# Clustering models in consumer segmentation

**Raymond Tsao**
Modern Statistical Prediction and Machine Learning
Final Project
r112358@berkeley.edu

## 1 Introduction

In the dynamic landscape of today's markets, businesses are continually innovating to connect with their diverse consumer base. Consumer segmentation, a pivotal tool in this pursuit, enables organizations to move beyond one-size-fits-all approaches and tailor marketing efforts to specific consumer segments.

One widely employed method for dissecting consumer behavior is through the implementation of the RFM model. This strategic approach involves assessing each transaction along three fundamental metrics: recency, frequency, and monetary value [1]. By scrutinizing how recently a customer made a purchase, how often they engage with the brand, and the monetary value of their transactions, businesses can categorize consumers into distinct segments. This segmentation not only aids in identifying high-value customers but also provides a nuanced understanding of diverse customer preferences and behaviors.

In this project, different clustering models is used to perform customer segmentation on an E-commerce dataset sourced from Kaggle [2]. The resulting segmentation and relevant metrics are subsequently compared across various methods, in order to evaluate the effectiveness of different clustering techniques in delineating customer segments.

## 2 Data preprocessing

### 2.1 Data

The E-commerce dataset is sourced from Kaggle, comprising 541,909 samples and 8 features. These features include invoice number, stock code, description, quantity, invoice date, unit price, consumer ID, and country. The first few entries of the data is shown below in Figure 1.

| | InvoiceNo | StockCode | Description | Quantity | InvoiceDate | UnitPrice | CustomerID | Country |
|---|---|---|---|---|---|---|---|---|
| **0** | 536365 | 85123A | WHITE HANGING HEART T-LIGHT HOLDER | 6 | 2010-12-01 08:26:00 | 2.55 | 17850.0 | United Kingdom |
| **1** | 536365 | 71053 | WHITE METAL LANTERN | 6 | 2010-12-01 08:26:00 | 3.39 | 17850.0 | United Kingdom |
| **2** | 536365 | 84406B | CREAM CUPID HEARTS COAT HANGER | 8 | 2010-12-01 08:26:00 | 2.75 | 17850.0 | United Kingdom |
| **3** | 536365 | 84029G | KNITTED UNION FLAG HOT WATER BOTTLE | 6 | 2010-12-01 08:26:00 | 3.39 | 17850.0 | United Kingdom |
| **4** | 536365 | 84029E | RED WOOLLY HOTTIE WHITE HEART. | 6 | 2010-12-01 08:26:00 | 3.39 | 17850.0 | United Kingdom |

**Figure 1**: Initial entries of the dataset

### 2.2 Preprocessing pipeline

The data preprocessing pipeline is shown in Figure 2.

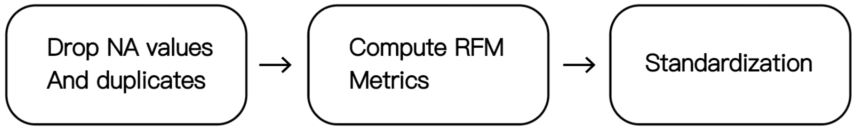



**Figure 2**: Data preprocessing workflow

Below, we provide a detailed explanation of the method used to compute the RFM metrics:

1. Recency: This metric represents the time difference between the most recent purchase and the last purchase.

2. Frequency: The frequency metric is calculated as the total number of purchases.

3. Monetary Value: For each purchase, the monetary value is computed by multiplying the price by the quantity purchased.

After computing these metrics, standardization is applied to address issues associated with varying scales among the features. The resulting feature dataset contains $4372$ samples.

| | recency | frequency | monetary_value |
|---|---|---|---|
| **0** | -0.842172 | -0.951603 | 59.590708 |
| **1** | -1.196993 | 0.365177 | 0.240981 |
| **2** | -1.120960 | -0.249321 | 0.381462 |
| **3** | 1.751405 | 2.208668 | 1.052594 |
| **4** | -0.715450 | -0.249321 | -0.051911 |

**Figure 3**: Initial entries of the preprocessed features

The preprocessed data is subsequently employed to train an unsupervised learning model, as elaborated in the next section.

# 3 Methods

We applied different unsupervised clustering models, namely the k-Means clustering model, hierarchical model, Gaussian mixture clustering, and BIRCH model, to the preprocessed data containing the RFM metrics. When necessary, we determined the optimal parameters, specifically the number of clusters, using the elbow method. The outcomes of these clustering algorithms were subsequently compared and analyzed collectively.

# 4 Results

## 4.1 k-Means clustering

We applied the preprocessed dataset to the k-Means clustering model using the `kMeans` package in `sklearn.cluster`. To determine the optimal number of clusters, we employed the elbow method and plotted the inertia as a function of the number of clusters. Initial inspection suggests that the optimal number of clusters is $n = 4$, with a Silhouette score of $0.46976$. The resulting plot of the clusters is shown below.

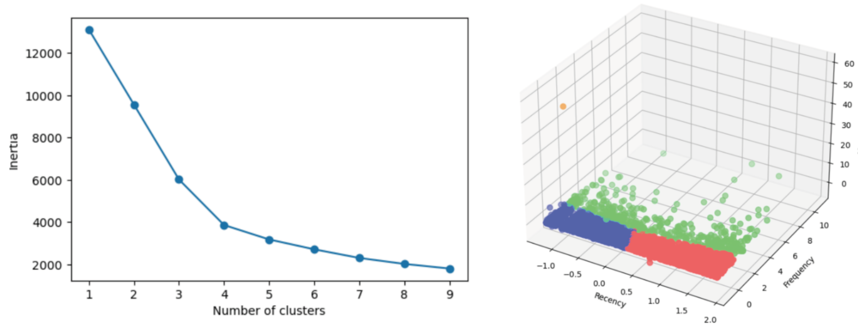



**Figure 4**: Fitted k-Means clustering model

However, based on the results, it is evident that there exists an outlier (colored in orange) that forms a distinct cluster. To streamline the model, we calculated the z-value for each data point and excluded those with a z-value greater than 2.5. The updated dataset was then clustered into 3 clusters, yielding a silhouette score of $0.47235$.

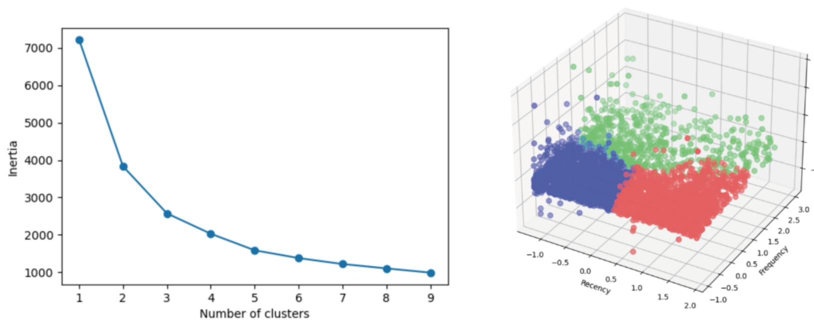



**Figure 5**: Fitted k-Means clustering model on the cleaned data

### 4.2 Hierarchical clustering

Hierarchical clustering is executed utilizing the `AgglomerativeClustering` package from `sklearn.cluster`. The resulting silhouette score is $0.37246$.

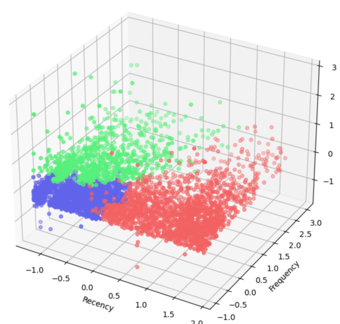



**Figure 6**: Fitted hierarchical clustering model on the cleaned data

### 4.3 Gaussian mixture clustering

We employed the Gaussian mixture clustering model with the `GaussianMixture` package from `sklearn.mixture`. The silhouette score is given by $0.177390$.

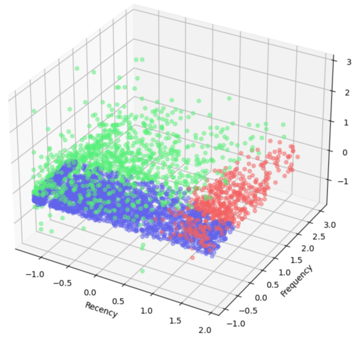



**Figure 7**: Fitted Gaussian mixture clustering model on the cleaned data

### 4.4 BIRCH clustering

BIRCH clustering is executed using the `Birch` package from `sklearn.cluster`. Initially, the parameter `nclusters` is set to `None`, allowing the algorithm to autonomously determine the optimal number of clusters. However, the results initially yield a total of 38 clusters. Through manual tuning of the number of clusters, it is determined that $4$ clusters yield the most distinctive clustering. The final silhouette score is given by $0.40247$.

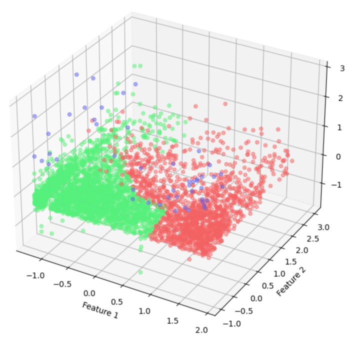



**Figure 8**: Fitted BIRCH clustering model on the cleaned data

## 5 Discussion

From the visualization and the silhouette scores, it is evident that the k-Means algorithm yields the highest silhouette score and the most distinctive clustering. K-Means clustering is also the most interpretable among all methods. We observe that k-Means clusters customers primarily based on the frequency and recency of customers, rather than the monetary value. One reason for this could be that the distribution of monetary value is highly centered, as shown in Figure 9.

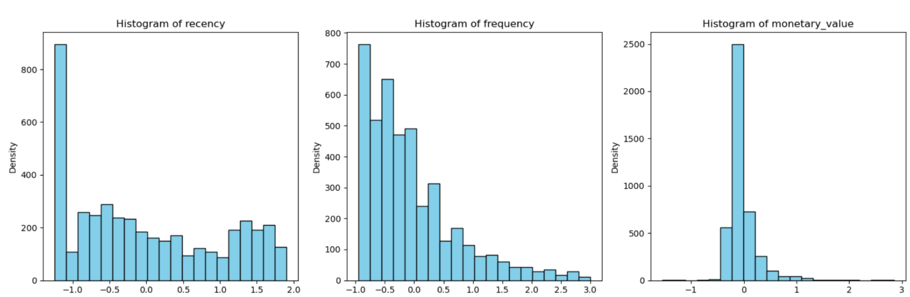



**Figure 9**: Distribution of the standardized features

Moreover, we note that k-Means makes a cut for frequency at around $0.25$. This justifies the distribution shown above as there is an obvious jump between the densities of the frequency metric. Overall, we can interpret the clusters produced by k-Means as:

1. Cluster 1 (Green): Frequent buyers.
2. Cluster 2 (Red): Non-frequent buyers but recently purchased the product.
3. Cluster 3 (Blue): Non-frequent buyers and haven't bought anything for a while.

Based on these clusters, one can formulate a marketing plan. For instance, instead of promoting new products to customers in Cluster 3, marketing and advertising efforts can be focused on customers in Clusters 1 and 2. One can also take a step further and analyze why customers in Cluster 3 stop buying the product.

Hierarchical clustering and Gaussian mixture clustering showed similar results to k-Means clustering. However, as observed in the plot, there are more overlaps between the clusters, making them harder to interpret. This is especially noticeable in the Gaussian mixture clustering model, which seems to also consider the monetary value metric in the clustering.

Unlike the previous three methods, BIRCH clustering, despite having a high silhouette score, produces results that are difficult to interpret. In particular, one can observe that the BIRCH model produces two big clusters, despite the input number of clusters being $4$. This suggests that maybe one can try removing more outliers to obtain a better result.

## 6 Conclusion

In this study, we assessed the effectiveness of different clustering models for customer segmentation using RFM metrics. Notably, k-Means clustering emerged as the most promising approach, showcasing superior performance. To further enhance the robustness of our segmentation strategy, future improvements could involve exploring advanced feature engineering techniques, experimenting with alternative distance metrics, and considering ensemble clustering methods. Additionally, incorporating additional relevant features or leveraging more sophisticated algorithms might contribute to a more nuanced understanding of customer behavior and enhance the overall efficacy of the segmentation process.

# References

[1] Customer segmentation based on Recency Frequency - DergiPark. (n.d.). https://dergipark.org.tr/en/download/article-file/951937

[2] Carrie. (2017, August 17). E-commerce data. Kaggle. https://www.kaggle.com/datasets/carrie1/ecommerce-data

In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
```

In [2]:
```python
red = "#f06262"
blue = "#6264f0"
orange = "#f0b062"
green = "#62f073"
```

# Data acquisition

In [3]:
```python
path = "/Users/raymondtsao/Desktop/STAT 154/Project/data.csv"
```

In [4]:
```python
data = pd.read_csv(path, encoding='unicode_escape')
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])
print(f"Data size: {data.shape}")
```

```
Data size: (541909, 8)
```

In [5]:
```python
data.head(5)
```

Out[5]:

| | InvoiceNo | StockCode | Description | Quantity | InvoiceDate | UnitPrice | CustomerID | Country |
|---|---|---|---|---|---|---|---|---|
| **0** | 536365 | 85123A | WHITE HANGING HEART T-LIGHT HOLDER | 6 | 2010-12-01 08:26:00 | 2.55 | 17850.0 | United Kingdom |
| **1** | 536365 | 71053 | WHITE METAL LANTERN | 6 | 2010-12-01 08:26:00 | 3.39 | 17850.0 | United Kingdom |
| **2** | 536365 | 84406B | CREAM CUPID HEARTS COAT HANGER | 8 | 2010-12-01 08:26:00 | 2.75 | 17850.0 | United Kingdom |
| **3** | 536365 | 84029G | KNITTED UNION FLAG HOT WATER BOTTLE | 6 | 2010-12-01 08:26:00 | 3.39 | 17850.0 | United Kingdom |
| **4** | 536365 | 84029E | RED WOOLLY HOTTIE WHITE HEART. | 6 | 2010-12-01 08:26:00 | 3.39 | 17850.0 | United Kingdom |

# Data preprocessing

In [6]:
```python
# Drop all the NA values and the duplicates
df = data.dropna(axis=0)
df = df.drop_duplicates()

print(f"Data size: {df.shape}")
```

```
Data size: (401604, 8)
```

In [7]:
```python
# Compute recency
df.loc[:, 'rank'] = df.sort_values(['CustomerID', 'InvoiceDate']).groupby(['CustomerID']
df = df[df['rank'] == 1]
df['recency'] = (df['InvoiceDate'] - pd.to_datetime(min(df['InvoiceDate']))).dt.days
df.head()
```

Out[7]:

| | InvoiceNo | StockCode | Description | Quantity | InvoiceDate | UnitPrice | CustomerID | Country | rank | recer |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 536365 | 85123A | WHITE | 6 | 2010-12-01 | 2.55 | 17850.0 | United | 1 | |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | HANGING HEART T-LIGHT HOLDER | | 08:26:00 | | | Kingdom | |
| **1** | 536365 | 71053 | WHITE METAL LANTERN | 6 | 2010-12-01 08:26:00 | 3.39 | 17850.0 | United Kingdom | 1 |
| **2** | 536365 | 84406B | CREAM CUPID HEARTS COAT HANGER | 8 | 2010-12-01 08:26:00 | 2.75 | 17850.0 | United Kingdom | 1 |
| **3** | 536365 | 84029G | KNITTED UNION FLAG HOT WATER BOTTLE | 6 | 2010-12-01 08:26:00 | 3.39 | 17850.0 | United Kingdom | 1 |
| **4** | 536365 | 84029E | RED WOOLLY HOTTIE WHITE HEART. | 6 | 2010-12-01 08:26:00 | 3.39 | 17850.0 | United Kingdom | 1 |

In [8]:

```
# Compute frequency
freq = df.groupby('CustomerID')['InvoiceDate'].count()
df_freq = pd.DataFrame(freq).reset_index()
df_freq.columns = ['CustomerID','frequency']
df = df_freq.merge(df, on='CustomerID')
df.head()
```

Out[8]:

| | CustomerID | frequency | InvoiceNo | StockCode | Description | Quantity | InvoiceDate | UnitPrice | Country |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 12346.0 | 1 | 541431 | 23166 | MEDIUM CERAMIC TOP STORAGE JAR | 74215 | 2011-01-18 10:01:00 | 1.04 | Unitec Kingdom |
| **1** | 12347.0 | 31 | 537626 | 85116 | BLACK CANDELABRA T-LIGHT HOLDER | 12 | 2010-12-07 14:57:00 | 2.10 | Icelanc |
| **2** | 12347.0 | 31 | 537626 | 22375 | AIRLINE BAG VINTAGE JET SET BROWN | 4 | 2010-12-07 14:57:00 | 4.25 | Icelanc |
| **3** | 12347.0 | 31 | 537626 | 71477 | COLOUR GLASS. STAR T-LIGHT HOLDER | 12 | 2010-12-07 14:57:00 | 3.25 | Icelanc |
| **4** | 12347.0 | 31 | 537626 | 22492 | MINI PAINT SET VINTAGE | 36 | 2010-12-07 14:57:00 | 0.65 | Icelanc |

In [9]:

```
# Compute monetary value
df['total'] = df['Quantity'] * df['UnitPrice']
value = df.groupby('CustomerID')['total'].sum()
value = pd.DataFrame(value).reset_index()
value.columns = ['CustomerID','monetary_value']
df = value.merge(df, on='CustomerID')
df.head()
```

Out[9]:

| | CustomerID | monetary_value | frequency | InvoiceNo | StockCode | Description | Quantity | InvoiceDate | U |
|---|---|---|---|---|---|---|---|---|---|

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **0** | 12346.0 | 77183.60 | 1 | 541431 | 23166 | MEDIUM CERAMIC TOP STORAGE JAR | 74215 | 2011-01-18 10:01:00 |
| **1** | 12347.0 | 711.79 | 31 | 537626 | 85116 | BLACK CANDELABRA T-LIGHT HOLDER | 12 | 2010-12-07 14:57:00 |
| **2** | 12347.0 | 711.79 | 31 | 537626 | 22375 | AIRLINE BAG VINTAGE JET SET BROWN | 4 | 2010-12-07 14:57:00 |
| **3** | 12347.0 | 711.79 | 31 | 537626 | 71477 | COLOUR GLASS. STAR T-LIGHT HOLDER | 12 | 2010-12-07 14:57:00 |
| **4** | 12347.0 | 711.79 | 31 | 537626 | 22492 | MINI PAINT SET VINTAGE | 36 | 2010-12-07 14:57:00 |

In [10]:

```
# Extract nessesary information (Customer ID and RFM data)
df = df[['CustomerID','recency','frequency','monetary_value']]
df = df.drop_duplicates()

print(f"Data size: {df.shape}")
```

```
Data size: (4372, 4)
```

In [11]:

```
# Perform standardization
from sklearn.preprocessing import StandardScaler

col_names = ['recency', 'frequency', 'monetary_value']
features = df[col_names]
scaler = StandardScaler().fit(features.values)
features = scaler.transform(features.values)
features = pd.DataFrame(features, columns = col_names)
features.head()

print(f"Data size: {features.shape}")
```

```
Data size: (4372, 3)
```

In [12]:

```
features.head()
```

Out[12]:

| | recency | frequency | monetary_value |
|---|---|---|---|
| **0** | -0.842172 | -0.951603 | 59.590708 |
| **1** | -1.196993 | 0.365177 | 0.240981 |
| **2** | -1.120960 | -0.249321 | 0.381462 |
| **3** | 1.751405 | 2.208668 | 1.052594 |
| **4** | -0.715450 | -0.249321 | -0.051911 |

# k-Means Clustering

In [13]:

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

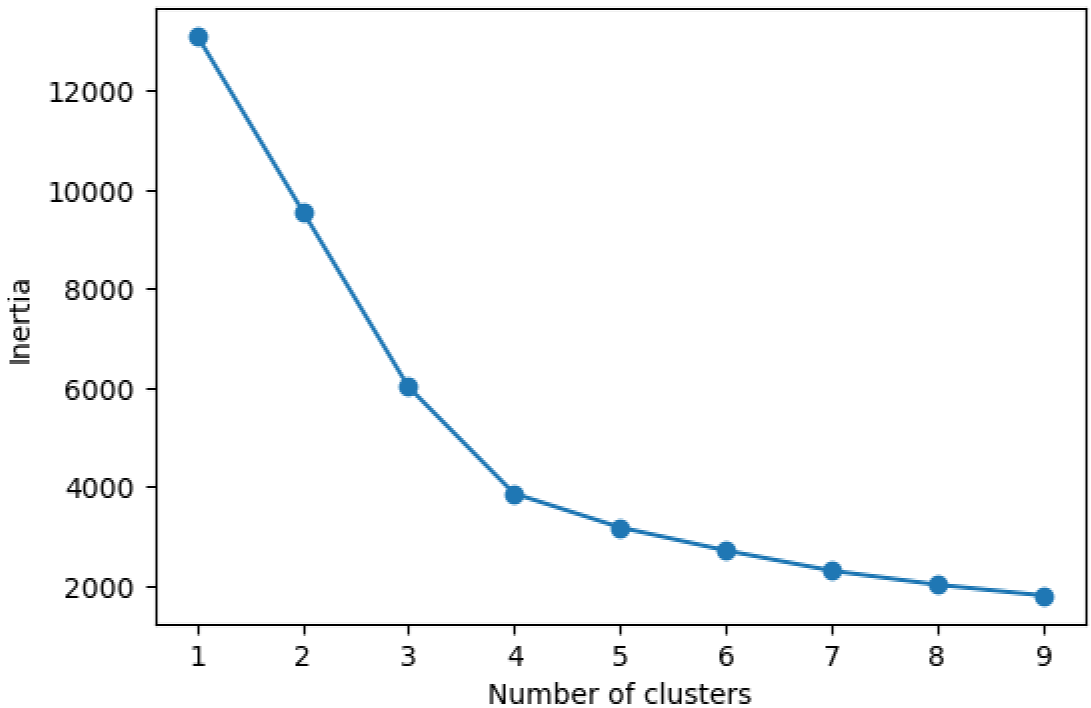
In [14]:

```
# Using the elbow method to determine the optimal number of clusters
```

```
sse = []
for cluster in range(1, 10):
    kmeans = KMeans(n_clusters=cluster, n_init=10)
    kmeans.fit(features)
    sse.append(kmeans.inertia_)
```

In [15]:
```
frame = pd.DataFrame({'Cluster':range(1,10), 'SSE':sse})
plt.figure(figsize=(6,4 ))
plt.plot(frame['Cluster'], frame['SSE'], marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
```

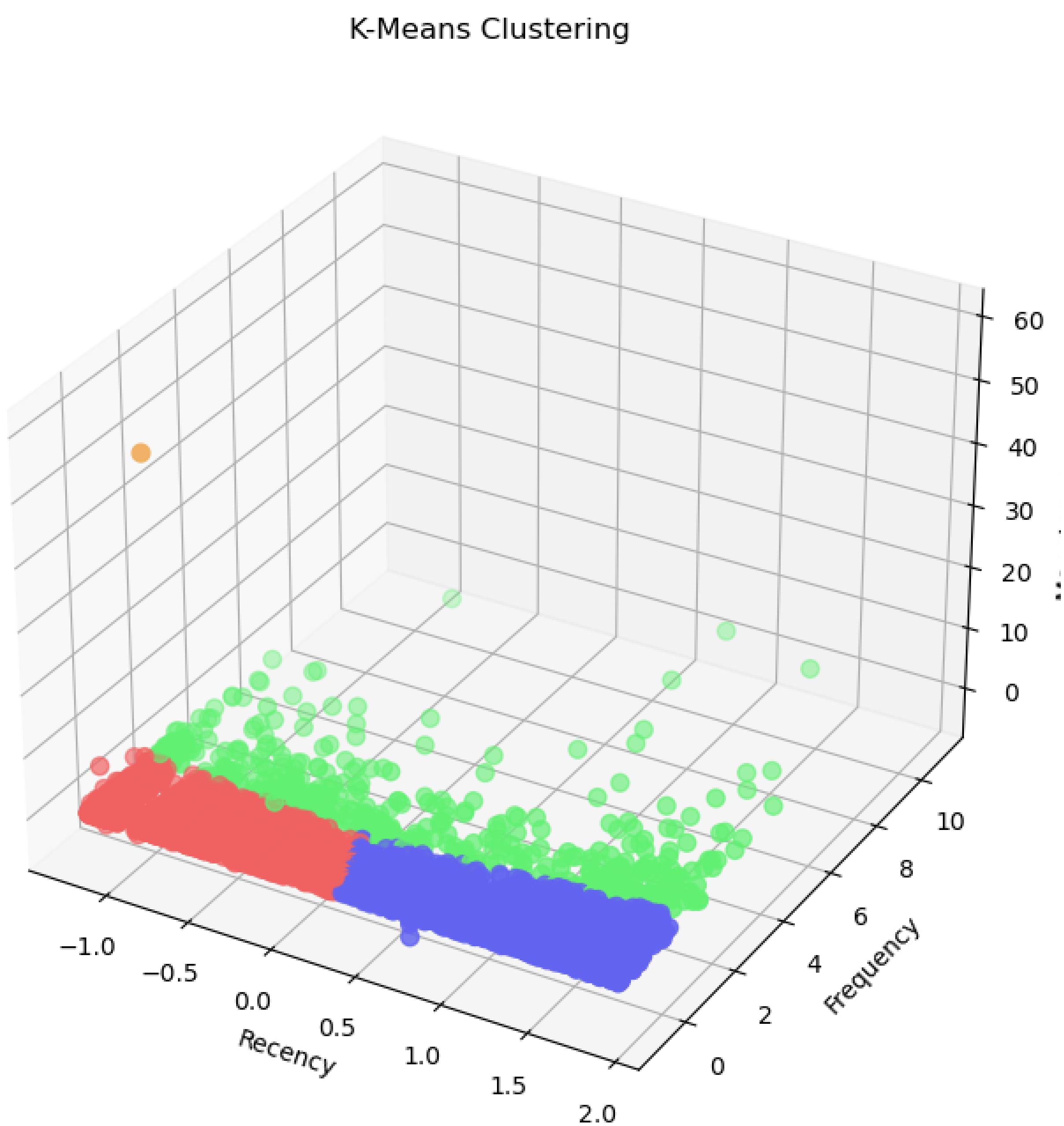
Out[15]:
```
Text(0, 0.5, 'Inertia')
```




In [16]:
```
# From the elbow method, we see that the optimal number of cluster is 4 clusters
kmeans = KMeans(n_clusters = 4, n_init=10)
kmeans.fit(features)
print(f"The silhouette score is: {silhouette_score(features, kmeans.labels_, metric='euc
```

```
The silhouette score is: 0.4697675293516605
```

In [17]:
```
custom_colors = [red, blue, orange, green]
cmap = ListedColormap(custom_colors)

labels = kmeans.labels_
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(features['recency'], features['frequency'], features['monetary_valu

ax.set_xlabel('Recency')
ax.set_ylabel('Frequency')
ax.set_zlabel('Monetary Value')
ax.set_title('K-Means Clustering')

plt.show()
```




From the above graph, we see that there is an outlier that itself forms a class. We now remove the outliers.

In [18]:
```
from scipy import stats

# Remove the outliers
z_scores = stats.zscore(features)
abs_z_scores = np.abs(z_scores)
filtered_entries = (abs_z_scores < 3).all(axis=1)
cleaned_features = features[filtered_entries]

print(f"The size of the cleaned features: {cleaned_features.shape}")
```

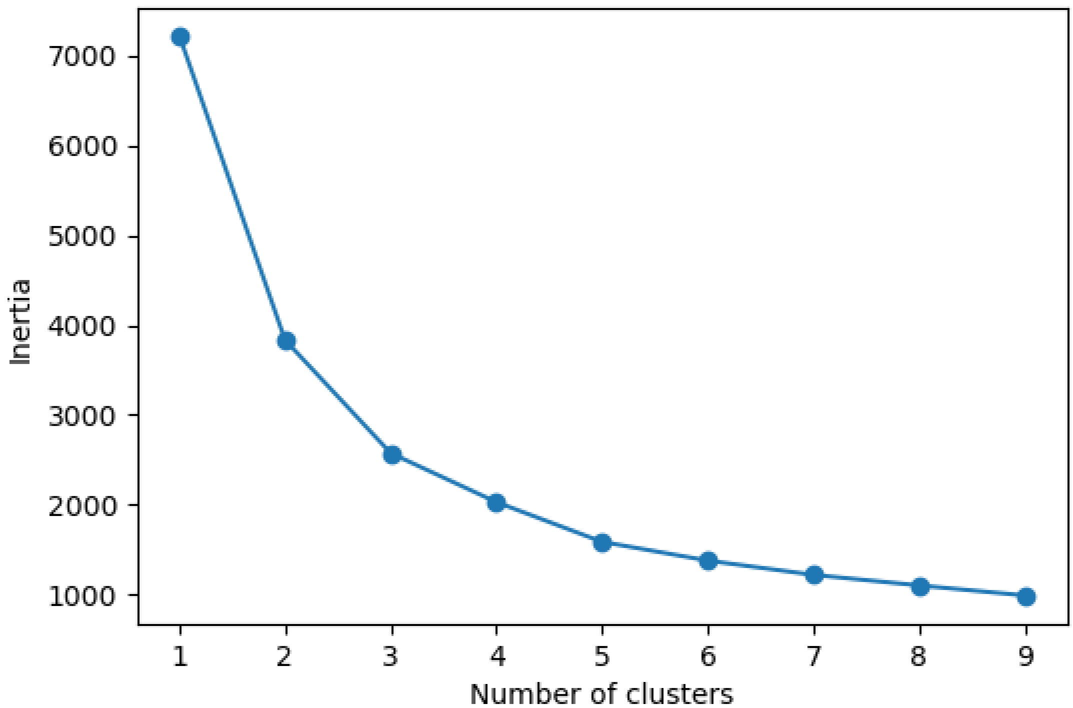
```
The size of the cleaned features: (4287, 3)
```

In [19]:
```
sse = []
for cluster in range(1, 10):
    kmeans = KMeans(n_clusters=cluster, n_init=10)
    kmeans.fit(cleaned_features)
    sse.append(kmeans.inertia_)
```

In [20]:
```
frame = pd.DataFrame({'Cluster':range(1,10), 'SSE':sse})
```

```python
plt.figure(figsize=(6,4))
plt.plot(frame['Cluster'], frame['SSE'], marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
```

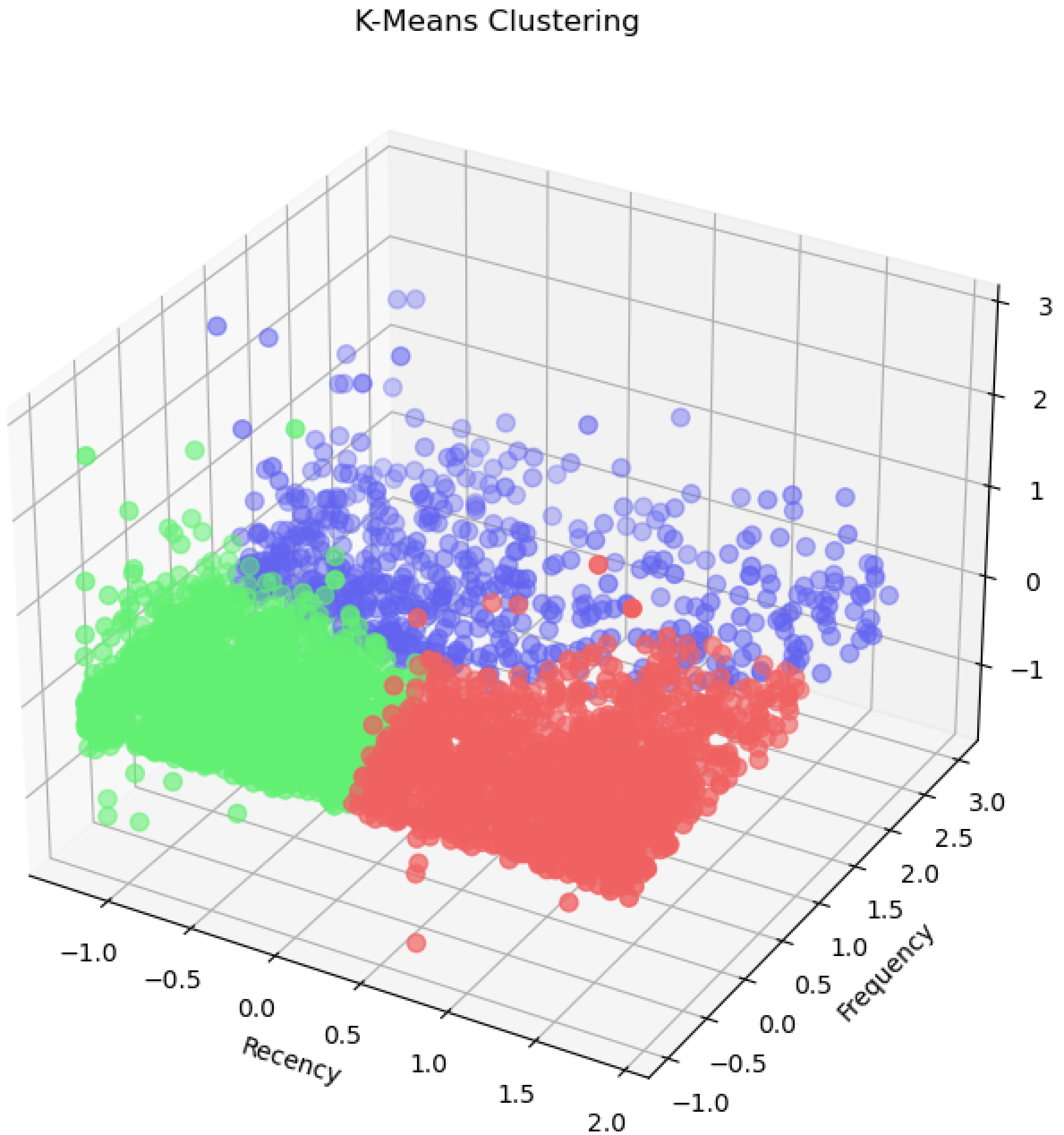
Out[20]:
```
Text(0, 0.5, 'Inertia')
```




In [21]:
```python
kmeans = KMeans(n_clusters = 3, n_init=10)
kmeans.fit(cleaned_features)
print(f"The silhouette score is: {silhouette_score(cleaned_features, kmeans.labels_, met
```

