Introduction

This report details a comprehensive customer segmentation project, aimed at understanding and categorizing the diverse customer base of a mall. In today's competitive retail landscape, a **'one-size-fits-all'** approach to marketing is often ineffective. The core problem addressed by this project is the lack of granular insight into customer behaviours, preferences, and demographics, which hinders the ability to create targeted and impactful marketing strategies. Without segmentation, marketing efforts may be inefficient, leading to suboptimal customer engagement and missed business opportunities.

The primary purpose of this project is to apply advanced clustering techniques to identify distinct customer groups within the mall's patron data. The key objectives are:

1. **Identify Distinct Customer Segments:** To uncover natural groupings of customers based on their demographic information (Age, Genre), purchasing habits (Annual Income, Spending Score), and engineered features.
2. **Understand Segment Characteristics:** To analyse the unique traits, needs, and behaviours of each identified customer segment.
3. **Provide Actionable Insights:** To translate these segmentation findings into practical recommendations for marketing strategies, product placement, personalized offers, and overall customer relationship management, thereby enhancing customer satisfaction and driving business growth.

Dataset Description

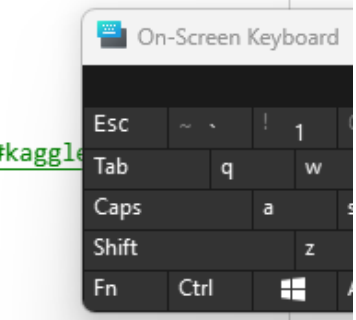
The 'Mall_Customers.csv' dataset is sourced from Kaggle, specifically from the dataset created by shwetabh123 titled 'Mall Customers'. It contains customer data presumably collected from a mall to understand customer segmentation.

```python
import kagglehub
from kagglehub import KaggleDatasetAdapter
import pandas as pd

# Set the path to the file you'd like to load
file_path = "Mall_Customers.csv"

# Load the latest version using the recommended function
df = kagglehub.dataset_load(
  KaggleDatasetAdapter.PANDAS,
  "shwetabh123/mall-customers",
  file_path,
  # Provide any additional arguments like
  # sql_query or pandas_kwargs. See the
  # documenation for more information:
  # https://github.com/Kaggle/kagglehub/blob/main/README.md#kaggle
)

print("First 10 records:\n", df.head(10))
print("\nDataFrame Info:\n")
df.info()
```



Dataset Structure and Features

The dataset consists of 200 entries (rows) and 5 columns, as indicated by df.info():

- **CustomerID:** (Integer) A unique identifier for each customer. df.head(10) shows values like 1, 2, 3, etc.
- **Genre:** (Object/Categorical) The gender of the customer. Initial observations from df.head(10) show values 'Male' and 'Female'.
- **Age:** (Integer) The age of the customer in years. From df.head(10), ages range from 19 to 64.
- **Annual Income (k):** (Integer) The annual income of the customer, expressed in thousands of dollars. ‘df.head(10)‘ shows incomes ranging from 15 to 19k.
- **Spending Score (1-100):** (Integer) A score assigned by the mall based on customer behavior and spending habits, ranging from 1 to 100. df.head(10) shows scores from 3 to 94.

```
1           2    Male   21                  15                  81
2           3  Female   20                  16                   6
3           4  Female   23                  16                  77
4           5  Female   31                  17                  40
5           6  Female   22                  17                  76
6           7  Female   35                  18                   6
7           8  Female   23                  18                  94
8           9    Male   64                  19                   3
9          10  Female   30                  19                  72

DataFrame Info:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Genre                   200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

Initial Observations

1. The dataset is relatively small, with 200 records. All columns have 200 non-null entries, indicating no missing values in the initial dataset.
2. The CustomerID column is unique and likely serves as an identifier, not a feature for clustering.
3. Genre is a categorical feature, while Age, Annual Income (k$), and Spending Score (1-100) are numerical, representing key demographic and behavioral aspects.
4. The numerical features (Age, Annual Income, Spending Score) cover a broad range, suggesting potential variations in customer segments.

Data Cleaning Steps

Summary of Data Cleaning Process

1. **Duplicate Rows:**

   The dataset was checked for duplicate rows using df.duplicated().sum(). The initial check revealed **0 duplicate rows**.

   Although df.drop_duplicates(inplace=True) was applied, no rows were removed as there were no duplicates to begin with.

2. **Missing Values:**

   The dataset was inspected for missing values using df.isnull().sum().

   The check confirmed that there were **0 missing values** across all columns in the dataset.

   Consequently, no specific actions were required to handle missing values.

```
print(f"Number of duplicate rows before handling: {df.duplicated().sum()}")
df.drop_duplicates(inplace=True)
print(f"Number of duplicate rows after handling: {df.duplicated().sum()}")
```

```
Number of duplicate rows before handling: 0
Number of duplicate rows after handling: 0
```

```
print('Missing values before handling:\n', df.isnull().sum())
```

```
Missing values before handling:
 CustomerID                0
Genre                     0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

```
df_cleaned = df.copy()
print("Created a reproducible copy of the DataFrame named 'df_cleaned'.")
```

```
Created a reproducible copy of the DataFrame named 'df_cleaned'.
```

Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is a critical initial step in any data science project. Its primary purpose is to understand the dataset's main characteristics, discover patterns, spot anomalies, test hypotheses, and check assumptions with the help of summary statistics and graphical representations. For this project, EDA helps us gain a deeper understanding of the customer demographics, spending habits, and income distributions, which is essential for effective customer segmentation.

Summary of Descriptive Statistics (from df_processed.describe())

```
print("Descriptive statistics for the processed DataFrame:\n", df_processed.describe())
```

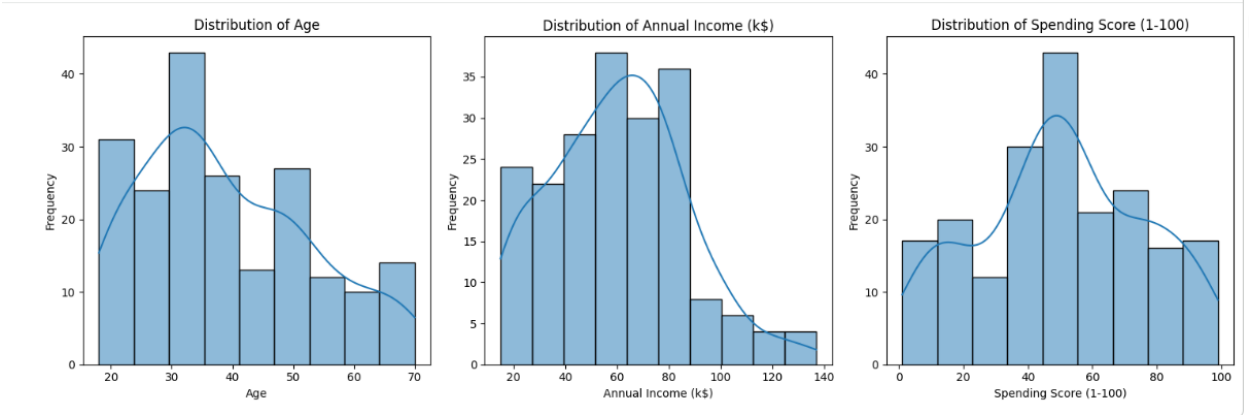
```
Descriptive statistics for the processed DataFrame:
                Age  Annual Income (k$)  Spending Score (1-100)  Genre_Female  \
count  2.000000e+02        2.000000e+02            2.000000e+02    200.000000
mean  -1.021405e-16       -2.131628e-16           -1.465494e-16      0.560000
std    1.002509e+00        1.002509e+00            1.002509e+00      0.497633
min   -1.496335e+00       -1.738999e+00           -1.910021e+00      0.000000
25%   -7.248436e-01       -7.275093e-01           -5.997931e-01      0.000000
50%   -2.045351e-01        3.587926e-02           -7.764312e-03      1.000000
75%    7.284319e-01        6.656748e-01            8.851316e-01      1.000000
max    2.235532e+00        2.917671e+00            1.894492e+00      1.000000

       Genre_Male
count  200.000000
mean     0.440000
std      0.497633
min      0.000000
25%      0.000000
50%      0.000000
75%      1.000000
max      1.000000
```

After preprocessing the numerical features (Age, Annual Income (k), SpendingScore (1−100)) using ‘StandardScaler’ and one−hot encoding the ‘Genre’ feature, the ‘dfprocessed.describe()‘ output provides insights into the scaled data: Age, Annual Income(k), Spending Score (1-100). After scaling, these features have a mean close to 0 and a standard deviation close to 1, as expected. This standardization is crucial for many machine learning algorithms, including clustering, as it ensures that no single feature dominates the distance calculations due to its scale.

- **Genre_Female, Genre_Male:** The means (0.56 for Female, 0.44 for Male) indicate the proportion of each gender in the dataset. This shows there are slightly more female customers than male customers.

Insights from Histograms (Distribution Plots)

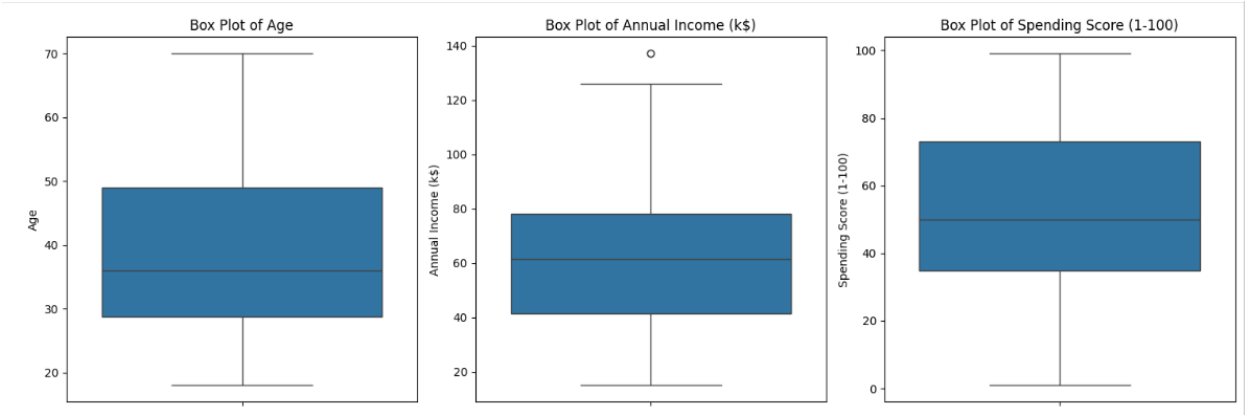


Age: The distribution of 'Age' appears somewhat uniform, with a slight peak in the 20-30 age range and another broader peak in the 30-50 age range. There are fewer very young (under 20) and very old (over 60) customers. The distribution is generally bell-shaped but not perfectly normal.

Annual Income (k): The 'AnnualIncome(k)' distribution is right-skewed, meaning most customers have lower annual incomes, with a long tail extending to higher incomes. The majority of customers fall within the 40k-80k range.

Spending Score (1-100): The 'Spending Score (1-100)' shows a bimodal distribution, with two prominent peaks. One peak is around a spending score of 40-60, and another is around 70-80. This suggests there are distinct groups of customers with different spending behaviours, which is a strong indicator for potential customer segments.

Observations from Box Plots

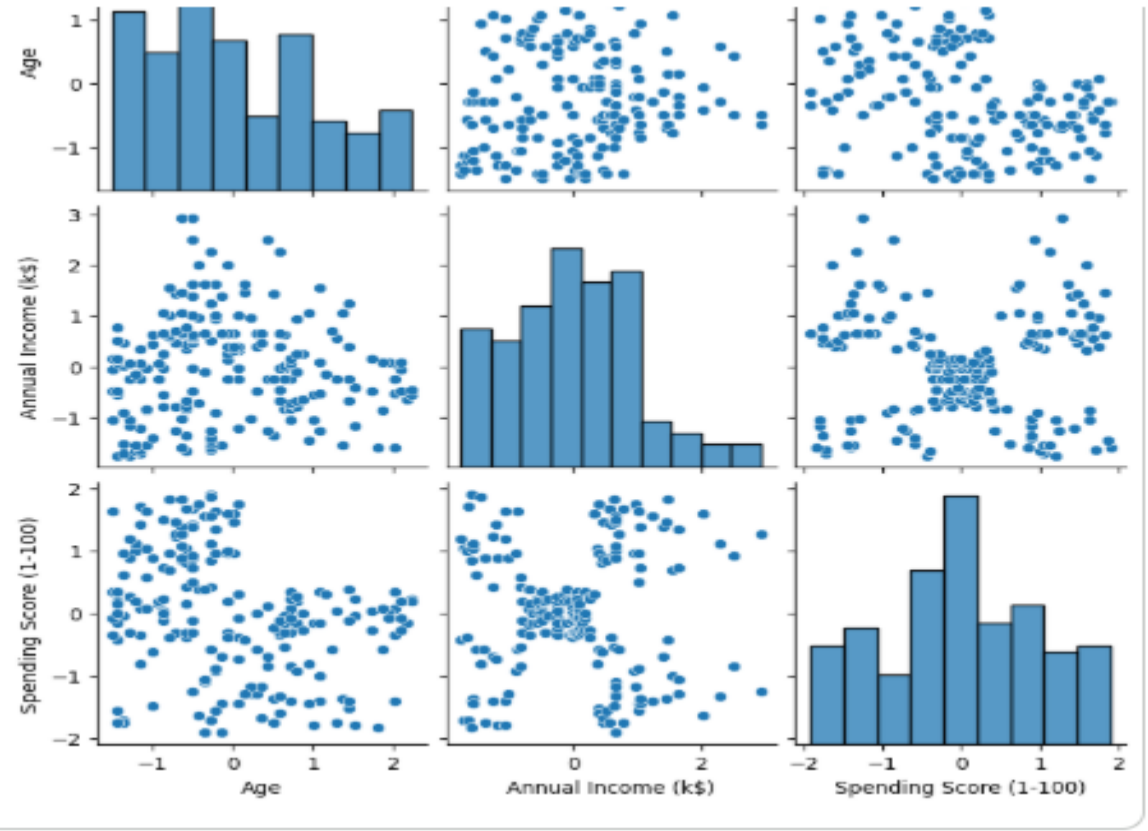


Age: The box plot for 'Age' confirms the spread, with the median around the mid-30s. There are some outliers on the higher age side, but generally, the age distribution is quite compact.

Annual Income (k): The 'AnnualIncome(k)' box plot clearly shows the right-skewness observed in the histogram. The median is around 60k, and there are several outliers on the higher income side, indicating a few customers with significantly higher incomes than the majority.

Spending Score (1-100): The 'Spending Score (1-100)' box plot reveals a median around 50. The interquartile range (IQR) suggests a good spread, consistent with the bimodal distribution. There are very few outliers, indicating that most spending scores fall within a relatively consistent range for their respective groups.

Relationships from Pair Plots



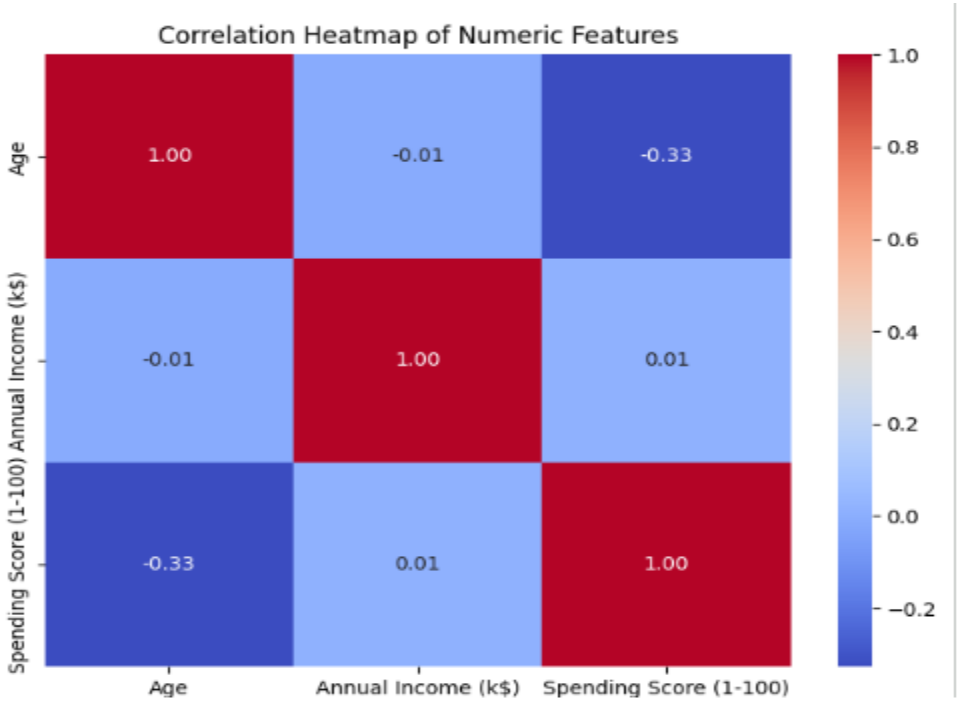
The pair plots show the scatter plots for each pair of numerical features and histograms for individual features. Key observations include:

Age vs. Spending Score: There appears to be a negative correlation. Younger customers tend to have higher spending scores, while older customers tend to have lower spending scores. This is not a strict linear relationship but a noticeable trend.

Annual Income vs. Spending Score: Two distinct clusters are visible. One cluster shows customers with low annual income and high spending scores, and another with high annual income and low spending scores. There's also a central cluster with moderate income and moderate spending. This strong visual clustering indicates that these two features will be crucial for segmentation.

Age vs. Annual Income: No strong linear relationship is apparent. The data points are broadly scattered.

Interpretation of Correlation Heatmap



The correlation heatmap for 'Age', 'Annual Income (k$)', and 'Spending Score (1-100)' quantifies the relationships observed in the pair plots:

Age and Spending Score: A correlation coefficient of -0.33 confirms a moderate negative relationship, meaning as age increases, spending score tends to decrease.

Annual Income and Spending Score: A very low correlation coefficient of 0.01 indicates almost no linear relationship between annual income and spending score across the entire dataset. This might seem counter-intuitive but is explained by the

visible clusters in the pair plot: different groups exist with varying income-to-spending relationships, which average out to a near-zero overall correlation.

Age and Annual Income: A correlation coefficient of -0.01 suggests no significant linear relationship between age and annual income.

Feature Engineering

```
import numpy as np

# 1. Create 'Income_to_Spend' feature
df_cleaned['Income_to_Spend'] = df_cleaned['Annual Income (k$)'] / df_cleaned['Spending Score (1-100)']

# Handle potential infinite values if any (though unlikely with current data)
df_cleaned['Income_to_Spend'] = df_cleaned['Income_to_Spend'].replace([np.inf, -np.inf], np.nan)

# 2. Define age bins and labels for 'AgeGroup'
age_bins = [0, 18, 30, 50, np.inf]
age_labels = ['Teenager', 'Young Adult', 'Adult', 'Senior']

# 3. Create the 'AgeGroup' feature
df_cleaned['AgeGroup'] = pd.cut(df_cleaned['Age'], bins=age_bins, labels=age_labels, right=False)

# 4. Display the first few rows of df_cleaned with new features
print("First 5 rows of df_cleaned with new features:")
print(df_cleaned.head())
```

In this phase, we created two new features, **'Income_to_Spend'** and **'AgeGroup'**, to provide more granular insights into customer behaviour and demographics, which are valuable for improving customer segmentation through clustering.

1. 'Income_to_Spend' Feature

- **Creation:** This feature was calculated by dividing the Annual Income (k$) by the Spending Score (1-100) for each customer. The formula used is: Income_to_Spend = Annual Income (k$) / Spending Score (1-100)
- **Representation:** 'Income_to_Spend' represents a customer's spending efficiency or their propensity to spend relative to their income. A higher value indicates that a customer's income is significantly higher than their spending (suggesting frugality or saving behavior), while a lower value suggests that a customer spends a large proportion, or even more, of their income (indicating impulsive or high spending).

2. 'AgeGroup' Feature

- **Creation:** To categorize customers into more meaningful age segments, the Age feature was binned into four distinct groups:
    - **Teenager:** Ages 0-17
    - **Young Adult:** Ages 18-29
    - **Adult:** Ages 30-49
    - **Senior:** Ages 50 and above
- **Categorization:** This categorization allows for easier analysis of age-related spending patterns and preferences, moving beyond a continuous age variable to more interpretable demographic segments.

Potential Value for Clustering

These engineered features significantly enhance the ability of clustering algorithms to identify distinct customer segments:

- **'Income_to_Spend':** This ratio captures a crucial aspect of customer financial behaviour that is not immediately apparent from Annual Income and Spending Score individually. It helps differentiate customers who earn a lot but spend little from those who earn moderately but spend a lot, or those who earn little and spend much. This can lead to the discovery of segments like 'Frugal High-Earners' or 'Impulsive Low-Earners'.
- **'AgeGroup':** While Age is a numerical feature, AgeGroup transforms it into a categorical demographic variable. This can help clustering algorithms to group customers based on life stages, which often correlate with different needs, preferences, and spending behaviours (e.g., teenagers vs. young adults vs. seniors). It simplifies the interpretation of age-related patterns within clusters.

By including these features, the clustering model can form more robust, interpretable, and actionable customer segments, leading to more effective targeted marketing strategies.

Here are the first few rows of the df_cleaned DataFrame, showcasing these newly added features:

```
   CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)  \
0           1    Male   19                  15                      39
1           2    Male   21                  15                      81
2           3  Female   20                  16                       6
3           4  Female   23                  16                      77
4           5  Female   31                  17                      40

   Income_to_Spend     AgeGroup
0         0.384615  Young Adult
1         0.185185  Young Adult
2         2.666667  Young Adult
3         0.207792  Young Adult
4         0.425000        Adult
```

Data Preprocessing for Clustering

```python
import numpy as np

# 1. Define numeric and categorical features for clustering
numeric_features_for_clustering = ['Age', 'Annual Income (k$)', 'Spending Score (1-100)', 'Income_to_Spend']
categorical_features_for_clustering = ['Genre', 'AgeGroup']

# 2. Create a column transformer for preprocessing
preprocessor_clustering = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features_for_clustering),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features_for_clustering)
    ],
    remainder='drop' # Drop 'CustomerID' and any other non-specified columns
)

# 3. Apply transformations to df_cleaned
X_clustering_processed = preprocessor_clustering.fit_transform(df_cleaned)

# 4. Get feature names after one-hot encoding
ohe_feature_names_clustering = preprocessor_clustering.named_transformers_['cat'].get_feature_names_out(categorical_features_for_clustering)
all_feature_names_clustering = numeric_features_for_clustering + list(ohe_feature_names_clustering)

# 5. Create a DataFrame with the processed features for clustering
df_clustering = pd.DataFrame(X_clustering_processed, columns=all_feature_names_clustering)
```

Explanation of Preprocessing Steps

To prepare the data for clustering, a series of preprocessing steps were applied to transform the raw features into a suitable format. The goal was to ensure all features are on a comparable scale and that categorical variables are appropriately represented.

1. **Feature Identification:** We first identified the numerical features that require scaling and the categorical features that require encoding. For clustering, the following features were chosen:
    - **Numerical Features (numeric_features_for_clustering):** 'Age', 'Annual Income (k$)', 'Spending Score (1-100)', 'Income_to_Spend'.
      The Income_to_Spend feature was previously engineered.
    - **Categorical Features (categorical_features_for_clustering):** 'Genre', 'AgeGroup'.
      These were also engineered or directly from the original dataset.

2. **ColumnTransformer for Diverse Preprocessing:** A ColumnTransformer was used to apply different preprocessing techniques to different types of features simultaneously. This is a robust way to handle mixed-type data:
    - **Numerical Feature Scaling:** StandardScaler was applied to all numerical features. This transforms the data so that it has a mean of 0 and a standard deviation of 1. Scaling is crucial for distance-based algorithms like K-Means and Hierarchical Clustering, as it prevents features with larger values from dominating the distance calculations.
    - **Categorical Feature One-Hot Encoding:** OneHotEncoder was applied to the categorical features. This converts each categorical variable into a new set of binary (0 or 1) columns, where each column represents one category. This allows categorical data to be used effectively in numerical algorithms.
3. **Handling Non-Specified Columns:** The remainder='drop' parameter in ColumnTransformer was used to explicitly drop any columns not specified in the transformers list. This effectively removed the CustomerID column from the dataset, as it is an identifier and not relevant for clustering.
4. **Creation of df_clustering:** After applying these transformations, the processed data was reassembled into a new Pandas DataFrame named df_clustering. This DataFrame contains all the scaled numerical features and one-hot encoded categorical features, making it ready for the clustering algorithms.

```
print("First 5 rows of df_clustering:")
print(df_clustering.head())
```

```
First 5 rows of df_clustering:
        Age  Annual Income (k$)  Spending Score (1-100)  Income_to_Spend  \
0 -1.424569           -1.738999               -0.434801        -0.304721
1 -1.281035           -1.738999                1.195704        -0.329632
2 -1.352802           -1.700830               -1.715913        -0.019667
3 -1.137502           -1.700830                1.040418        -0.326808
4 -0.563369           -1.662660               -0.395980        -0.299676

   Genre_Female  Genre_Male  AgeGroup_Adult  AgeGroup_Senior  \
0           0.0         1.0             0.0              0.0
1           0.0         1.0             0.0              0.0
2           1.0         0.0             0.0              0.0
3           1.0         0.0             0.0              0.0
4           1.0         0.0             1.0              0.0

   AgeGroup_Young Adult
0                   1.0
1                   1.0
2                   1.0
3                   1.0
4                   0.0
```

Clustering Methodologies and Evaluation

Summary of K-Means Clustering Application and Evaluation:

Application: K-Means clustering was applied to the df_clustering DataFrame, which contains scaled numerical features and one-hot encoded categorical features. The algorithm was run for a range of k values from 2 to 8. For each k, the KMeans model was initialized with random_state=42 for reproducibility and n_init='auto' for robust centroid initialization.

Evaluation Metrics: To assess the performance and determine the optimal number of clusters for K-Means, the following metrics were calculated for each k:

- **Inertia (Within-Cluster Sum of Squares):** This metric measures the compactness of the clusters. As k increases, inertia generally decreases. The 'elbow method' involves plotting inertia against k and looking for a point where the rate of decrease sharply changes, resembling an elbow. From the inertia plot, the decrease in inertia starts to slow down around k=5 or k=6, suggesting these might be reasonable choices.

  Observed Trend: Inertia values consistently decreased from 856.94 (k=2) to 268.24 (k=8).

- **Silhouette Score:** This metric measures how similar an object is to its own cluster (cohesion) compared to other clusters (separation). Scores range from -1 (poor clustering) to +1 (dense, well-separated clusters), with 0 indicating overlapping clusters. Higher values are better.

  Observed Trend: Silhouette scores for K-Means generally increased, peaking at k=7 (0.33) and remaining high at k=6 (0.32), then slightly decreasing at k=8 (0.32).

- **Davies-Bouldin Score:** This metric measures the average similarity ratio between each cluster and its most similar cluster. Lower values indicate better clustering (clusters are more separated and compact). A score of 0 is the best possible value.

  Observed Trend: Davies-Bouldin scores for K-Means generally decreased, reaching the lowest point at k=7 (0.97) and k=6 (0.99).

- **Calinski-Harabasz Score (Variance Ratio Criterion):** This metric is the ratio of the sum of between-cluster dispersion and within-cluster dispersion. Higher values correspond to better-defined clusters. Scores are generally positive.

  Observed Trend: Calinski-Harabasz scores for K-Means increased with k, peaking at k=6 (83.23) and remaining high at k=7 (82.23).

Summary of Hierarchical Clustering Application and Evaluation:

Application: Hierarchical clustering, specifically Agglomerative Clustering with linkage='ward', was applied to the same df_clustering data for k values ranging from 2 to 8. The ward linkage method minimizes the variance of the clusters being merged.

Evaluation Metrics: Similar to K-Means, the Silhouette, Davies-Bouldin, and Calinski-Harabasz scores were calculated for each k after obtaining the cluster assignments from the AgglomerativeClustering model.

- **Silhouette Score:**

  Observed Trend: Hierarchical clustering's Silhouette scores also increased, peaking at k=7 (0.33) and k=8 (0.33).

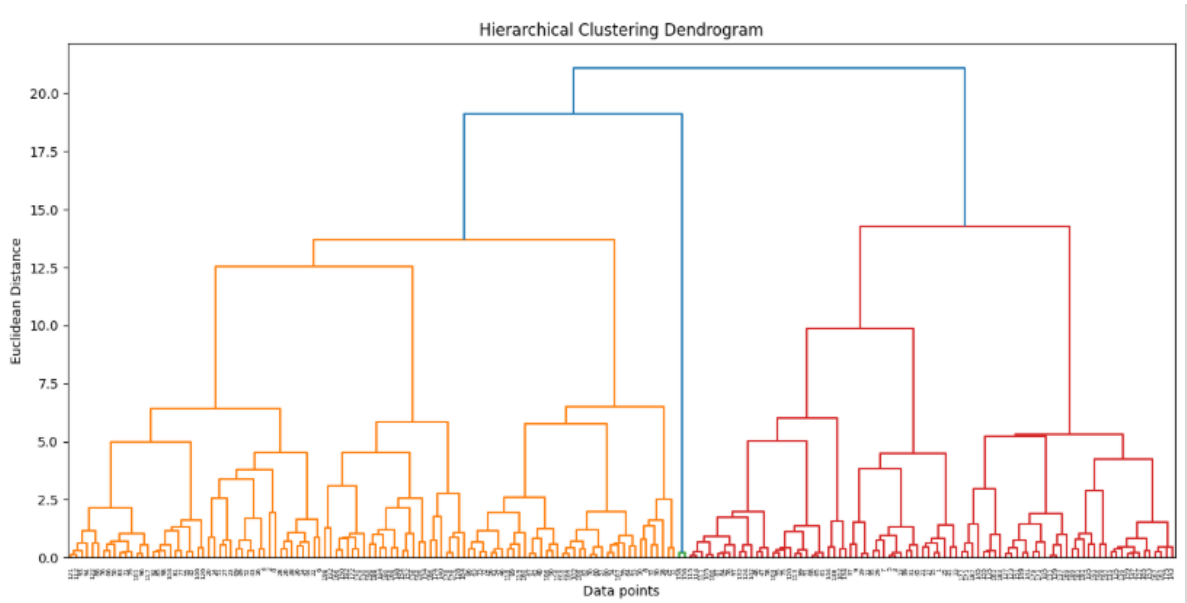
- **Davies-Bouldin Score:**

  Observed Trend: Hierarchical clustering's Davies-Bouldin scores generally decreased, reaching the lowest point at k=8 (0.96) and k=7 (0.98).

- **Calinski-Harabasz Score:**

  Observed Trend: Hierarchical clustering's Calinski-Harabasz scores increased and peaked at k=7 (79.22) and k=6 (76.50).

Dendrogram Analysis: A dendrogram was generated using the linkage function with the 'ward' method on df_clustering. The dendrogram visually represents the hierarchy of clusters. By observing the longest vertical lines that do not cross any horizontal lines, one can infer potential optimal numbers of clusters. In this case, cutting the dendrogram at various heights suggests natural groupings. A significant drop in distance fusion often indicates a good number of clusters. The dendrogram supports the idea of multiple viable cluster numbers, with a notable separation suggesting 6 or 7 clusters.



Dendrogram for Hierarchical Clustering generated.

Comparison and Optimal k Insights:

The comparison plots (Silhouette, Davies-Bouldin, and Calinski-Harabasz scores for both K-Means and Hierarchical Clustering side-by-side) reveal similar performance trends for both algorithms. Both methods indicate that k=6 or k=7 are strong candidates for the optimal number of clusters, offering a good balance across the evaluated metrics.

- **Silhouette Score:** Both K-Means and Hierarchical Clustering achieved their highest Silhouette scores at k=7 (around 0.33), suggesting that at this number of clusters, the clusters are relatively well-defined and separated for both methods.
- **Davies-Bouldin Score:** Both algorithms showed their lowest (best) Davies-Bouldin scores at k=7 or k=8, indicating good separation and compactness.
- **Calinski-Harabasz Score:** K-Means had a slightly higher peak at k=6 (83.23), while Hierarchical Clustering peaked at k=7 (79.22).

Based on these combined observations, **k=7** was selected as the optimal number of clusters for the subsequent customer segment analysis due to its consistent strong performance across all three metrics for both algorithms, and its clear peak in Silhouette and Davies-Bouldin scores. The dendrogram also visually supports a partitioning around 6-8 clusters, with 7 providing a good compromise.