Lab9: PCA + Clustering

In [67]:
```
ID = 1631938
Name = 'MIN SOE HTUT'
```

Task 1: Pre-done: Loading the data and preparing the data

In [68]:
```
from datetime import datetime
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv('https://raw.githubusercontent.com/bpfa/data_for_compx310_2024/main/marketing_campaign.csv', sep=',')
# Display the first few rows to inspect the data
print(df.columns)
```

```
Index(['ID', 'Year_Birth', 'Education', 'Marital_Status', 'Income', 'Kidhom
e',
       'Teenhome', 'Dt_Customer', 'Recency', 'MntWines', 'MntFruits',
       'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts',
       'MntGoldProds', 'NumDealsPurchases', 'NumWebPurchases',
       'NumCatalogPurchases', 'NumStorePurchases', 'NumWebVisitsMonth',
       'AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1',
       'AcceptedCmp2', 'Complain', 'Z_CostContact', 'Z_Revenue', 'Response'],
      dtype='object')
```

Creating new features for customer profiles

In [69]:
```
#  Creates a new field to store the age of the customer
df['Age']=2022-df['Year_Birth']

#  Recodes the customer's education level to numeric form (0: high-school, 1: diploma, 2: bachelors, 3: masters, and 4: doctorates)
df["Education"].replace({"Basic":0,"2n Cycle":1, "Graduation":2, "Master":3, "PhD":4},inplace=True)

#  Recodes the customer's marital status to numeric form (0: not living with a partner, 1: living with a partner)
df['Marital_Status'].replace({"Married":1, "Together":1, "Absurd":0, "Widow":0, "YOLO":0, "Divorced":0, "Single":0,"Alone":0},inplace=True)

#  creates a new field to store the number of children in the household
df['Children']=df['Kidhome']+df['Teenhome']

#  creates a new field to store the household size
df['Family_Size']=df['Marital_Status']+df['Children']+1
```

Creating new features for customer spending behavior

In [70]:
```
#  creates a new field to store the total spending of the customer
df['Total_Spending']=df["MntWines"]+ df["MntFruits"]+ df["MntMeatProducts"]+ df["MntFishProducts"]+ df["MntSweetProducts"]+ df["MntGoldProds"]

#  creates subsequent fields to store the spending proportion for each product by the customer
df['Prop_Wines']=df["MntWines"]/df["Total_Spending"]
df['Prop_Fruits']=df["MntFruits"]/df["Total_Spending"]
df['Prop_MeatProducts']=df["MntMeatProducts"]/df["Total_Spending"]
df['Prop_FishProducts']=df["MntFishProducts"]/df["Total_Spending"]
df['Prop_SweetProducts']=df["MntSweetProducts"]/df["Total_Spending"]
df['Prop_GoldProds']=df["MntGoldProds"]/df["Total_Spending"]
```

Additional customer features

In [71]:
```
# Converts the customer start date into a useful feature representing how long they have been a customer
df['Dt_Customer'] = pd.to_datetime(df['Dt_Customer'], dayfirst=True)
# Get the current date
today = datetime.today()
# Calculate how long each customer has been with the company
df['Days_as_Customer'] = (today - df['Dt_Customer']).dt.days
# Combine the number of offers a customer responded to into a single feature
df['Offers_Responded_To'] = df['AcceptedCmp1'] + df['AcceptedCmp2'] + df['AcceptedCmp3'] + df['AcceptedCmp4'] + df['AcceptedCmp5'] + df['Response']
```

Cleaning the data by removing outliers

In [72]:
```
#  Remove outliers when we do customer segmentation, as we are more interested
in the general population rather than the outliers
df = df[(df["Age"]<90)]
df = df[(df["Income"]<110000)]
df = df[(df["NumWebVisitsMonth"]<11)]
df = df[(df["NumWebPurchases"]<20)]
df = df[(df["NumCatalogPurchases"]<20)]
```

Dropping irrelevant or unhelpful fields for clustering

In [73]:
```
fields_to_drop=['ID','Year_Birth','Dt_Customer','Z_CostContact','Z_Revenue','A
cceptedCmp1',
        'AcceptedCmp2','AcceptedCmp3','AcceptedCmp4','AcceptedCmp5','Respons
e','Complain',
         'MntFruits','MntWines','MntMeatProducts','MntFishProducts','MntSweetP
roducts','MntGoldProds']
df.drop(fields_to_drop,axis=1,inplace=True)

# Remove any remaining missing values
df.dropna(inplace=True)
```

3D scatter plots

In [74]:
```
def scatter_3d(x,y,z,c=None):
    fig = plt.figure(figsize=(10,8))
    ax = plt.subplot(111, projection='3d', label="bla")
    ax.scatter(x, y, z, s=40, c=c, marker='o',cmap=plt.cm.viridis)
    ax.set_title("The Plot Of The Clusters")
    plt.show()
```

Task 2: PCA

a) Apply StandardScaler preprocessing on the dataframe df, and assigned the fit_transformed values as df_scaled

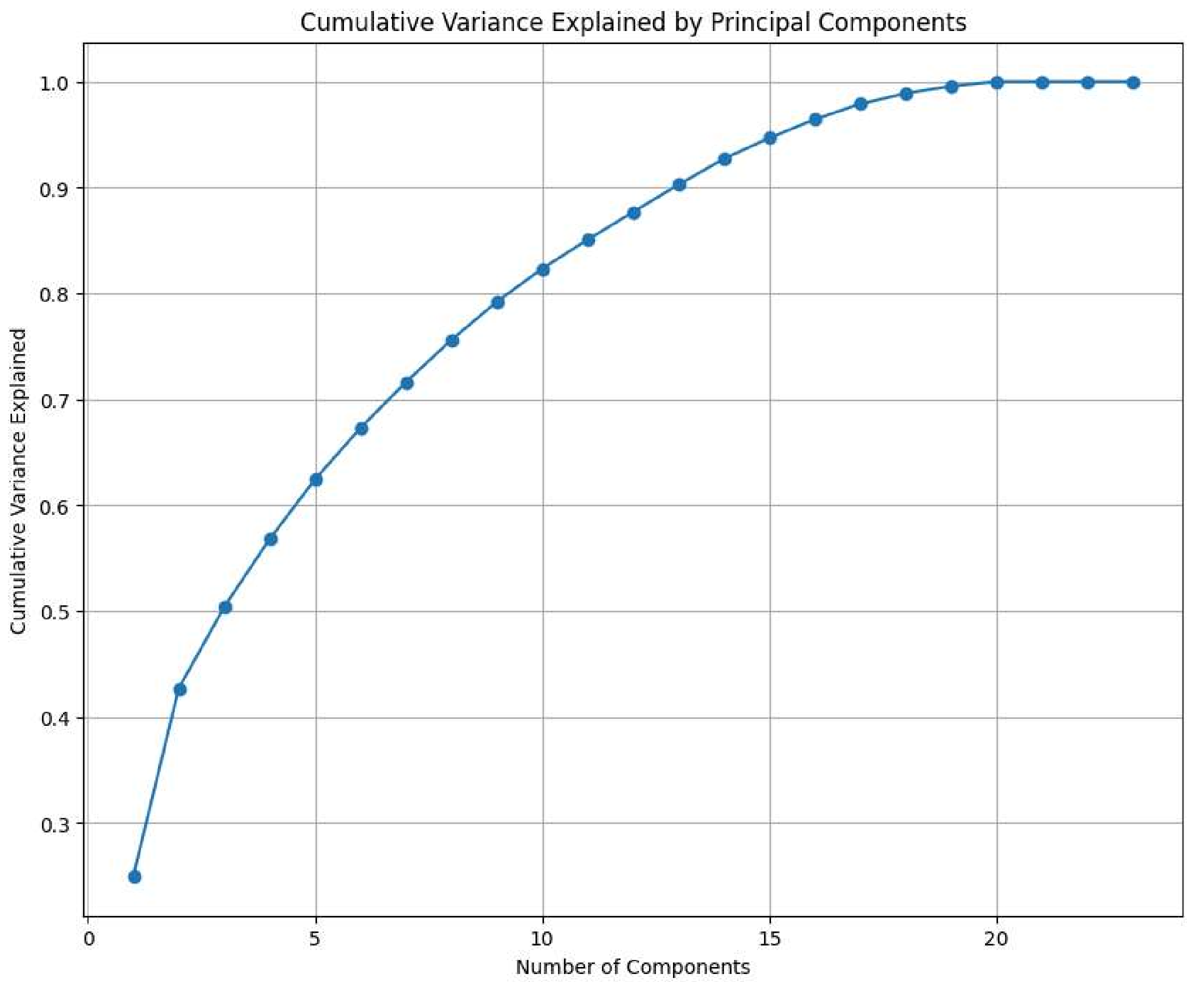
In [75]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
```

b) Apply PCA transformation on df_scaled using all 23 components

In [76]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=23)
df_pca = pca.fit_transform(df_scaled)
```

c) Plot the cummulative

In [77]:
```
cumsum = np.cumsum(pca.explained_variance_ratio_)

plt.figure(figsize=(10, 8))
plt.plot(range(1, len(cumsum) + 1), cumsum, marker='o')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Variance Explained')
plt.title('Cumulative Variance Explained by Principal Components')
plt.grid(True)
plt.show()
```



In [78]:
```
# Determine the number of components needed for 80% variance
n_components = np.argmax(cumsum >= 0.80) + 1
print(f'Number of components explaining 80% of the variance: {n_components}')
```

```
Number of components explaining 80% of the variance: 10
```

According to the plot 10 Principal Componenents is needed for a least 80% of the variance is explained since for 82% , 10 Principal Componenents is needed.

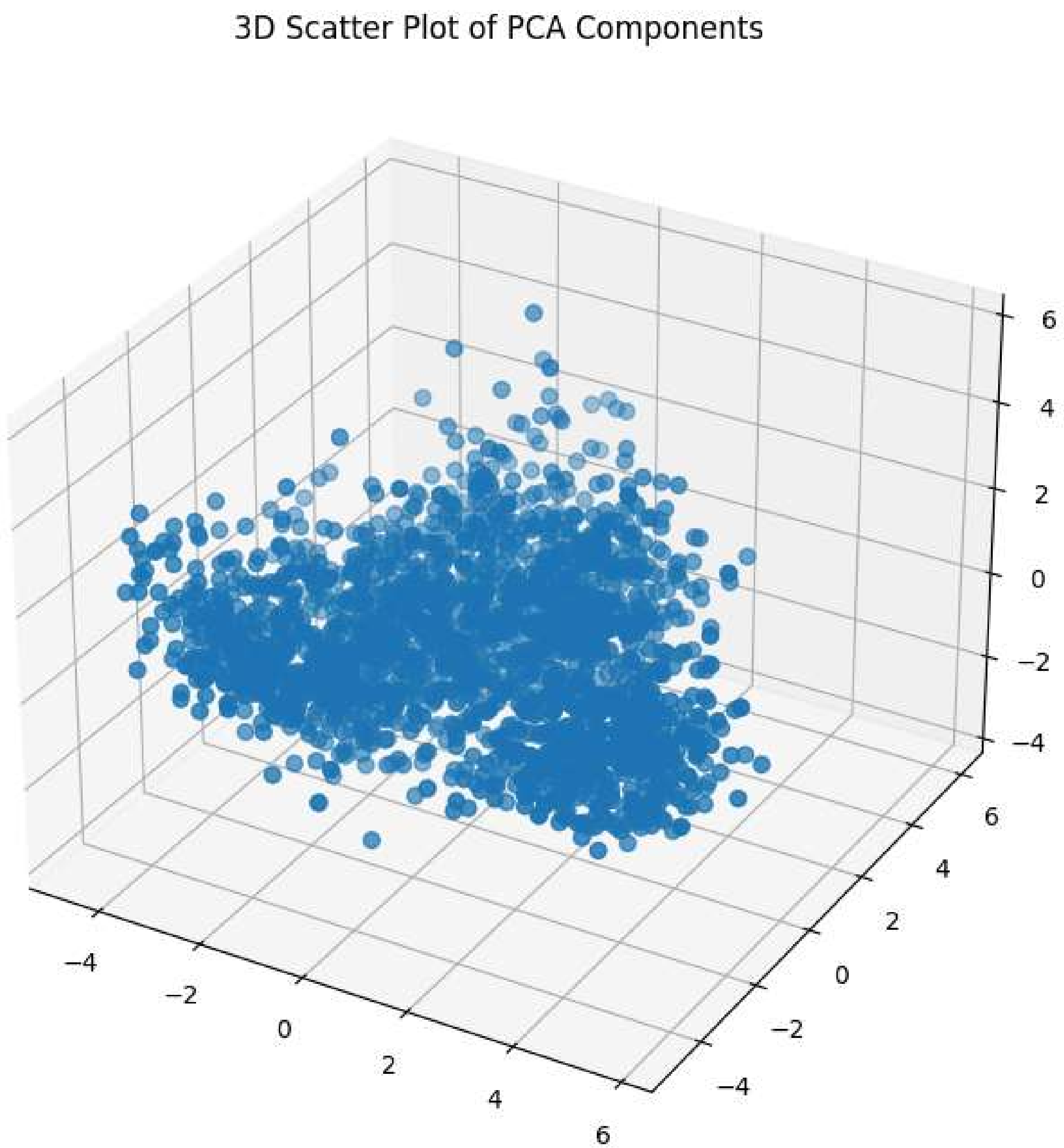
d) Redo the PCA transformation on df_scaled using the same number of componenents as the value from 2c

In [79]:
```
pca = PCA(n_components=n_components)
df_transformed = pca.fit_transform(df_scaled)
```

e) Visualize the first 3 components of df_transformed using a 3d scatter plot

In [80]:
```
def scatter_3d(x, y, z, c=None):
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')
    sc = ax.scatter(x, y, z, s=40, c=c, marker='o', cmap=plt.cm.viridis)
    if c is not None:
        fig.colorbar(sc)
    ax.set_title("3D Scatter Plot of PCA Components")
    plt.show()

# Visualize the first 3 PCA components
scatter_3d(df_transformed[:, 0], df_transformed[:, 1], df_transformed[:, 2])
```

```
<ipython-input-80-7f4b135a8134>:4: UserWarning: No data for colormapping prov
ided via 'c'. Parameters 'cmap' will be ignored
  sc = ax.scatter(x, y, z, s=40, c=c, marker='o', cmap=plt.cm.viridis)
```



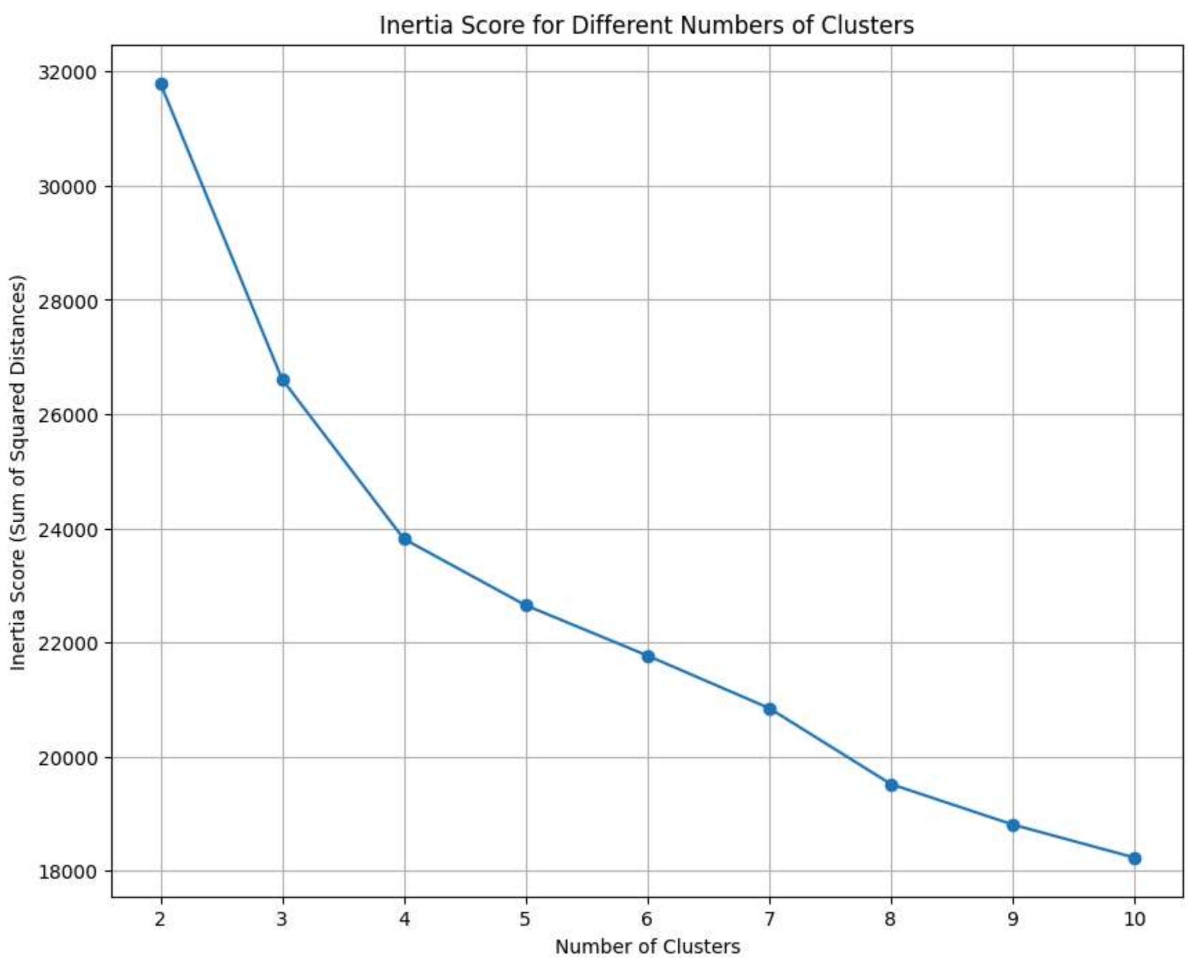
Task 3: KMeans Clustering

a) Apply KMeans clustering to df_transformed and measure the inertia score for n_clusters between 2 and 10

In [81]:
```
from sklearn.cluster import KMeans

inertia_scores = []
cluster_range = range(2, 11)

for i in cluster_range:
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit(df_transformed)
    inertia_scores.append(kmeans.inertia_)
```

b) Plot the inertia score.

In [82]:
```
plt.figure(figsize=(10, 8))
plt.plot(cluster_range, inertia_scores, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia Score (Sum of Squared Distances)')
plt.title('Inertia Score for Different Numbers of Clusters')
plt.grid(True)
plt.show()
```



What would you say is the best number of clusters for this dataset? Why?

The best number of the clusters is 4 because according to the plot, it can be seen that the number of clusters being 4 and the inertia score being just below 24000 is the elbow point where both the number of clusters and inertia score are balanced and equally lowest. Before this point, the number of clusters is low, but the inertia score is pretty high and then the inertia score decreases sharply but the number of clusters increases. Therefore, this elbow point is the best point for both the number of clusters and inertia score.

c) Find the cluster labels of KMeans(n_clusters=4) on df_transformed, (It doesn't matter what your answer for 3b is, set n_clusters=4)

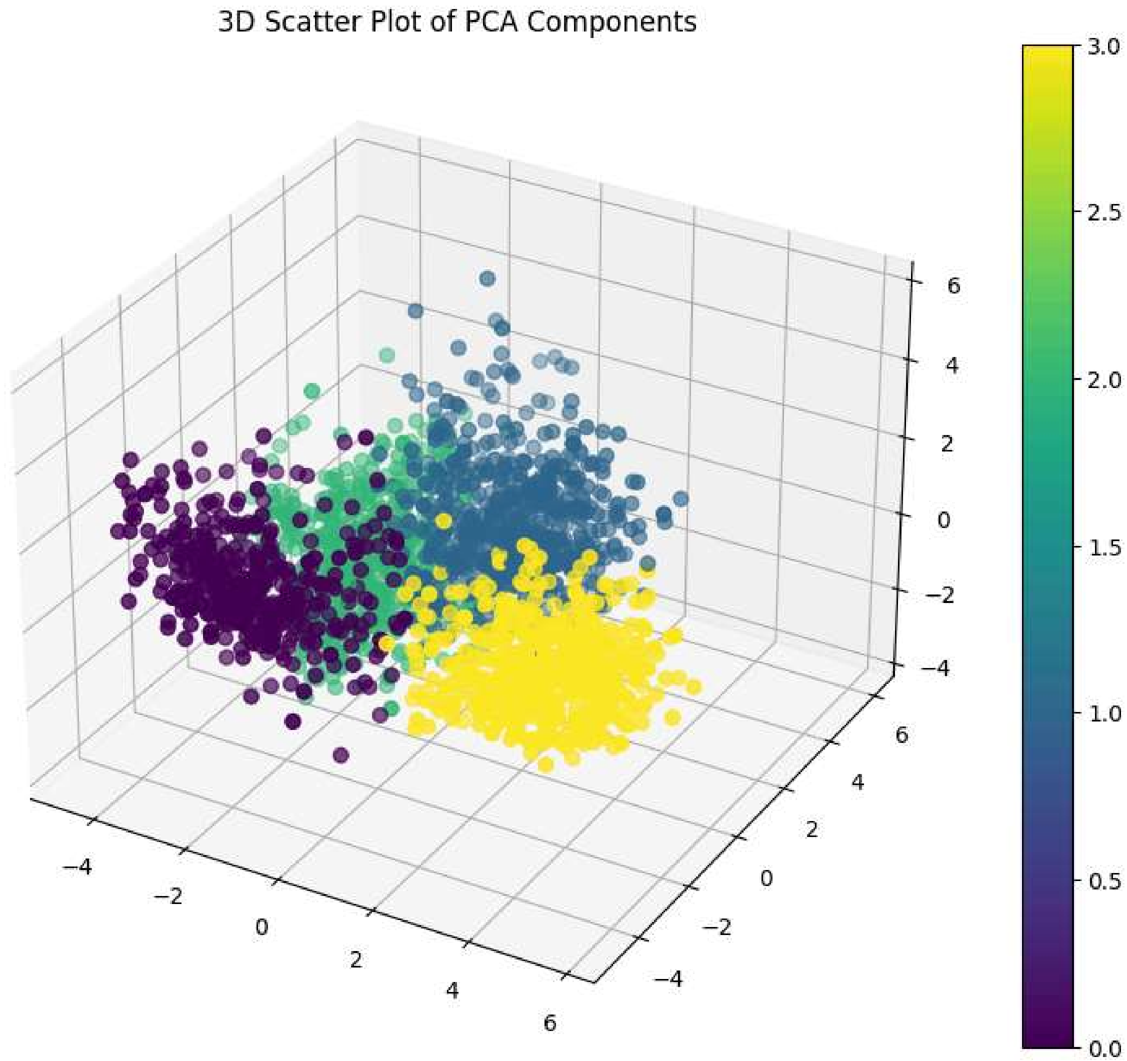
In [83]:
```
kmeans = KMeans(n_clusters=4, random_state= ID)
cluster_labels = kmeans.fit_predict(df_transformed)
```

d) Assign the cluster labels as df['Clusters']

In [84]:
```
df['Clusters'] = cluster_labels
```

e) Visualize the first 3 components of df_transformed using a 3d scatter plot, with the data points coloured according to the clusters

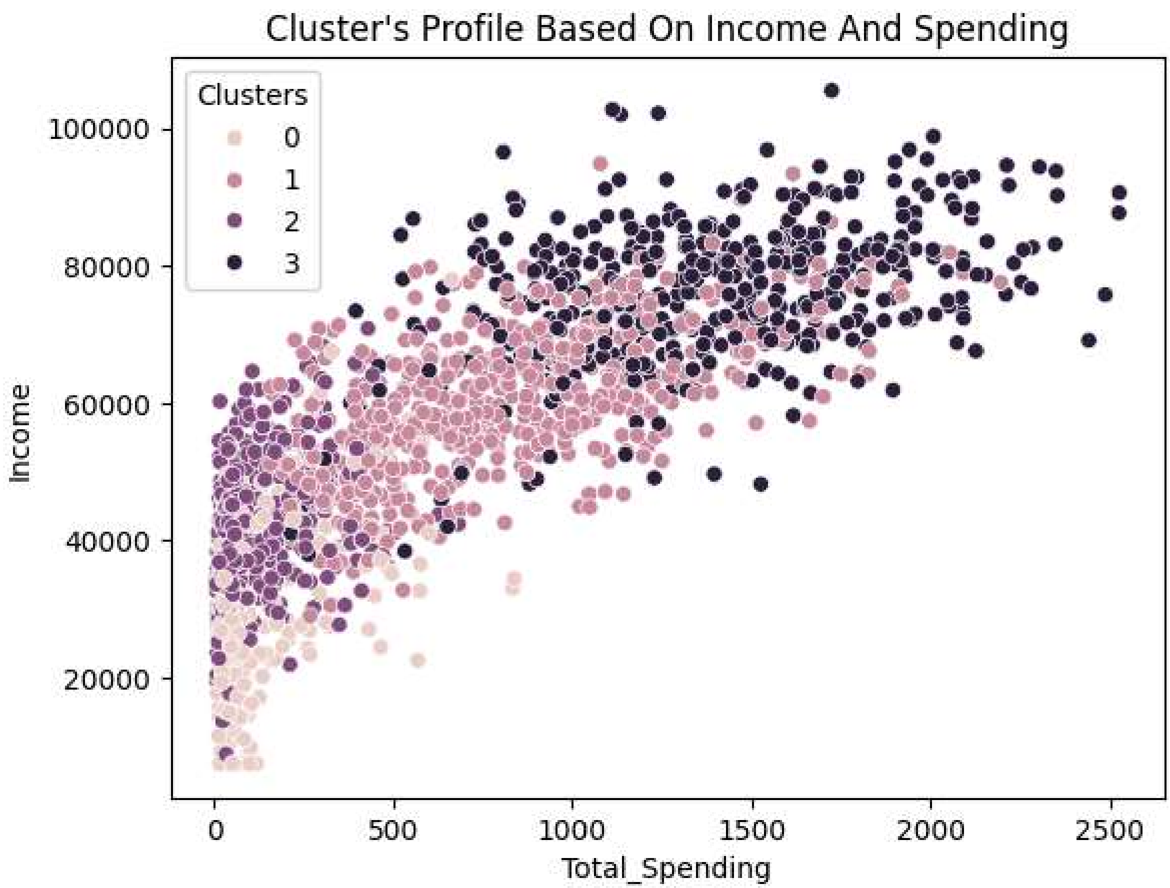
In [85]:
```
scatter_3d(df_transformed[:, 0], df_transformed[:, 1], df_transformed[:, 2], c
=df['Clusters'])
```



Task 4: Interpretating our results

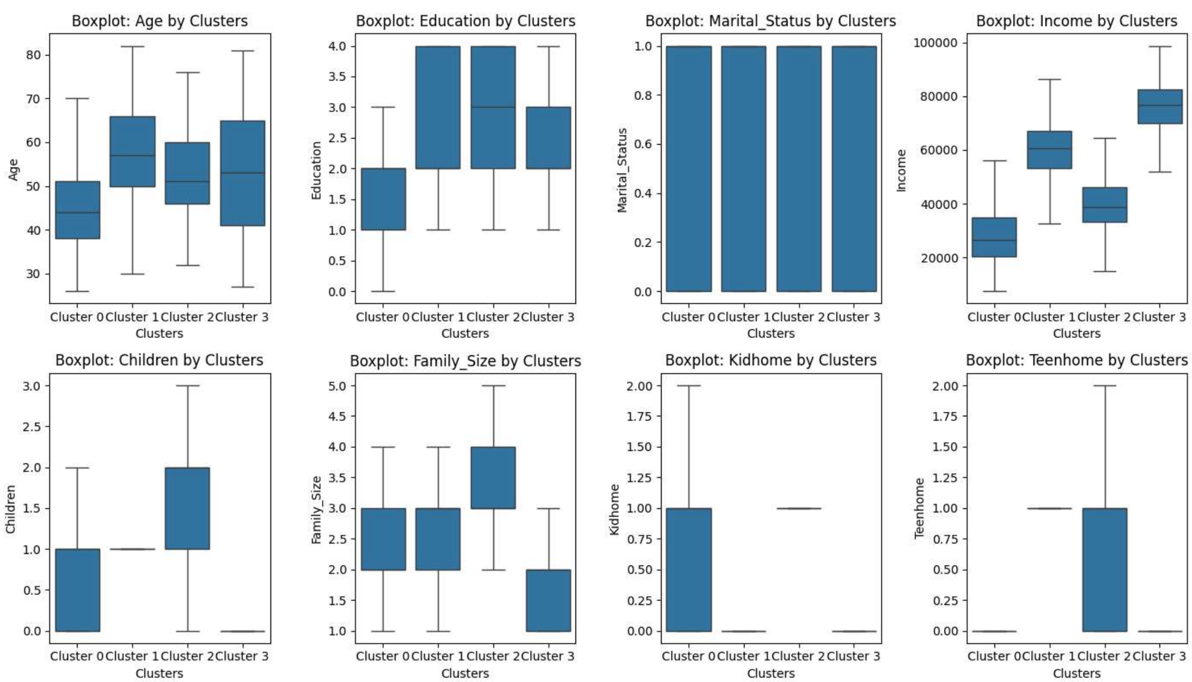
a) Do a scatter plot between df['Income'] and df['Total_Spending'] and colour the data points according the clusters.

In [86]:
```
pl = sns.scatterplot(data=df, x="Total_Spending", y="Income",hue="Clusters")
pl.set_title("Cluster's Profile Based On Income And Spending")
plt.show()
```



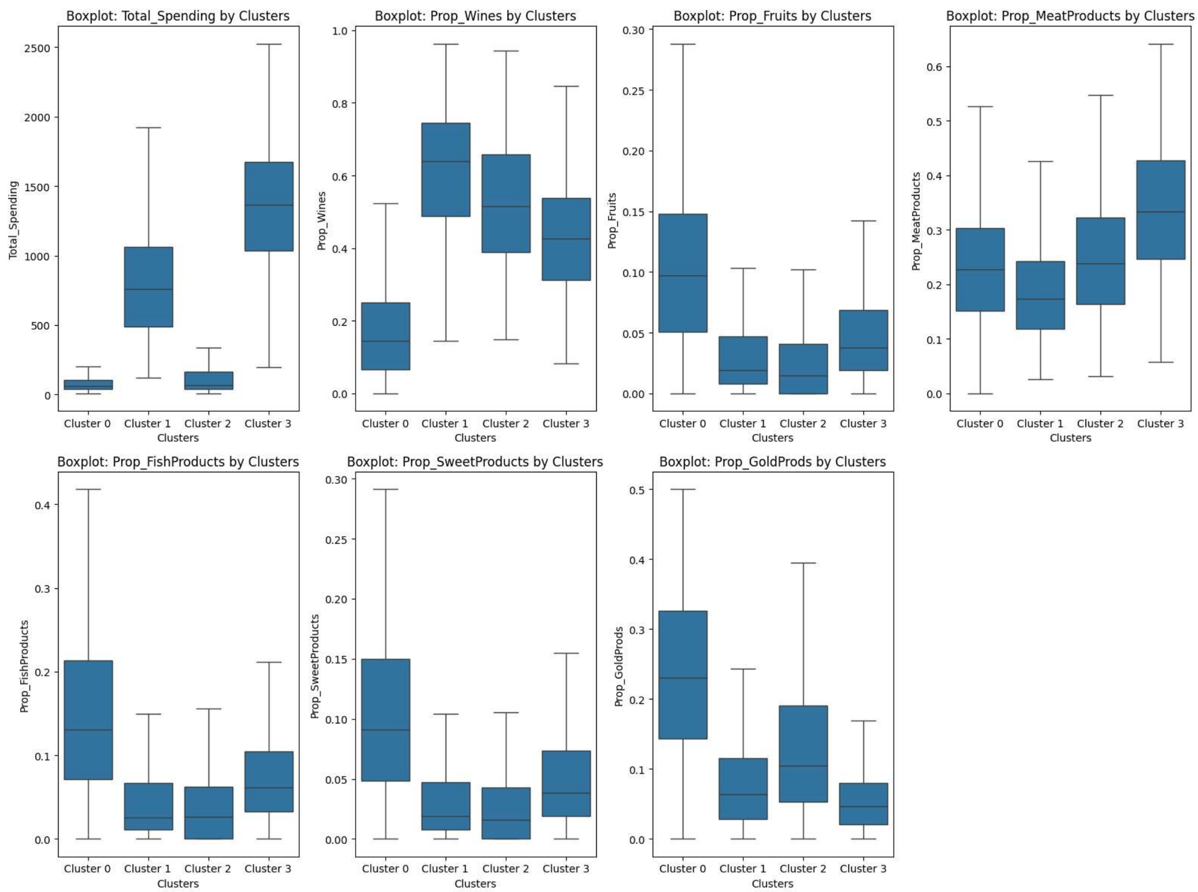
b) Do a boxplot by clusters, for each of these following fields describing the attributes of the customer ['Age'.'Education','Marital_Status','Income', 'Children', 'Family_Size','Kidhome','Teenhome']

In [87]:
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Get the number of unique clusters
n_clusters = df['Clusters'].nunique()

# Boxplots for customer attributes by clusters
attributes = ['Age', 'Education', 'Marital_Status', 'Income', 'Children', 'Family_Size', 'Kidhome', 'Teenhome']

plt.figure(figsize=(14, 8))
for i, attribute in enumerate(attributes):
    plt.subplot(2, 4, i + 1)  # Create subplots in a 2x4 grid
    sns.boxplot(x='Clusters', y=attribute, data=df, showfliers=False)
    plt.title(f'Boxplot: {attribute} by Clusters')
    plt.xticks(range(n_clusters), [f'Cluster {i}' for i in range(n_clusters)])
plt.tight_layout()
plt.show()
```



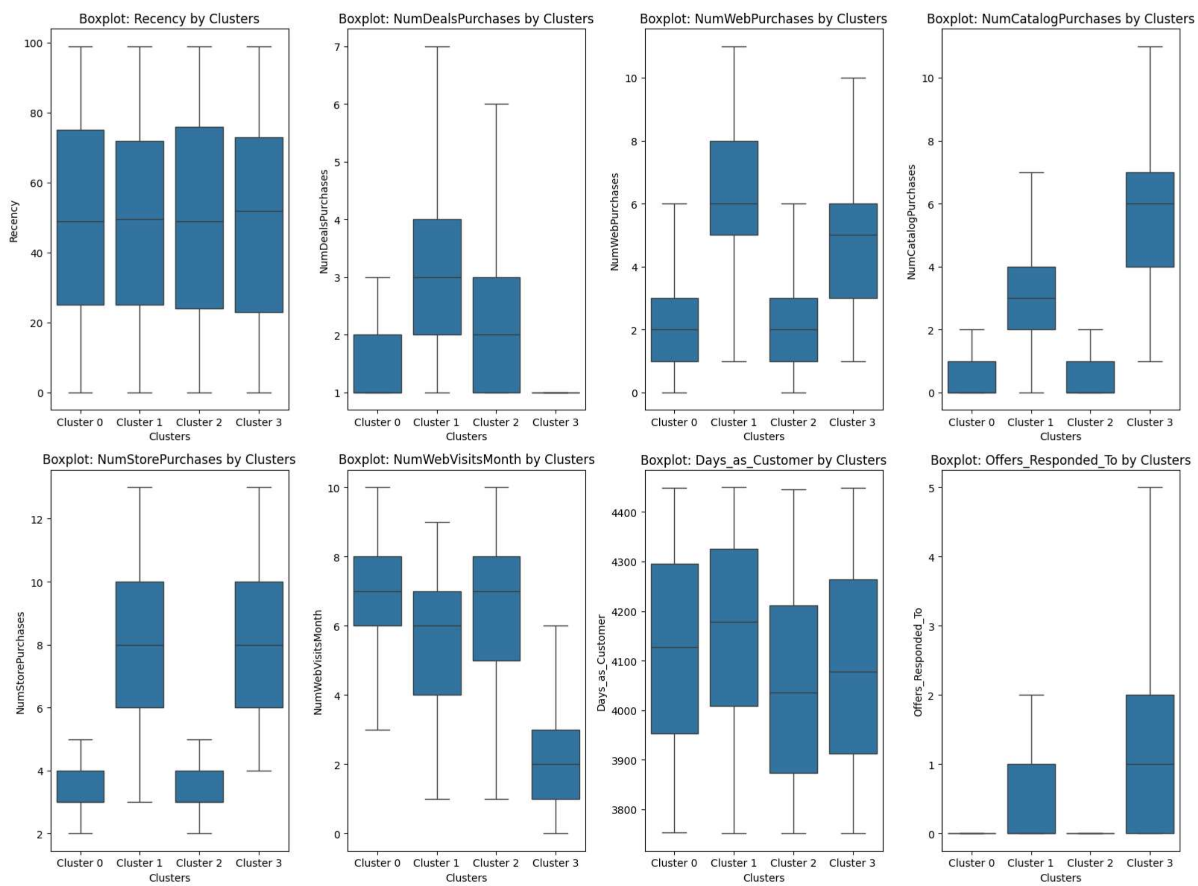
c) Do a boxplot by clusters, for each of these following fields describing the customer preference ['Total_Spending','Prop_Wines','Prop_Fruits','Prop_MeatProducts','Prop_FishProducts','Prop_SweetProdu

In [88]:
```
# Boxplots for customer preferences by clusters
preferences = ['Total_Spending', 'Prop_Wines', 'Prop_Fruits', 'Prop_MeatProduc
ts',
               'Prop_FishProducts', 'Prop_SweetProducts', 'Prop_GoldProds']

plt.figure(figsize=(16, 12))
for i, preference in enumerate(preferences):
    plt.subplot(2, 4, i + 1)
    sns.boxplot(x='Clusters', y=preference, data=df, showfliers=False)
    plt.title(f'Boxplot: {preference} by Clusters')
    plt.xticks(range(n_clusters), [f'Cluster {i}' for i in range(n_clusters)])
plt.tight_layout()
plt.show()
```



d) Do a boxplot by clusters, for each of these following fields describing the customer behaivoir ['Recency', 'NumDealsPurchases', 'NumWebPurchases', 'NumCatalogPurchases', 'NumStorePurchases', 'NumWebVisitsMonth', 'Days_as_Customer', 'Offers_Responded_To']

In [89]:
```
# Get the number of unique clusters dynamically
n_clusters = df['Clusters'].nunique()

# Boxplots for customer behavior by clusters
behaviors = ['Recency', 'NumDealsPurchases', 'NumWebPurchases', 'NumCatalogPur
chases',
             'NumStorePurchases', 'NumWebVisitsMonth', 'Days_as_Customer', 'Of
fers_Responded_To']

plt.figure(figsize=(16, 12))
for i, behavior in enumerate(behaviors):
    plt.subplot(2, 4, i + 1)
    sns.boxplot(x='Clusters', y=behavior, data=df, showfliers=False)
    plt.title(f'Boxplot: {behavior} by Clusters')
    plt.xticks(range(n_clusters), [f'Cluster {i}' for i in range(n_clusters)])
# Handle different numbers of clusters
plt.tight_layout()
plt.show()
```



e) Obtain the means grouped by the cluster, for each of the fields. (See hints for a 1-liner code)

In [90]:
```
# Calculate the means of the features grouped by Clusters
cluster_means = df.groupby('Clusters').mean().T
print(cluster_means)
```

```
Clusters                         0             1             2             3
Education                 1.664319      2.656151      2.753647      2.506796
Marital_Status            0.624413      0.675079      0.675851      0.592233
Income                28299.960094  60160.159306  39556.867099  75822.452427
Kidhome                   0.678404      0.156151      0.925446      0.023301
Teenhome                  0.096244      0.968454      0.696921      0.048544
Recency                  49.117371     48.676656     49.264182     49.225243
NumDealsPurchases         1.821596      3.488959      2.470016      1.046602
NumWebPurchases           2.173709      6.361199      2.406807      4.893204
NumCatalogPurchases       0.591549      3.361199      0.619125      5.899029
NumStorePurchases         3.293427      7.973186      3.411669      8.300971
NumWebVisitsMonth         6.546948      5.550473      6.385737      2.642718
Age                      45.730047     57.913249     53.076175     53.264078
Children                  0.774648      1.124606      1.622366      0.071845
Family_Size               2.399061      2.799685      3.298217      1.664078
Total_Spending          111.453052    807.957413    117.638574   1363.409709
Prop_Wines                0.167191      0.616963      0.524853      0.433415
Prop_Fruits               0.106278      0.034664      0.025610      0.049006
Prop_MeatProducts         0.233265      0.185796      0.251463      0.334373
Prop_FishProducts         0.151586      0.046297      0.041214      0.074083
Prop_SweetProducts        0.106718      0.035482      0.026846      0.050715
Prop_GoldProds            0.234962      0.080799      0.130014      0.058409
Days_as_Customer       4123.093897   4154.750789   4054.682334   4090.203883
Offers_Responded_To       0.180751      0.386435      0.183144      1.081553
```