Problem Statement: Customer Segmentation using Machine Learning

In the given problem statement, the objective is to perform customer segmentation using machine learning techniques. Customer segmentation is a critical aspect of marketing and business strategy, as it involves categorizing a company's customers into distinct groups based on certain characteristics, behaviors, or demographics. This segmentation helps businesses gain a deeper understanding of their customer base, tailor marketing strategies, and improve customer satisfaction.

Problem Context:

The dataset provided contains information about customer transactions, including details such as purchase frequency, recency, and monetary value. The task is to apply machine learning algorithms to this dataset in order to segment customers into meaningful groups.

Steps of the Algorithm

The primary technique used is K-Means Clustering, which is an unsupervised machine learning algorithm. K-Means is employed to group customers into clusters based on their behavior and transactional patterns. Data preprocessing, feature transformation, and data visualization steps are performed to gain insights from the customer data. The optimal number of clusters is determined using the Elbow Method, and the results are visualized to aid interpretation and decision-making for the business.

The steps involved in the customer segmentation code are as follows:

1. Import Libraries: Import necessary libraries for data manipulation, visualization, and machine learning.
2. Load and Inspect Data: Load the customer transaction data from an Excel file. Create a deep copy of the original dataset.
3. Data Preprocessing: Sample 10,000 rows randomly for faster processing. Convert 'InvoiceDate' to date format. Calculate 'TotalSum' as the product of 'Quantity' and 'UnitPrice'. Determine 'snapshot_date' as the maximum 'InvoiceDate' plus one day. Group data by 'CustomerID' and compute 'Recency,' 'Frequency,' and 'MonetaryValue.'
4. Exploratory Data Analysis (EDA): Display data summary and statistics. Create distribution plots for 'Recency,' 'Frequency,' and 'MonetaryValue.'
5. Feature Transformation: Create copies for various transformations: logarithmic, square root, cube root, and Box-Cox. Apply these transformations to 'Recency' and 'Frequency.' Visualize transformed data using distribution plots.
6. Fix and Normalize Data: Create 'customers_fix' DataFrame. Apply Box-Cox transformation to 'Recency' and 'Frequency,' cube root transformation to 'MonetaryValue.' Normalize data using StandardScaler.
7. Determine Optimal Number of Clusters (K): Calculate SSE for different K values using K-Means. Plot SSE vs. K to find the optimal number of clusters.

8. K-Means Clustering: Initialize K-Means model with chosen K (e.g., 3 clusters). Fit model to normalized and standardized customer data. Add 'Cluster' column to the original customer DataFrame.
9. Cluster Analysis and Visualization: Group customer data by clusters and calculate mean values and counts. Create a DataFrame for normalized customer data, cluster labels, and attributes. Melt DataFrame for visualization. Visualize attribute values by cluster using line plots.
10. Conclusion and Insights: Interpret customer segmentation results. Provide actionable insights for marketing strategies or business decisions.

About the Dataset

This dataset is related to retail or e-commerce transactions, where each row represents a specific purchase made by a customer. It includes information about the products purchased, their quantities, prices, and the customer involved. The dataset can be used for various analyses, including customer segmentation, sales forecasting, and market trend analysis. The dataset has 8 columns and 541910 rows The dataset consists of the following columns:

1. InvoiceNo: A unique identifier for each transaction or invoice.
2. StockCode: A code that represents the specific product or item being sold.
3. Description: A brief description of the product or item.
4. Quantity: The quantity of the product or item purchased in the transaction.
5. InvoiceDate: The date and time when the transaction occurred.
6. UnitPrice: The price of a single unit of the product or item.
7. CustomerID: A unique identifier for each customer.
8. Country: The country where the customer is located or where the transaction took place.

Code

In [18]:
```python
# Import necessary libraries
import os
import math
import scipy
import datetime
import numpy as np
import pandas as pd
import seaborn as sns
from scipy import stats
from scipy.stats import randint
from scipy.stats import loguniform
from IPython.display import display

# Import specific functions and classes from scikit-learn
from sklearn.cluster import KMeans
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Import matplotlib for visualization
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [10,6]

# Suppress warnings
import warnings
warnings.filterwarnings('ignore')

!pip install openpyxl
```

```
Requirement already satisfied: openpyxl in c:\users\barunaditya mohanty\anaco
nda3\lib\site-packages (3.0.10)
Requirement already satisfied: et_xmlfile in c:\users\barunaditya mohanty\ana
conda3\lib\site-packages (from openpyxl) (1.1.0)
```

In [19]:
```
# Read data from an Excel file into a Pandas DataFrame
df = pd.read_excel('Online Retail.xlsx')


# Create a deep copy of the original DataFrame
original_df = df.copy(deep=True)
display(df.head())  # Display the first few rows of the DataFrame

# Print information about the dataset (number of features and samples)
print('\n\033[1mInference:\033[0m The Datset consists of {} features & {} sampl
```

	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

Inference: The Datset consists of 8 features & 541909 samples.

In [20]:
```
df = pd.read_excel('Online Retail.xlsx')

original_df = df.copy(deep=True)
display(df.head())

print('\n\033[1mInference:\033[0m The Datset consists of {} features & {} sampl
```

	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

Inference: The Datset consists of 8 features & 541909 samples.

In [21]:
```
# Randomly sample 10,000 rows from the DataFrame with a fixed random seed
df_s = df.sample(10000, random_state=42)
df_s["InvoiceDate"] = df_s["InvoiceDate"].dt.date # Convert the "InvoiceDate" c
df_s["TotalSum"] = df_s["Quantity"] * df_s["UnitPrice"] # Calculate the total m
snapshot_date = max(df_s.InvoiceDate) + datetime.timedelta(days=1)
customers = df_s.groupby(['CustomerID']).agg({'InvoiceDate': lambda x: (snapsho
                                              'InvoiceNo': 'count',
                                              'TotalSum': 'sum'})
# Rename the columns for better readability
customers.rename(columns = {'InvoiceDate': 'Recency','InvoiceNo': 'Frequency',
                            'TotalSum': 'MonetaryValue'}, inplace=True)

# Display the first few rows of the aggregated customer data
display(customers.head())
```

	Recency	Frequency	MonetaryValue
CustomerID			
12347.0	3	5	81.60
12349.0	19	1	19.90
12353.0	205	1	39.80
12354.0	233	2	25.45
12356.0	326	1	50.00

In [22]:
```
customers.info() # Print information about the customer DataFrame
```

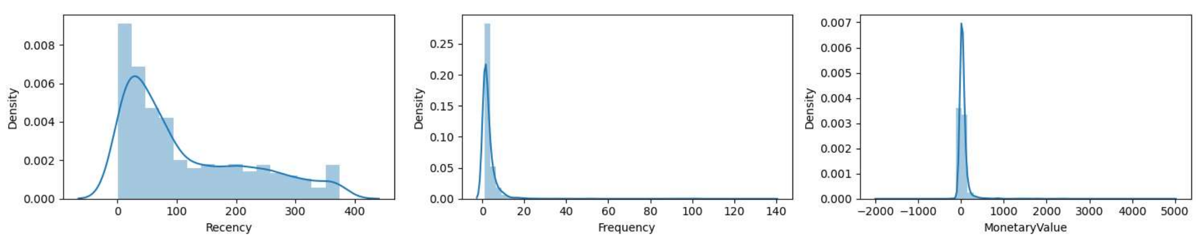
```
<class 'pandas.core.frame.DataFrame'>
Float64Index: 2433 entries, 12347.0 to 18287.0
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Recency        2433 non-null   int64  
 1   Frequency      2433 non-null   int64  
 2   MonetaryValue  2433 non-null   float64
dtypes: float64(1), int64(2)
memory usage: 76.0 KB
```

In [23]:
```
display(customers.describe()) # Display summary statistics of the customer Data
```

	Recency	Frequency	MonetaryValue
count	2433.000000	2433.000000	2433.000000
mean	115.114673	3.076038	60.757185
std	105.746852	5.693838	205.094177
min	1.000000	1.000000	-1867.860000
25%	30.000000	1.000000	12.400000
50%	73.000000	2.000000	24.770000
75%	191.000000	3.000000	53.100000
max	374.000000	137.000000	4887.330000

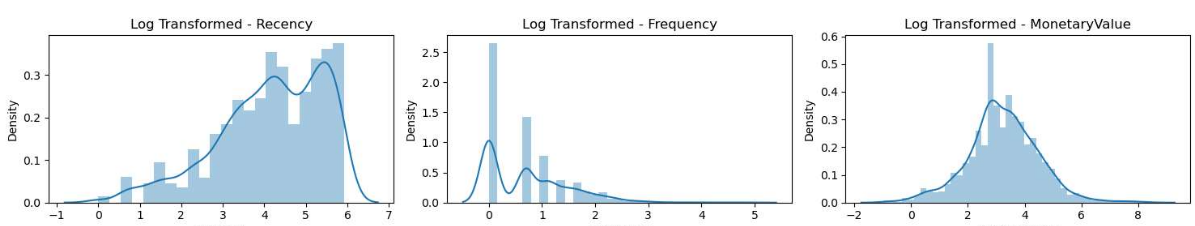
In [24]:
```
print('\033[1mRMF Variables Distribution'.center(100))

n=3 # Define the number of subplots per row and list of customer metrics
nf = [i for i in customers.columns]

# Create distribution plots for each customer metric
plt.figure(figsize=[15,3*math.ceil(len(nf)/n)])
for c in range(len(nf)):
    plt.subplot(math.ceil(len(nf)/n),n,c+1)
    sns.distplot(customers[nf[c]])
plt.tight_layout()
plt.show()
```

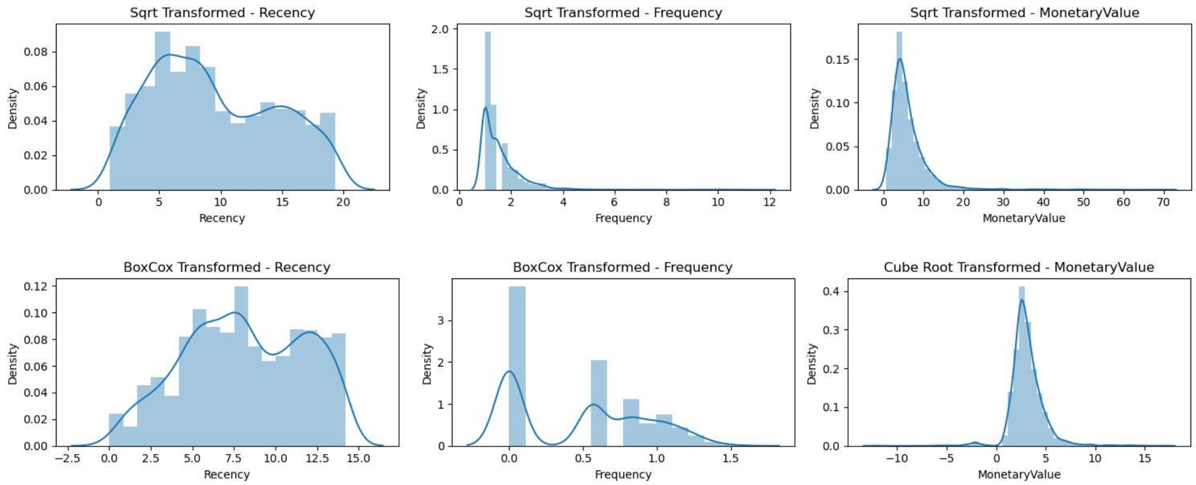
RMF Variables Distribution



In [25]:

```
# Create deep copies of the customer data for different transformations
cutomers_logT = customers.copy(deep=True)
cutomers_sqrtT = customers.copy(deep=True)
cutomers_cbrtT = customers.copy(deep=True)
cutomers_bxcxT = customers.copy(deep=True)

# Apply logarithmic, square root, cube root, and Box-Cox transformations to cus
for i in customers.columns:
    cutomers_logT[i] = np.log(customers[i])
    cutomers_sqrtT[i] = np.sqrt(customers[i])
    cutomers_cbrtT[i] = np.cbrt(customers[i])
    if i != 'MonetaryValue':
        cutomers_bxcxT[i] = stats.boxcox(customers[i])[0]

# Create distribution plots for the Log Transformed metrics
plt.figure(figsize=[15,3*math.ceil(len(nf)/n)])
for c in range(len(nf)):
    plt.subplot(math.ceil(len(nf)/n),n,c+1)
    sns.distplot(cutomers_logT[nf[c]])
    plt.title('Log Transformed - {}'.format(nf[c]))
plt.tight_layout()
plt.show()

# Create distribution plots for the Square Root Transformed metrics
plt.figure(figsize=[15,3*math.ceil(len(nf)/n)])
for c in range(len(nf)):
    plt.subplot(math.ceil(len(nf)/n),n,c+1)
    sns.distplot(cutomers_sqrtT[nf[c]])
    plt.title('Sqrt Transformed - {}'.format(nf[c]))
plt.tight_layout()
plt.show()

# Create distribution plots for the Box-Cox Transformed metrics (excluding Mone
plt.figure(figsize=[15,3*math.ceil(len(nf)/n)])
for c in range(len(nf)-1):
    plt.subplot(1,3,c+1)
    sns.distplot(cutomers_bxcxT[nf[c]])
    plt.title('BoxCox Transformed - {}'.format(nf[c]))
plt.subplot(1,3,3)
sns.distplot(cutomers_cbrtT[nf[2]])
plt.title('Cube Root Transformed - {}'.format(nf[2]))
plt.tight_layout()
plt.show()
```





In [26]:
```python
# Create a DataFrame for the fixed and transformed customer metrics
customers_fix = pd.DataFrame()
customers_fix["Recency"] = stats.boxcox(customers['Recency'])[0]
customers_fix["Frequency"] = stats.boxcox(customers['Frequency'])[0]
customers_fix["MonetaryValue"] = pd.Series(np.cbrt(customers['MonetaryValue']))

# Display the last few rows of the fixed and transformed DataFrame
customers_fix.tail()
```

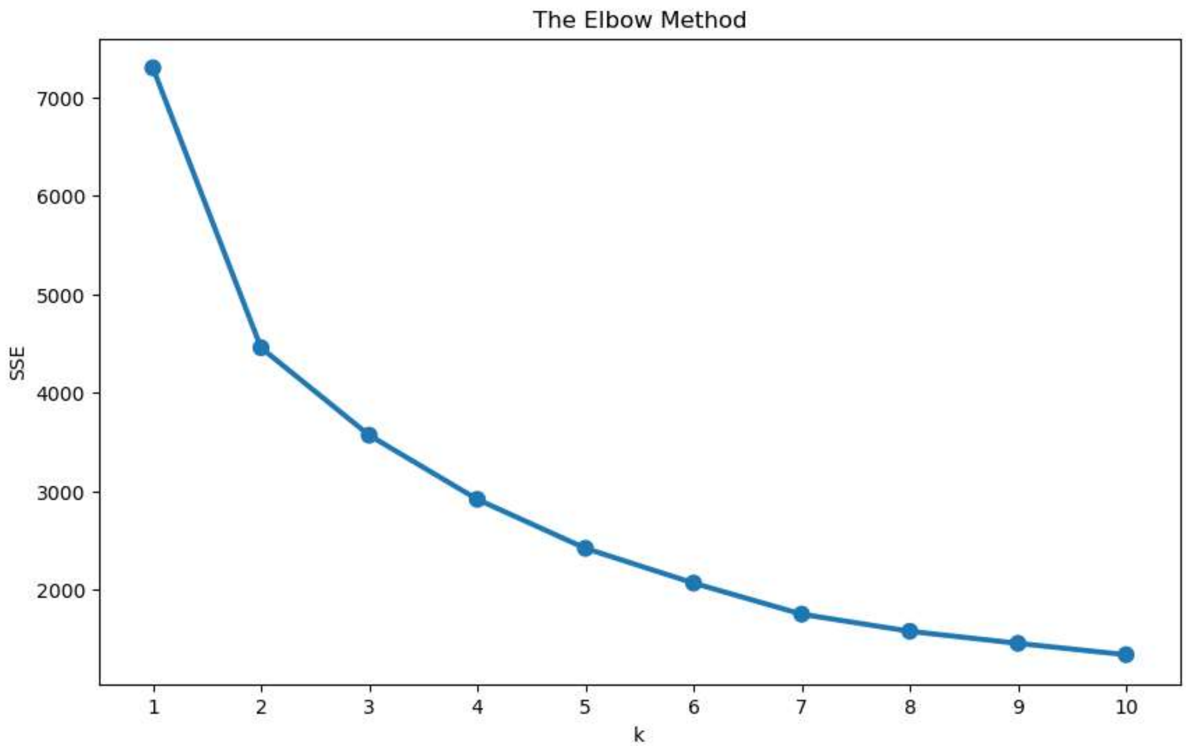
Out[26]:

	Recency	Frequency	MonetaryValue
2428	5.144506	1.021167	4.861252
2429	6.148622	0.564199	4.091635
2430	1.272970	0.798349	3.737290
2431	1.671379	1.253008	3.013275
2432	11.514709	0.000000	2.482545

In [27]:
```python
# Initialize a StandardScaler and fit it to the fixed and transformed customer
scaler = StandardScaler()
scaler = StandardScaler()
scaler.fit(customers_fix)
# Transform the customer data using StandardScaler
customers_normalized = scaler.transform(customers_fix)

# Print the mean and standard deviation of the normalized customer data
print(customers_normalized.mean(axis = 0).round(2)) # [0. -0. 0.]
print(customers_normalized.std(axis = 0).round(2))
```

```
[-0. -0. -0.]
[1. 1. 1.]
```

In [28]:
```
sse = {} # Initialize a dictionary to store Sum of Squared Errors (SSE) for dif
# Perform K-means clustering for different numbers of clusters (k)
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(customers_normalized)
    sse[k] = kmeans.inertia_

# Create an Elbow Method plot to determine the optimal number of clusters
plt.title('The Elbow Method')
plt.xlabel('k')
plt.ylabel('SSE')
sns.pointplot(x=list(sse.keys()), y=list(sse.values()))
plt.show()
```

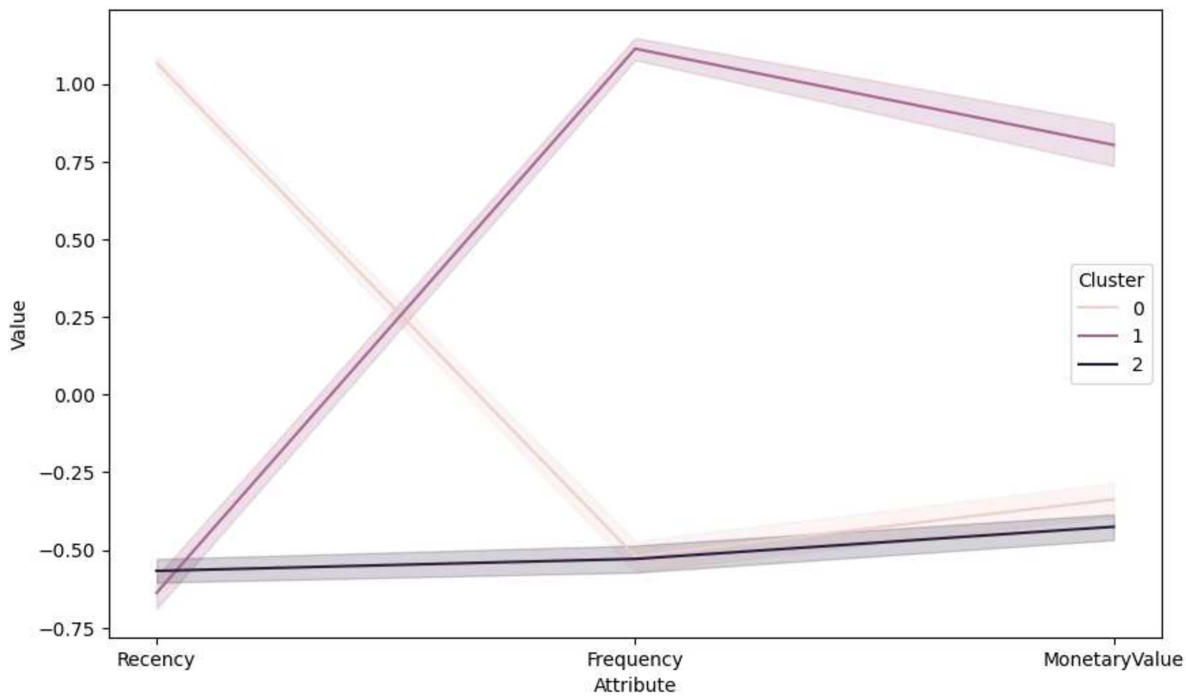


In [29]:
```
# Initialize a KMeans model with the chosen number of clusters (3 in this case)
model = KMeans(n_clusters=3, random_state=42)
model.fit(customers_normalized) # Fit the KMeans model to the normalized custom

# Group the customer data by cluster and calculate mean values and counts
customers["Cluster"] = model.labels_
customers.groupby('Cluster').agg({
    'Recency':'mean',
    'Frequency':'mean',
    'MonetaryValue':['mean', 'count']}).round(2)
```

Out[29]:

	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
Cluster				
0	233.71	1.53	23.22	878
1	50.45	6.41	144.10	779
2	45.84	1.48	19.56	776

In [30]:
```
# Create a DataFrame with normalized customer data, cluster labels, and attribu
df_normalized = pd.DataFrame(customers_normalized, columns=['Recency', 'Frequer
df_normalized['ID'] = customers.index
df_normalized['Cluster'] = model.labels_

# Melt the DataFrame to prepare for visualization
df_nor_melt = pd.melt(df_normalized.reset_index(),
                      id_vars=['ID', 'Cluster'],
                      value_vars=['Recency','Frequency','MonetaryValue'],
                      var_name='Attribute',
                      value_name='Value')
df_nor_melt.head()
# Create a line plot to visualize the attribute values by cluster
sns.lineplot(x='Attribute', y='Value', hue='Cluster', data=df_nor_melt)
plt.show()
```



In [ ]:

Dataset References

https://www.kaggle.com/code/fabiendaniel/customer-segmentation/input