

Customer Segmentation

Murtuja Afshar

Debug-Tech Task 2

February 23, 2025

Data Collection:

Data was collected from Kaggle from a survey conducted to assess the spending habits of population that visited a superstore. The following columns have been created:

Columns:

id: Unique identifier for each customer.

age: Age of the customer.

gender: Gender of the customer (Male, Female, Other).

income: Annual income of the customer (in USD).

spending_score: Spending score (1-100), indicating the customer's spending behavior and loyalty.

membership_years: Number of years the customer has been a member.

purchase_frequency: Number of purchases made by the customer in the last year.

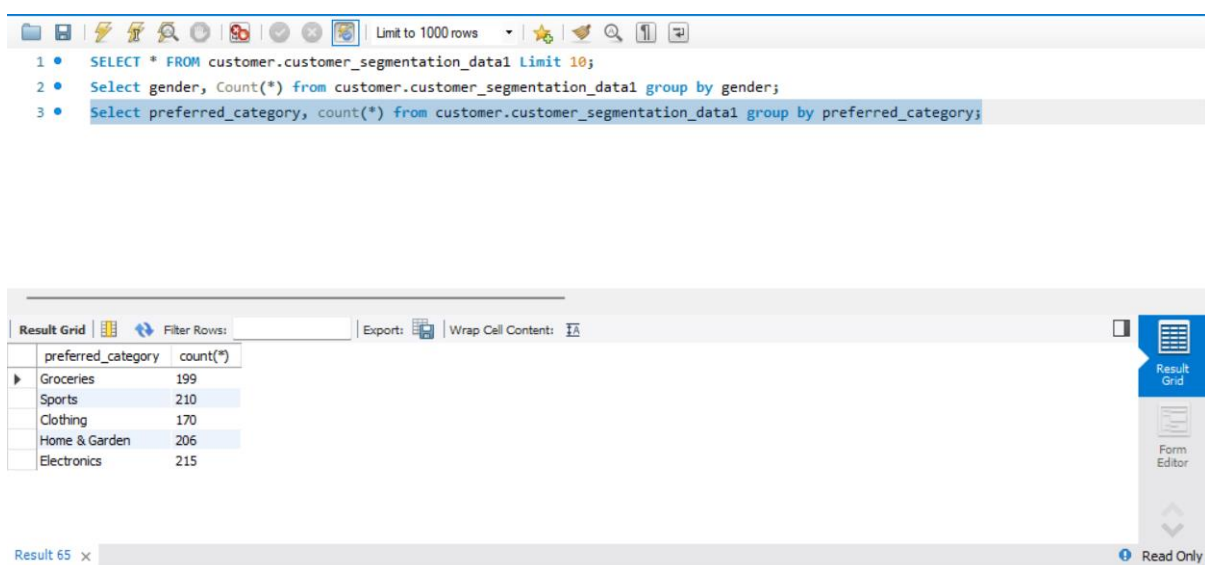
preferred_category: Preferred shopping category (Electronics, Clothing, Groceries, Home & Garden, Sports).

last_purchase_amount: Amount spent by the customer on their last purchase (in USD).

Data Cleaning:

Data was uploaded in MySQL workbench and SQL queries were passed to create a cleaning process:

Identified different categories in gender and preferred category columns. Then treated null values and Duplicates



The screenshot displays the MySQL Workbench interface. The top section shows three SQL queries in the editor:

- `SELECT * FROM customer.customer_segmentation_data1 Limit 10;`
- `Select gender, Count(*) from customer.customer_segmentation_data1 group by gender;`
- `Select preferred_category, count(*) from customer.customer_segmentation_data1 group by preferred_category;`

The bottom section shows the 'Result Grid' for the third query. It contains a table with two columns: 'preferred_category' and 'count(*)'.

preferred_category	count(*)
Groceries	199
Sports	210
Clothing	170
Home & Garden	206
Electronics	215

The interface also includes a toolbar at the top with icons for various functions, a 'Filter Rows' field, and an 'Exports' button. The bottom right corner shows a 'Read Only' status.

Null values:

Null values in a database represent missing, unknown, or inapplicable data. They indicate the absence of a value rather than zero or an empty string. In databases, null values can create challenges in data analysis, as they may distort calculations, affect relationships between tables, and lead to logical inconsistencies. The following SQL queries were used to find Null values of individual columns:

The screenshot shows a SQL query editor with the following query:

```
4 • Select
5     SUM(case when age is null then 1 else 0 end) as null_age,
6     SUM(case when gender is null then 1 else 0 end) as null_gender,
7     SUM(case when income is null then 1 else 0 end) as null_income,
8     SUM(case when spending_score is null then 1 else 0 end) as null_spending_score,
9     SUM(case when membership_years is null then 1 else 0 end) as null_membership_years,
10    SUM(case when purchase_frequency is null then 1 else 0 end) as null_purchase_frequency,
11    SUM(case when last_purchase_amount is null then 1 else 0 end) as null_last_purchase_amount
12 FROM customer.customer_segmentation_data1;
```

The result grid below the query shows the following data:

null_age	null_gender	null_income	null_spending_score	null_membership_years	null_purchase_frequency	null_last_purchase_amount
0	0	0	0	0	0	0

The interface includes a toolbar with icons for saving, undo, redo, and other functions. The bottom status bar shows the current query is 'customer_segmentation_data166' and the result is 'Result 67'.

Duplicate:

Duplicates in a dataset refer to repeated records that contain identical or highly similar values across one or more fields. They can occur due to data entry errors, system glitches, or multiple data sources merging. Duplicates can distort analysis, inflate statistics, and mislead decision-making by overrepresenting certain values. SQL helps manage duplicates by detecting, filtering, and removing them to ensure data accuracy and consistency. It allows identifying duplicate records based on specific criteria and provides methods to retain only unique entries. By handling duplicates effectively, SQL enhances data quality, ensuring reliable and meaningful analysis. The following SQL queries were used to find Duplicate values of data using age, income and preferred_category:

The screenshot shows a SQL query editor with the following query:

```
9     SUM(case when membership_years is null then 1 else 0 end) as null_membership_years,
10    SUM(case when purchase_frequency is null then 1 else 0 end) as null_purchase_frequency,
11    SUM(case when last_purchase_amount is null then 1 else 0 end) as null_last_purchase_amount
12 FROM customer.customer_segmentation_data1;
13
14 • Select age, gender, income, preferred_category, count(*)
15 from customer.customer_segmentation_data1
16 group by age, gender, income, preferred_category
17 having count(*) > 1;
```

The result grid below the query shows the following data:

age	gender	income	preferred_category	count(*)
-----	--------	--------	--------------------	----------

The interface includes a toolbar with icons for saving, undo, redo, and other functions. The bottom status bar shows the current query is 'customer_segmentation_data187' and the result is 'Result 88'.

Once data was cleaned we imported the data in jupyter notebook.

Data Loading:

First import the libraries that are important and load the data using read.csv function. Also describe the data to get an idea of the data:

```
In [27]: import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from matplotlib import colors
import seaborn as sns
```

```
In [28]: data = pd.read_csv(r"C:\Users\Murtuja_afshar\OneDrive\Desktop\job\task 2\customer_segmentation_data1.csv")
data.head()
```

```
Out[28]:
```

	id	age	gender	income	spending_score	membership_years	purchase_frequency	preferred_category	last_purchase_amount
0	1	38	Female	99342	90	3	24	Groceries	113.53
1	2	21	Female	78852	60	2	42	Sports	41.93
2	3	60	Female	126573	30	2	28	Clothing	424.36
3	4	40	Other	47099	74	9	5	Home & Garden	991.93
4	5	65	Female	140621	21	3	25	Electronics	347.08

```
In [33]: data.describe()
```

```
Out[33]:
```

	id	age	income	spending_score	membership_years	purchase_frequency	last_purchase_amount
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	500.500000	43.783000	88500.800000	50.685000	5.469000	26.596000	492.348670
std	288.819436	15.042213	34230.771122	28.955175	2.85573	14.243654	295.744253
min	1.000000	18.000000	30004.000000	1.000000	1.000000	1.000000	10.400000
25%	250.750000	30.000000	57911.750000	26.000000	3.000000	15.000000	218.762500

Preprocessing Data:

Since we are using unsupervised machine learning algorithms and for that we will be using K-Means classification. It is important to preprocess the data to carry out the analysis.

Label Encoder

The data contains columns that have string variables which cannot be used for K-means, thus we use label encoder to turn them to float data:

```
In [5]: s = (data.dtypes == 'object')
object_cols = list(s[s].index)

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

Categorical variables in the dataset: ['gender', 'preferred_category']
```

```
In [11]: from sklearn.preprocessing import LabelEncoder
LE = LabelEncoder()

# Encode all categorical columns
for i in object_cols:
    data[i] = LE.fit_transform(data[i])

# Create mapping for 'preferred_category' only
data['preferred_category'] = LE.fit_transform(data['preferred_category']) # Re-encode to ensure LE has the right classes
category_mapping = dict(enumerate(LE.classes_))
```

Standard Scaler

The data also has values that range over a wide variety. For example the age data varies between 18 to 69 whereas the income ranges between 30004 to 149973. The variation is huge

which will cause the classification features with larger numeric ranges to dominate the clustering process. Since K-Means uses Euclidean distance to assign data points to clusters, the algorithm may give more importance to features with larger values, leading to biased cluster assignments.

```
from sklearn.preprocessing import StandardScaler
features = ['age', 'income', 'spending_score', 'membership_years', 'purchase_frequency', 'preferred_category', 'last_purchase_amount']
x = data[features]
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
df = pd.DataFrame(x_scaled, columns=features)
df.head()
```

	age	income	spending_score	membership_years	purchase_frequency	preferred_category	last_purchase_amount
0	-0.384644	0.316868	1.358468	-0.865010	-0.182348	-0.051028	-1.281540
1	-1.515362	-0.282016	0.321865	-1.215358	1.082005	1.386386	-1.523763
2	1.078639	1.112778	-0.714738	-1.215358	0.098620	-1.488442	-0.230005
3	-0.251618	-1.210096	0.805613	1.237080	-1.516943	0.667679	1.690080
4	1.411203	1.523374	-1.025718	-0.865010	-0.112106	-0.769735	-0.491443

Data Clustering:

Initially a cluster of 3 is formed and a random state of 48 is used. Cluster of 3 is for initial process if there is improper clustering or need more detailing we can increase the number of clusters. Random state ensures that this initial selection is reproducible when you run the algorithm multiple times.. The initial centroids can affect the final clusters, as K-Means converges to local minima.

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 3, random_state = 48 )
data['cluster'] = kmeans.fit_predict(df)
print(data.groupby('cluster')[features].mean())
```

	age	income	spending_score	membership_years	\
cluster					
0	42.013378	96810.344482	49.575251	6.297659	
1	39.829341	92523.883234	52.059880	4.769461	
2	48.822888	78069.564033	50.337875	5.430518	

	purchase_frequency	preferred_category	last_purchase_amount
cluster			
0	26.394649	0.715719	329.754181
1	25.661677	3.266467	317.351916
2	27.610354	2.087193	784.078011

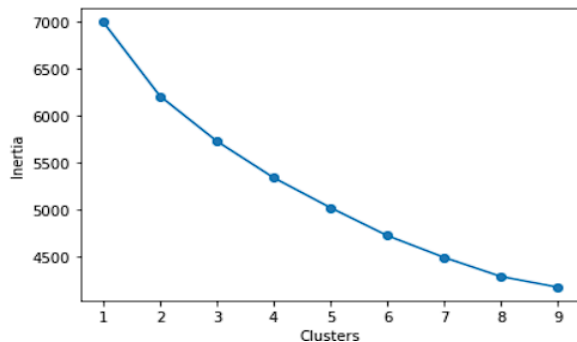
The elbow plot is a graphical method used to determine the optimal number of clusters for K-means clustering. It plots the inertia (within-cluster sum of squares) against the number of clusters. Inertia measures how compact the clusters are; lower inertia indicates more compact clusters. As the number of clusters increases, inertia decreases, but after a certain point, the reduction in inertia becomes very less. We can increase the number of clusters to 7, 8 or 9 but I wanted to create lesser confusion and broader category I have gone with 3 clusters.

```

inertia = []
for k in range(1,10):
    kmeans = KMeans(n_clusters = k, random_state = 48 )
    kmeans.fit(df)
    inertia.append(kmeans.inertia_)
plt.plot(range(1,10), inertia, marker = 'o')
plt.xlabel('Clusters')
plt.ylabel('Inertia')
plt.show()

```

C:\Users\Murtuja afshar\anaconda3\lib\site-packages\sklearn\cluster\k_means_.py:130: UserWarning: A possible memory leak on Windows with MKL, when there are less chunks than available OMP_NUM_THREADS=4.



Evaluating Models:

After classification I have done a 3-D scatter plot and 2 2-D scatter plots using age, income and Preferred category as main references:

```

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(14, 10))
ax = fig.add_subplot(111, projection='3d')

scatter = ax.scatter(data['age'], data['income'], data['preferred_category'],
                    c=data['cluster'], cmap='viridis')

ax.set_xlabel('Age')
ax.set_ylabel('Income')
ax.set_zlabel('Preferred Category')
ax.set_title('Clusters by Age, Income, and Preferred Category')

ticks = np.sort(data['preferred_category'].unique())
ax.set_zticks(ticks)
ax.set_zticklabels([category_mapping[int(t)] for t in ticks])

cbar = plt.colorbar(scatter, label='Cluster')

category_text = "\n".join([f"{k}: {v}" for k, v in category_mapping.items()])
props = dict(boxstyle='round', facecolor='white', alpha=0.7)
ax.text2D(0.05, 0.05, f"Preferred Category Mapping:\n{category_text}", transform=ax.transAxes, fontsize=10, verticalalignment='top')

plt.show()

```

Here in ax variable subplot(111) refers to 1 row, 1 column, and the first subplot. In the code cmap='viridis', cmap refers to the colormap used for mapping the values (in this case, the cluster labels in data['cluster']) to colors in the scatter plot. 'viridis' is a specific colormap that is perceptually uniform. Also since the label encoder has changed the name of the z axis labels to numbers we will use the below process to create the 3d plot with preferred category labels.

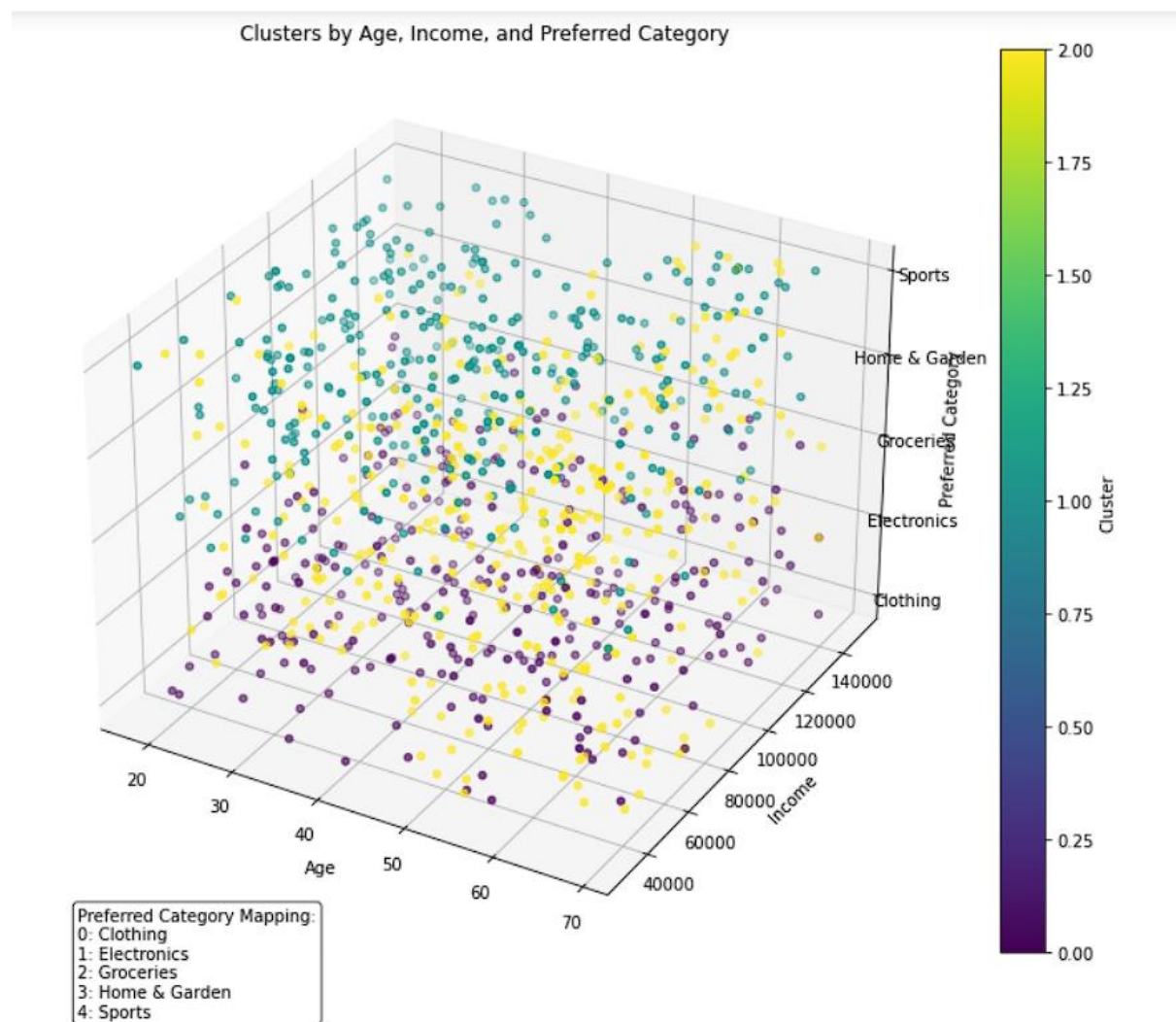
1. ticks = np.sort(data['preferred_category'].unique()): It gets the unique values from the preferred_category column, sorts them, and stores them in ticks.

2. `ax.set_zticks(ticks)`: It sets the tick positions on the Z-axis to the sorted unique values (ticks).
3. `ax.set_zticklabels([category_mapping[int(t)] for t in ticks])`: It maps the numeric preferred_category values to their corresponding labels from category_mapping and sets them as the tick labels on the Z-axis.

This code displays the category_mapping dictionary as a text box on the plot:

1. `category_text = "\n".join([f"{k}: {v}" for k, v in category_mapping.items()])`: It creates a formatted string by joining key-value pairs from the category_mapping dictionary, with each pair on a new line.
2. `props = dict(boxstyle='round', facecolor='white', alpha=0.7)`: It defines the properties of the text box, setting a rounded style, white background color, and 70% transparency (alpha).
3. `ax.text2D(...)`: It adds the formatted category_text as a text box to the plot at the coordinates (0.05, 0.05) within the axes. The `transform=ax.transAxes` ensures the position is relative to the axes, and the text box is styled using the props dictionary.

The final 3-D plot:



I also created two 2d plots between x and z axis and also y and z axis:

```
scatter = ax.scatter(data['age'], data['preferred_category'], c=data['cluster'], cmap='viridis', alpha=0.8, edgecolors='k')
ax.set_xlabel('Age')
ax.set_ylabel('Preferred Category')
ax.set_title('Clusters by Age and Preferred Category')

ticks = np.sort(data['preferred_category'].unique())
ax.set_yticks(ticks)
ax.set_yticklabels([category_mapping[int(t)] for t in ticks])

plt.colorbar(scatter, label='Cluster')
plt.show()
```



```
fig, ax = plt.subplots(figsize=(10, 6))

scatter = ax.scatter(data['income'], data['preferred_category'], c=data['cluster'], cmap='viridis', alpha=0.8, edgecolors='k')
ax.set_xlabel('Income')
ax.set_ylabel('Preferred Category')
ax.set_title('Clusters by Age and Preferred Category')

ticks = np.sort(data['preferred_category'].unique())
ax.set_yticks(ticks)
ax.set_yticklabels([category_mapping[int(t)] for t in ticks])

plt.colorbar(scatter, label='Cluster')
plt.show()
```

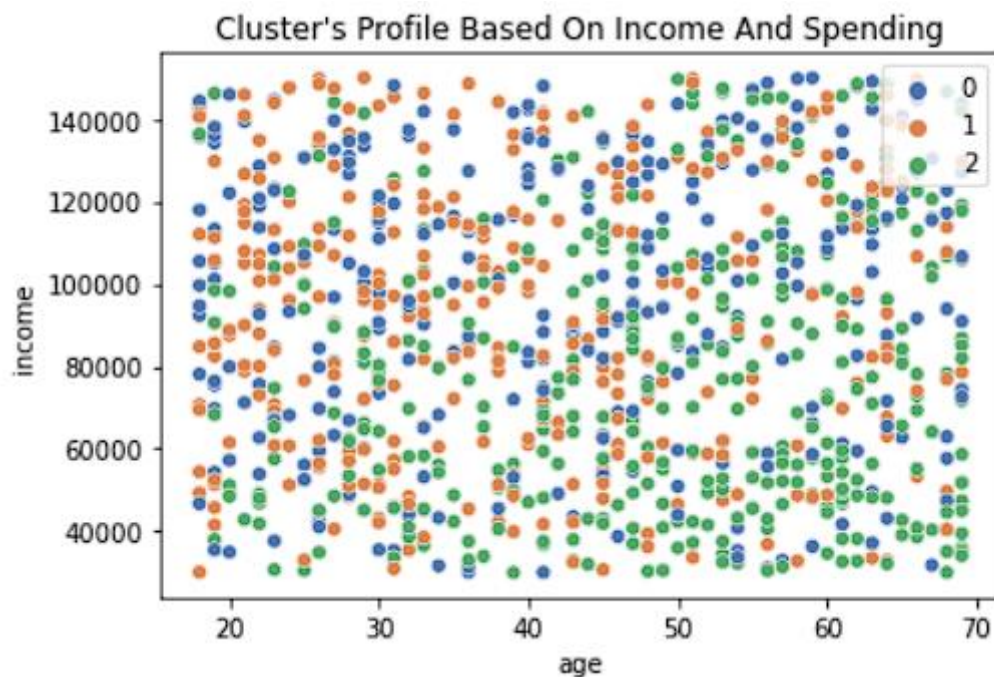


I have also created two scatter plots between age and income and also age and preferred category:


```

pl = sns.scatterplot(data = data,x=data["age"], y=data["income"],hue=data["cluster"], palette= "deep")
pl.set_title("Cluster's Profile Based On Income And Spending")
plt.legend()
plt.show()

```



```

pl = sns.scatterplot(data = data,x=data["age"], y=data["preferred_category"],hue=data["cluster"], palette= "deep")
pl.set_title("Cluster's Profile Based On Income And Spending")
plt.legend()
plt.show()

```



Final Analysis:

- **Cluster 0 (Blue):**

- **Profile:** Likely lower-income customers (\$40,000–\$80,000) spread across all ages, with a concentration on younger (20–40) and older (60–70) customers preferring Clothing (purple dots in the plots).
- **Fashion Preference:** These customers may prioritize affordable, practical, or essential casual wear due to budget constraints. They're less likely to invest in high-end, fashion-forward items.
- **Product Recommendations:**
 - Develop budget-friendly casual wear lines, such as basic t-shirts, jeans, and everyday outfits in neutral or classic colors.
 - Offer seasonal sales or loyalty discounts to appeal to price-sensitive shoppers.
 - Focus on durable, versatile clothing that can be mixed and matched for everyday use.

- **Cluster 1 (Orange):**

- **Profile:** Mid to higher-income customers (\$80,000–\$140,000) across all ages, with a notable presence among middle-aged (30–50) customers who show interest in Electronics and Sports but also Clothing.
- **Fashion Preference:** This segment is more likely to be fashion-forward or willing to invest in trendy, stylish clothing. They may seek premium or branded fashion items due to their higher disposable income.
- **Product Recommendations:**
 - Create fashion-forward collections with bold patterns, trendy designs, and premium materials (e.g., designer jeans, statement jackets, or seasonal fashion pieces).
 - Introduce limited-edition or exclusive lines to attract these customers, leveraging their interest in unique, high-value products.
 - Offer complementary accessories (e.g., belts, scarves) or upscale casual wear like athleisure for those also interested in Sports.

- **Cluster 2 (Green):**

- **Profile:** A broad, mixed segment with incomes ranging from \$40,000–\$120,000 and all ages, showing a balanced interest in Groceries, Home & Garden, and Clothing.
- **Fashion Preference:** These customers likely prefer versatile, functional casual wear that fits into their daily lives (e.g., family-oriented or practical outfits). They may not prioritize high-fashion but value comfort and affordability.
- **Product Recommendations:**
 - Design casual wear lines that balance style and comfort, such as comfortable leggings, hoodies, or work-appropriate outfits.
 - Focus on family-sized packs or multi-purpose clothing (e.g., items suitable for both work and leisure).
 - Use eco-friendly or sustainable materials to appeal to environmentally conscious customers within this segment.