# RFM Analysis and Customer Segmentation using KMeans

In the dynamic world of retail, understanding customer behavior is the compass that guides data-driven decision-making. The RFM (Recency, Frequency, Monetary) analysis technique, when paired with the robust KMeans clustering algorithm, opens a door to uncovering hidden patterns and actionable insights within your customer data.

## Introduction

In this notebook, we embark on a journey into the realm of customer segmentation through RFM analysis and KMeans clustering. We'll explore the following key aspects:

1. **RFM Analysis**: We begin by breaking down customer behavior into three essential dimensions:

   - **Recency (R)**: How recently did a customer make a purchase?
   - **Frequency (F)**: How often does a customer make purchases?
   - **Monetary (M)**: How much has a customer spent?

   By quantifying these dimensions, we gain a comprehensive understanding of each customer's engagement and value.

2. **KMeans Clustering**: With the RFM indicators in hand, we harness the power of the KMeans clustering algorithm. KMeans is a versatile tool that groups similar customers together based on their RFM scores. This segmentation unveils distinct customer clusters, each with its own story to tell.

3. **Visualizations**: We go beyond the numbers, using visualizations to illustrate the clusters and the unique characteristics of each segment. Visual representations breathe life into the data, making it easier to grasp and act upon.

The ultimate goal of this analysis is to empower data-driven marketing and business strategies. By segmenting customers based on their behaviors and preferences, we can tailor our approaches to meet their specific needs, drive engagement, and optimize marketing efforts.

This notebook is not just a technical exploration; it's a practical guide to leveraging data for real-world business decisions. Let's dive in and discover the insights waiting to be unlocked in your customer data.

---

**Note**: Before proceeding, make sure you have the necessary libraries installed and your data prepared for analysis. Let's embark on this data-driven journey into customer segmentation!

## Import Libraries

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import datetime
from sklearn.cluster import KMeans
import plotly.express as px
from plotly.offline import init_notebook_mode
import seaborn as sns
```

## Read Dataset

In [ ]:
```
df = pd.read_excel("Online Retail.xlsx")
print('number of dataset rows: ', df.shape[0])
print('number 0f dataset columns: ', df.shape[1])
```

```
number of dataset rows:  541909
number 0f dataset columns:  8
```

## Preview OF Dataset

In [ ]:
```
df.head()
```

Out[ ]:

| | 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 |

# Preprocessing

- drop stock items with letter `C` in their StockCode. This letter indicates that these items were returned to the store and so, it shouldn't be considered in our analysis.
- in this dataset there is records which the amount of `Quantity` or `UnitPrice` is negative. So, these records must be dropped in our dataset.
- duplicates value should be dropped from our dataset.
- there are rows with missing `CustomerID` and as we are doing our analysis based on customers, we should drop these rows.

In [ ]:
```
df = df[~df["StockCode"].str.contains('C', na=False)]
df = df[(df.Quantity>0) & (df.UnitPrice> 0)]
df.drop_duplicates(inplace=True)
print(df.isnull().sum())
df.dropna(inplace=True)
```

```
InvoiceNo           0
StockCode           0
Description         0
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     129925
Country             0
dtype: int64
```

## Adding a column indicating TotalAmount in each order

In [ ]:
```
df["TotalAmount"] = df["UnitPrice"] * df["Quantity"]
```

## Assuming a last day w.r.t to dataset

- last day assumed one day after the most recent InvoiceDate in the dataset. (the last date was 2011-12-9 and the day that this analysis happened which is the variable `today` was 2011-12-10)

In [ ]:
```
first_date = df["InvoiceDate"].min()
last_date = df["InvoiceDate"].max()
print(last_date)
today = last_date.to_pydatetime() + datetime.timedelta(days=1)
```

```
2011-12-09 12:50:00
```

# Creating a dataframe for RFM indicators

With combining and processing columns in the dataset, RFM indicators (Recency, Frequency, Monetary) were created and added to the `rfm_df` which is the dataframe for RFM analysis.

---

## Understanding RFM Indicators

RFM stands for Recency, Frequency, and Monetary Value. These three key indicators are fundamental for analyzing customer behavior in the context of retail and e-commerce businesses.

- **Recency (R)**: This metric measures how recently a customer has made a purchase. It provides insights into customer engagement and loyalty. A lower recency score indicates a recent purchase and suggests an active and engaged customer.
- **Frequency (F)**: Frequency quantifies how often a customer makes purchases. It reveals customer activity and buying patterns. A high frequency score implies a customer who frequently interacts with your business.
- **Monetary (M)**: The monetary indicator assesses the total amount a customer has spent. It quantifies the customer's financial value to the business. A high monetary score represents a valuable customer in terms of revenue.

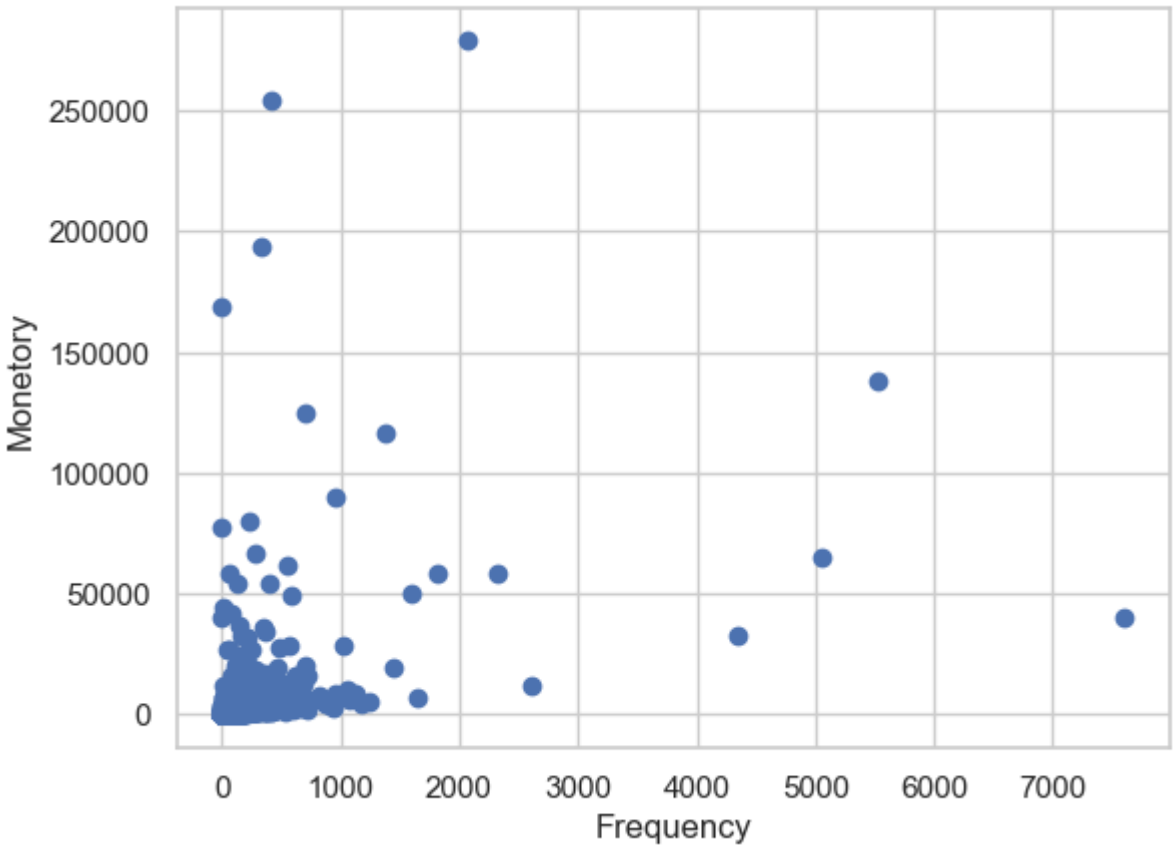
In summary, RFM indicators offer a multi-dimensional view of customer behavior, enabling businesses to segment customers based on their engagement, purchase frequency, and financial contribution. By understanding these indicators, businesses can tailor marketing strategies and campaigns to meet the unique needs of different customer segments.

In [ ]:
```
rfm_df = df.groupby(["CustomerID"]).agg({'InvoiceNo': lambda x: len(x),
                                         'TotalAmount': lambda price: price.sum(),
                                         'InvoiceDate': lambda date: (today - date.dt.to_pydatetime().max()).days})\
                                         .reset_index().sort_values('InvoiceNo')
rfm_df.columns = ["CustomerID", "Frequency", "Monetary", "Recency"]
rfm_df.reset_index(inplace=True)
```

# Scatter Chart

- Scatter chart for two variables `Monetary` and `Frequency` for the dataset customer

In [ ]:
```
%matplotlib inline
plt.scatter(x=rfm_df["Frequency"], y=rfm_df["Monetary"])
plt.xlabel("Frequency")
plt.ylabel("Monetory")
plt.show()
```

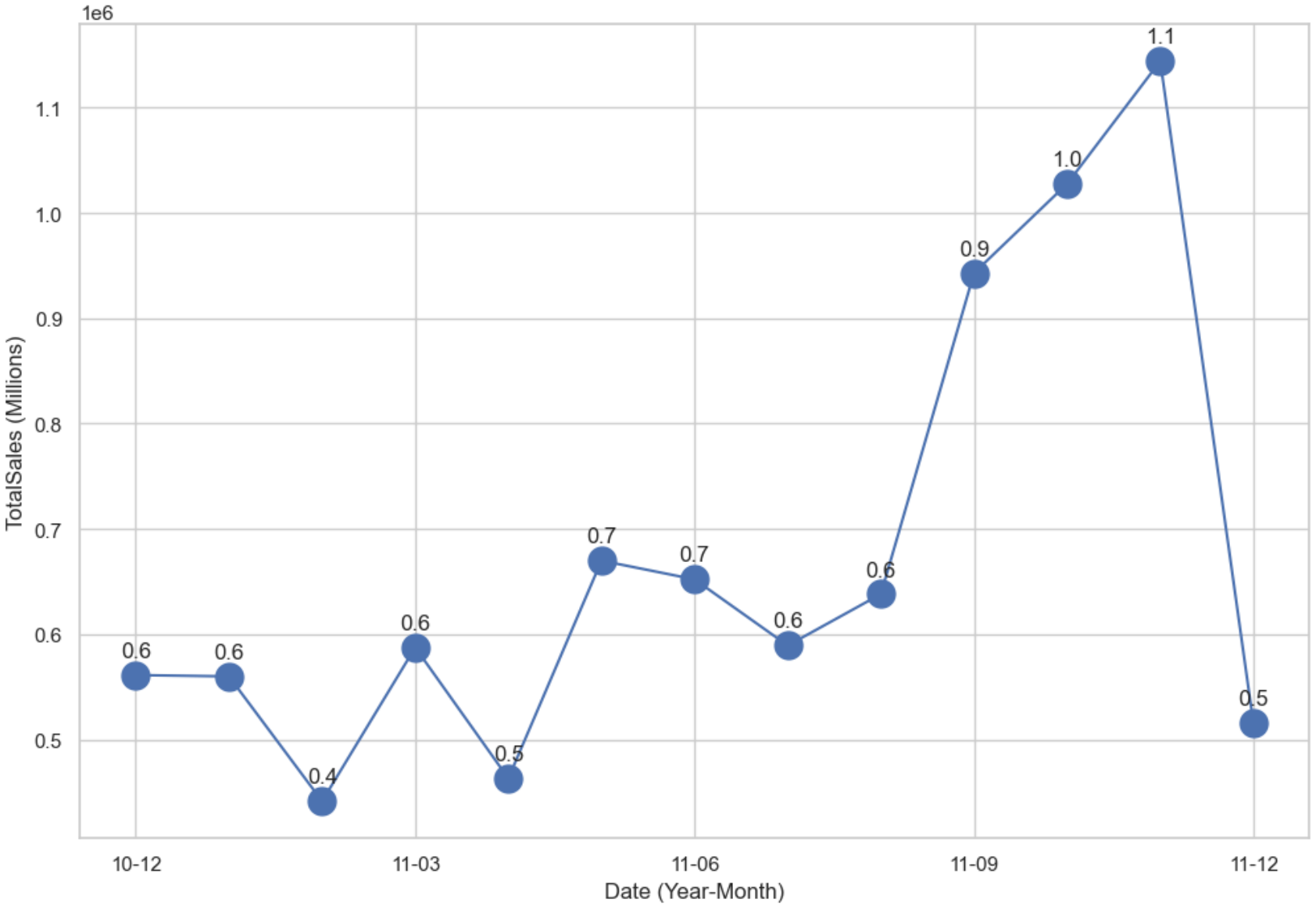



## Analysing trend of sales amount

- sales amounts were groupped by each month in the dataset.
- the sales amount of each month is illustrated in the following chart.

In [ ]:
```
%matplotlib inline
df["TrimmedDate"] = df["InvoiceDate"].dt.strftime('%y-%m')
df_trend = df.groupby('TrimmedDate').agg({'TotalAmount': lambda x: x.sum()}).reset_index().sort_values(by='TrimmedDat
fig = plt.figure(figsize=(12, 8))
plt.plot(df_trend["TrimmedDate"], df_trend["TotalAmount"], marker='o', ms=15)
plt.xlabel('Date (Year-Month)')
plt.ylabel('TotalSales (Millions)')
plt.xticks(['10-12', '11-03', '11-06', '11-09', '11-12'])
```

```
for i, (amount, date) in df_trend.iterrows():
    plt.annotate("{:.1f}".format(date/1e+6), (amount, date), textcoords="offset points", ha='center', xytext=(0,10))
plt.show()
```




## Bucketing RFM indicators

- bucket indicators into 5 points.
- for `F(Frequency)` and `M(Monetary)` we split our data from 1 to 5 but for `R(Recency)` data is transformed from 5 to 1 because the least number is the best number in this indicator.

In [ ]:
```
rfm_df["R"] = pd.qcut(rfm_df["Recency"], 5, labels=[5, 4, 3, 2, 1])
rfm_df["M"] = pd.qcut(rfm_df["Monetary"], 5, labels=[1, 2, 3, 4, 5])
rfm_df["F"] = pd.qcut(rfm_df["Frequency"], 5, labels=[1, 2, 3, 4, 5])
```

## Based on the buckets these segments were introduced

In [ ]:
```
rfm_df['TotalRFM'] = rfm_df['R'].astype('int') + rfm_df['F'].astype('int') + rfm_df['M'].astype('int')

# Define segments based on Total RFM Score
segments = {
    'HighValue_HighEngagement': (15, 15),
    'HighValue_Frequent': (14, 14),
    'HighValue_ModFrequent': (13, 13),
    'HighValue_LowFrequent': (12, 12),
    'ModValue_HighEngagement': (11, 11),
    'ModValue_ModFrequent': (10, 10),
    'ModValue_LowFrequent': (9, 9),
    'LowValue_HighEngagement': (8, 8),
    'LowValue_ModFrequent': (7, 7),
    'LowValue_LowFrequent': (6, 6)
}

# Function to assign segments based on Total RFM Score
def assign_segment(score):
    for segment, (min_score, max_score) in segments.items():
        if min_score <= score <= max_score:
            return segment
    return 'Other'

# Create a new column for the segments
rfm_df['Segment'] = rfm_df['TotalRFM'].apply(assign_segment)
```

In [ ]:
```
rfm_df
```

Out[ ]:

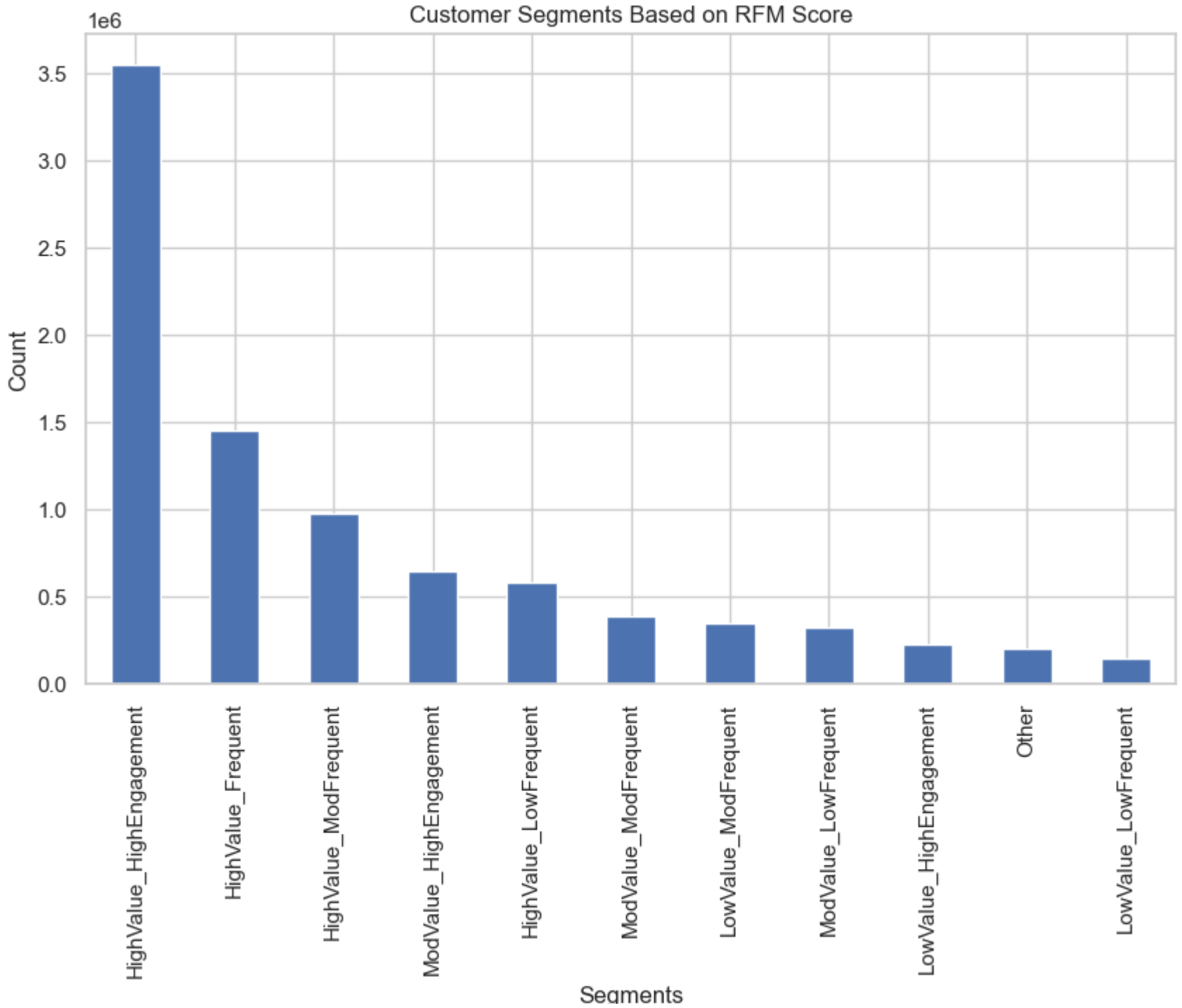
| | index | CustomerID | Frequency | Monetary | Recency | R | M | F | TotalRFM | Segment |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 12346.0 | 1 | 77183.60 | 326 | 1 | 5 | 1 | 7 | LowValue_ModFrequent |
| **1** | 4204 | 18113.0 | 1 | 76.32 | 369 | 1 | 1 | 1 | 3 | Other |
| **2** | 2026 | 15118.0 | 1 | 244.80 | 134 | 2 | 1 | 1 | 4 | Other |
| **3** | 1035 | 13747.0 | 1 | 79.60 | 374 | 1 | 1 | 1 | 3 | Other |
| **4** | 4219 | 18133.0 | 1 | 931.50 | 212 | 1 | 4 | 1 | 6 | LowValue_LowFrequent |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **4328** | 1659 | 14606.0 | 2603 | 11718.65 | 1 | 5 | 5 | 5 | 15 | HighValue_HighEngagement |
| **4329** | 326 | 12748.0 | 4345 | 32509.54 | 1 | 5 | 5 | 5 | 15 | HighValue_HighEngagement |
| **4330** | 1287 | 14096.0 | 5059 | 64824.04 | 4 | 5 | 5 | 5 | 15 | HighValue_HighEngagement |
| **4331** | 1876 | 14911.0 | 5529 | 138460.32 | 1 | 5 | 5 | 5 | 15 | HighValue_HighEngagement |
| **4332** | 4005 | 17841.0 | 7604 | 40177.67 | 2 | 5 | 5 | 5 | 15 | HighValue_HighEngagement |

4333 rows × 10 columns

## Count of customers per segment

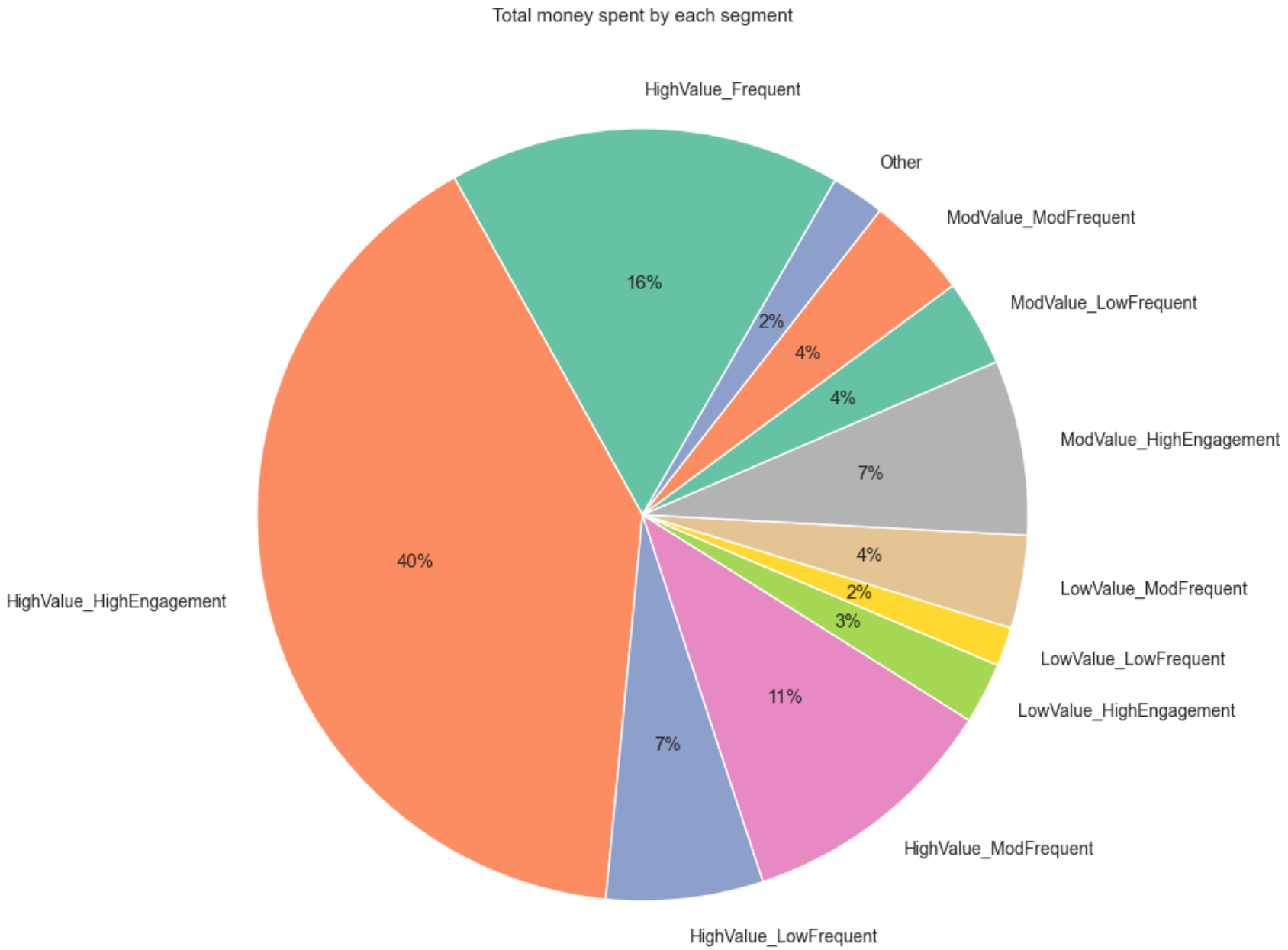
In [ ]:

```
segments_count = rfm_df.groupby("Segment")["Monetary"].sum().sort_values(ascending=False)
plt.figure(figsize=(10, 6))
segments_count.plot(kind='bar')
plt.title('Customer Segments Based on RFM Score')
plt.xlabel('Segments')
plt.ylabel('Count')
# plt.xticks(rotation=45)
plt.show()
```


Customer Segments Based on RFM Score
1e6
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Count
HighValue_HighEngagement
HighValue_Frequent
HighValue_ModFrequent
ModValue_HighEngagement
HighValue_LowFrequent
ModValue_ModFrequent
LowValue_ModFrequent
ModValue_LowFrequent
LowValue_HighEngagement
Other
LowValue_LowFrequent
Segments

## Total money spent by each segment

In [ ]:
```
sns.set(font_scale = .9)
plt.figure(figsize=(10, 10))
ax = plt.pie(rfm_df.groupby('Segment').agg({'Monetary': lambda x: x.sum()}).reset_index().sort_values('Segment')["Mon
plt.title('Total money spent by each segment')
plt.show()
```




In [ ]:
```
rfm_df["RFM"] = rfm_df["R"].astype("str") + rfm_df["F"].astype("str") + rfm_df["M"].astype("str")
```

# Customer segmentation using KMeans with RFM indicators

In [ ]:
```
df_cluster = rfm_df[["CustomerID","R", "F", "M", "Recency", "Frequency", "Monetary", 'TotalRFM']]

kmeans = KMeans(n_clusters=5, random_state=42)
kmeans.fit(df_cluster)


df_cluster["Cluster"] = kmeans.fit_predict(df_cluster)
```

```
f:\Projects\.venv\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning

f:\Projects\.venv\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning

C:\Users\ASUS\AppData\Local\Temp\ipykernel_15388\938301458.py:7: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning
-a-view-versus-a-copy
```

In [ ]:
```
df_cluster.head()
```

Out[ ]:

| | CustomerID | R | F | M | Recency | Frequency | Monetary | TotalRFM | Cluster |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 12346.0 | 1 | 1 | 5 | 326 | 1 | 77183.60 | 7 | 2 |
| **1** | 18113.0 | 1 | 1 | 1 | 369 | 1 | 76.32 | 3 | 0 |
| **2** | 15118.0 | 2 | 1 | 1 | 134 | 1 | 244.80 | 4 | 0 |
| **3** | 13747.0 | 1 | 1 | 1 | 374 | 1 | 79.60 | 3 | 0 |
| **4** | 18133.0 | 1 | 1 | 4 | 212 | 1 | 931.50 | 6 | 0 |

## Saving clusters into csv

In [ ]:
```
df_cluster.to_csv('rfm.csv')
```

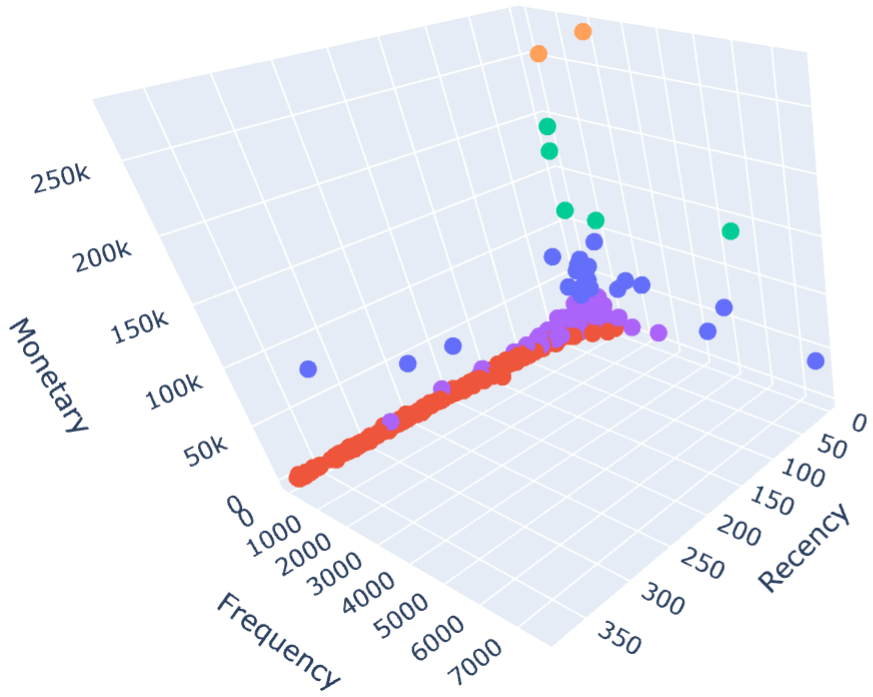
## Labeling each cluster

In [ ]:
```
empty_arr = np.zeros(shape=(df_cluster.shape[0]))
cluster_detail = pd.Series(empty_arr, name='ClusterName')
df_cluster = pd.concat([df_cluster, cluster_detail], axis=1)
for idx, row in df_cluster.iterrows():
    if row[-2] == 1:
        df_cluster.iloc[idx, -1] = 'loyal customers'
    elif row[-2] == 2:
        df_cluster.iloc[idx, -1] = 'potential loyalist'
    elif row[-2] == 3:
        df_cluster.iloc[idx, -1] = 'legendary'
    elif row[-2] == 4:
        df_cluster.iloc[idx, -1] = 'need attention'
    else:
        df_cluster.iloc[idx, -1] ='hibernate'
```

## 3D scatter plot for RFM indicators for each segment

In [ ]:
```
%matplotlib inline
import plotly.io as pio
```

```
init_notebook_mode(connected=True)
pio.renderers.default = "notebook_connected"
fig = px.scatter_3d(df_cluster, x='Recency', y='Frequency', z='Monetary',
                    color='ClusterName')
fig.update_traces(marker_size=5)
fig.update_layout(showlegend=True)
fig.show()
```



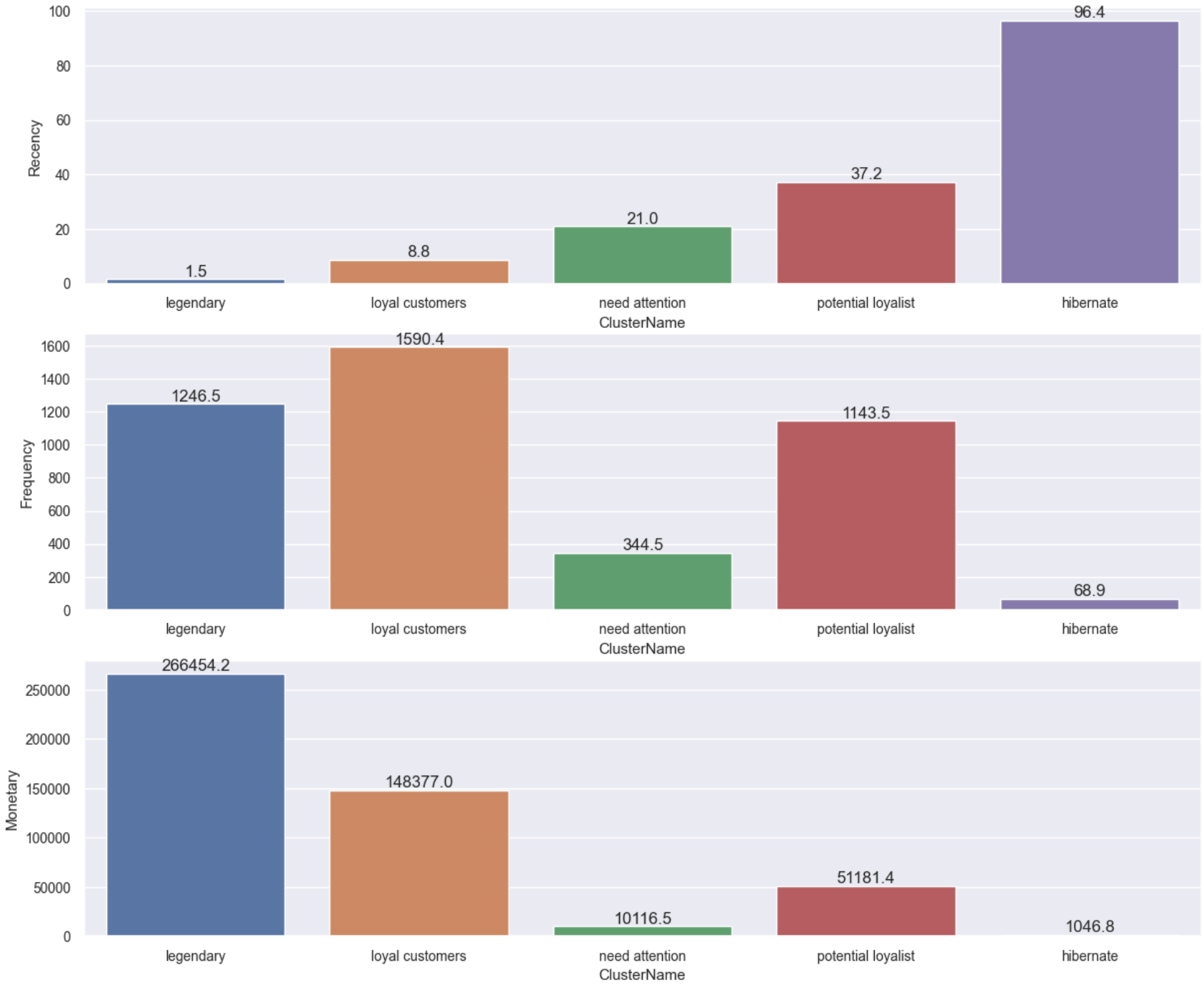
## Dataframe for comparing the average of each indicator per segment

In [ ]:
```
df_analysis = df_cluster.groupby('ClusterName').agg({'Recency': lambda r: r.mean(),
                                                     'Frequency': lambda f: f.mean(),
                                                     'Monetary': lambda m: m.mean()}).sort_values(by='Recency').reset_index()
```

In [ ]:
```
figure, axes = plt.subplots(3, 1, figsize=(12,10), )
sns.set_theme(style='whitegrid')

col_list = ['Recency', 'Frequency', 'Monetary']
plt.tight_layout()
for i, ax in enumerate(axes):
    axis = sns.barplot(data=df_analysis, x='ClusterName', y=col_list[i], ax=axes[i])
    axis.bar_label(axis.containers[0], fmt='%.1f')

plt.show()
```


Recency
100
80
60
40
20
0
1.5
8.8
21.0
37.2
96.4
legendary
loyal customers
need attention
potential loyalist
hibernate
ClusterName
Frequency
1600
1400
1200
1000
800
600
400
200
0
1246.5
1590.4
344.5
1143.5
68.9
legendary
loyal customers
need attention
potential loyalist
hibernate
ClusterName
Monetary
250000
200000
150000
100000
50000
0
266454.2
148377.0
10116.5
51181.4
1046.8
legendary
loyal customers
need attention
potential loyalist
hibernate
ClusterName

# RFM Customer Segmentation Analysis

## Introduction

RFM (Recency, Frequency, Monetary) analysis was conducted on customer transaction data to identify and profile key customer segments. Customers were scored on three metrics - how recently they made a purchase, their purchase frequency, and the total money they spent. These RFM scores were then used to divide customers into five distinct segments via clustering algorithms.

The analysis revealed five customer segments - **legendary**, **loyal**, **potential loyalists**, **need attention**, and **hibernate** customers. Below is a comparison of these segments and recommendations for each.

## Customer Segment Comparisons

### Legendary Customers

- Have the highest recency, frequency, and monetary scores
- Made a purchase recently, purchase very frequently (1246 transactions on average), and spend the most ($266k on average)
- Most valuable customer segment
- Recommendations: Provide VIP services, loyalty rewards, special offers, and personalized engagement

### Loyal Customers

- Have moderately high recency, frequency, and monetary scores
- Made a purchase semi-recently, purchase frequently (1590 transactions), and have high spend ($148k on average)
- Very valuable but just below legendary customers
- Recommendations: Thank them for their loyalty, provide incentives to upgrade to legendary status

### Potential Loyalists

- Moderate recency, high frequency, and decent average spend

- Haven't purchased recently but when they do, they purchase often (1143 transactions) and spend a good amount ($51k on average)
- Show potential to become loyal customers
- Recommendations: Identify reasons for lapses, re-engage with personalized offers, incentivize purchasing again

### Need Attention

- Relatively low recency, frequency, and monetary scores
- Don't purchase often (345 transactions) and have low spend overall ($10k on average)
- At risk of churning, need attention to reactivate and create loyalty
- Recommendations: Survey why purchasing dropped, provide VIP treatment, target with win-back offers

### Hibernate Customers

- Lowest recency, frequency, and spend by far
- Haven't purchased in a long time (recency of 96), rarely purchase (69 transactions), and have very low spend ($1k on average)
- Mostly dormant/lost customers
- Recommendations: Reactivate with specialized campaigns if worth the effort, otherwise focus on other segments

## Conclusion

In summary, this analysis distinguishes the high value customers to prioritize, the medium value customers to nurture and grow, and the low value customers to re-engage or not focus on. These insights empower an organization to implement targeted strategies per customer segment and optimize customer lifetime value.