RFM Analysis Case Study

Tasks:

1. Data preprocessing - 10%

2. Build a Recency Frequency Monetary Model - 30%

3. Perform customer segmentation with KMeans Clustering - 30%

4. Create segments to determine total customer value for the retail - 30%

In [3]:

```
# Importing the basic libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

In [4]:

```
# Load the datasets
train_df = pd.read_excel("train.xlsx")
test_df = pd.read_excel("test.xlsx")
```

In [5]:

```
# Explore the datasets as per the below outputs
print("Shape of train data")
print("-" * 20)
print(train_df.shape)
print("-" * 20)

print("Columns of train data")
print("-" * 20)
print(train_df.columns.tolist())
print("-" * 20)

print("Types of train columns")
print("-" * 20)
print(train_df.info())
print("-" * 20)
```

```python
print("Shape of test data")
print("-" * 20)
print(test_df.shape)
print("-" * 20)

print("Columns of test data")
print("-" * 20)
print(test_df.columns.tolist())
print("-" * 20)

print("Types of test columns")
print("-" * 20)
print(test_df.info())
```

```
Shape of train data
--------------------
(379336, 8)
--------------------
Columns of train data
--------------------
['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate', 'UnitPrice', 'CustomerID', 'Country']
--------------------
Types of train columns
--------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 379336 entries, 0 to 379335
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    379336 non-null  object
 1   StockCode    379336 non-null  object
 2   Description  378373 non-null  object
 3   Quantity     379336 non-null  int64
 4   InvoiceDate  379336 non-null  datetime64[ns]
 5   UnitPrice    379336 non-null  float64
 6   CustomerID   285076 non-null  float64
 7   Country      379336 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 23.2+ MB
None
--------------------
Shape of test data
--------------------
(162573, 8)
--------------------
Columns of test data
--------------------
['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate', 'UnitPrice', 'CustomerID', 'Country']
--------------------
Types of test columns
--------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 162573 entries, 0 to 162572
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    162573 non-null  object
 1   StockCode    162573 non-null  object
 2   Description  162082 non-null  object
 3   Quantity     162573 non-null  int64
 4   InvoiceDate  162573 non-null  datetime64[ns]
 5   UnitPrice    162573 non-null  float64
```

```
 6   CustomerID   121753 non-null  float64
 7   Country      162573 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 9.9+ MB
None
```

In [6]:
```
# Basic sumary statiscal analysis as shown below
train_df.describe(include='all')
```

Out[6]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
count	379336.0	379336	378373	379336.000000	379336	379336.000000	285076.000000	379336
unique	23857.0	4008	4132	NaN	NaN	NaN	NaN	38
top	573585.0	85123A	WHITE HANGING HEART T-LIGHT HOLDER	NaN	NaN	NaN	NaN	United Kingdom
freq	774.0	1611	1649	NaN	NaN	NaN	NaN	346854
mean	NaN	NaN	NaN	9.517272	2011-07-04 11:02:43.239080960	4.681474	15288.302463	NaN
min	NaN	NaN	NaN	-80995.000000	2010-12-01 08:26:00	-11062.060000	12346.000000	NaN
25%	NaN	NaN	NaN	1.000000	2011-03-28 11:34:00	1.250000	13958.750000	NaN
50%	NaN	NaN	NaN	3.000000	2011-07-19 15:23:00	2.080000	15152.000000	NaN
75%	NaN	NaN	NaN	10.000000	2011-10-19 09:43:00	4.130000	16791.000000	NaN
max	NaN	NaN	NaN	80995.000000	2011-12-09 12:50:00	38970.000000	18287.000000	NaN
std	NaN	NaN	NaN	259.070548	NaN	105.799352	1712.323663	NaN

In [7]:
```
# Looking for missing values along with NaN from the dataset as shown below
print(train_df.isnull().sum())
print(test_df.isnull().sum())
```

```
InvoiceNo          0
StockCode          0
Description      963
Quantity           0
InvoiceDate        0
UnitPrice          0
CustomerID     94260
Country            0
dtype: int64
InvoiceNo          0
StockCode          0
Description      491
Quantity           0
InvoiceDate        0
UnitPrice          0
CustomerID     40820
Country            0
dtype: int64
```

In [8]:

```
# Drop null values for customer id since it is not an important categorical data and use mode to replace the null value

# Drop vals where customer id is null
train_df = train_df.dropna(subset=['CustomerID'])
test_df = test_df.dropna(subset=['CustomerID'])

# Replace nulls in Description with mode
desc_mode = train_df['Description'].mode()[0]

train_df['Description'] = train_df['Description'].fillna(desc_mode)
test_df['Description'] = test_df['Description'].fillna(desc_mode)
```

In [9]:

```
# Display duplicate records as shown below
print(train_df[train_df.duplicated()])
```

```
        InvoiceNo StockCode                          Description  Quantity  \
2878       575117     21098                 CHRISTMAS TOILET ROLL         1
5729       542107     21755              LOVE BUILDING BLOCK WORD         1
7615       577778     21733       RED HANGING HEART T-LIGHT HOLDER         1
8997       578781     22988                       SOLDIERS EGG CUP          1
14797      575583     20893   HANGING BAUBLE T-LIGHT HOLDER SMALL         1
...           ...       ...                                  ...       ...
378899     577773     23507        MINI PLAYING CARDS BUFFALO BILL          1
379020     571682     23182          TOILET SIGN OCCUPIED OR VACANT         1
379073     564729     22208                WOOD STAMP SET THANK YOU         2
379205     538368     22759             SET OF 3 NOTEBOOKS IN PARCEL         1
379226     578041     22726               ALARM CLOCK BAKELIKE GREEN         1

                InvoiceDate  UnitPrice  CustomerID         Country
2878    2011-11-08 14:22:00       1.25     12748.0  United Kingdom
5729    2011-01-25 13:38:00       5.95     16222.0  United Kingdom
7615    2011-11-21 16:10:00       2.95     16549.0  United Kingdom
8997    2011-11-25 11:54:00       1.25     15872.0  United Kingdom
14797   2011-11-10 11:55:00       2.55     14456.0  United Kingdom
...                     ...        ...         ...             ...
378899 2011-11-21 15:57:00       0.42     16712.0  United Kingdom
379020 2011-10-18 14:00:00       0.83     14179.0  United Kingdom
379073 2011-08-28 12:44:00       0.83     13137.0  United Kingdom
379205 2010-12-12 10:57:00       1.65     15503.0  United Kingdom
379226 2011-11-22 14:23:00       3.75     17338.0  United Kingdom

[2656 rows x 8 columns]
```

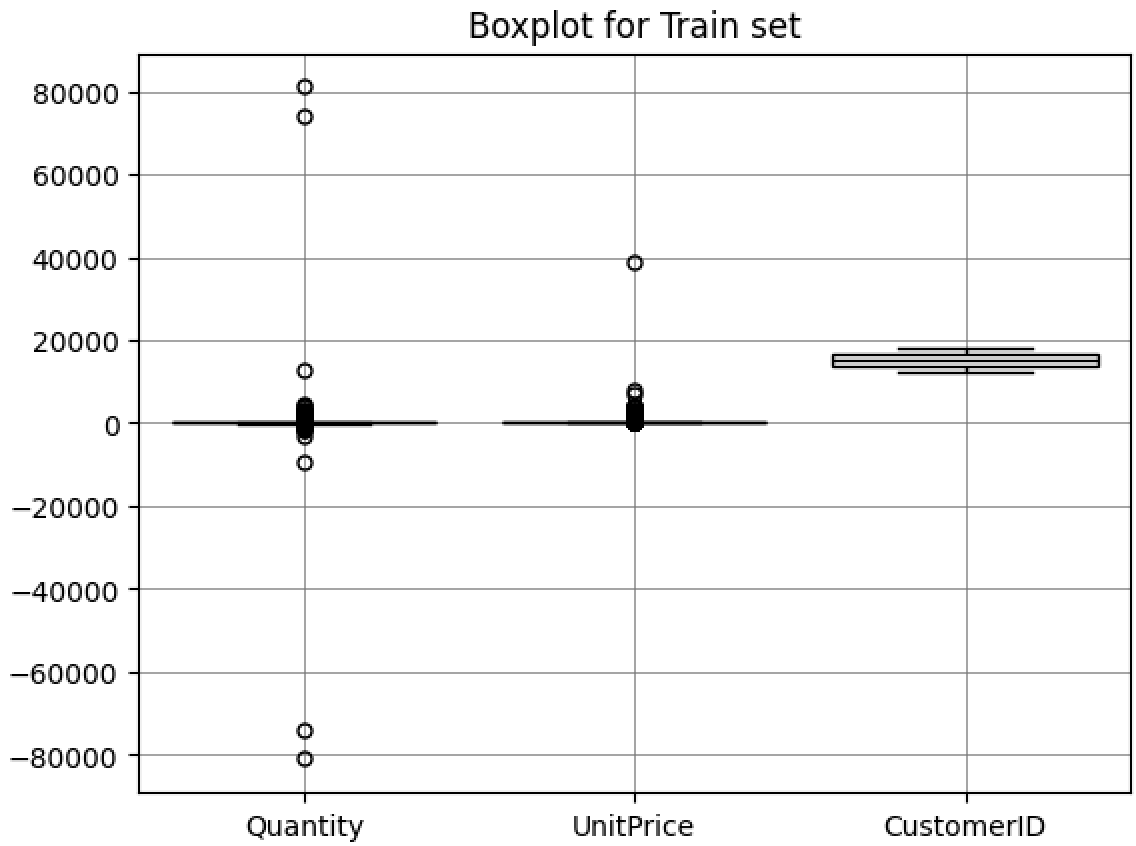
In [10]:

```python
# Drop duplicate rows
train_df = train_df.drop_duplicates()
test_df = test_df.drop_duplicates()
```

In [11]:

```python
# Display train dataset outliers with the box plot method as shown below
numeric_cols = ['Quantity', 'UnitPrice', 'CustomerID']

plt.title('Boxplot for Train set')

sns.boxplot(
    data=train_df[['Quantity', 'UnitPrice', 'CustomerID']],
    palette=['white']*3,
    fliersize=5,
    linewidth=1,
    boxprops=dict(edgecolor='black'),
    medianprops=dict(color='black'),
    whiskerprops=dict(color='black'),
    capprops=dict(color='black'),
    flierprops=dict(marker='o', color='black', markersize=5, markeredgecolor='black')
```

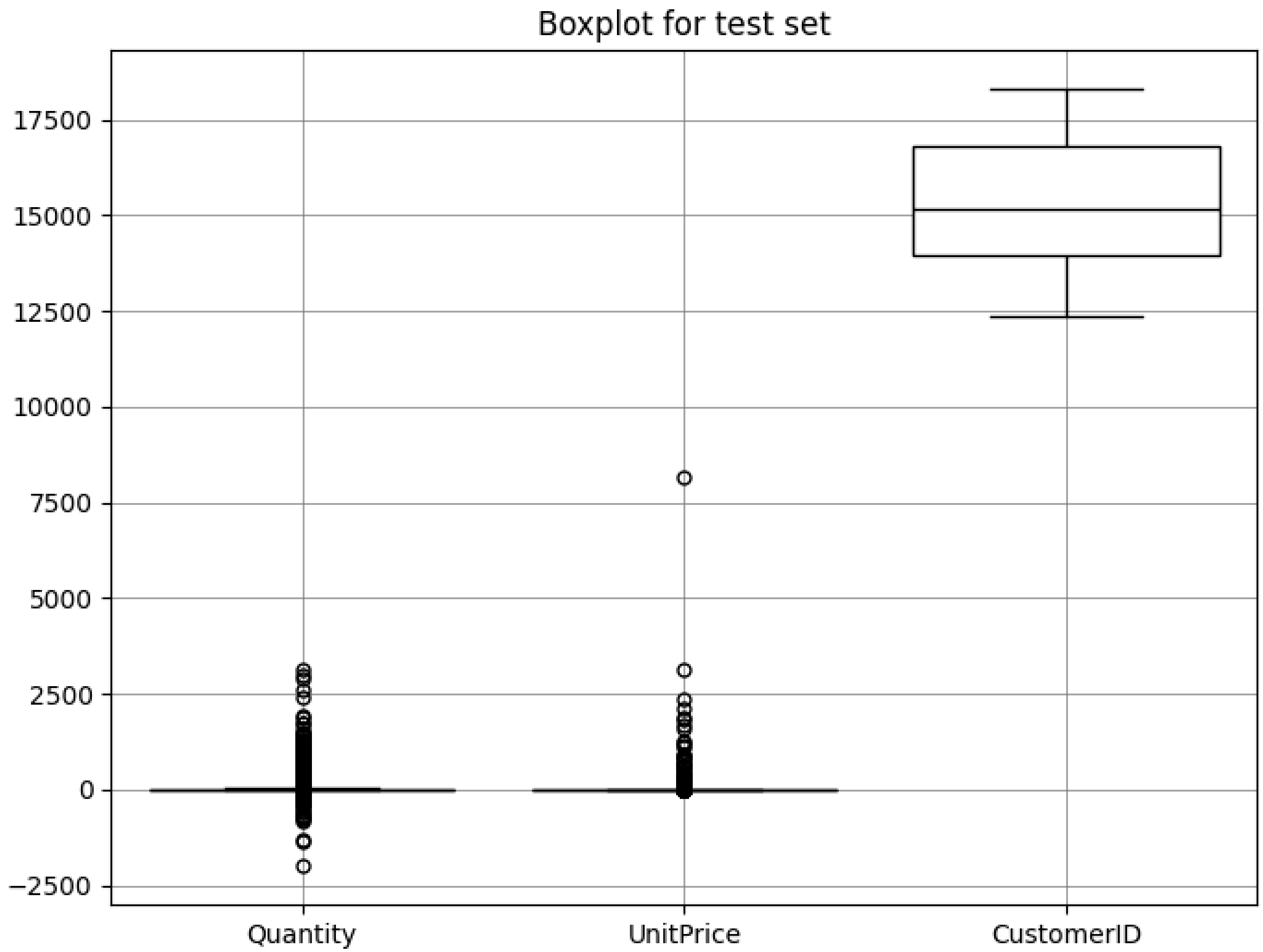
```
)

plt.grid(True, color='gray', linestyle='-', linewidth=0.5)
plt.show()
```



In [12]:

```
# Display test dataset outliers with the box plot method as shown below

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

sns.boxplot(
    data=test_df[['Quantity', 'UnitPrice', 'CustomerID']],
    palette=['white']*3,
    fliersize=5,
    linewidth=1,
    boxprops=dict(facecolor='white', edgecolor='black'),
    medianprops=dict(color='black'),
    whiskerprops=dict(color='black'),
    capprops=dict(color='black'),
    flierprops=dict(marker='o', color='black', markersize=5, markeredgecolor='black')
)
```

```
plt.title('Boxplot for test set')
plt.grid(True, color='gray', linestyle='-', linewidth=0.5)
plt.show()
```



In [13]:
```
# Create monthly groupings on the basis of customer id and invoice date while converting cust_id to integer

# convert customer id to int
train_df['CustomerID'] = train_df['CustomerID'].astype(int)

#Convert InvoiceDate to datetime
train_df['InvoiceDate'] = pd.to_datetime(train_df['InvoiceDate'])

#create a new column for Invoice Month
train_df['InvoiceMonth'] = train_df['InvoiceDate'].dt.to_period('M')
```

```python
# group by CustomerID and InvoiceMonth
monthly_grouped = train_df.groupby(['CustomerID', 'InvoiceMonth'])
```

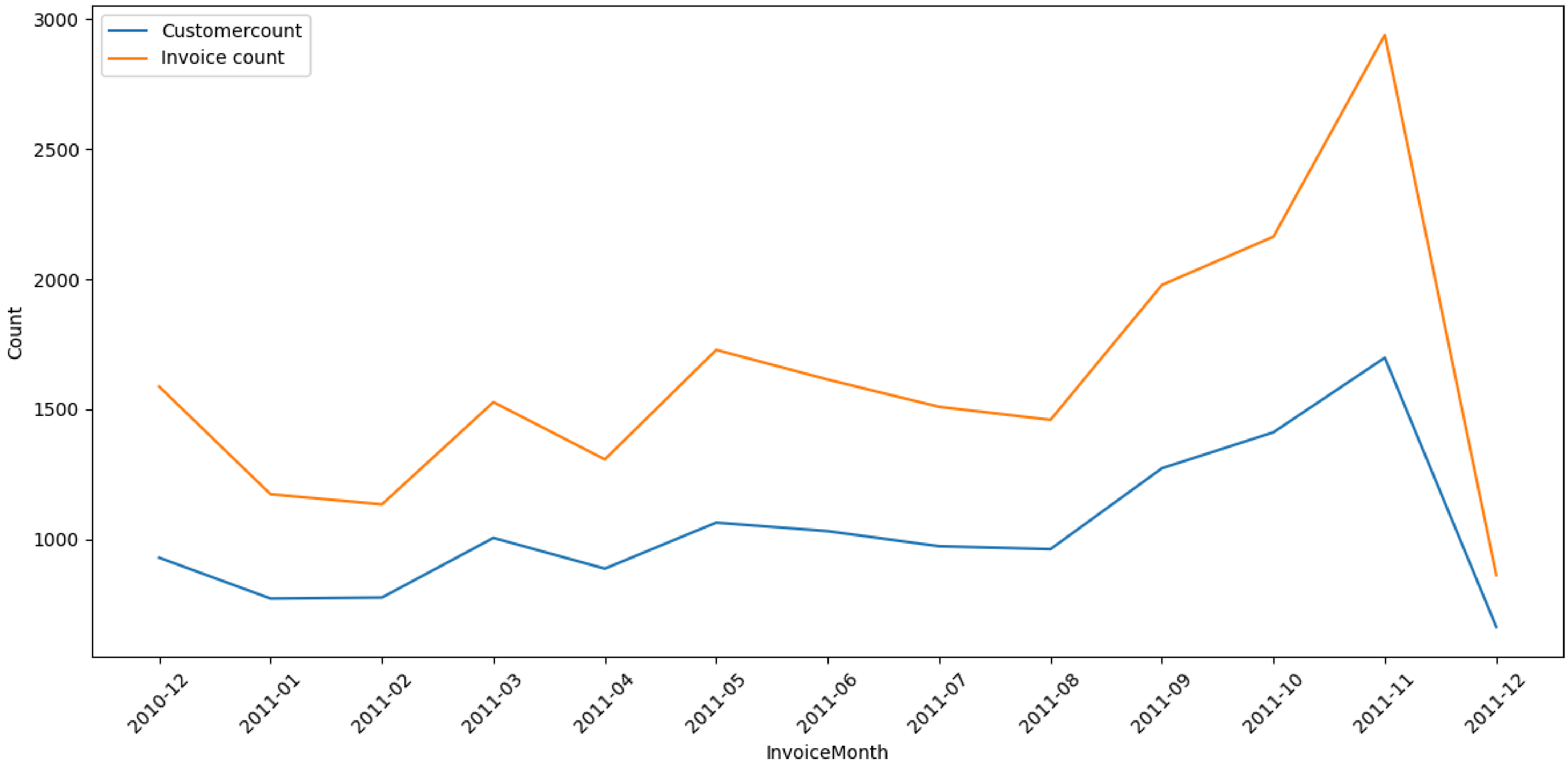
In [14]:
```python
# Looking for unique values in invoice date on the basis of years and months as shown below

# Total Unique vals in InvoiceDate
print(train_df['InvoiceDate'].nunique())

#Unique years
print(train_df['InvoiceDate'].dt.year.unique())

# unique months
print(train_df['InvoiceDate'].dt.month.unique())
```

```
19427
[2011 2010]
[ 6  5  1 12  9 10  2 11  7  8  3  4]
```

In [15]:
```python
# Using the above to extract string from time and showing the number of unique months as shown below
train_df['InvoiceMonthStr'] =train_df['InvoiceDate'].dt.to_period('M').astype(str)

np.unique(train_df['InvoiceMonthStr'].values)
```

Out[15]:
```
array(['2010-12', '2011-01', '2011-02', '2011-03', '2011-04', '2011-05',
       '2011-06', '2011-07', '2011-08', '2011-09', '2011-10', '2011-11',
       '2011-12'], dtype=object)
```

In [16]:
```python
# Plot the customer count and invoice count across the unique months as shown below

# Counting the unique customers and invoices per month
monthly_summary = train_df.groupby('InvoiceMonth').agg({
    'CustomerID': 'nunique',
    'InvoiceNo': 'nunique'
}).rename(columns={'CustomerID': 'Customercount', 'InvoiceNo': 'Invoice count'})

plt.figure(figsize=(12, 6))
plt.plot(monthly_summary.index.astype(str), monthly_summary['Customercount'], label='Customercount')
plt.plot(monthly_summary.index.astype(str), monthly_summary['Invoice count'], label='Invoice count')
plt.xlabel('InvoiceMonth')
plt.ylabel('Count')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Building Recency Frequency Monetary (RFM)

In [18]:
```
# Calculate the total sales for both train and test datasets. Dispaly the below using the train dataset as shown below

train_df['Sales'] = train_df['Quantity'] *train_df['UnitPrice']

sales_summary = train_df.groupby(['InvoiceNo','InvoiceDate','CustomerID'], as_index=False)['Sales'].sum()

sales_summary.head()
```

Out[18]:

	InvoiceNo	InvoiceDate	CustomerID	Sales
0	536365	2010-12-01 08:26:00	17850	113.62
1	536366	2010-12-01 08:28:00	17850	11.10
2	536367	2010-12-01 08:34:00	13047	149.93
3	536368	2010-12-01 08:34:00	13047	55.20
4	536369	2010-12-01 08:35:00	13047	17.85

In [19]:

```
# Recency has to be claculated from a specific date. Show the below anchor/reference date
reference_date = train_df['InvoiceDate'].max()
reference_date
```

Out[19]:

```
Timestamp('2011-12-09 12:50:00')
```

In [20]:

```
# Grouping by cust id and create a new table as shown below
reference_date = train_df['InvoiceDate'].max()

rfm = train_df.groupby('CustomerID').agg({
    'InvoiceDate': lambda x: (reference_date - x.max()).days, # Recency
    'InvoiceNo': 'nunique', # Frequency
    'Sales': 'sum' # MonetaryValue
})

rfm.columns = ['Recency', 'Frequency', 'MonetaryValue']

rfm.head()
```

Out[20]:

CustomerID	Recency	Frequency	MonetaryValue
12346	325	2	0.00
12347	1	7	3124.96
12348	74	4	1009.88
12349	18	1	1344.17
12350	309	1	213.30

Give recency, frequency, and monetary scores individually by dividing them into quartiles, Combine three ratings to get a RFM segment (as strings), Get the RFM score by adding up the three ratings, Analyze the RFM segments by summarizing them and comment on the findings

Rate “recency" for customer who has been active more recently higher than the less recent customer. Rate “frequency" and “monetary" higher, because the company wants the customer to visit more often and spend more money

In [22]:
```
# Calculate RFM groups,labels and quartiles with the qcut function where recency_labels = range(4, 0, -1),
# frequency_labels = range(1, 5), montary_labels = range(1, 5)
r_labels = range(4, 0, -1)
f_labels = range(1, 5)
m_labels = range(1, 5)

# Assign these labels to 4 equal percentile groups for recency
rfm['R'] = pd.qcut(rfm['Recency'], q=4, labels=r_labels)

# Assign these labels to 4 equal percentile groups for frequency
rfm['F'] = pd.qcut(rfm['Frequency'].rank(method='first'), q=4, labels=f_labels)

# Assign these labels to 4 equal percentile groups for montary value
rfm['M'] = pd.qcut(rfm['MonetaryValue'].rank(method='first'), q=4, labels=m_labels)
```

In [23]:
```
#Display the below output
print(rfm['Recency'])
print(rfm['Frequency'])
print(rfm['MonetaryValue'])
```

```
CustomerID
12346    325
12347      1
12348     74
12349     18
12350    309
        ...
18280    277
18281    180
18282      7
18283      3
18287     42
Name: Recency, Length: 4353, dtype: int64
CustomerID
12346     2
12347     7
12348     4
12349     1
12350     1
         ..
18280     1
18281     1
18282     3
18283    16
18287     3
Name: Frequency, Length: 4353, dtype: int64
CustomerID
12346       0.00
12347    3124.96
12348    1009.88
12349    1344.17
12350     213.30
          ...
18280      91.70
18281      59.28
18282     118.16
18283    1450.29
18287    1430.78
Name: MonetaryValue, Length: 4353, dtype: float64
```

In [24]:

```
# Adding the new columns to original rmf as shown below

# copy of original RFM table
rfm_scored = rfm.copy()

r_labels = range(4, 0, -1)
f_labels = range(1, 5)
m_labels = range(1, 5)
```

```
rfm_scored['R'] = pd.qcut(rfm_scored['Recency'], q=4, labels=r_labels)
rfm_scored['F'] = pd.qcut(rfm_scored['Frequency'].rank(method='first'), q=4, labels=f_labels)
rfm_scored['M'] = pd.qcut(rfm_scored['MonetaryValue'].rank(method='first'), q=4, labels=m_labels)

rfm_scored.head()
```

Out[24]:

CustomerID	Recency	Frequency	MonetaryValue	R	F	M
12346	325	2	0.00	1	2	1
12347	1	7	3124.96	4	4	4
12348	74	4	1009.88	2	3	3
12349	18	1	1344.17	3	1	4
12350	309	1	213.30	1	1	2

In [25]:

```
# Combine three ratings to get a RFM segment as shown below
rfm_scored['RFM_segment'] = (
    rfm_scored['R'].astype(str) +
    rfm_scored['F'].astype(str) +
    rfm_scored['M'].astype(str)
)

rfm_scored.head()
```

Out[25]:

CustomerID	Recency	Frequency	MonetaryValue	R	F	M	RFM_segment
12346	325	2	0.00	1	2	1	121
12347	1	7	3124.96	4	4	4	444
12348	74	4	1009.88	2	3	3	233
12349	18	1	1344.17	3	1	4	314
12350	309	1	213.30	1	1	2	112

In [26]:
```python
# Get the RFM score by adding up the three ratings as shown below.
rfm_scored['RFM_Score'] = (
    rfm_scored[['R', 'F', 'M']].astype(int).sum(axis=1)
)
rfm_scored.head()
```

Out[26]:

CustomerID	Recency	Frequency	MonetaryValue	R	F	M	RFM_segment	RFM_Score
12346	325	2	0.00	1	2	1	121	4
12347	1	7	3124.96	4	4	4	444	12
12348	74	4	1009.88	2	3	3	233	8
12349	18	1	1344.17	3	1	4	314	8
12350	309	1	213.30	1	1	2	112	4

In [27]:
```python
# show the number of unique segments, unique RFM_Scores, and print the best customers as shown below
print(rfm_scored['RFM_segment'].nunique())
print(sorted(rfm_scored['RFM_Score'].unique()))


best_customers = rfm_scored[rfm_scored['RFM_Score'] == 12]
best_customers.head()
```

```
63
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

Out[27]:

CustomerID	Recency	Frequency	MonetaryValue	R	F	M	RFM_segment	RFM_Score
12347	1	7	3124.96	4	4	4	444	12
12359	7	6	4681.17	4	4	4	444	12
12362	2	13	3463.01	4	4	4	444	12
12388	15	6	2235.13	4	4	4	444	12
12395	15	14	2189.10	4	4	4	444	12

In [28]:
```python
# Define rfm_level function on the basis of importance using the following criteria: Important (RFM_Score>= 9),
# Good (RFM_Score >= 8 and < 9), Okay (RFM_Score >= 7 and < 8), Neutral (RFM_Score >= 6 and < 7),
# Might (RFM_Score >= 5) and < 6), Needs Attention (RFM_Score >= 4 and < 5) Otherwise Activate
```

```python
def rfm_level(score):
    if score >= 9:
        return 'Important'
    elif score >= 8:
        return 'Good'
    elif score >= 7:
        return 'Okay'
    elif score >= 6:
        return 'Neutral'
    elif score >= 5:
        return 'Might'
    elif score >= 4:
        return 'Needs Attention'
    else:
        return 'Activate'

rfm_scored['RFM_Level'] = rfm_scored['RFM_Score'].apply(rfm_level)
rfm_scored.head()
```

Out[28]:

CustomerID	Recency	Frequency	MonetaryValue	R	F	M	RFM_segment	RFM_Score	RFM_Level
12346	325	2	0.00	1	2	1	121	4	Needs Attention
12347	1	7	3124.96	4	4	4	444	12	Important
12348	74	4	1009.88	2	3	3	233	8	Good
12349	18	1	1344.17	3	1	4	314	8	Good
12350	309	1	213.30	1	1	2	112	4	Needs Attention

In [29]:

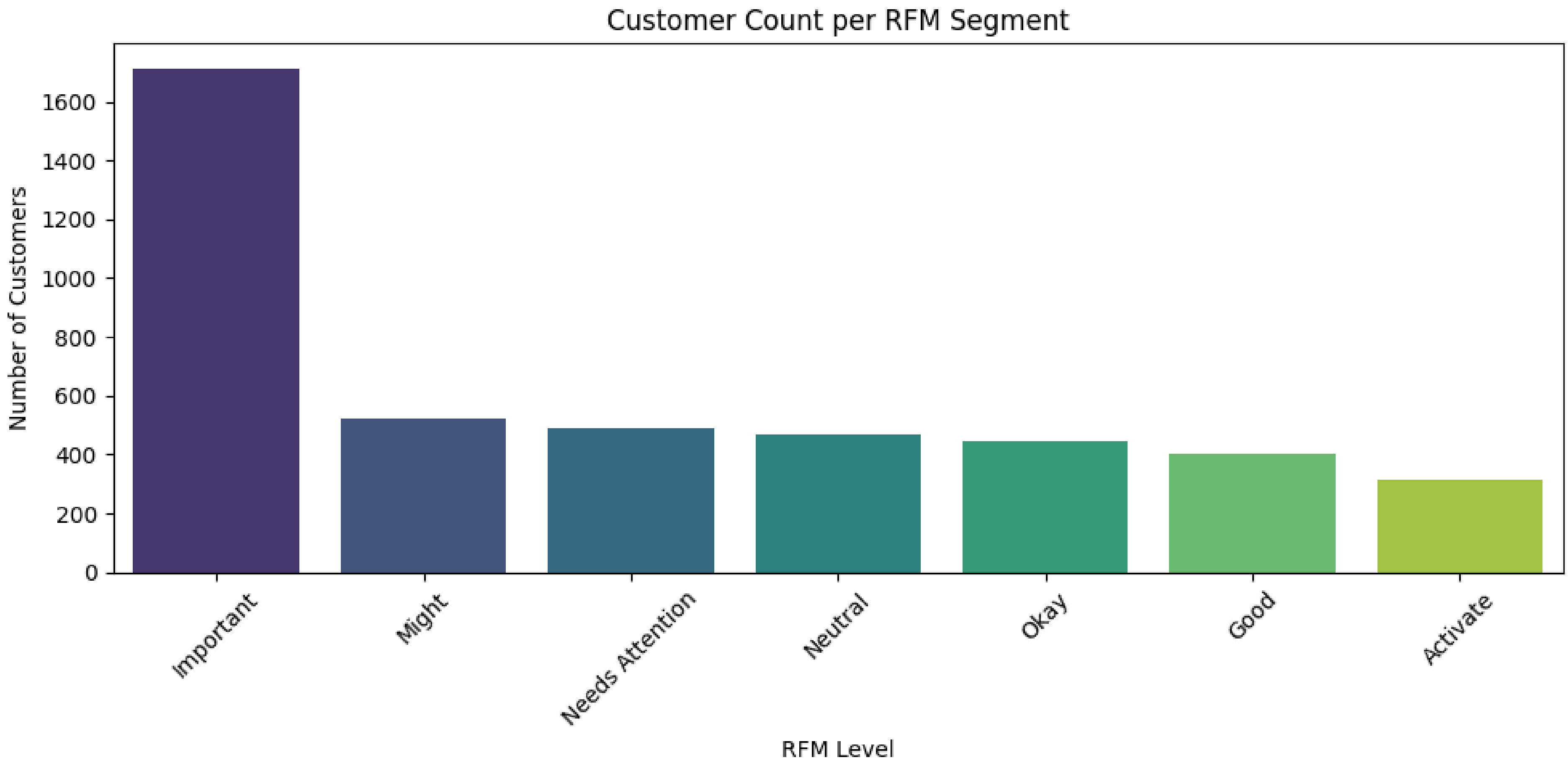
```python
# Calculate average values for each RFM_Level, and return a size of each segment as shown below
rfm_summary = rfm_scored.groupby('RFM_Level').agg({
    'Recency': 'mean',
    'Frequency': 'mean',
    'MonetaryValue': 'mean',
    'RFM_Score': 'count'
}).rename(columns={'RFM_Score': 'count'})

rfm_summary.head()
```

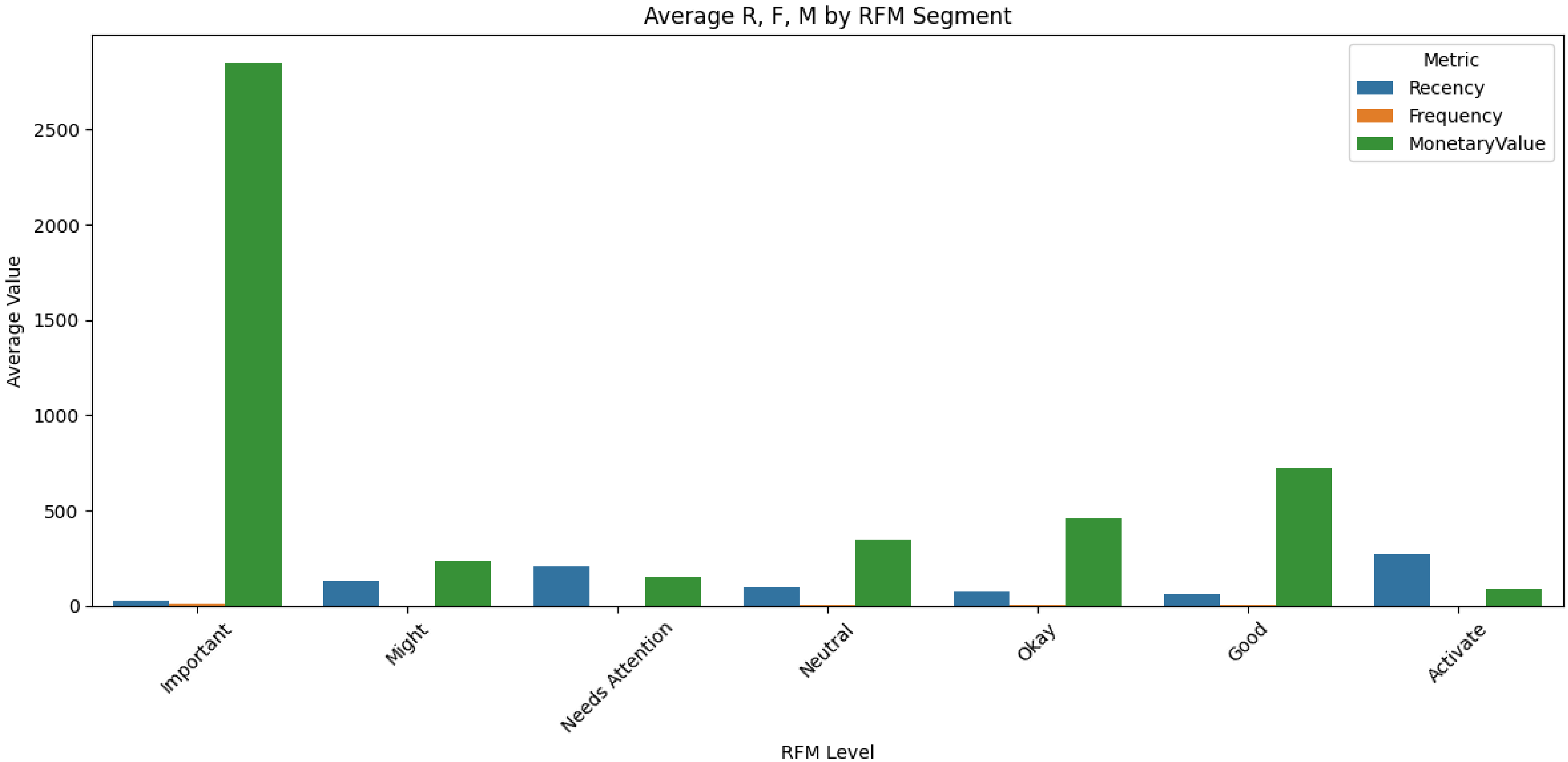
Out[29]:

RFM_Level	Recency	Frequency	MonetaryValue	count
Activate	270.394904	1.000000	92.132229	314
Good	60.727047	3.096774	726.174891	403
Important	24.761098	9.474299	2853.218867	1712
Might	131.676245	1.409962	238.093333	522
Needs Attention	204.605317	1.184049	151.744622	489

In [30]:

```
# Plot the above information and draw your insight


#Plotting the bar chart of segment sizes
rfm_summary_sorted = rfm_summary.sort_values('count', ascending=False)

plt.figure(figsize=(10, 5))
sns.barplot(x=rfm_summary_sorted.index, y=rfm_summary_sorted['count'], hue=rfm_summary_sorted.index, palette='viridis',
plt.title('Customer Count per RFM Segment')
plt.ylabel('Number of Customers')
plt.xlabel('RFM Level')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()


# Plotting the average RFM metrics per segment

rfm_summary_melted = rfm_summary_sorted.drop(columns='count').reset_index().melt(id_vars='RFM_Level')

plt.figure(figsize=(12, 6))
sns.barplot(data=rfm_summary_melted, x='RFM_Level', y='value', hue='variable')
plt.title('Average R, F, M by RFM Segment')
plt.ylabel('Average Value')
plt.xlabel('RFM Level')
plt.legend(title='Metric')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```


Customer Count per RFM Segment
Number of Customers
0
200
400
600
800
1000
1200
1400
1600
Important
Might
Needs Attention
Neutral
Okay
Good
Activate
RFM Level



My Insights

- **Important** customers are the largest and most valuable segment; they purchase often, spend the most, and are recently active.
- **Activate** segment includes the least engaged customers with high recency, low frequency, and low spending — consider reactivation campaigns.
- **Needs Attention**, **Might**, and **Neutral** segments show moderate activity; they are at risk of churn without targeted engagement.
- **Good** and **Okay** segments spend relatively well but are smaller; these are potential candidates for upselling or conversion to "Important".
- Spending is heavily concentrated in the **Important** group — indicating a small group of customers drives most revenue (Pareto effect).

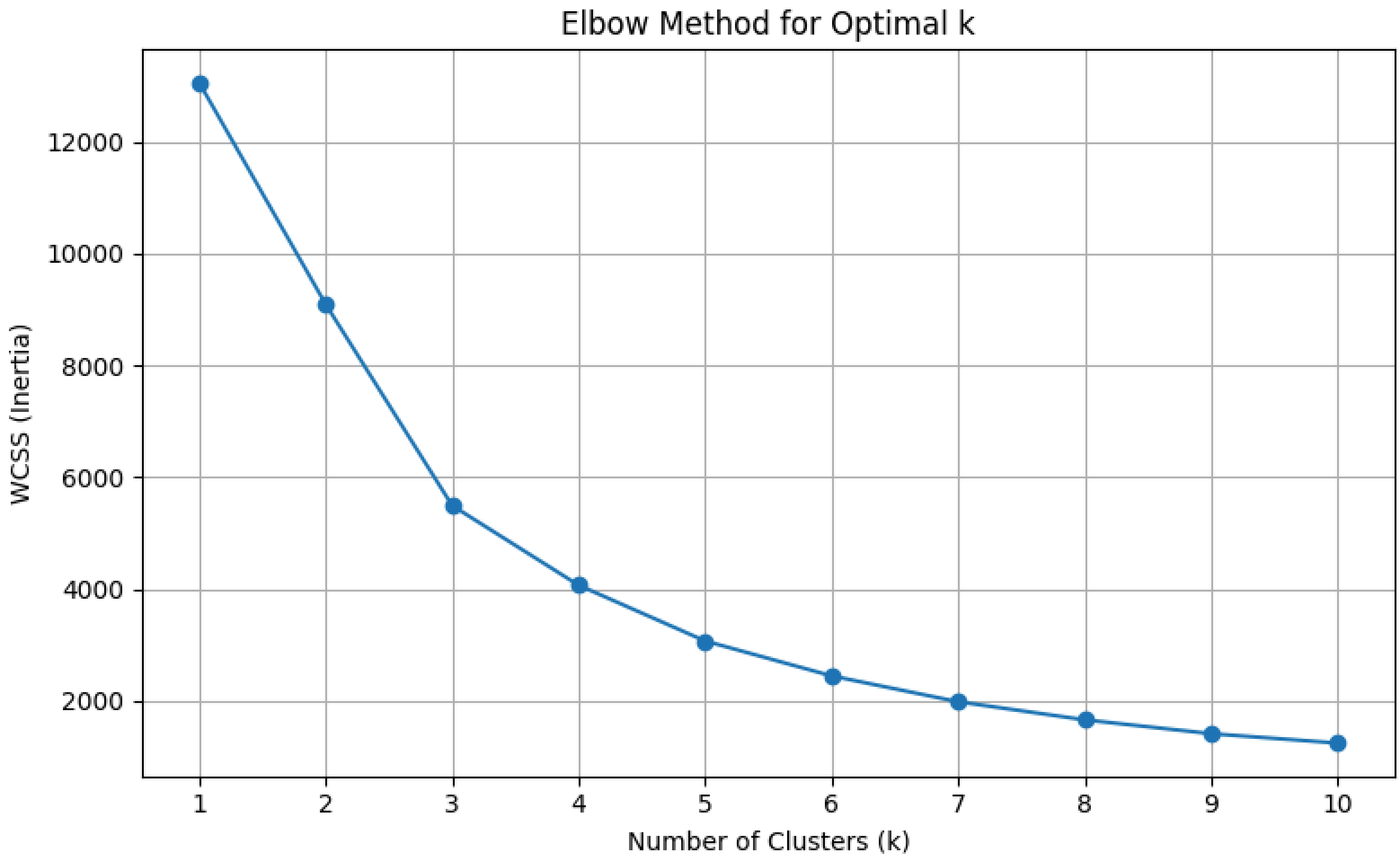
** Modeling training data with Kmeans **

In [33]:
```
# Explore the optimum number of clusters/cluster sum of squares (WCSS) with max_iter   = 300 and n_init =10
X = rfm_scored[['Recency', 'Frequency', 'MonetaryValue']]

scaler = StandardScaler()
```

```python
X_scaled = scaler.fit_transform(X)

wcss = []
K_range = range(1, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, max_iter=300, n_init=10, random_state=5503)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)
```

In [34]:

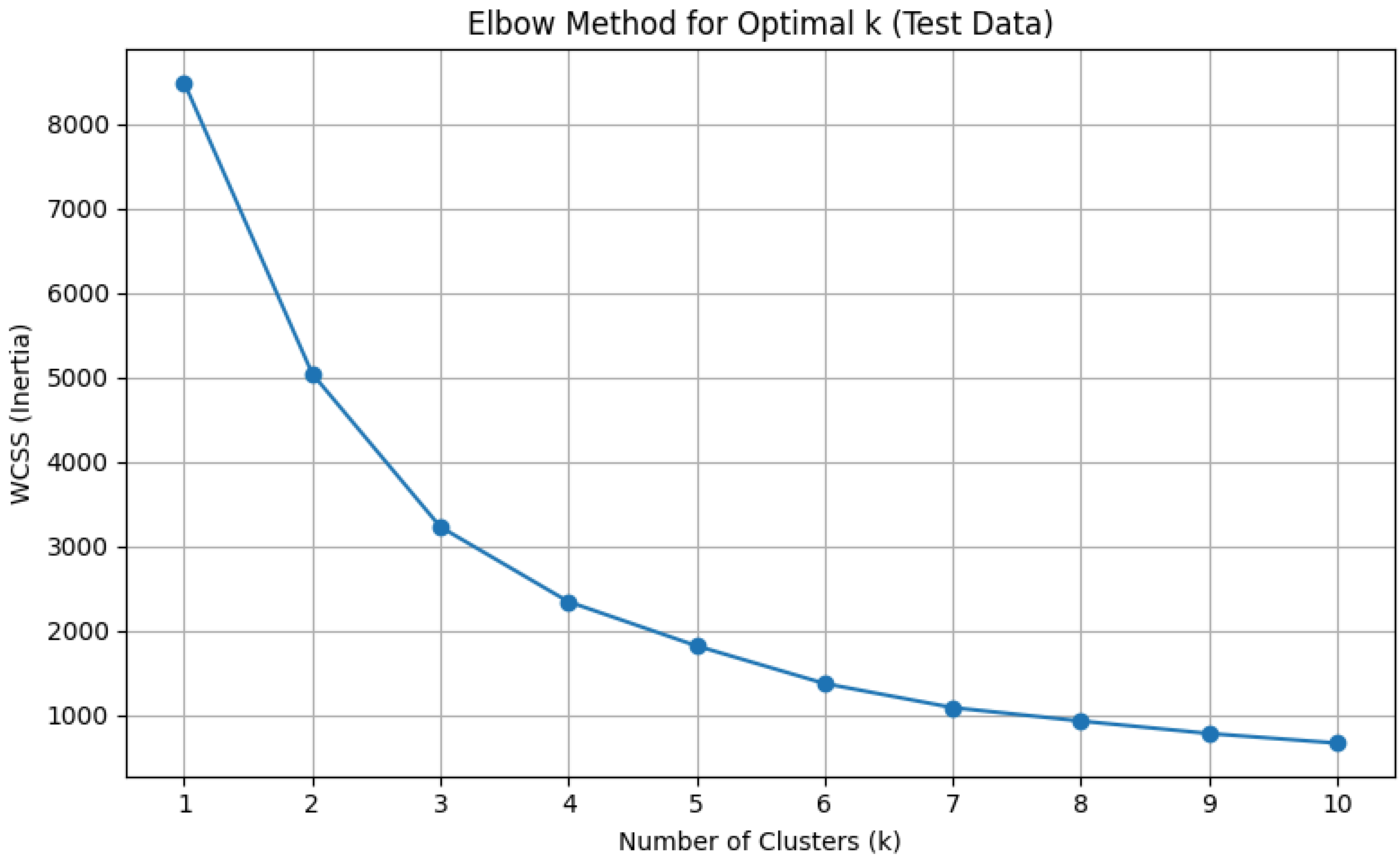
```python
# Plot the above results into a line graph and determine the optimum number of clusters.
plt.figure(figsize=(8, 5))
plt.plot(K_range, wcss, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('WCSS (Inertia)')
plt.xticks(K_range)
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [35]:
```
# What insight can you gather from the above graph?

# I can gather that the WCSS drops sharply until k=4, after which the rate of decrease slows down.
# This indicates that 4 is likely the optimal number of clusters
```

In [36]:
```
# Repeating the above steps for test data

test_df['Sales'] = test_df['Quantity'] * test_df['UnitPrice']

# reference date for test data
test_reference_date = test_df['InvoiceDate'].max()

# compute rfm table
rfm_test = test_df.groupby('CustomerID').agg({
    'InvoiceDate': lambda x: (test_reference_date - x.max()).days,
    'InvoiceNo': 'nunique',
    'Sales': 'sum'
```

```python
})

rfm_test.columns = ['Recency', 'Frequency', 'MonetaryValue']

# ccale test rfm features
X_test = rfm_test[['Recency', 'Frequency', 'MonetaryValue']]
X_test_scaled = scaler.transform(X_test)

# WCSS loop and elbow plot
wcss_test = []
for k in K_range:
    kmeans = KMeans(n_clusters=k, max_iter=300, n_init=10, random_state=5503)
    kmeans.fit(X_test_scaled)
    wcss_test.append(kmeans.inertia_)

plt.figure(figsize=(8, 5))
plt.plot(K_range, wcss_test, marker='o')
plt.title('Elbow Method for Optimal k (Test Data)')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('WCSS (Inertia)')
plt.xticks(K_range)
plt.grid(True)
plt.tight_layout()
plt.show()
```



** Create segments to determine total customer value for the retail **

In [38]:
```
# Applying kmeans using the optimal number of clusters
kmeans = KMeans(n_clusters=4, max_iter=300, n_init=10, random_state=5503)
kmeans.fit(X_scaled)

rfm_scored['Cluster'] = kmeans.labels_
```

In [39]:
```
# Create a new cloumn for your predictions
rfm_test['Cluster'] = kmeans.predict(X_test_scaled)
```

In [40]:
```
# As shown below, show the number of Cluster along with the number of customers in it. What's your insight here?
print(rfm_test['Cluster'].value_counts())
```

```
Cluster
3    3034
0    1058
1     127
2       5
Name: count, dtype: int64
```

My Insight

- Most customers fall into **Cluster 3**, indicating a dominant behavior pattern in the test set.
- **Cluster 0** is significantly smaller, but still represents a noticeable customer group.
- **Cluster 1** and especially **Cluster 2** are very small, suggesting niche segments, likely outliers or high-value customers.
- The sharp drop from Cluster 3 to Clusters 1 and 2 implies that the majority of customers share similar RFM characteristics, while only a few deviate meaningfully.

In [42]:
```python
# Analyze the customers within each cluster seperately to gather insight for each customer segment.


# Avg RFM values per cluster
cluster_summary = rfm_test.groupby('Cluster')[['Recency', 'Frequency','MonetaryValue']].mean()
cluster_counts = rfm_test['Cluster'].value_counts().rename('count')

cluster_insights = cluster_summary.join(cluster_counts)

print(cluster_insights)
```

```
            Recency   Frequency  MonetaryValue  count
Cluster
0        246.904537    1.636106     158.728034   1058
1          6.929134   29.055118    5481.148189    127
2          1.600000  111.000000   58813.214000      5
3         41.954515    4.031971     447.323817   3034
```

My Insights

- **Cluster 0**: These customers haven't bought anything in a long time, and when they did, they didn't spend much or shop often.
    - They may be inactive or have lost interest. Consider sending reactivation emails or limited-time offers.
- **Cluster 1**: These are loyal, high-value customers who buy often and spend a lot.
    - Focus on keeping them happy with loyalty rewards, special deals, or early access to new products.
- **Cluster 2**: This small group contains your very best customers, they buy frequently, spend the most, and shop very recently.
    - Treat them as VIPs. Offer premium support, exclusive perks, or personalized messages to retain them.

- **Cluster 3**: Most customers fall into this group. They buy occasionally, spend a little, and have shopped fairly recently.
    - They show potential. Try encouraging repeat purchases through follow-up emails, recommendations, or small incentives.

In [ ]: