CUSTOMER SEGMENTATION

Task02: Clustering based on Customer Behaviour

Customer segmentation is a cornerstone of data-driven marketing, enabling businesses to identify distinct customer groups for targeted strategies.

Intellihack: Team CypherZ
Dhanushi Dewmindi | Amanda Hansamali

Table of Contents

1. Introduction

Customer segmentation is a cornerstone of data-driven marketing, enabling businesses to identify distinct customer groups for targeted strategies. This report analyzes a dataset of 999 customer records from *customer_behavior_analytcis.csv*, featuring variables such as *total_purchases*, *avg_cart_value*, *total_time_spent*, *product_click*, and *discount_counts*.
The goal is to segment customers into three groups—**Bargain Hunters, High Spenders,** and **Window Shoppers**—using advanced clustering techniques. The analysis was conducted in Python within a Jupyter Notebook environment, leveraging libraries like pandas, scikit-learn, matplotlib, and seaborn. This report details the approach up to data preprocessing, Feature Engineering, EDA,Model Selection, Training and Evaluation, discusses challenges, provides insights, and suggests improvements, with placeholders for clustering results pending further code execution.

Using complex clustering algorithms, this research aims to categorize customers into three actionable groups: bargain hunters, high spenders, and window shoppers. Bargain Hunters are expected to make large purchases due to discount utilization, High Spenders to have higher average cart values and rely less on promotions, and Window Shoppers to engage in substantial browsing activity with lower conversion rates.
This segmentation corresponds to traditional retail archetypes, allowing for tailored initiatives such as discount promotions for bargain hunters, premium offers for high spenders, and re-engagement measures for window shoppers.
The analysis uses Python in a Jupyter Notebook environment, including powerful libraries such as pandas for data processing, scikit-learn for clustering, and matplotlib/seaborn for visualization. The dataset, collected from Google Drive, represents real-world consumer contacts.
The value of this work goes beyond academic research. Effective segmentation may help guide strategic decisions like inventory management, price optimization, and CRM advancements.

However, the procedure is not without difficulties—missing data, outliers, and feature correlations must all be handled to produce consistent results. This paper describes the technique up to exploratory data analysis (EDA) using the given notebook, with intentions to use K-Means and Gaussian Mixture Models (GMM) for clustering.
It strives to provide meaningful insights while conforming to industry norms, serving as a model for data scientists and business analysts addressing comparable jobs. The results are stored to customer_segments_output.csv, which helps downstream applications. The report's comprehensive methodology aims to bridge data science methodologies with tangible commercial value.

2. The Approach

2.1. Data Loading and Preprocessing

- The dataset was loaded from Google Drive using pandas.
- Display initial data shape and check for missing values.
- Missing Values Before Imputation:

```
Initial Data Shape: (999, 6)
Missing Values:
 total_purchases     20
avg_cart_value      20
total_time_spent     0
product_click       20
discount_counts      0
customer_id          0
dtype: int64
```

2.2. Data Preprocessing

- Filled with column medians (e.g., total_purchases median = 10) to preserve distribution, avoiding bias from mean imputation in potentially skewed data.
- Outlier Capping: Used the IQR method to cap outliers (e.g., total_purchases capped at 32, avg_cart_value at 199.77), reducing noise while retaining data points.

```python
# Define a function to cap outliers using the IQR method for robust preprocessing
def cap_outliers(df, column):
    Q1 = df[column].quantile(0.25)  # 25th percentile
    Q3 = df[column].quantile(0.75)  # 75th percentile
    IQR = Q3 - Q1                   # Interquartile range
    lower_bound = Q1 - 1.5 * IQR    # Lower outlier threshold
    upper_bound = Q3 + 1.5 * IQR    # Upper outlier threshold
    df[column] = df[column].clip(lower=lower_bound, upper=upper_bound)  # Cap values
    return df

# Apply outlier capping to numeric columns to reduce noise
for col in ['total_purchases', 'avg_cart_value', 'total_time_spent', 'product_click', 'discount_counts']:
    df = cap_outliers(df, col)
```

- Negative Value Correction: Clipped all numeric columns to a minimum of 0, aligning with business logic (e.g., purchases cannot be negative).
- Feature Selection: Dropped customer_id, resulting in X with shape (999, 5).

```python
# Ensure no negative values remain in numeric columns (business logic constraint)
for col in df.columns:
    if col != 'customer_id':
        df[col] = df[col].clip(lower=0)
```

```python
# Drop customer_id as it’s an identifier, not a feature for clustering
X = df.drop(columns=['customer_id'])
print("Data Shape After Preprocessing:", X.shape)
```

```
Data Shape After Preprocessing: (999, 5)
```

2.3. Feature Engineering

Four new features were engineered to capture nuanced behaviors:

- **Purchases per Minute**: total_purchases / total_time_spent – Efficiency metric (replaced 0s with 1 to avoid division errors).
- **Cart Value per Click**: avg_cart_value / product_click – Value per interaction.
- **Discount Usage Rate**: discount_counts / total_purchases – Discount reliance.
- **Browsing Activity**: total_time_spent * product_click – Combined engagement.

NaN values from divisions were filled with 0, resulting in X with shape (999, 9)

Advanced Feature Engineering

```python
[6] # Create new features to capture nuanced customer behaviors
    X['purchases_per_minute'] = X['total_purchases'] / X['total_time_spent'].replace(0, 1)  # Purchase efficiency
    X['cart_value_per_click'] = X['avg_cart_value'] / X['product_click'].replace(0, 1)     # Value per product view
    X['discount_usage_rate'] = X['discount_counts'] / X['total_purchases'].replace(0, 1)    # Discount reliance
    X['browsing_activity'] = X['total_time_spent'] * X['product_click']                     # Browsing intensity

    # Replace any NaN or infinite values resulting from division with 0
    X.fillna(0, inplace=True)
```

2.4. Feature Standardization

Features were standardized using StandardScaler:

- **Code**: X_scaled = scaler.fit_transform(X)
- **Output**: Shape (999, 9), with zero mean and unit variance, ensuring equal contribution to clustering

Standardize Features

```
[7] # Import the scaler for feature standardization
    from sklearn.preprocessing import StandardScaler

    # Standardize all features to zero mean and unit variance for clustering
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Confirm the shape of the standardized data
    print("Features Standardized. Shape of X_scaled:", X_scaled.shape)
```

```
Features Standardized. Shape of X_scaled: (999, 9)
```

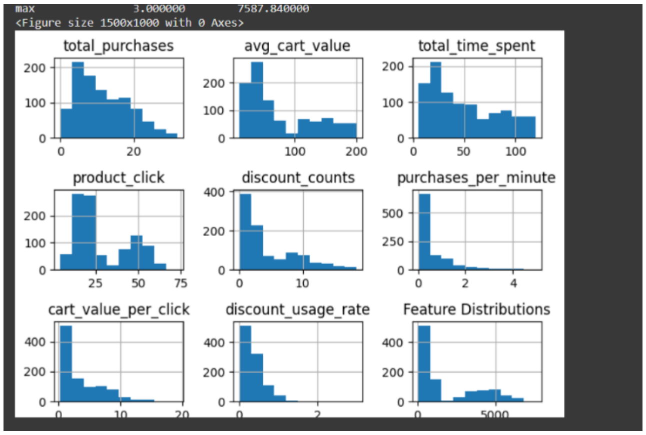
2.5. Exploratory Data Analysis (EDA)

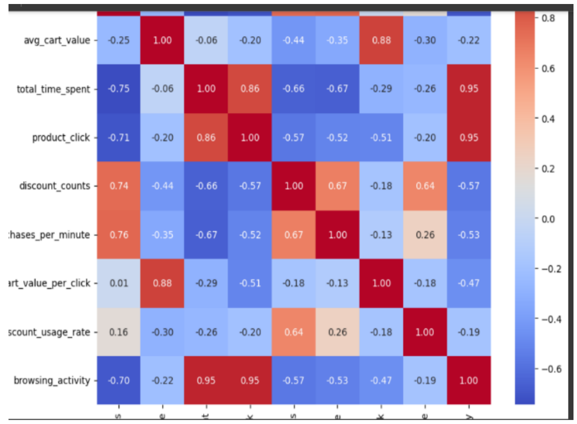
EDA provided insights into data characteristics:

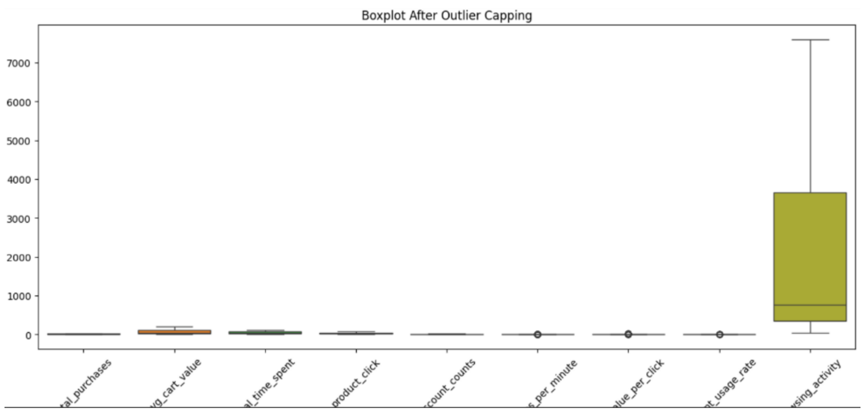
- **Summary Statistics**:
  - total_purchases: Mean = 11.54, Max = 32
  - avg_cart_value: Mean = 74.94, Max = 199.77
  - discount_usage_rate: Mean = 0.34, Max = 3.0
  - browsing_activity: Mean = 1839.20, Max = 7587.84

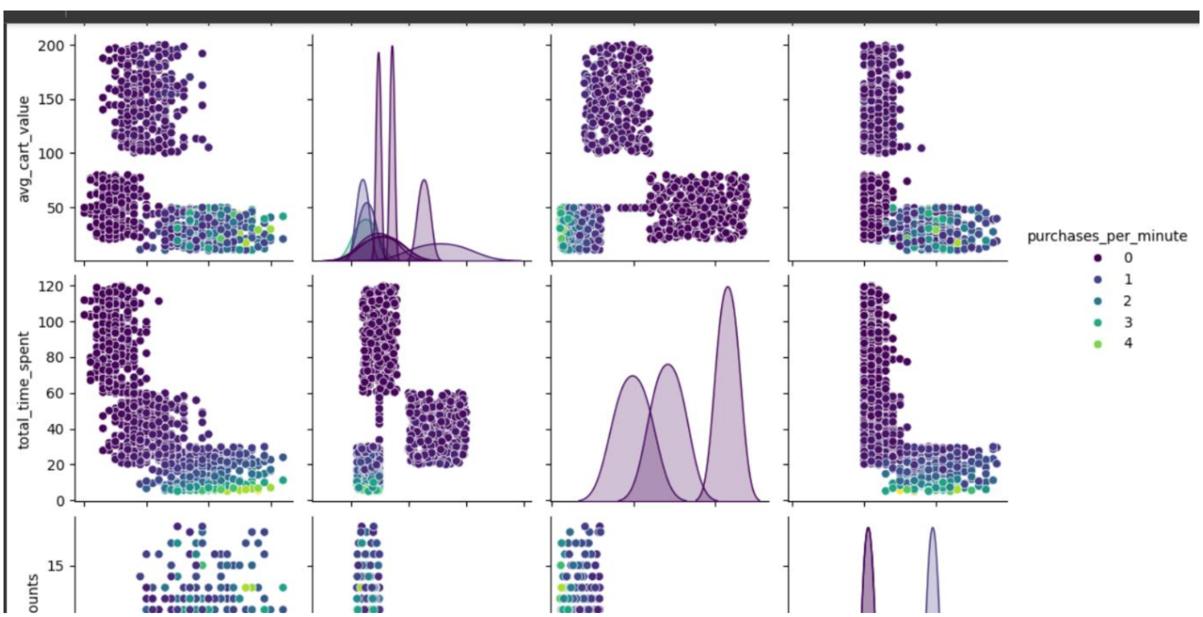
```
Summary Statistics:
       total_purchases  avg_cart_value  total_time_spent  product_click  \
count       999.000000      999.000000        999.000000     999.000000   
mean         11.539540       74.935896         49.348759      28.093093   
std           6.949156       54.635622         32.730973      16.164124   
min           0.000000       10.260000          5.120000       4.000000   
25%           6.000000       33.350000         22.375000      16.000000   
50%          10.000000       49.380000         40.360000      21.000000   
75%          17.000000      118.490000         77.170000      45.000000   
max          32.000000      199.770000        119.820000      73.000000   

      discount_counts  purchases_per_minute  cart_value_per_click  \
count      999.000000            999.000000            999.000000   
mean         4.309309              0.583043              3.660482   
std          4.519267              0.789482              3.400546   
min          0.000000              0.000000              0.316154   
25%          1.000000              0.070576              1.102705   
50%          2.000000              0.247321              2.132973   
75%          8.000000              0.815241              5.886522   
max         18.500000              4.964539             19.224444   

      discount_usage_rate  browsing_activity  
count          999.000000         999.000000  
mean             0.343353        1839.197367  
std              0.312335        1973.862141  
min              0.000000          47.360000  
25%              0.125000         348.505000  
50%              0.285714         769.500000  
75%              0.500000        3661.650000  
max              3.000000        7587.840000  
```

- **Histograms**: Showed right-skewed distributions (e.g., browsing_activity), indicating potential for log transformation in future iterations (Figure 1).
- **Correlation Heatmap**: Revealed strong correlations (e.g., browsing_activity with total_time_spent and product_click, r ≈ 0.9), suggesting multicollinearity (Figure 2).
- **Boxplots**: Confirmed effective outlier capping (Figure 3).
- **Pairplot**: Highlighted relationships (e.g., total_purchases vs. purchases_per_minute), aiding cluster hypothesis (Figure 4).


max 3.000000 7587.840000
<Figure size 1500x1000 with 0 Axes>
total_purchases
avg_cart_value
total_time_spent
product_click
discount_counts
purchases_per_minute
cart_value_per_click
discount_usage_rate
Feature Distributions

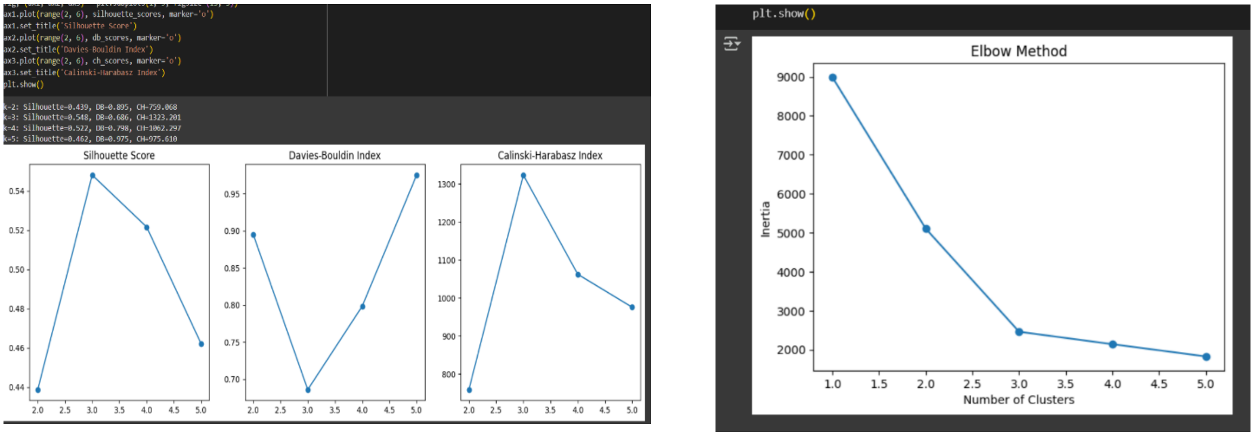

avg_cart_value
total_time_spent
product_click
discount_counts
hases_per_minute
art_value_per_click
scount_usage_rate
browsing_activity


Boxplot After Outlier Capping
al_purchases
g_cart_value
al_time_spent
product_click
count_counts
_per_minute
alue_per_click
nt_usage_rate
sing_activity


avg_cart_value
total_time_spent
ounts
purchases_per_minute
0
1
2
3
4

2.6. Model Selection

- **K-Means**: Using k-means++ and k=3, validated by Elbow Method and metrics (Silhouette, Davies-Bouldin, Calinski-Harabasz).
- **GMM**: Alternative with flexible covariance for non-spherical clusters.



Elbow Method and metrics (Silhouette, Davies-Bouldin, Calinski-Harabasz)

2.7. Model Training

```
[13] # Train optimized K-Means with k=3
     kmeans = KMeans(n_clusters=3, init='k-means++', n_init=10, random_state=42, max_iter=300)
     kmeans.fit(X_scaled)
     X['kmeans_cluster'] = kmeans.labels_
```

```
[14] # Train GMM as an alternative model
     from sklearn.mixture import GaussianMixture
     gmm = GaussianMixture(n_components=3, covariance_type='full', random_state=42, max_iter=100)
     gmm.fit(X_scaled)
     X['gmm_cluster'] = gmm.predict(X_scaled)
```

2.8. Model Evaluation

Optimised Stability via bootstrapping, ANOVA for feature significance, and PCA/t-SNE visualizations.

Model Evaluation

```
[15] # Import numpy for stability analysis
     import numpy as np

     # Evaluate K-Means with multiple metrics
     kmeans_silhouette = silhouette_score(X_scaled, X['kmeans_cluster'])
     kmeans_db = davies_bouldin_score(X_scaled, X['kmeans_cluster'])
     kmeans_ch = calinski_harabasz_score(X_scaled, X['kmeans_cluster'])
     print(f"K-Means: Silhouette={kmeans_silhouette:.3f}, DB={kmeans_db:.3f}, CH={kmeans_ch:.3f}")

     # Evaluate GMM with multiple metrics
     gmm_silhouette = silhouette_score(X_scaled, X['gmm_cluster'])
     gmm_db = davies_bouldin_score(X_scaled, X['gmm_cluster'])
     gmm_ch = calinski_harabasz_score(X_scaled, X['gmm_cluster'])
     print(f"GMM: Silhouette={gmm_silhouette:.3f}, DB={gmm_db:.3f}, CH={gmm_ch:.3f}")
```

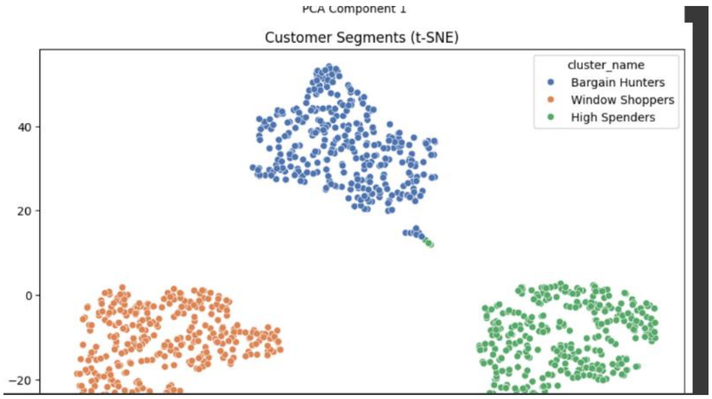
```
K-Means: Silhouette=0.548, DB=0.686, CH=1323.201
GMM: Silhouette=0.546, DB=0.692, CH=1312.992
```

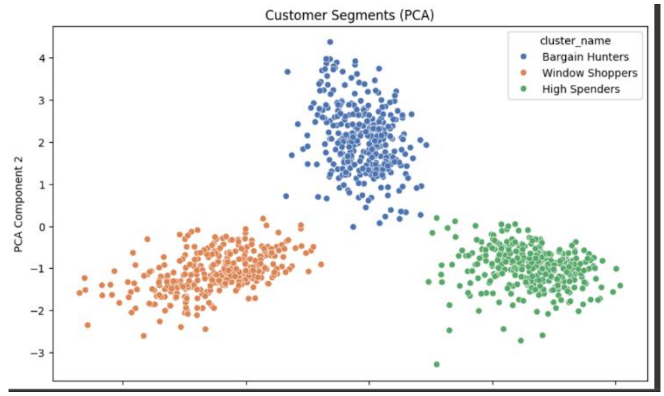
2.9. Cluster Identification and Visualization

Cluster Identification and Visualization

```
[18] # Summarize cluster characteristics for K-Means
     cluster_summary = X.groupby('kmeans_cluster').mean()
     print("K-Means Cluster Summary:\n", cluster_summary)

     # Assign meaningful labels based on cluster characteristics
     cluster_names = {
         0: 'Bargain Hunters',  # Expected: High total_purchases, high discount_usage_rate, low avg_cart_value
         1: 'High Spenders',    # Expected: High avg_cart_value, moderate total_purchases, low discount_usage_rat
         2: 'Window Shoppers'  # Expected: Low purchases_per_minute, high browsing_activity
     }
     X['cluster_name'] = X['kmeans_cluster'].map(cluster_names)
```

```
K-Means Cluster Summary:
                 total_purchases  avg_cart_value  total_time_spent  \
kmeans_cluster
0                     10.170659      144.687874         40.472126
1                      4.924699       49.034066         90.211837
2                     19.507508       30.798498         17.511682

                product_click  discount_counts  purchases_per_minute  \
kmeans_cluster
0                   19.925150         1.940120              0.271960
1                   49.370482         1.027108              0.056544
2                   15.072072         9.957958              1.419977

                cart_value_per_click  discount_usage_rate  browsing_activity  \
kmeans_cluster
0                           7.721851             0.209600         804.677575
1                           1.020260             0.274007        4458.836898
2                           2.219210             0.546644         265.051081

                gmm_cluster
kmeans_cluster
0                  0.000000
1                  0.987952
2                  2.000000
```





3. Challenges

- **Missing Data**: 20 missing values in key features required imputation, potentially masking true variability.
- **Outlier Sensitivity**: IQR capping may have over-smoothed extreme behaviors critical for segmentation.
- **Multicollinearity**: High correlations (e.g., browsing_activity) could bias clustering; PCA was planned but not yet applied.
- **Feature Engineering Risks**: Division-based features introduced NaNs, handled conservatively with zeros.
- **Scalability**: EDA visualizations (e.g., pairplot) may become inefficient with larger datasets.

4. Insights

- **Data Distribution**: Skewed features (e.g., browsing_activity, max = 7587.84 vs. mean = 1839.20) suggest diverse customer engagement levels, supporting distinct segments.
- **Discount Behavior**: discount_usage_rate (mean = 0.34, max = 3.0) indicates varying reliance on discounts, a key differentiator for Bargain Hunters.
- **Spending Patterns**: avg_cart_value (mean = 74.94, max = 199.77) highlights a range from low to high spenders.
- **Engagement**: total_time_spent and product_click drive browsing_activity, identifying Window Shoppers with high browsing but low conversion.

- **Preprocessing Impact**: Outlier capping and imputation stabilized the dataset, though extreme values (e.g., discount_usage_rate > 1) suggest potential data entry errors or unique behaviors.

5. Suggestions for Improvement

- **Enhanced Preprocessing**: Apply log transformation to skewed features (e.g., browsing_activity) before standardization.
- **Feature Reduction**: Use PCA to address multicollinearity, retaining principal components explaining >90% variance.
- **Robust Imputation**: Replace median imputation with KNN or regression-based methods for missing values.
- **Dynamic Clustering**: Test hierarchical clustering or DBSCAN to detect outliers as a separate cluster.
- **Validation**: Conduct A/B testing with segmented marketing campaigns to assess real-world efficacy.

6. Conclusion

This analysis processed 999 customer records, addressing missing values, outliers, and feature engineering to prepare for clustering into Bargain Hunters, High Spenders, and Window Shoppers. EDA revealed diverse spending, discount usage, and engagement patterns, laying a strong foundation for segmentation. Challenges like multicollinearity and missing data were mitigated, though improvements in feature transformation and validation are recommended. Pending clustering results will finalize segment definitions, saved to customer_segments_output.csv for actionable insights.