```
The silhouette score is: 0.4723099914197541
```

In [22]:
```python
custom_colors = [blue, red, green]
cmap = ListedColormap(custom_colors)

labels = kmeans.labels_

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(cleaned_features['recency'], cleaned_features['frequency'], cleaned

ax.set_xlabel('Recency')
ax.set_ylabel('Frequency')
ax.set_zlabel('Monetary Value')
ax.set_title('K-Means Clustering')

plt.show()
```

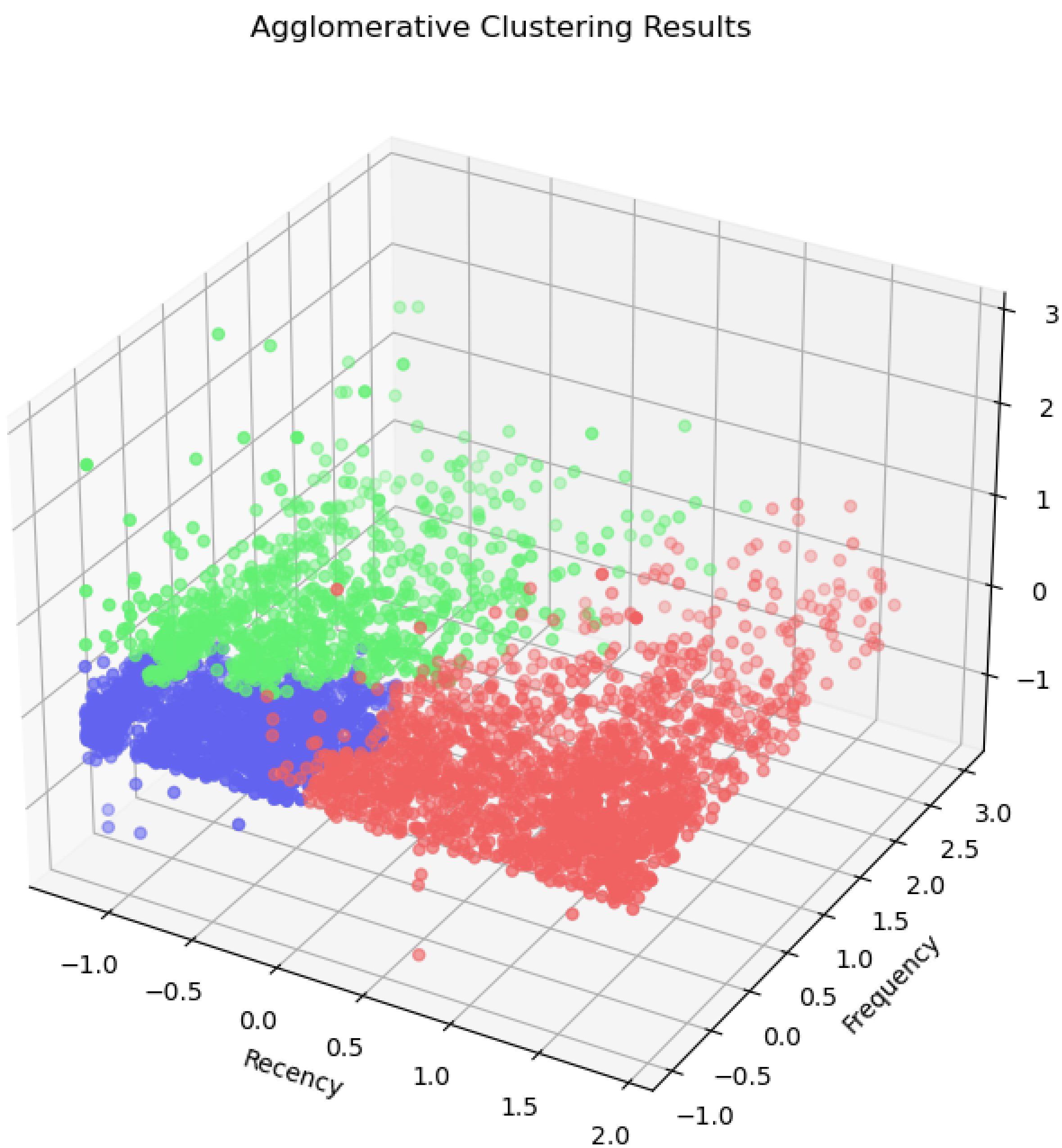



## Hierarchical Clustering

In [23]:
```
import numpy as np
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

In [24]:
```
model = AgglomerativeClustering(n_clusters=3, metric='euclidean', linkage='ward')
clusters = model.fit_predict(cleaned_features)

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

color = [red, green, blue]


for cluster_label in np.unique(clusters):
    cluster_points = cleaned_features[clusters == cluster_label]
    ax.scatter(cluster_points.iloc[:, 0], cluster_points.iloc[:, 1], cluster_points.iloc

ax.set_xlabel('Recency')
```

```
ax.set_ylabel('Frequency')
ax.set_zlabel('Monetary value')
ax.set_title('Agglomerative Clustering Results')
plt.show()
```



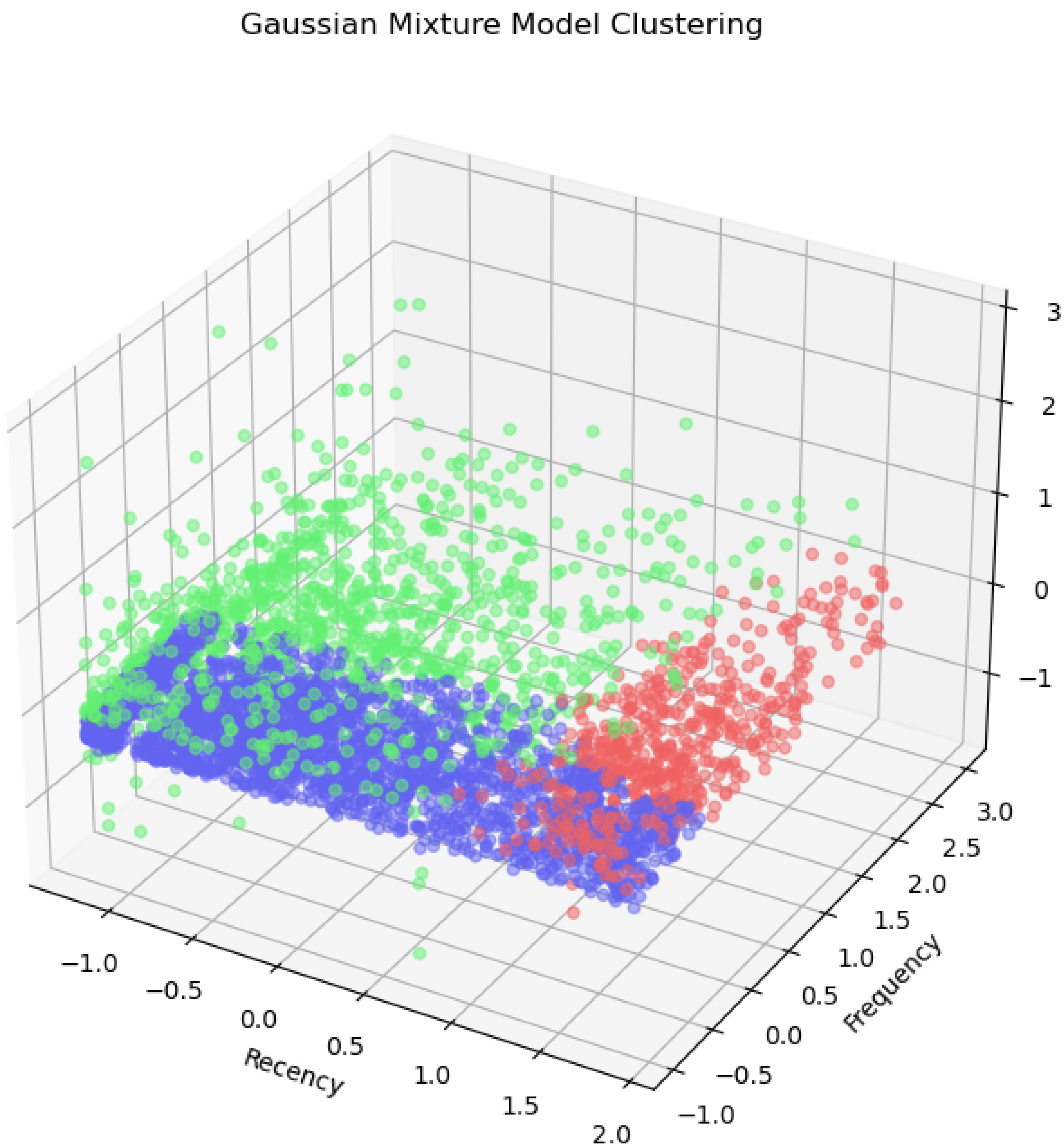

In [26]:
```
# Compute the Silhouette score
silhouette_avg = silhouette_score(cleaned_features, clusters)
print(f'Silhouette Score: {silhouette_avg}')
```

```
Silhouette Score: 0.3724629958331841
```

## Gaussian mixture clustering

In [27]:
```
import numpy as np
from sklearn.mixture import GaussianMixture
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D  # For 3D plotting

# Assuming your data is stored in a variable called 'data'
# data.shape should be (4000, 3)
# If not, you might need to adjust accordingly
```

```
# Assuming 'data' is a NumPy array
gmm = GaussianMixture(n_components=3, random_state=42)
clusters = gmm.fit_predict(cleaned_features)
custom_colors = [blue, green, red]
cmap = ListedColormap(custom_colors)

# Visualize the clusters in 3D (assuming 3 features)
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(cleaned_features.iloc[:, 0], cleaned_features.iloc[:, 1], cleaned_features.il
ax.set_title('Gaussian Mixture Model Clustering')
ax.set_xlabel('Recency')
ax.set_ylabel('Frequency')
ax.set_zlabel('Monetary value')
plt.show()
```




In [28]:
```
# Compute the Silhouette score
silhouette_avg = silhouette_score(cleaned_features, clusters)
print(f'Silhouette Score: {silhouette_avg}')
```

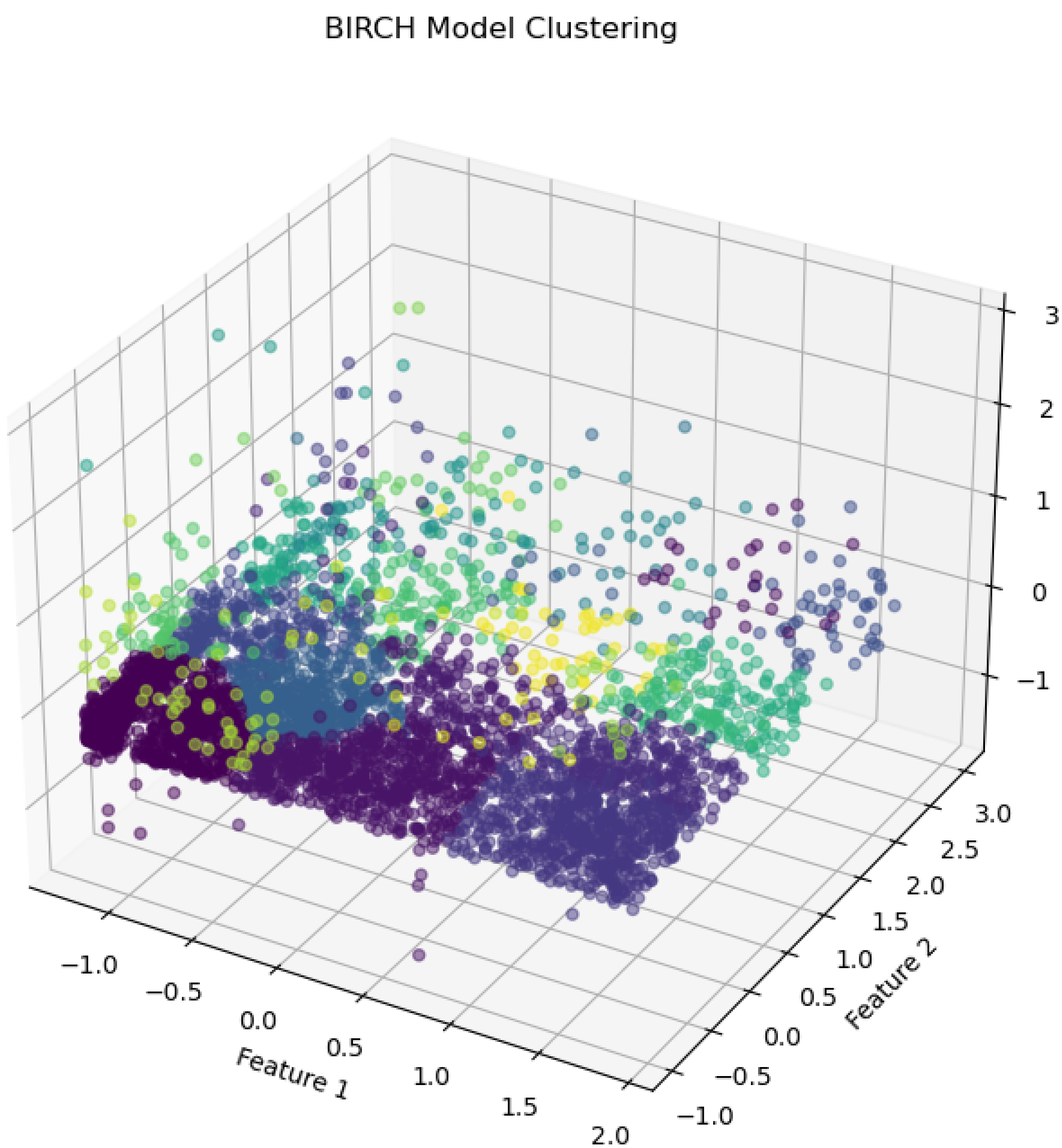
```
Silhouette Score: 0.1773901481114778
```

# BIRCH clustering model

In [29]:
```
from sklearn.cluster import Birch
```

In [30]:
```
birch_model = Birch(n_clusters=None)

birch_model.fit(cleaned_features)

cluster_labels = birch_model.predict(cleaned_features)
```

In [31]:
```
custom_colors = [blue, red, green]
cmap = ListedColormap(custom_colors)

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(cleaned_features.iloc[:, 0], cleaned_features.iloc[:, 1], cleaned_features.il
ax.set_title('BIRCH Model Clustering')
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Feature 3')
plt.show()
```

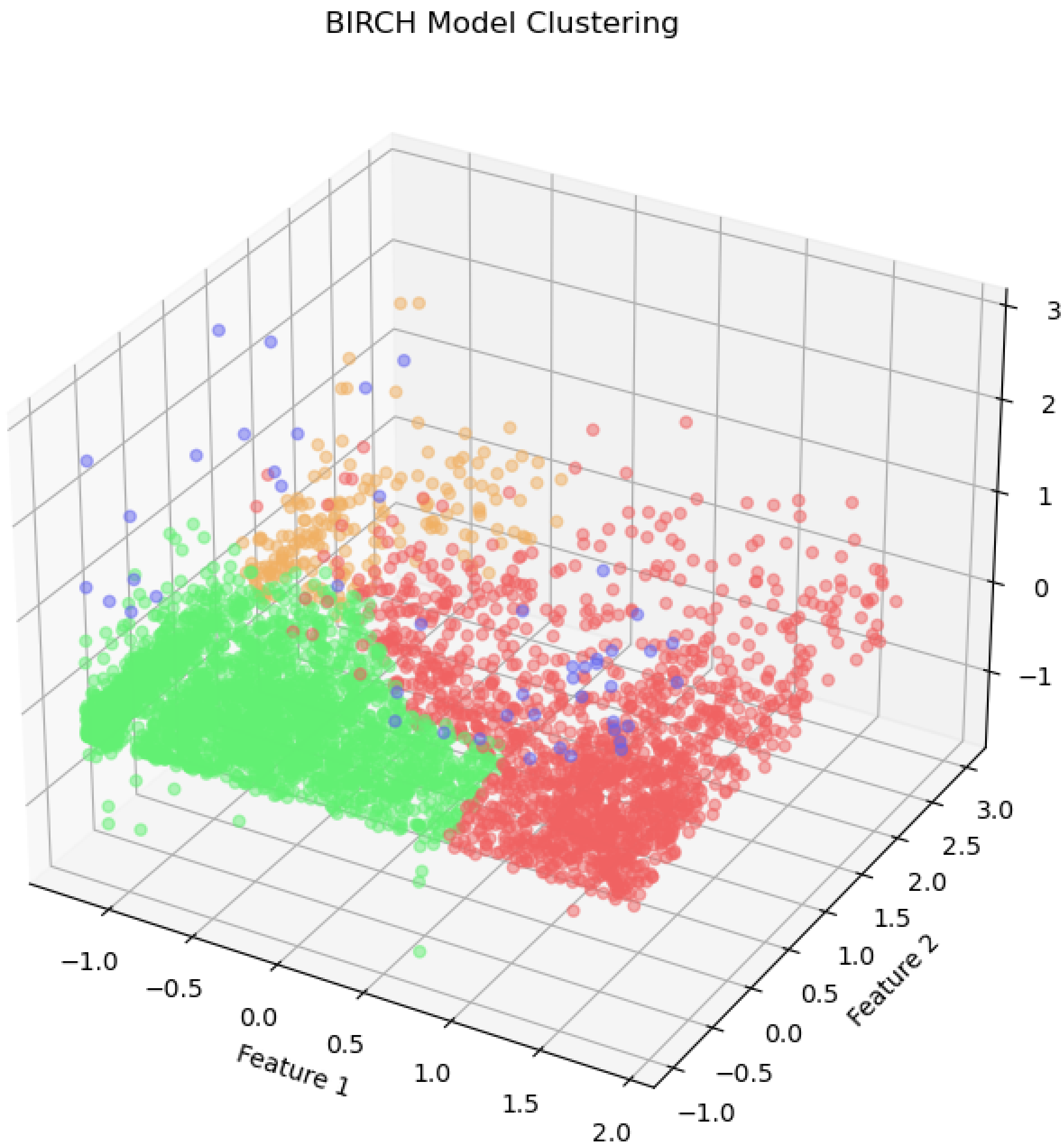



In [32]:
```
print(f"There are a total of {len(np.unique(cluster_labels))} clusters")
```

```
There are a total of 38 clusters
```

In [33]:
```python
birch_model = Birch(n_clusters=4)

birch_model.fit(cleaned_features)

cluster_labels = birch_model.predict(cleaned_features)
```

In [34]:
```python
custom_colors = [blue, red, green, orange]
cmap = ListedColormap(custom_colors)

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(cleaned_features.iloc[:, 0], cleaned_features.iloc[:, 1], cleaned_features.il
ax.set_title('BIRCH Model Clustering')
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Feature 3')
plt.show()
```




In [35]:
```python
# Compute the Silhouette score
silhouette_avg = silhouette_score(cleaned_features, cluster_labels)
print(f'Silhouette Score: {silhouette_avg}')
```

```
Silhouette Score: 0.4024760840210064
```

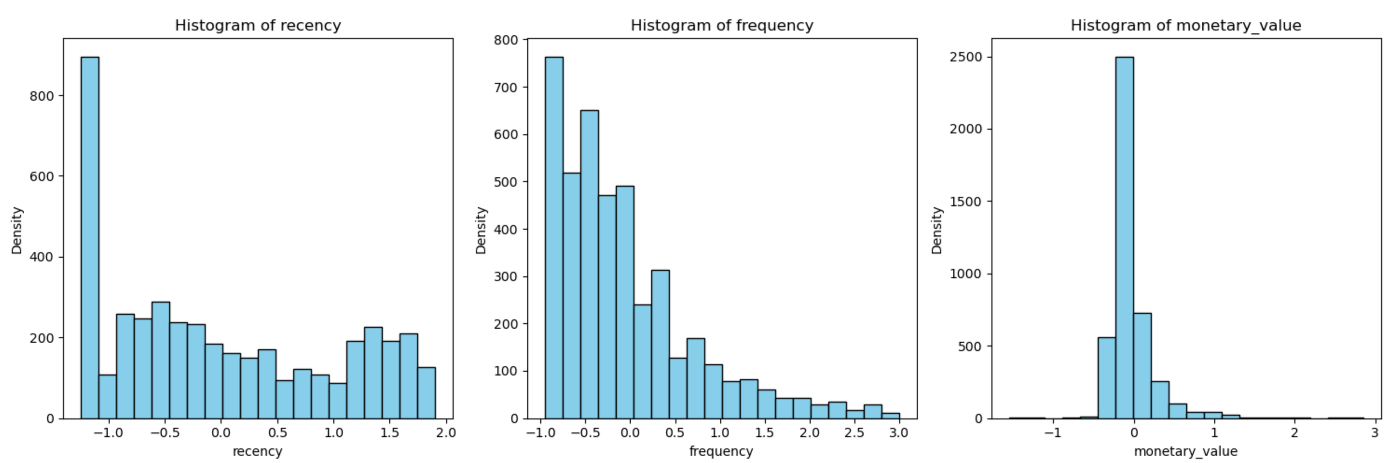
In [36]:
```
cleaned_features
```

Out[36]:

| | recency | frequency | monetary_value |
|---|---|---|---|
| **1** | -1.196993 | 0.365177 | 0.240981 |
| **2** | -1.120960 | -0.249321 | 0.381462 |
| **3** | 1.751405 | 2.208668 | 1.052594 |
| **4** | -0.715450 | -0.249321 | -0.051911 |
| **5** | -0.597176 | -0.337106 | -0.081326 |
| **...** | ... | ... | ... |
| **4367** | -0.436661 | -0.556569 | -0.171276 |
| **4368** | 0.382807 | -0.688247 | -0.248715 |
| **4369** | 0.839007 | -0.688247 | -0.233666 |
| **4370** | -0.943549 | 1.506386 | -0.227271 |
| **4371** | 0.205397 | 0.277391 | 0.282494 |

4287 rows × 3 columns

In [38]:
```
# Get the column names (feature names)
feature_names = cleaned_features.columns

# Create subplots for each feature
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))

# Plot histograms for each feature
for i, feature in enumerate(feature_names):
    axes[i].hist(cleaned_features[feature], bins=20, color='skyblue', edgecolor='black')
    axes[i].set_title(f'Histogram of {feature}')
    axes[i].set_xlabel(feature)
    axes[i].set_ylabel('Density')

plt.tight_layout()
plt.show()
```




In [ ]: