Algorithm implementation

1)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

# Load customer data

customer\_data = pd.read\_csv('customer\_data.csv')

# Select relevant features for segmentation (you can customize this)

selected\_features = ['Feature1', 'Feature2', 'Feature3']

X = customer\_data[selected\_features]

# Standardize the data to have mean=0 and standard deviation=1

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Determine the optimal number of clusters using the Elbow method

wcss = []

for i in range(1, 11):

    kmeans = KMeans(n\_clusters=i, init='k-means++', max\_iter=300, n\_init=10, random\_state=0)

    kmeans.fit(X\_scaled)

    wcss.append(kmeans.inertia\_)

# Plot the Elbow method graph

plt.figure(figsize=(8, 6))

plt.plot(range(1, 11), wcss)

plt.title('Elbow Method for Optimal k')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS (Within-Cluster Sum of Squares)')

plt.show()

# Based on the Elbow method, choose the optimal number of clusters

optimal\_k = 3  # Adjust this based on the plot

# Perform K-Means clustering with the selected number of clusters

kmeans = KMeans(n\_clusters=optimal\_k, init='k-means++', max\_iter=300, n\_init=10, random\_state=0)

kmeans.fit(X\_scaled)

# Add the cluster labels to the original customer data

customer\_data['Cluster'] = kmeans.labels\_

# Analyze the segments

segmented\_data = customer\_data.groupby('Cluster').mean()

# Print the summary statistics of each cluster

print(segmented\_data)

# Visualize the clusters (for 2D or 3D data)

if len(selected\_features) == 2:

    plt.figure(figsize=(8, 6))

    for cluster in range(optimal\_k):

        plt.scatter(X\_scaled[kmeans.labels\_ == cluster][:, 0], X\_scaled[kmeans.labels\_ == cluster][:, 1], label=f'Cluster {cluster}')

    plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s=300, c='red', label='Centroids')

    plt.title('Customer Segmentation')

    plt.xlabel('Feature1')

    plt.ylabel('Feature2')

    plt.legend()

    plt.show()

elif len(selected\_features) == 3:

    from mpl\_toolkits.mplot3d import Axes3D

    fig = plt.figure(figsize=(10, 8))

    ax = fig.add\_subplot(111, projection='3d')

    for cluster in range(optimal\_k):

        ax.scatter(X\_scaled[kmeans.labels\_ == cluster][:, 0], X\_scaled[kmeans.labels\_ == cluster][:, 1], X\_scaled[kmeans.labels\_ == cluster][:, 2], label=f'Cluster {cluster}')

    ax.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], kmeans.cluster\_centers\_[:, 2], s=300, c='red', label='Centroids')

    ax.set\_title('Customer Segmentation')

    ax.set\_xlabel('Feature1')

    ax.set\_ylabel('Feature2')

    ax.set\_zlabel('Feature3')

    ax.legend()

    plt.show()

In this code, we load customer data, select relevant features for

segmentation, standardize the data, and determine the optimal number

of clusters using the Elbow method. After performing K-Means

clustering, we add cluster labels to the original data and analyze

the segments. Finally, we visualize the clusters if the data is 2D

or 3D

2)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Load customer transaction data (sample data)

data = pd.read\_csv('customer\_transactions.csv')

# Assuming the dataset has columns like 'CustomerID', 'TransactionDate', 'AmountSpent', 'ProductCategory', etc.

# Calculate total spending per customer

total\_spending = data.groupby('CustomerID')['AmountSpent'].sum()

# Calculate the frequency of transactions per customer

transaction\_frequency = data.groupby('CustomerID').size()

# Calculate the average transaction amount per customer

average\_transaction\_amount = data.groupby('CustomerID')['AmountSpent'].mean()

# Calculate the recency of transactions (assuming a reference date)

reference\_date = pd.to\_datetime('2023-01-01')

data['TransactionDate'] = pd.to\_datetime(data['TransactionDate'])

recency = data.groupby('CustomerID')['TransactionDate'].max()

recency = (reference\_date - recency).dt.days  # Recency in days

# Create a customer behavior dataframe

customer\_behavior = pd.DataFrame({

    'TotalSpending': total\_spending,

    'TransactionFrequency': transaction\_frequency,

    'AverageTransactionAmount': average\_transaction\_amount,

    'Recency': recency

})

# Visualize customer behavior

plt.figure(figsize=(12, 8))

# Plot Total Spending vs. Transaction Frequency

plt.subplot(2, 2, 1)

plt.scatter(customer\_behavior['TotalSpending'], customer\_behavior['TransactionFrequency'])

plt.title('Total Spending vs. Transaction Frequency')

plt.xlabel('Total Spending')

plt.ylabel('Transaction Frequency')

# Plot Total Spending vs. Average Transaction Amount

plt.subplot(2, 2, 2)

plt.scatter(customer\_behavior['TotalSpending'], customer\_behavior['AverageTransactionAmount'])

plt.title('Total Spending vs. Average Transaction Amount')

plt.xlabel('Total Spending')

plt.ylabel('Average Transaction Amount')

# Plot Total Spending vs. Recency

plt.subplot(2, 2, 3)

plt.scatter(customer\_behavior['TotalSpending'], customer\_behavior['Recency'])

plt.title('Total Spending vs. Recency')

plt.xlabel('Total Spending')

plt.ylabel('Recency (days)')

# Plot Transaction Frequency vs. Recency

plt.subplot(2, 2, 4)

plt.scatter(customer\_behavior['TransactionFrequency'], customer\_behavior['Recency'])

plt.title('Transaction Frequency vs. Recency')

plt.xlabel('Transaction Frequency')

plt.ylabel('Recency (days)')

plt.tight\_layout()

plt.show()

In this code, we calculate customer behavior metrics such as total spending, transaction frequency, average transaction amount, and recency. Then, we visualize the relationships between these metrics. The code plots four scatter plots to visualize how total spending correlates with other behavior metrics.

3)

import pandas as pd

import datetime as dt

# Load customer transaction data (sample data)

data = pd.read\_csv('customer\_transactions.csv')

# Assuming the dataset has a 'CustomerID', 'TransactionDate', and 'Amount' columns

# Convert 'TransactionDate' to datetime

data['TransactionDate'] = pd.to\_datetime(data['TransactionDate'])

# Calculate the current date

current\_date = max(data['TransactionDate'])

# Calculate Recency, Frequency, and Monetary Value for each customer

rfm\_data = data.groupby('CustomerID').agg({

    'TransactionDate': lambda x: (current\_date - x.max()).days,  # Recency

    'TransactionID': 'count',  # Frequency

    'Amount': 'sum'  # Monetary Value

})

# Rename the columns

rfm\_data.rename(columns={

    'TransactionDate': 'Recency',

    'TransactionID': 'Frequency',

    'Amount': 'Monetary'

}, inplace=True)

# Print the RFM data

print(rfm\_data)

# Define RFM score quartiles

quantiles = rfm\_data.quantile(q=[0.25, 0.5, 0.75])

# Function to create RFM segments

def create\_rfm\_segments(data, quantiles):

    r\_segment, f\_segment, m\_segment = '', '', ''

    if data['Recency'] <= quantiles['Recency'][0.25]:

        r\_segment = '4'

    elif data['Recency'] <= quantiles['Recency'][0.5]:

        r\_segment = '3'

    elif data['Recency'] <= quantiles['Recency'][0.75]:

        r\_segment = '2'

    else:

        r\_segment = '1'

    if data['Frequency'] <= quantiles['Frequency'][0.25]:

        f\_segment = '1'

    elif data['Frequency'] <= quantiles['Frequency'][0.5]:

        f\_segment = '2'

    elif data['Frequency'] <= quantiles['Frequency'][0.75]:

        f\_segment = '3'

    else:

        f\_segment = '4'

    if data['Monetary'] <= quantiles['Monetary'][0.25]:

        m\_segment = '1'

    elif data['Monetary'] <= quantiles['Monetary'][0.5]:

        m\_segment = '2'

    elif data['Monetary'] <= quantiles['Monetary'][0.75]:

        m\_segment = '3'

    else:

        m\_segment = '4'

    return r\_segment + f\_segment + m\_segment

# Apply the create\_rfm\_segments function to create RFM segments

rfm\_data['RFM\_Segment'] = rfm\_data.apply(create\_rfm\_segments, args=(quantiles,), axis=1)

# Calculate RFM score

rfm\_data['RFM\_Score'] = rfm\_data['Recency'].astype(str) + rfm\_data['Frequency'].astype(str) + rfm\_data['Monetary'].astype(str)

# Print the RFM segments and scores

print(rfm\_data[['RFM\_Segment', 'RFM\_Score']])

In this code, we first load the customer transaction data, calculate

Recency (how recently a customer made a purchase), Frequency (how

often a customer makes a purchase), and Monetary Value (how much a

customer spends), and then create RFM segments and scores. You can

use these segments and scores for customer segmentation and targeted

marketing.

4)

import pandas as pd

from lifetimes import BetaGeoFitter, GammaGammaFitter

import matplotlib.pyplot as plt

# Load your transaction data into a Pandas DataFrame

data = pd.read\_csv('customer\_data.csv')

# Data Preprocessing

data['date'] = pd.to\_datetime(data['date'])  # Convert the date column to a datetime format

summary = pd.pivot\_table(data, values='monetary\_value', index='customer\_id', aggfunc=['count', 'mean', 'sum'])

summary.columns = ['frequency', 'monetary\_mean', 'monetary\_sum']

# Split your data into calibration and holdout periods

calibration\_period\_end = '2022-01-01'

holdout\_period\_start = '2022-01-02'

calibration\_data = data[data['date'] <= calibration\_period\_end]

holdout\_data = data[(data['date'] > calibration\_period\_end) & (data['date'] >= holdout\_period\_start)]

# Initialize and fit the BG/NBD model

bgf = BetaGeoFitter(penalizer\_coef=0.0)

bgf.fit(calibration\_data['frequency'], calibration\_data['recency'], calibration\_data['T'])

# Calculate the expected number of future purchases for a given customer

summary['predicted\_purchases'] = bgf.predict(holdout\_data['frequency'], holdout\_data['recency'], holdout\_data['T'])

# Initialize and fit the Gamma-Gamma model

ggf = GammaGammaFitter(penalizer\_coef=0.0)

ggf.fit(summary['frequency'], summary['monetary\_mean'])

# Calculate conditional expected average profit

summary['predicted\_monetary\_value'] = ggf.conditional\_expected\_average\_profit(summary['frequency'], summary['monetary\_mean'])

# Calculate CLV for each customer

summary['CLV'] = summary['predicted\_purchases'] \* summary['predicted\_monetary\_value']

# Sort customers by CLV to identify high-value customers

sorted\_summary = summary.sort\_values(by='CLV', ascending=False)

# Print the top N customers by CLV

N = 10

top\_customers = sorted\_summary.head(N)

print("Top {} Customers by CLV:".format(N))

print(top\_customers)

# Visualize the distribution of CLV

plt.figure(figsize=(10, 5))

plt.hist(summary['CLV'], bins=30, edgecolor='k')

plt.title('CLV Distribution')

plt.xlabel('Customer Lifetime Value')

plt.ylabel('Number of Customers')

plt.show()

# Export the results to a CSV file or any other desired format

top\_customers.to\_csv('top\_customers\_clv.csv', index=False)

In this code:

The code loads your transaction data and preprocesses it to calculate frequency, monetary mean, and monetary sum for each customer.

It splits the data into calibration and holdout periods for model training and validation.

The BG/NBD model is trained using the calibration data to predict future purchases in the holdout period.

The Gamma-Gamma model is trained using summary statistics to estimate conditional expected average profit.

CLV is calculated for each customer based on these predictions.

The code visualizes the CLV distribution using a histogram.

The top N customers by CLV are printed and can be exported to a CSV file.