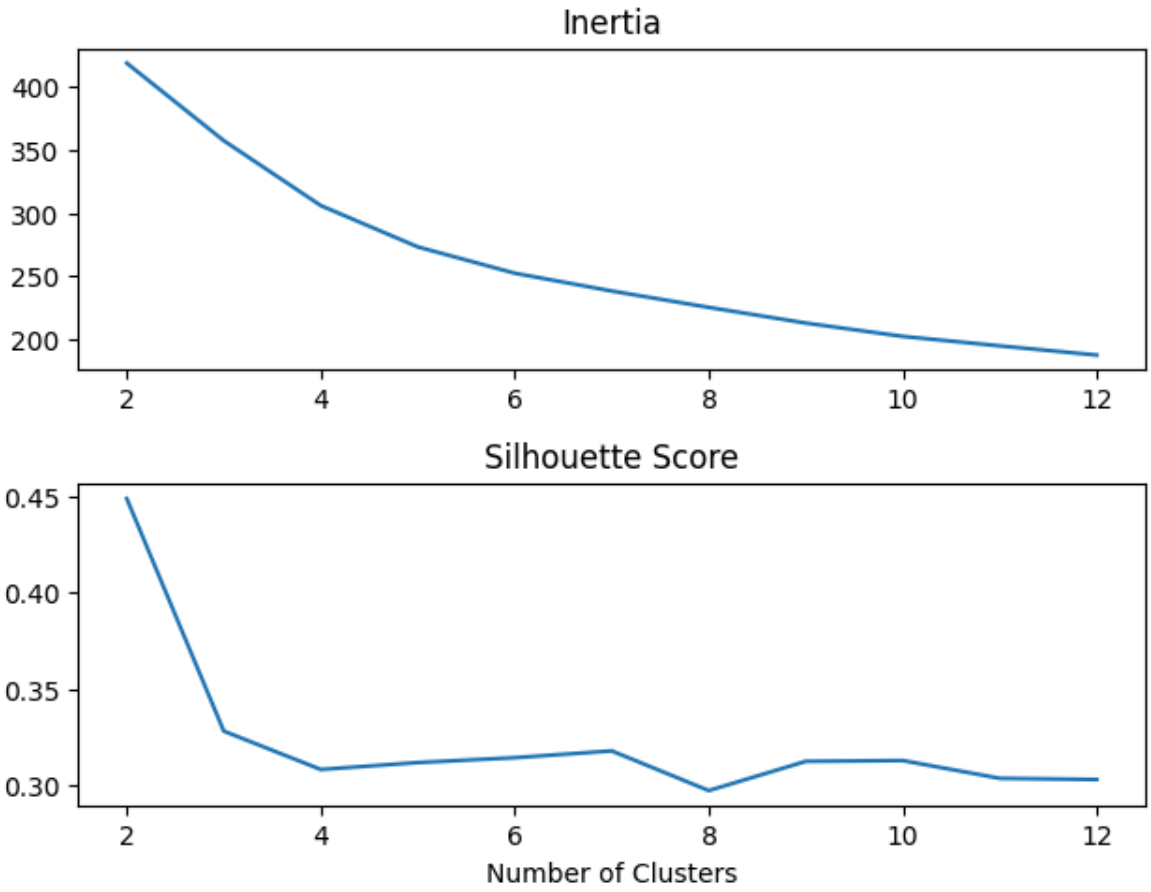
	Num_children	Income	Wines	Fruits	Meat	Fish	Sweets	Gold
0	0.000000	0.503625	0.425318	0.442211	0.316522	0.664093	0.335878	0.274143
1	0.666667	0.398325	0.007368	0.005025	0.003478	0.007722	0.003817	0.018692
2	0.000000	0.623933	0.285332	0.246231	0.073623	0.428571	0.080153	0.130841
3	0.333333	0.222456	0.007368	0.020101	0.011594	0.038610	0.011450	0.015576
4	0.333333	0.505009	0.115874	0.216080	0.068406	0.177606	0.103053	0.046729

K Means Clustering

We will use inertia and silhouette scores to determine what the best number of clusters is

In [103…
```
clusters = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
scores   = {}
inertias = {}

for i in clusters:
    kmeans = KMeans(n_clusters=i, n_init=50)  # , algorithm='full')
    kmeans.fit(X_scaled)
    scores[i] = silhouette_score(X_scaled, kmeans.labels_)
    inertias[i] = kmeans.inertia_
```

In [104…
```
# Two subplots
fig, (ax1, ax2) = plt.subplots(nrows=2)

# Inertia
ax1.plot(list(inertias.keys()), list(inertias.values()))
ax1.set_title('Inertia')

# Silhouette Score
ax2.plot(list(scores.keys()), list(scores.values()))
ax2.set_title('Silhouette Score')

# Spacing
fig.tight_layout()

# X-Axis Label
plt.xlabel('Number of Clusters')

plt.show()
```



Credit

The code in the following cell comes directly from the Scikit-learn documentation.

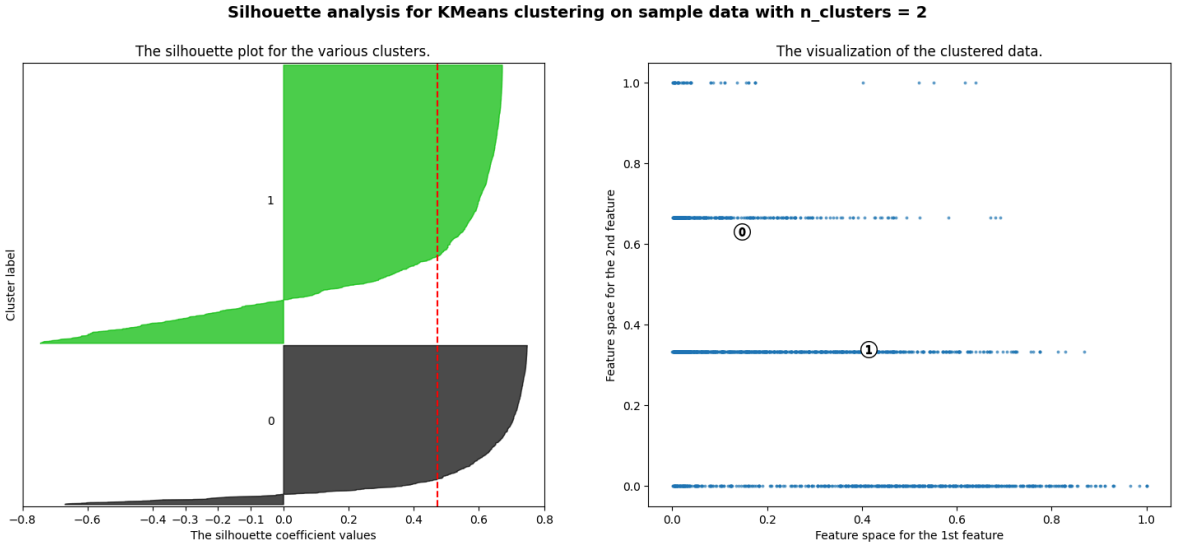
In [105…

```
cluster_range = [2, 3, 4, 5, 6, 7, 8, 9, 10]

for n_clusters in cluster_range:
    fig, (ax1, ax2) = plt.subplots(1,2)
    fig.set_size_inches(18, 7)

    ax1.set_xlim([-0.8, 0.8])

    ax1.set_ylim([0, len(X_scaled) + (n_clusters + 1) * 10])

    # Cluster
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(X_scaled)

    silhouette_avg = silhouette_score(X, cluster_labels)
    print(
        "For n_clusters =",
        n_clusters,
        "The average silhouette_score is :",
        silhouette_avg,
    )
```

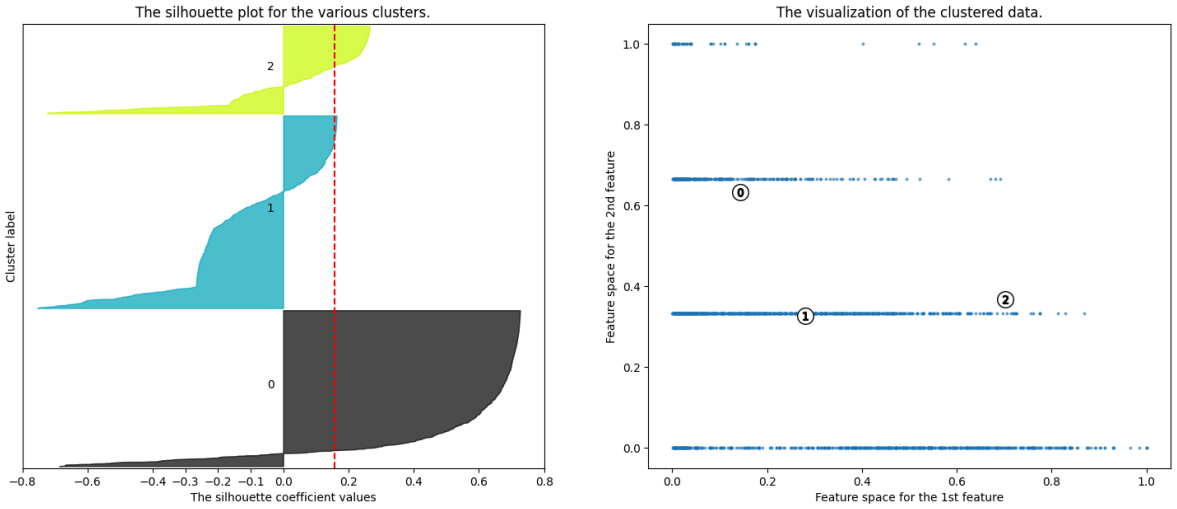
```
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(
            np.arange(y_lower, y_upper),
            0,
            ith_cluster_silhouette_values,
            facecolor=color,
            edgecolor=color,
            alpha=0.7,
        )

        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.8, -0.6, -0.4, -0.3, -0.2, -0.1, 0, 0.2, 0.4, 0.6, 0.8])

    # 2nd Plot showing the actual clusters formed
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(
        X_scaled['Total'], X_scaled['Num_children'], marker=".", s=30, lw=0, alp
    )

    # Labeling the clusters
    centers = clusterer.cluster_centers_
    # Draw white circles at cluster centers
    ax2.scatter(
        centers[:, 0],
        centers[:, 1],
        marker="o",
        c="white",
        alpha=1,
        s=200,
```

```
        edgecolor="k",
    )

    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker="$%d$" % i, alpha=1, s=50, edgecolor="k")

    ax2.set_title("The visualization of the clustered data.")
    ax2.set_xlabel("Feature space for the 1st feature")
    ax2.set_ylabel("Feature space for the 2nd feature")

    plt.suptitle(
        "Silhouette analysis for KMeans clustering on sample data with n_cluster
        % n_clusters,
        fontsize=14,
        fontweight="bold",
    )

    plt.show()
```

For n_clusters = 2 The average silhouette_score is : 0.47236915421742814

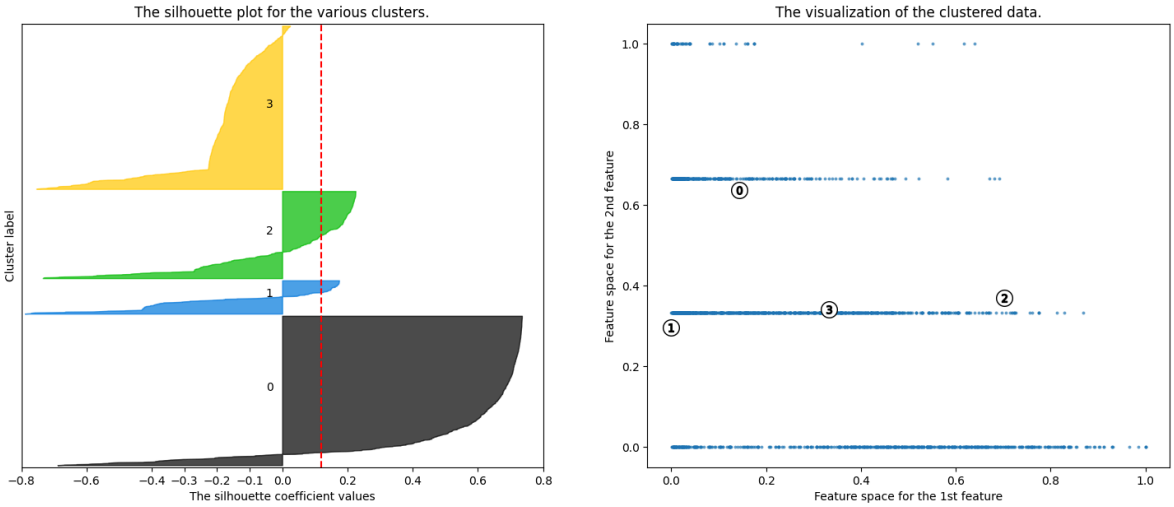


For n_clusters = 3 The average silhouette_score is : 0.15816357399607045



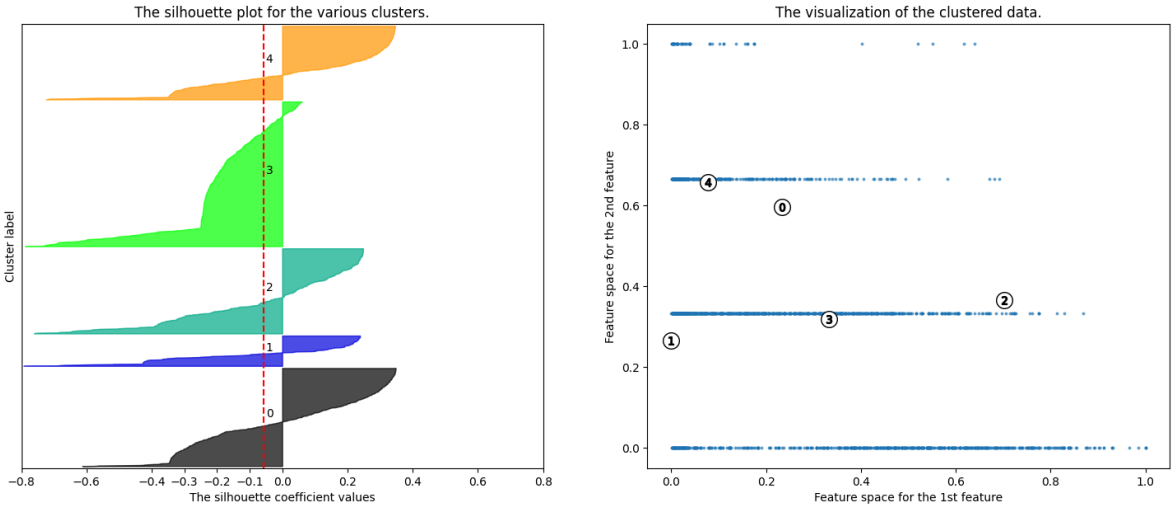
For n_clusters = 4 The average silhouette_score is : 0.11951188620164022

Silhouette analysis for KMeans clustering on sample data with n_clusters = 4



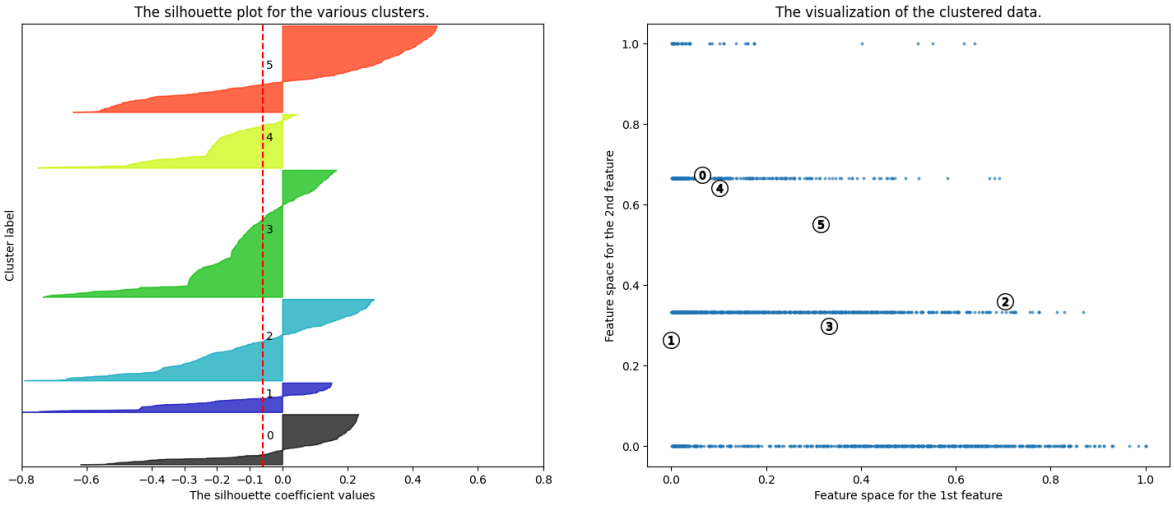
For n_clusters = 5 The average silhouette_score is : -0.05580554364327878

Silhouette analysis for KMeans clustering on sample data with n_clusters = 5

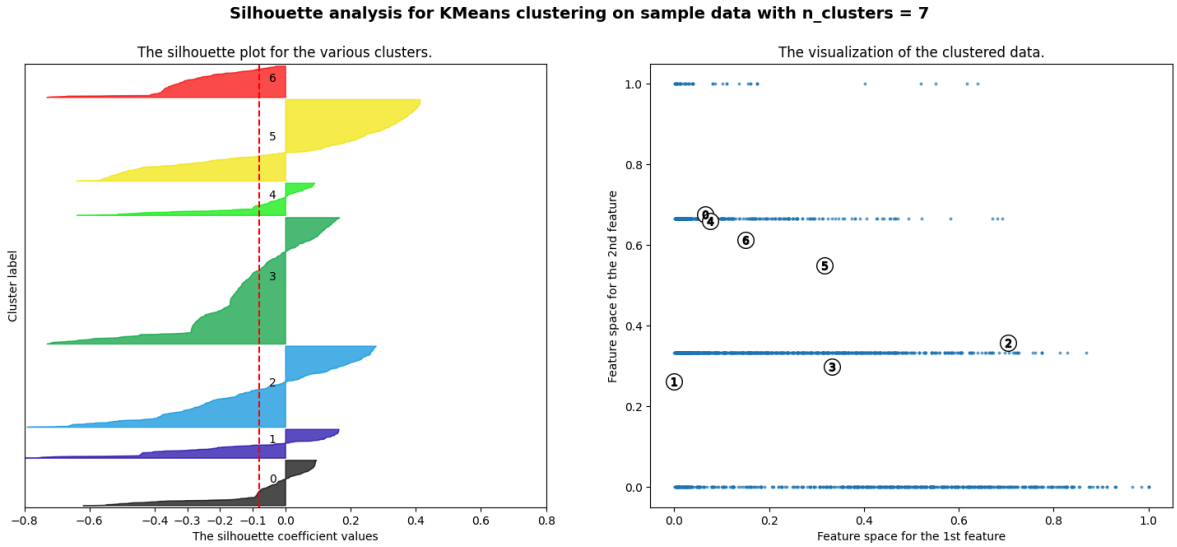


For n_clusters = 6 The average silhouette_score is : -0.058883794277863166

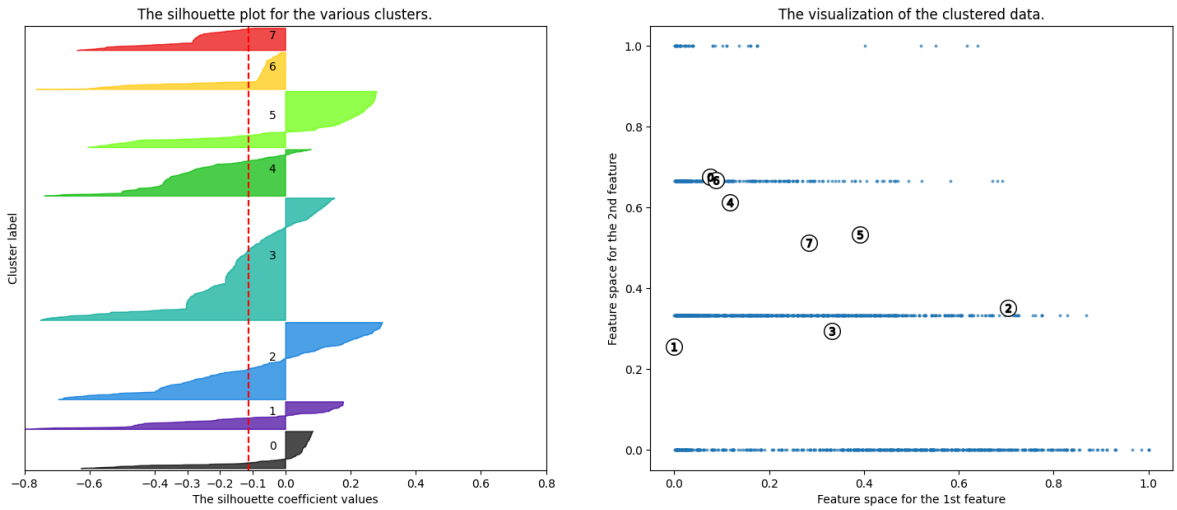
Silhouette analysis for KMeans clustering on sample data with n_clusters = 6



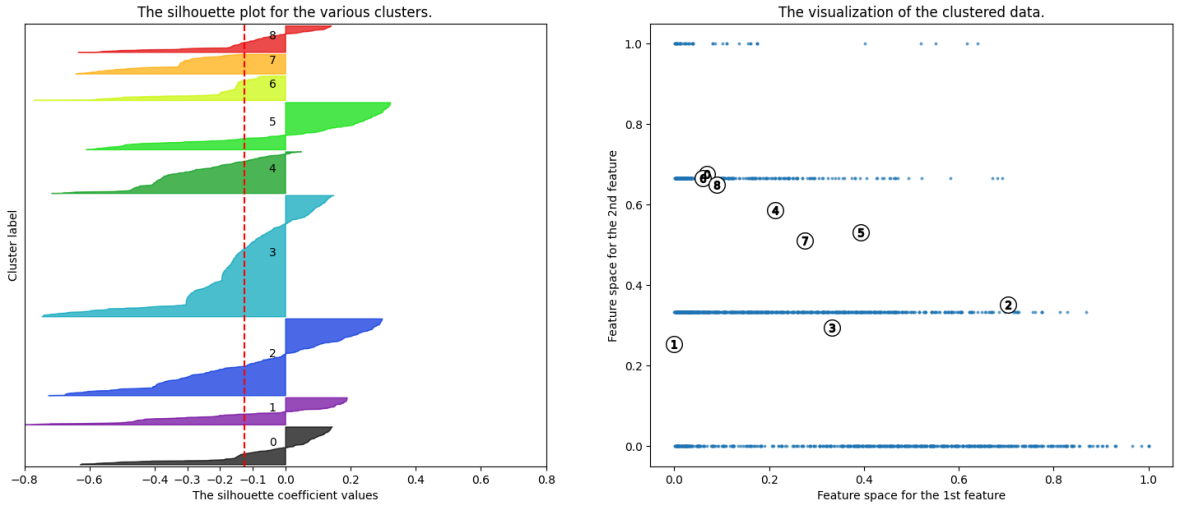
For n_clusters = 7 The average silhouette_score is : -0.0800796774776505



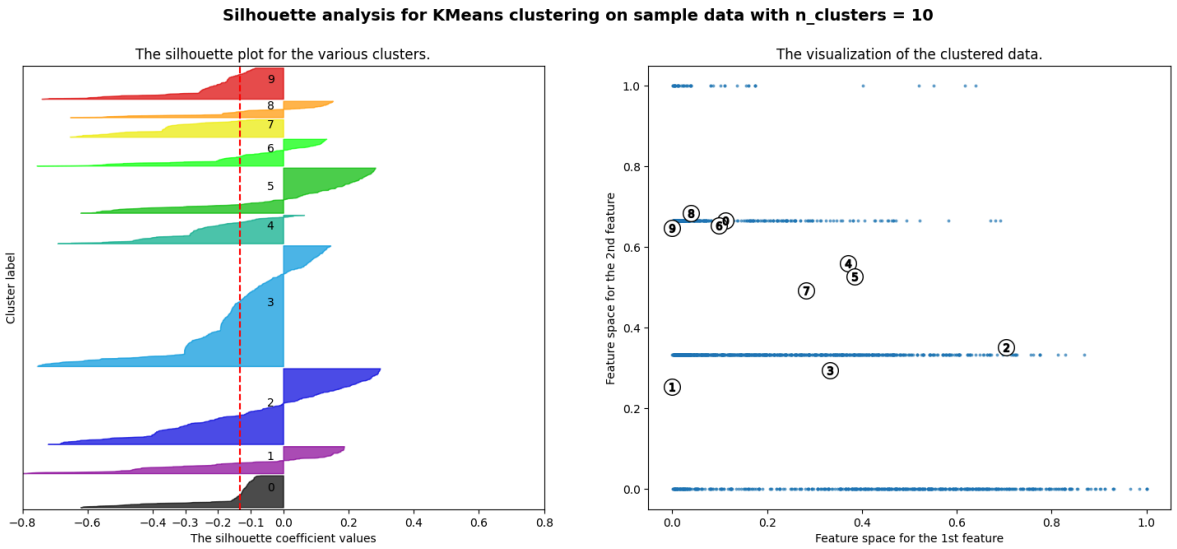
For n_clusters = 8 The average silhouette_score is : -0.1124454308903397



For n_clusters = 9 The average silhouette_score is : -0.12446301085041107



For n_clusters = 10 The average silhouette_score is : -0.13218092734060352



3 clusters appears to be the best, according to the inertia and silhouette score

In [106…
```
kmeans = KMeans(n_clusters=3, n_init=100, algorithm='lloyd', random_state=10)
kmeans.fit(X_scaled)
```

Out[106…

KMeans

```
KMeans(n_clusters=3, n_init=100, random_state=10)
```

Add the labels to the datasets

In [107…
```
X_scaled['labels'] = kmeans.labels_
X['labels'] = kmeans.labels_
```

In [108…
```
X.head()
```

Out[108…

	Num_children	Income	Wines	Fruits	Meat	Fish	Sweets	Gold	Total	labels
0	0	58138	635	88	546	172	88	88	1617	2
1	2	46344	11	1	6	2	1	6	27	1
2	0	71613	426	49	127	111	21	42	776	0
3	1	26646	11	4	20	10	3	5	53	1
4	1	58293	173	43	118	46	27	15	422	0

In [109…
```
X_scaled.head()
```

Out[109…

	Num_children	Income	Wines	Fruits	Meat	Fish	Sweets	Gold
0	0.000000	0.503625	0.425318	0.442211	0.316522	0.664093	0.335878	0.274143
1	0.666667	0.398325	0.007368	0.005025	0.003478	0.007722	0.003817	0.018692
2	0.000000	0.623933	0.285332	0.246231	0.073623	0.428571	0.080153	0.130841
3	0.333333	0.222456	0.007368	0.020101	0.011594	0.038610	0.011450	0.015576
4	0.333333	0.505009	0.115874	0.216080	0.068406	0.177606	0.103053	0.046729

In [110…

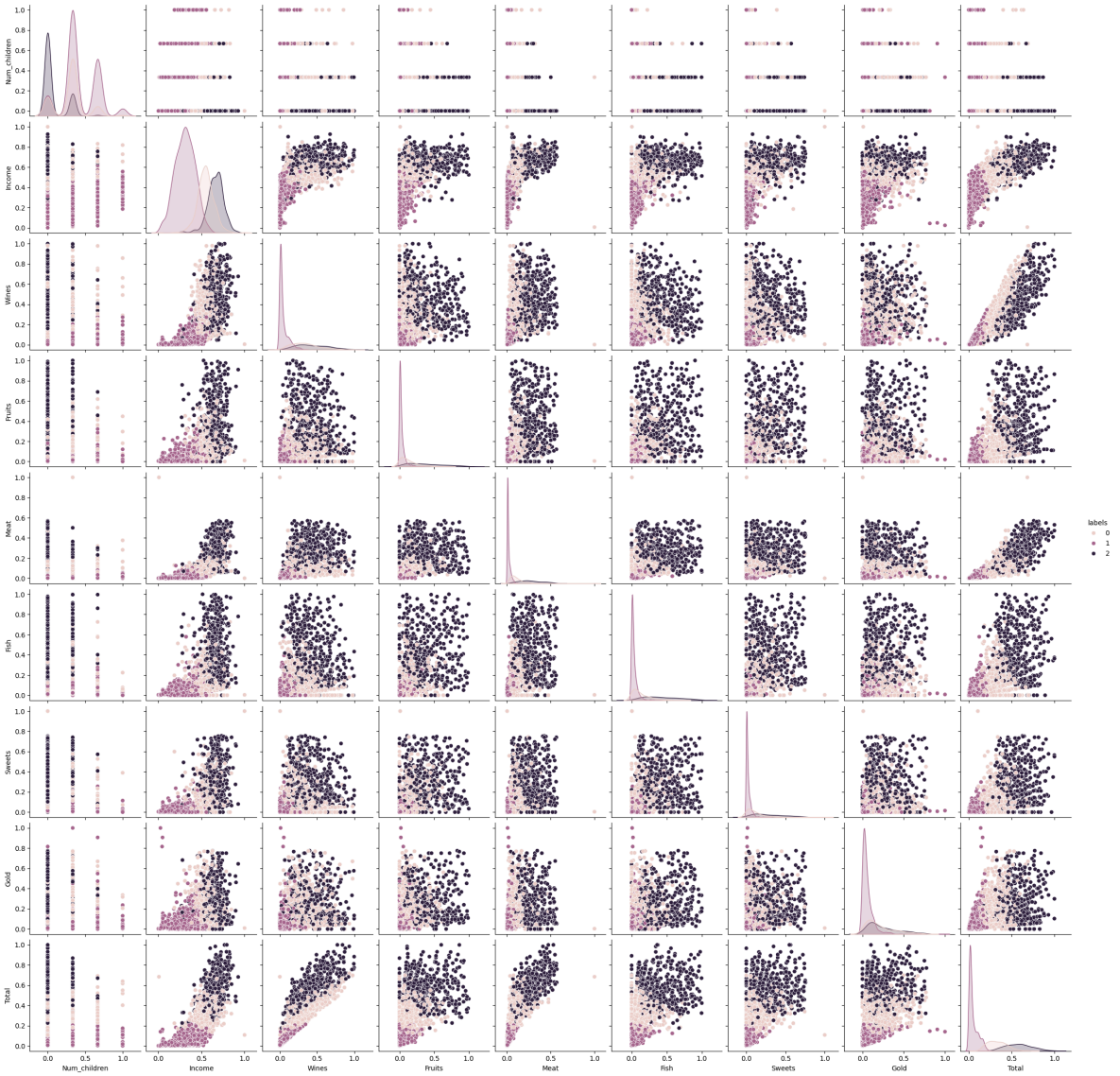
```
X_scaled.corr()['labels'].sort_values(ascending=False)
```

Out[110…

```
labels          1.000000
Fish            0.460942
Meat            0.417765
Fruits          0.402861
Sweets          0.363023
Total           0.321392
Income          0.185322
Wines           0.127312
Gold            0.064167
Num_children   -0.296996
Name: labels, dtype: float64
```

In [111…

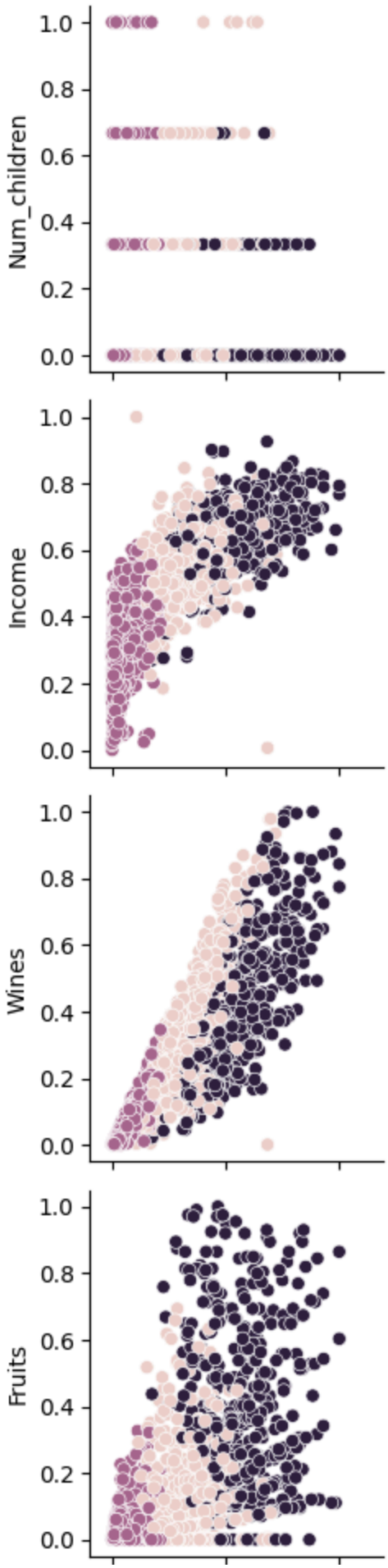
```
sns.pairplot(X_scaled, hue='labels')
plt.show()
```

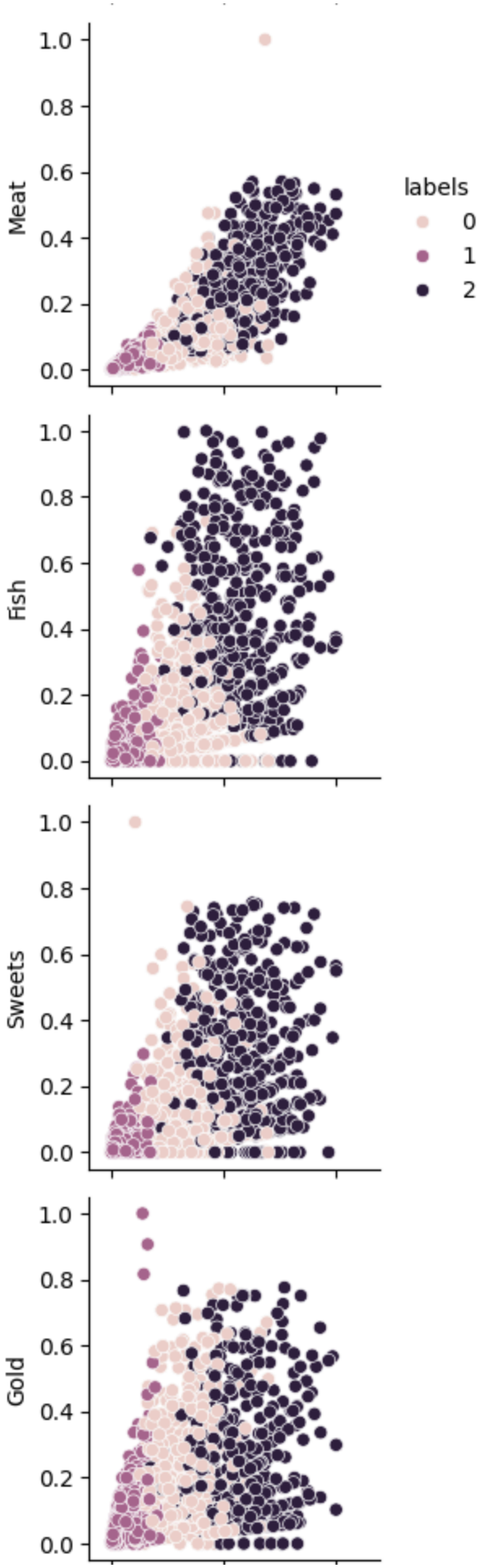


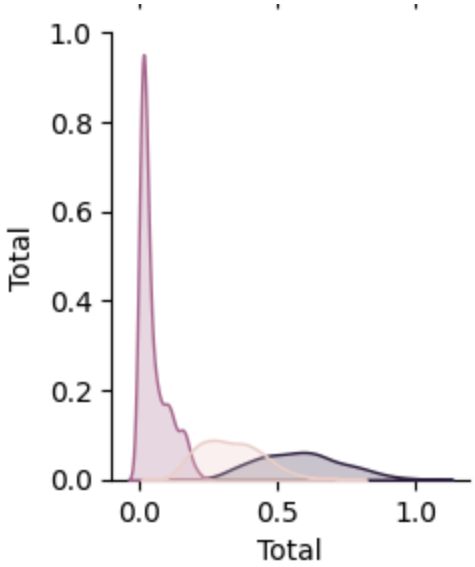
Let's take a closer look at the `Total` column

In [112…

```
sns.pairplot(X_scaled, hue='labels', x_vars=['Total'])
plt.show()
```


Num_children
Income
Wines
Fruits
0.0
0.2
0.4
0.6
0.8
1.0


Meat
Fish
Sweets
Gold
1.0
0.8
0.6
0.4
0.2
0.0
labels
0
1
2



Analyze the Results

We've cleaned the data and trained a K Means model on that data. Now it's time to explore the results.

Size of the Clusters

Generally speaking, we're looking for clusters of similar sizes

In [113…
```
df.head()
```

Out[113…

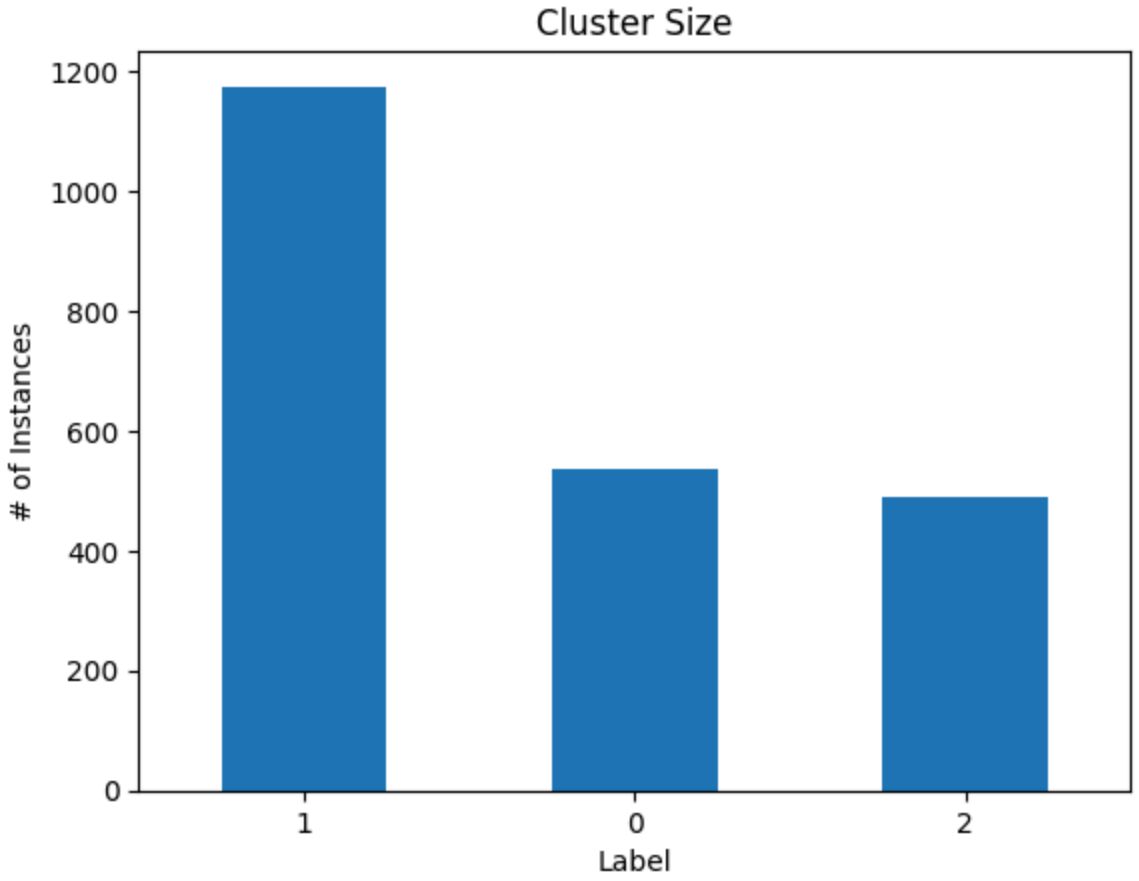
	Education	Income	Kidhome	Teenhome	Recency	Wines	Fruits	Meat	Fish	Swee
0	Graduation	58138	0	0	58	635	88	546	172	
1	Graduation	46344	1	1	38	11	1	6	2	
2	Graduation	71613	0	0	26	426	49	127	111	
3	Graduation	26646	1	0	26	11	4	20	10	
4	PhD	58293	1	0	94	173	43	118	46	

5 rows × 36 columns

In [114…
```
#df['cluster_labels'] = kmeans.labels_
df['labels'] = kmeans.labels_
```

In [115…
```
# Bar graph of total instances in each label
#fig = df['cluster_labels'].value_counts().plot.bar()
fig = df['labels'].value_counts().plot.bar()
```

```
# Set axes to horizontal
plt.tick_params(axis='x', labelrotation=0)

# Chart titles
plt.title('Cluster Size')
plt.xlabel('Label')
plt.ylabel('# of Instances')

plt.show()
```



In [116…
```
top_df = df[['Total', 'Income', 'Meat', 'Wines', 'Fish', 'labels']].copy()
```

Divide DataFrame

Create three smaller DataFrames; one for each label.

In [117…
```
# Group 0
label0 = top_df[top_df['labels'] == 0].copy()
label0.drop(columns=['labels'], inplace=True)
label0.reset_index(inplace=True, drop=True)

# Group 1
label1 = top_df[top_df['labels'] == 1].copy()
label1.drop(columns=['labels'], inplace=True)
label1.reset_index(inplace=True, drop=True)
```

```
# Group 2
label2 = top_df[top_df['labels'] == 2].copy()
label2.drop(columns=['labels'], inplace=True)
label2.reset_index(inplace=True, drop=True)
```

In [118…
```
label0.describe()
```

Out[118…

	Total	Income	Meat	Wines	Fish
count	537.000000	537.000000	537.000000	537.000000	537.000000
mean	851.821229	63039.093110	186.294227	506.962756	34.595903
std	291.831219	10967.238927	143.604138	280.332942	34.783534
min	277.000000	2447.000000	3.000000	1.000000	0.000000
25%	615.000000	56559.000000	94.000000	293.000000	10.000000
50%	819.000000	62845.000000	149.000000	478.000000	23.000000
75%	1038.000000	69508.000000	238.000000	656.000000	52.000000
max	1829.000000	113734.000000	1725.000000	1462.000000	188.000000

In [119…
```
label1.describe()
```

Out[119…

	Total	Income	Meat	Wines	Fish
count	1174.000000	1174.000000	1174.000000	1174.000000	1174.000000
mean	134.417376	36232.798126	29.601363	67.324532	8.189097
std	129.342700	12727.878116	32.385643	88.793809	12.887209
min	5.000000	1730.000000	0.000000	0.000000	0.000000
25%	42.250000	27100.000000	8.000000	9.000000	2.000000
50%	75.000000	36425.500000	17.000000	27.000000	4.000000
75%	197.250000	45578.250000	38.750000	88.750000	11.000000
max	595.000000	70844.000000	217.000000	516.000000	150.000000

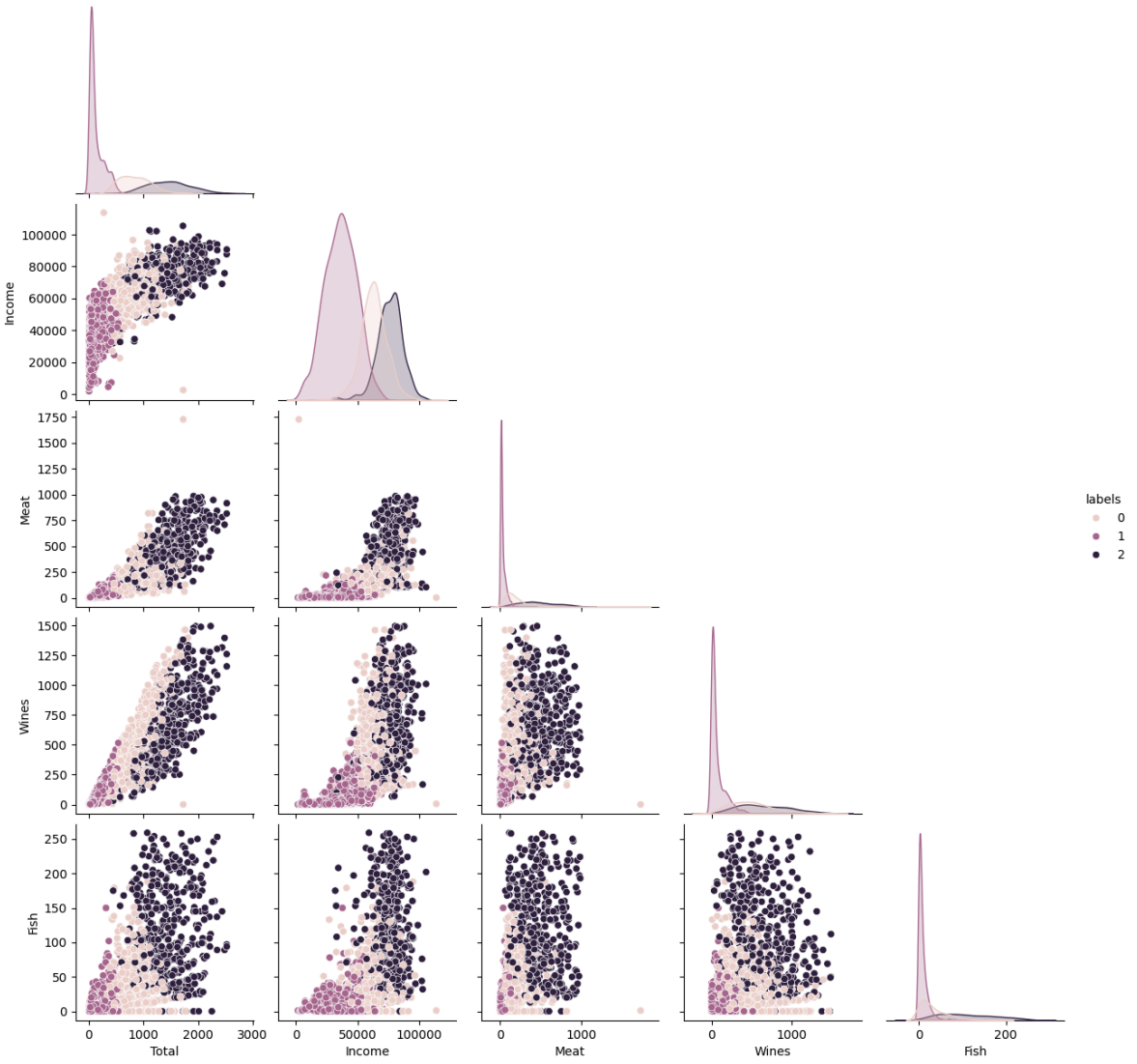
In [120…
```
label2.describe()
```

Out[120…

	Total	Income	Meat	Wines	Fish
count	490.000000	490.000000	490.000000	490.000000	490.000000
mean	1468.520408	75909.848980	467.338776	658.081633	111.514286
std	388.441036	10413.687668	228.606679	320.821315	64.355899
min	449.000000	31907.000000	64.000000	33.000000	0.000000
25%	1170.000000	70046.250000	282.000000	398.000000	58.250000
50%	1457.500000	76504.500000	430.000000	605.500000	103.500000
75%	1722.000000	82343.500000	628.000000	900.500000	160.000000
max	2525.000000	105471.000000	984.000000	1493.000000	259.000000

In [121…

```
sns.pairplot(top_df, hue='labels', corner=True)
plt.show()
```

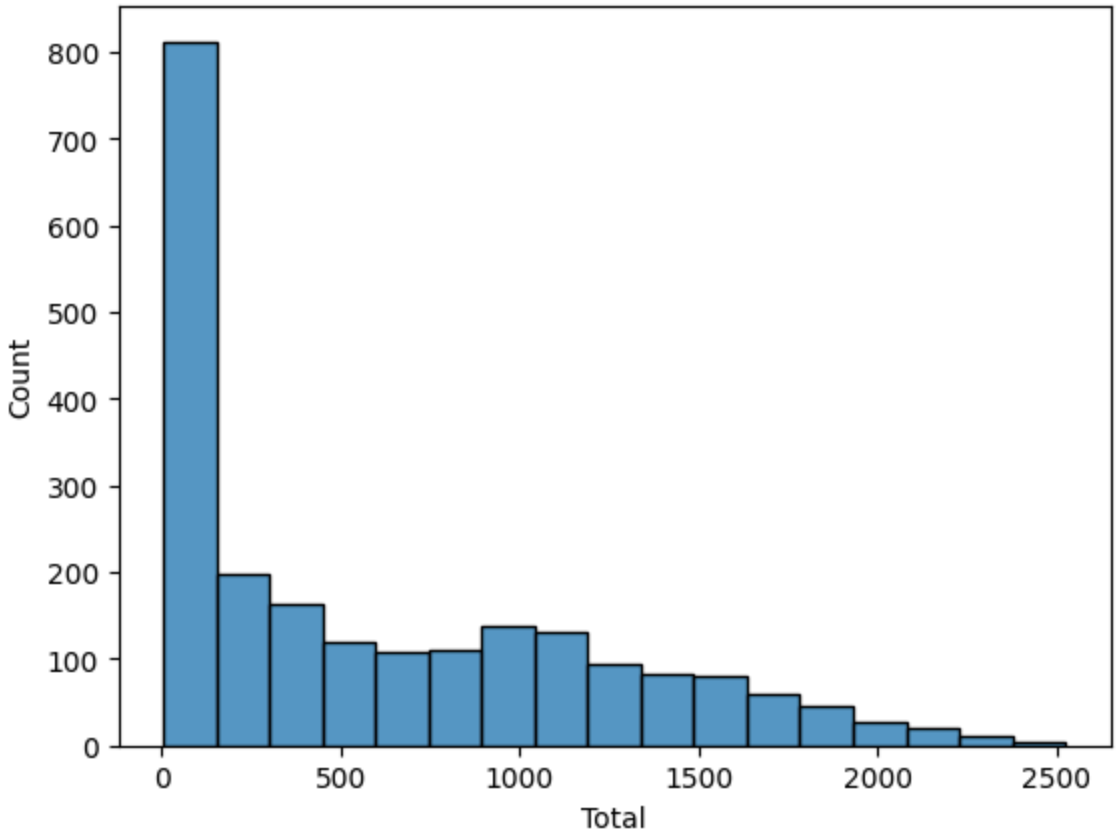


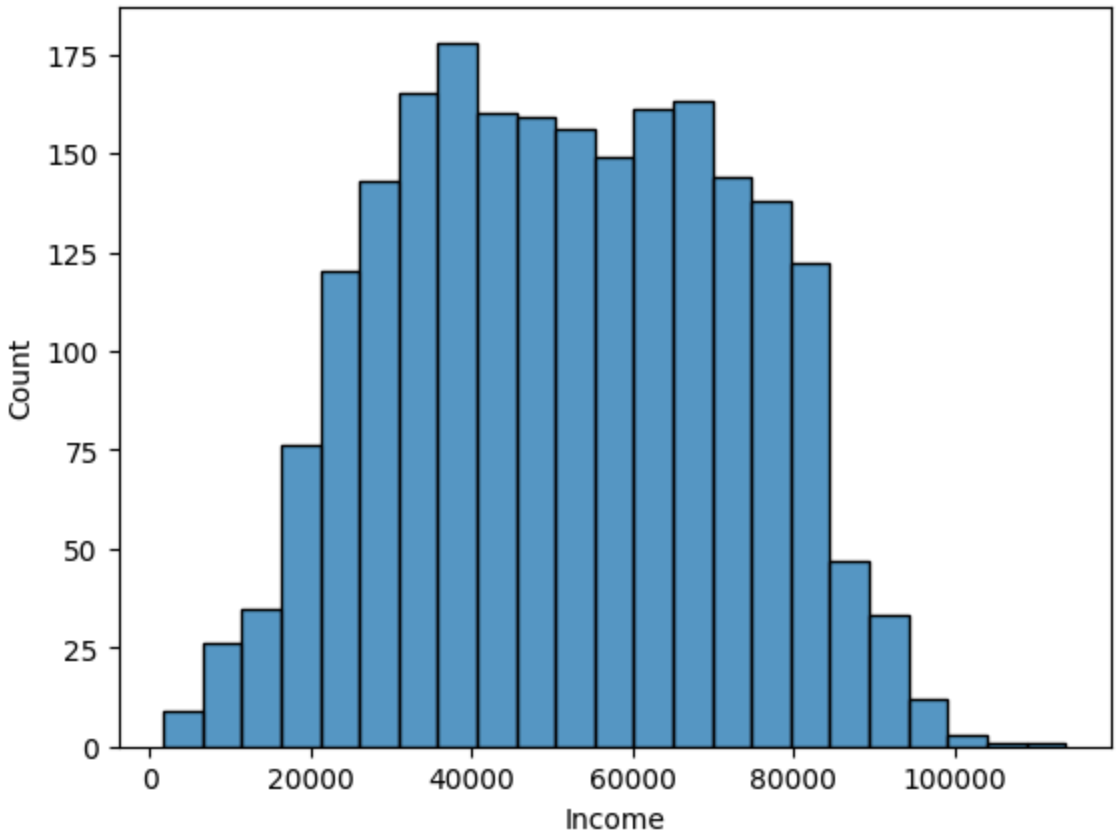
Initial Findings

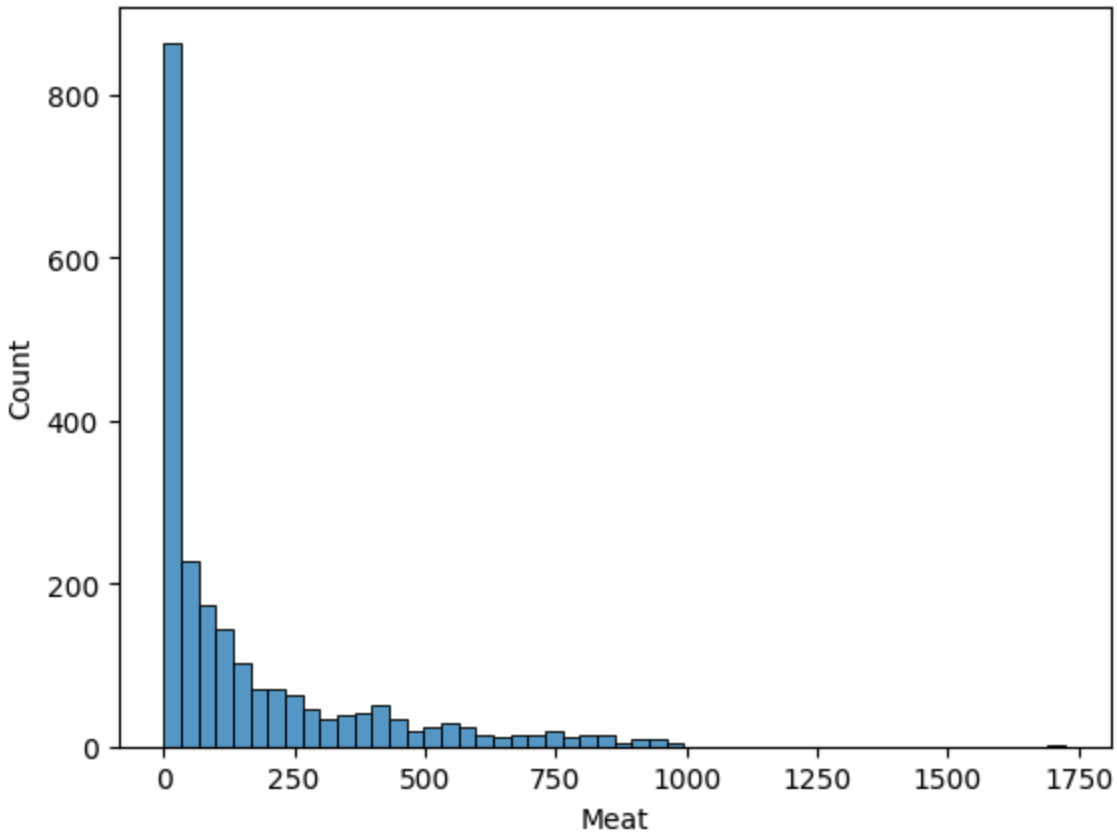
- Unbalanced
  - label 0 is as much as labels 1 and 2 combined
- Distinct clusters
  - it seems like the clusters are separated as low, medium, and high

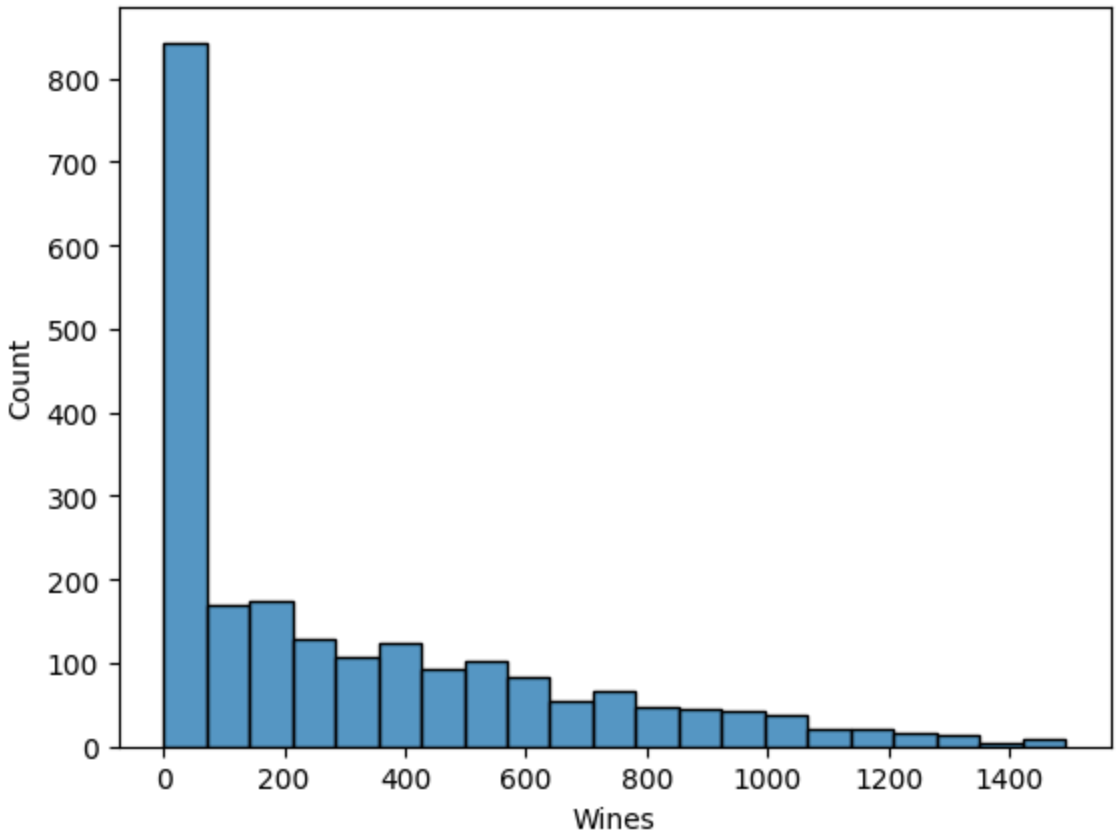
In [122…
```
#sns.histplot(data = top_df, hue='labels')
```

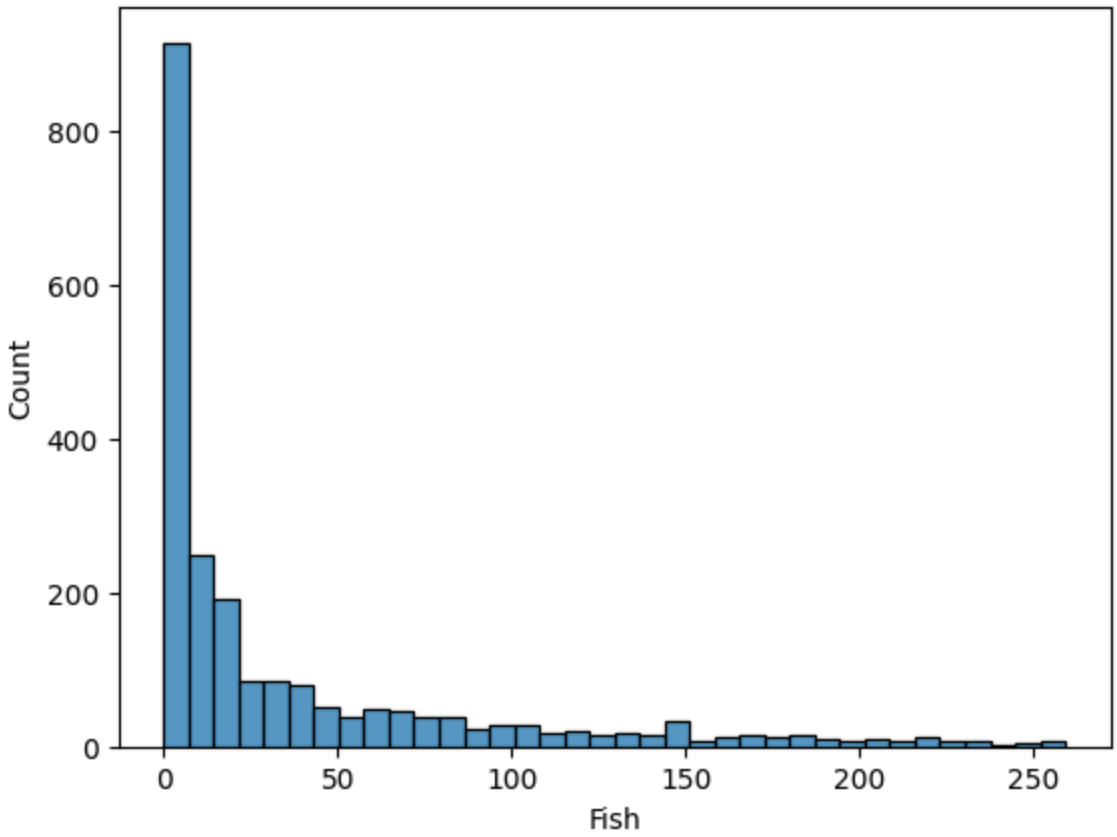
In [123…
```
# Histogram
for col in top_df.columns:
    sns.histplot(top_df[col])
    plt.show()
```

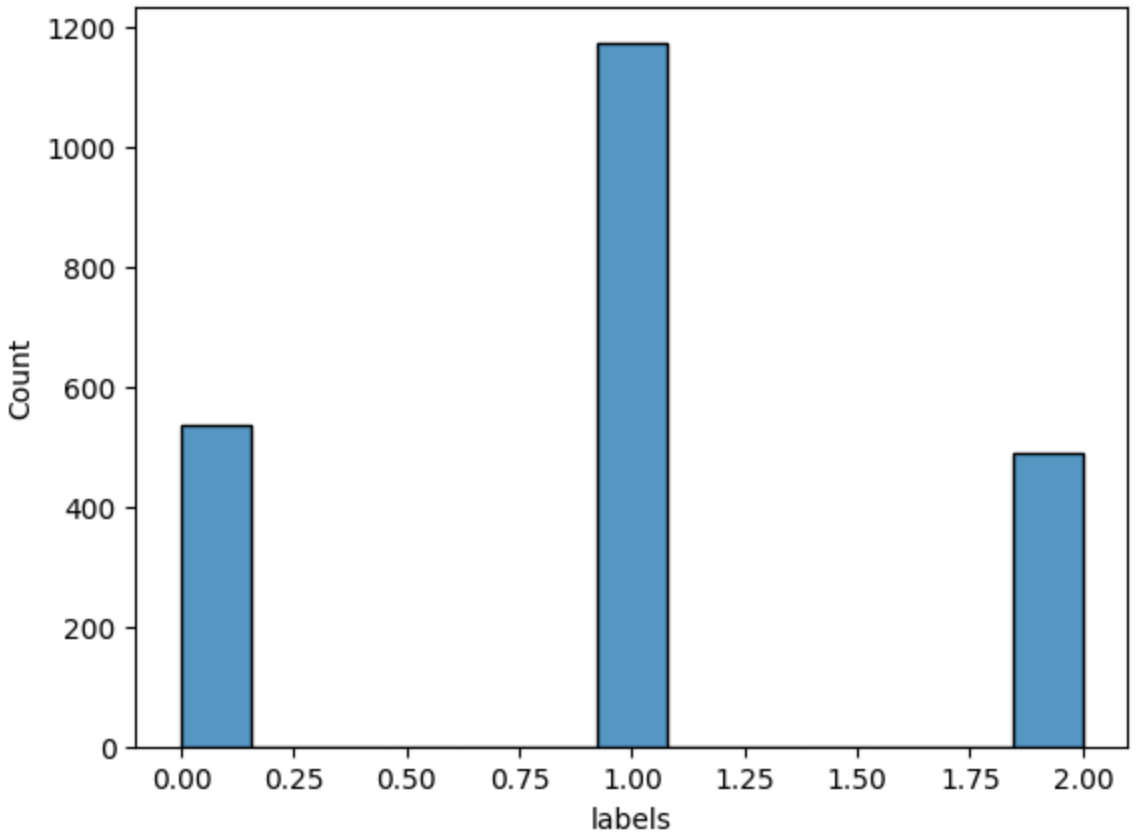



175
150
125
100
75
50
25
0
Count
0
20000
40000
60000
80000
100000
Income


800
600
400
200
0
Count
0
250
500
750
1000
1250
1500
1750
Meat


Count
800
700
600
500
400
300
200
100
0
0
200
400
600
800
1000
1200
1400
Wines


Count
800
600
400
200
0
0
50
100
150
200
250
Fish



In [124…
```
top_df.head()
```

Out[124…

	Total	Income	Meat	Wines	Fish	labels
0	1617	58138	546	635	172	2
1	27	46344	6	11	2	1
2	776	71613	127	426	111	0
3	53	26646	20	11	10	1
4	422	58293	118	173	46	0

TO DO

- Scale the data with `MinMaxScaler`
- Check out each histogram in greater detail
  - Use **Plotly** to make the graph interactive

In [125…
```
columns_to_scale = ['Income', 'Meat', 'Wines', 'Fish', 'Total']
df_to_scale = top_df[columns_to_scale].copy()
scaled_data = scaler.fit_transform(df_to_scale)
df_scaled = pd.DataFrame(scaled_data, columns=columns_to_scale)
```

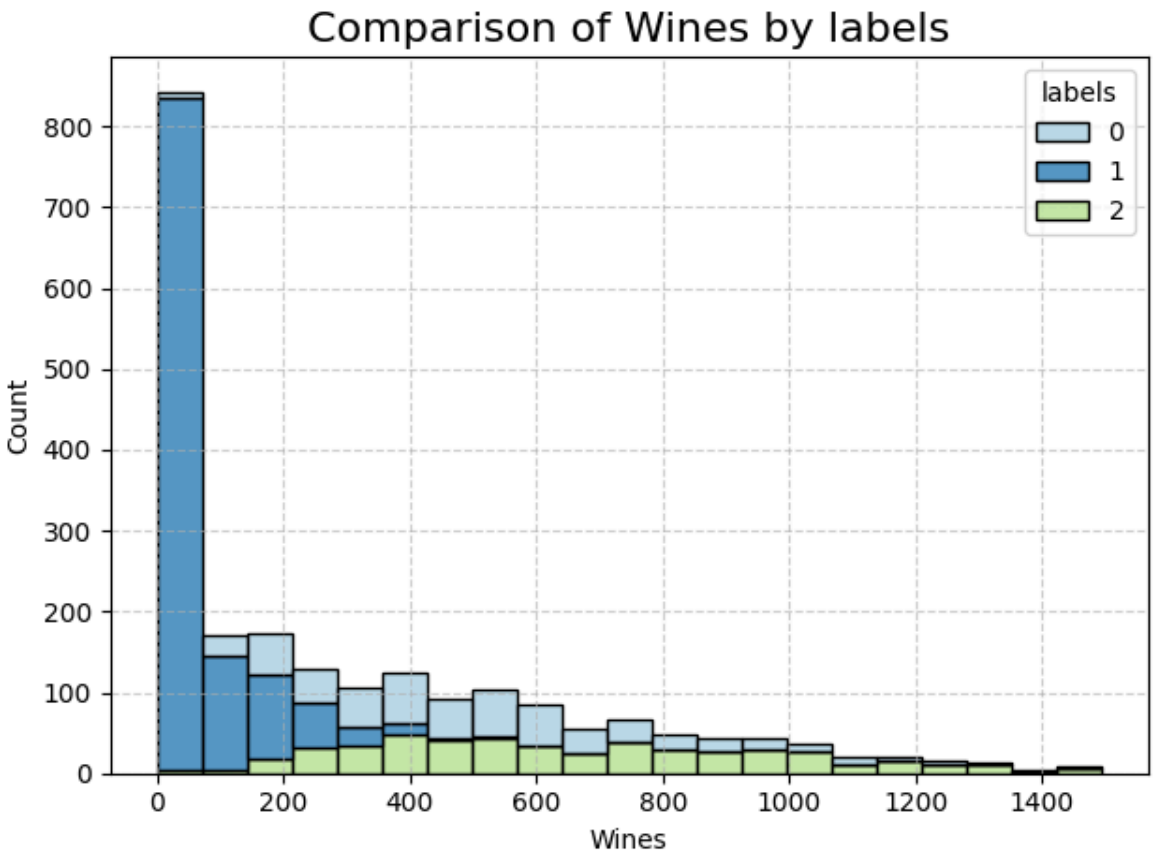
In [126…
```
df_scaled = pd.DataFrame(scaled_data, columns=columns_to_scale)
```

In [127…
```
top_df.head()
```

Out[127…

	Total	Income	Meat	Wines	Fish	labels
0	1617	58138	546	635	172	2
1	27	46344	6	11	2	1
2	776	71613	127	426	111	0
3	53	26646	20	11	10	1
4	422	58293	118	173	46	0

In [128…
```
fig = sns.histplot(top_df, x='Wines', hue='labels', multiple="stack", palette="P

# Improve the grid and layout
plt.grid(True, linestyle='--', alpha=0.6)
plt.title('Comparison of Wines by labels', fontsize=16)
# Show the plot
plt.tight_layout()  # Automatically adjusts subplots for better spacing

plt.show()
```



In [129…
```
total = top_df['Total'].copy()
meat = top_df['Meat'].copy()
```

```
wine = top_df['Wines'].copy()
```

In [130…

```
# Plot histograms with KDE and better aesthetics
plt.figure(figsize=(10, 6))  # Increase figure size for better visualization

sns.histplot(total, color='royalblue', label='Total', stat='density', alpha=0.6,
sns.histplot(meat, color='darkorange', label='Meat', stat='density', alpha=0.6,
sns.histplot(wine, color='forestgreen', label='Wines', stat='density', alpha=0.6

# Add a legend to differentiate the histograms
plt.legend(title='Categories')

# Title and labels for the axes
plt.title('Comparison of Distributions: Total, Meat, and Wines', fontsize=16)
plt.xlabel('Value', fontsize=12)
plt.ylabel('Density', fontsize=12)

# Improve the grid and layout
plt.grid(True, linestyle='--', alpha=0.6)

# Show the plot
plt.tight_layout()  # Automatically adjusts subplots for better spacing
plt.show()
```