crm-analysis-2

October 8, 2024

Name: Shagun Wadhwa

Business Case: CRM analysis CRM analysis is focused on understanding customer behavior through data. This project looks into the dataset to identify patterns and help segment customers based on behavior using RFM (Recency, Frequency, Monetary) scores. The aim is to help businesses make informed decisions that improve customer satisfaction and sales.

What is CRM ? Customer relationship management (CRM) is a tool that allows us to see the relationship between our customers and our company. According to the author the four core objectives of CRM's are;

1.Boost Customer Satisfaction 2.Improve The Efficiency Of Your Business 3.Gain New Customers 4.Strengthen Your Sales And Support Teams

RFM And CLTV As I mentioned before, today we will discuss two CRM analysis concept which are RFM(Recency — Frequency — Monetary) and CLTV(Customer Life Time Value).

RFM RFM is an analysis that allows us to segment our customers into specific segments based on their behavior.

Recency: Time since last purchase.

Frequency: Total repeat purchases.

Monetary: Average earnings per purchase.

Variable Description InvoiceNo: Invoice number that consists 6 digits. If this code starts with letter 'c', it indicates a cancellation. StockCode: Product code that consists 5 digits. Description: Product name. Quantity: The quantities of each product per transaction. InvoiceDate: This represents the day and time when each transaction was generated. UnitPrice: Product price per unit. CustomerID: Customer number that consists 5 digits. Each customer has a unique customer ID. Country: Name of the country where each customer resides.

[374]:
```
#Necessary imports
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter('ignore')
```

[375]:
```
# Load dataset
df_crm=pd.read_csv("Ecom_CRM_analysis.ipynb.csv",encoding="ISO-8859-1")
df_crm
```

[375]:
```
       InvoiceNo StockCode                          Description  Quantity  \
0         536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1         536365     71053                  WHITE METAL LANTERN         6
2         536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3         536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4         536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6
…            …         …                                  …       …
541904    581587     22613          PACK OF 20 SPACEBOY NAPKINS        12
541905    581587     22899         CHILDREN'S APRON DOLLY GIRL          6
541906    581587     23254        CHILDRENS CUTLERY DOLLY GIRL          4
541907    581587     23255      CHILDRENS CUTLERY CIRCUS PARADE         4
541908    581587     22138        BAKING SET 9 PIECE RETROSPOT          3

            InvoiceDate  UnitPrice  CustomerID         Country
0        12/1/2010 8:26       2.55     17850.0  United Kingdom
1        12/1/2010 8:26       3.39     17850.0  United Kingdom
2        12/1/2010 8:26       2.75     17850.0  United Kingdom
3        12/1/2010 8:26       3.39     17850.0  United Kingdom
4        12/1/2010 8:26       3.39     17850.0  United Kingdom
…                   …          …           …             …
541904  12/9/2011 12:50       0.85     12680.0          France
541905  12/9/2011 12:50       2.10     12680.0          France
541906  12/9/2011 12:50       4.15     12680.0          France
541907  12/9/2011 12:50       4.15     12680.0          France
541908  12/9/2011 12:50       4.95     12680.0          France

[541909 rows x 8 columns]
```

[376]:
```
# Check the first few rows of the dataset
df_crm.head()
```

[376]:
```
  InvoiceNo StockCode                          Description  Quantity  \
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                  WHITE METAL LANTERN         6
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6

      InvoiceDate  UnitPrice  CustomerID         Country
0  12/1/2010 8:26       2.55     17850.0  United Kingdom
1  12/1/2010 8:26       3.39     17850.0  United Kingdom
2  12/1/2010 8:26       2.75     17850.0  United Kingdom
3  12/1/2010 8:26       3.39     17850.0  United Kingdom
```

```
4  12/1/2010 8:26       3.39     17850.0  United Kingdom
```

Basic Information about the Dataset

[377]:
```
# Summary of the dataset (columns, data types, missing values)
df_crm.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    541909 non-null  object
 1   StockCode    541909 non-null  object
 2   Description  540455 non-null  object
 3   Quantity     541909 non-null  int64
 4   InvoiceDate  541909 non-null  object
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB
```

[378]:
```
#changing the datatype of InvoiceDate column
df_crm['InvoiceDate'] = pd.to_datetime(df_crm['InvoiceDate'], errors='coerce')
```

[379]:
```
df_crm.shape
```

[379]:
```
(541909, 8)
```

[380]:
```
# Basic statistical summary of the numerical features
df_crm.describe()
```

[380]:
```
            Quantity                    InvoiceDate      UnitPrice  \
count  541909.000000                         541909  541909.000000
mean        9.552250  2011-07-04 13:34:57.156386048       4.611114
min    -80995.000000            2010-12-01 08:26:00  -11062.060000
25%         1.000000            2011-03-28 11:34:00       1.250000
50%         3.000000            2011-07-19 17:17:00       2.080000
75%        10.000000            2011-10-19 11:27:00       4.130000
max     80995.000000            2011-12-09 12:50:00   38970.000000
std       218.081158                            NaN      96.759853

          CustomerID
count  406829.000000
mean    15287.690570
min     12346.000000
25%     13953.000000
```

```
50%      15152.000000
75%      16791.000000
max      18287.000000
std       1713.600303
```

What are the columns present in the dataset?

[381]:
```
column=df_crm.columns
column
```

[381]:
```
Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'UnitPrice', 'CustomerID', 'Country'],
      dtype='object')
```

What is the datatype of the columns ?

[382]:
```
df_crm.dtypes
```

[382]:
```
InvoiceNo              object
StockCode              object
Description            object
Quantity                int64
InvoiceDate    datetime64[ns]
UnitPrice             float64
CustomerID            float64
Country                object
dtype: object
```

How many unique entries present in each column ?

[383]:
```
for i in df_crm.columns:
    print(f"Unique entries for column {i:<30} = {df_crm[i].nunique()}")
```

```
Unique entries for column InvoiceNo                      = 25900
Unique entries for column StockCode                      = 4070
Unique entries for column Description                    = 4223
Unique entries for column Quantity                       = 722
Unique entries for column InvoiceDate                    = 23260
Unique entries for column UnitPrice                      = 1630
Unique entries for column CustomerID                     = 4372
Unique entries for column Country                        = 38
```

Data Preperation: Are there any missing observations in the dataset? If yes, how many missing observations in each variable?

[384]:
```
# Handling missing values (if any)
df_crm.isnull().sum()
```

[384]:

```
InvoiceNo           0
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135080
Country             0
dtype: int64
```

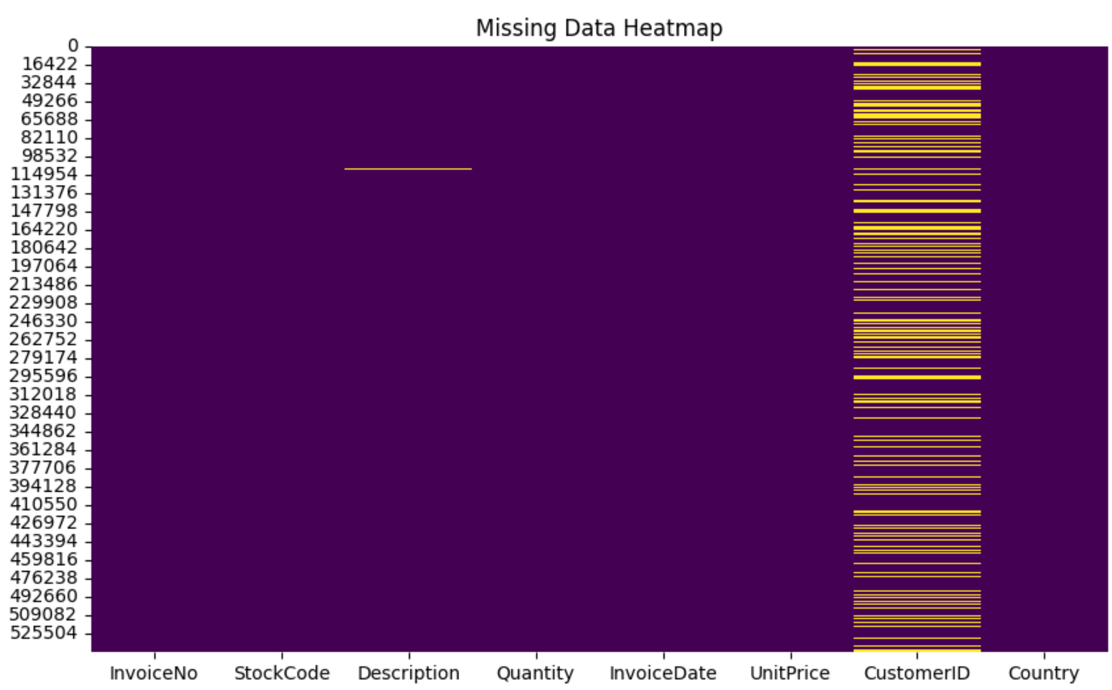
[385]:

```python
# Check the percentage of missing values in CustomerID
missing_percentage = df_crm['CustomerID'].isnull().mean() * 100
print(f"Percentage of missing CustomerID: {missing_percentage:.2f}%")
```

```
Percentage of missing CustomerID: 24.93%
```

[386]:

```python
# Visualize missing data using a heatmap
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10,6))
sns.heatmap(df_crm.isnull(), cbar=False, cmap='viridis')
plt.title('Missing Data Heatmap')
plt.show()
```



[387]:
```
#Impute missing CustomerID based on InvoiceNo
df_crm['CustomerID'] = df_crm ['CustomerID'].fillna("UnknownCustomerID")
```

[388]:
```
missing_after_imputation = df_crm['CustomerID'].isnull().mean() * 100
print(f"Percentage of missing CustomerID after imputation:␣
↪{missing_after_imputation:.2f}%")
```

```
Percentage of missing CustomerID after imputation: 0.00%
```

[389]:
```
# Impute missing descriptions with 'Unknown' where Description is missing for␣
↪the same StockCode
df_crm[df_crm["UnitPrice"] == 0]["Description"].fillna("Free item",␣
↪inplace=True)
df_crm['Description_imputed'] = df_crm.groupby('StockCode')['Description'].
↪transform(lambda x: x.fillna('Unknown'))

# Check if there are still any missing values in the imputed column
print(df_crm['Description_imputed'].isnull().sum())
```

```
0
```

Total quantities of products

[390]:
```
df_crm['Description'].value_counts().head(10)
```

[390]:
```
Description
WHITE HANGING HEART T-LIGHT HOLDER    2369
REGENCY CAKESTAND 3 TIER              2200
JUMBO BAG RED RETROSPOT               2159
PARTY BUNTING                         1727
LUNCH BAG RED RETROSPOT               1638
ASSORTED COLOUR BIRD ORNAMENT         1501
SET OF 3 CAKE TINS PANTRY DESIGN      1473
PACK OF 72 RETROSPOT CAKE CASES       1385
LUNCH BAG  BLACK SKULL.               1350
NATURAL SLATE HEART CHALKBOARD        1280
Name: count, dtype: int64
```

Sorting the most ordered products from most to least

[391]:
```
df_crm.groupby('Description').agg({'Quantity':'sum'}).sort_values('Quantity',␣
↪ascending=False)
```

[391]:
```
                                   Quantity
Description
WORLD WAR 2 GLIDERS ASSTD DESIGNS     53847
JUMBO BAG RED RETROSPOT               47363
ASSORTED COLOUR BIRD ORNAMENT         36381
```

```
POPCORN HOLDER                           36334
PACK OF 72 RETROSPOT CAKE CASES          36039
…                                          …
Damaged                                  -7540
Printing smudges/thrown away             -9058
check                                   -12030
Unsaleable, destroyed.                  -15644
printing smudges/thrown away            -19200

[4223 rows x 1 columns]
```

'C' in invoices indicates canceled transactions. Let's remove the canceled transactions from the dataset.

[392]:
```
df_crm = df_crm[~df_crm['InvoiceNo'].apply(str).str.contains('C', na=False)]
```

[393]:
```
#Let's create a variable called 'TotalPrice' that represents the total earnings␣
↪per invoice.
df_crm["TotalPrice"] = df_crm["Quantity"] * df_crm["UnitPrice"]
```

[394]:
```
df_crm.head()
```

[394]:
```
  InvoiceNo StockCode                          Description  Quantity  \
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                  WHITE METAL LANTERN         6
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6

          InvoiceDate  UnitPrice  CustomerID         Country  \
0 2010-12-01 08:26:00       2.55     17850.0  United Kingdom
1 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
2 2010-12-01 08:26:00       2.75     17850.0  United Kingdom
3 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
4 2010-12-01 08:26:00       3.39     17850.0  United Kingdom

                  Description_imputed  TotalPrice
0   WHITE HANGING HEART T-LIGHT HOLDER       15.30
1                 WHITE METAL LANTERN        20.34
2       CREAM CUPID HEARTS COAT HANGER       22.00
3  KNITTED UNION FLAG HOT WATER BOTTLE       20.34
4       RED WOOLLY HOTTIE WHITE HEART.       20.34
```

[395]:
```
# Descriptive statistics for important columns (e.g., 'Quantity', 'UnitPrice',␣
↪'TotalPrice')
print(df_crm[['Quantity', 'UnitPrice', 'TotalPrice']].describe())
```

```
            Quantity      UnitPrice     TotalPrice
```

```
count   532621.000000  532621.000000  532621.000000
mean        10.239972       3.847621      19.985244
std        159.593551      41.758023     270.574241
min      -9600.000000  -11062.060000  -11062.060000
25%          1.000000       1.250000       3.750000
50%          3.000000       2.080000       9.900000
75%         10.000000       4.130000      17.700000
max      80995.000000   13541.330000  168469.600000
```

[396]:

```
# Check for duplicate entries
duplicate_rows = df_crm[df_crm.duplicated()]
print(f"Number of duplicate rows: {duplicate_rows.shape[0]}")
```

```
Number of duplicate rows: 5231
```

[397]:

```
#Remove Duplicates:
df_crm = df_crm.drop_duplicates()
```

[398]:

```
# Function to detect outliers based on IQR
def detect_outliers_iqr(df_crm,column):
    Q1 = df_crm[column].quantile(0.25)
    Q3 = df_crm[column].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Outliers condition
    outliers = df_crm[(df_crm[column] < lower_bound) | (df_crm[column] >␣
 ↪upper_bound)]
    return outliers
```

[399]:

```
# Extract month and year from InvoiceDate
df_crm['YearMonth'] = df_crm['InvoiceDate'].dt.to_period('M')
df_crm
```

[399]:

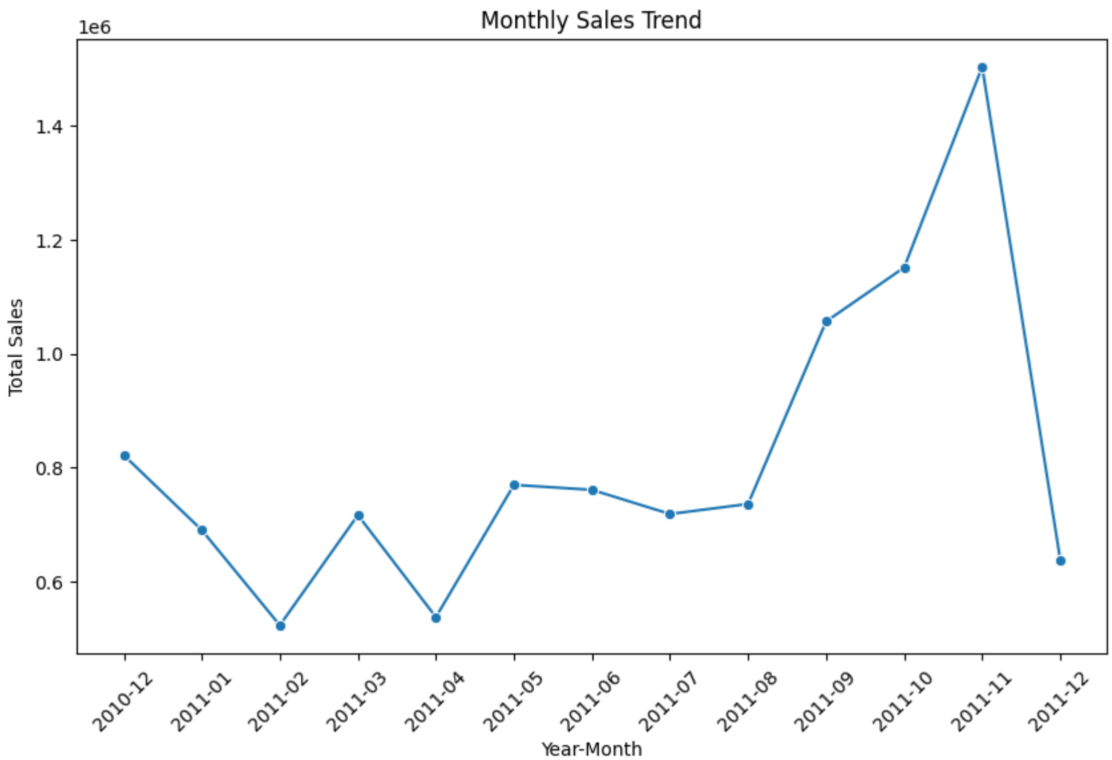
```
        InvoiceNo StockCode                          Description  Quantity  \
0          536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1          536365     71053                  WHITE METAL LANTERN         6
2          536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3          536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4          536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6
…             …         …                                   …       …
541904     581587     22613          PACK OF 20 SPACEBOY NAPKINS        12
541905     581587     22899         CHILDREN'S APRON DOLLY GIRL          6
541906     581587     23254        CHILDRENS CUTLERY DOLLY GIRL          4
541907     581587     23255      CHILDRENS CUTLERY CIRCUS PARADE         4
```

```
541908    581587     22138        BAKING SET 9 PIECE RETROSPOT          3

                InvoiceDate  UnitPrice CustomerID         Country  \
0       2010-12-01 08:26:00       2.55    17850.0  United Kingdom
1       2010-12-01 08:26:00       3.39    17850.0  United Kingdom
2       2010-12-01 08:26:00       2.75    17850.0  United Kingdom
3       2010-12-01 08:26:00       3.39    17850.0  United Kingdom
4       2010-12-01 08:26:00       3.39    17850.0  United Kingdom
…                       …          …        …             …
541904  2011-12-09 12:50:00       0.85    12680.0          France
541905  2011-12-09 12:50:00       2.10    12680.0          France
541906  2011-12-09 12:50:00       4.15    12680.0          France
541907  2011-12-09 12:50:00       4.15    12680.0          France
541908  2011-12-09 12:50:00       4.95    12680.0          France

                       Description_imputed  TotalPrice YearMonth
0        WHITE HANGING HEART T-LIGHT HOLDER       15.30   2010-12
1                       WHITE METAL LANTERN       20.34   2010-12
2            CREAM CUPID HEARTS COAT HANGER       22.00   2010-12
3       KNITTED UNION FLAG HOT WATER BOTTLE       20.34   2010-12
4            RED WOOLLY HOTTIE WHITE HEART.       20.34   2010-12
…                                      …          …       …
541904          PACK OF 20 SPACEBOY NAPKINS       10.20   2011-12
541905         CHILDREN'S APRON DOLLY GIRL        12.60   2011-12
541906        CHILDRENS CUTLERY DOLLY GIRL        16.60   2011-12
541907      CHILDRENS CUTLERY CIRCUS PARADE       16.60   2011-12
541908        BAKING SET 9 PIECE RETROSPOT        14.85   2011-12

[527390 rows x 11 columns]
```

[400]:

```python
df_crm['InvoiceDate'] = pd.to_datetime(df_crm['InvoiceDate'])

# Create a YearMonth column for grouping
df_crm['YearMonth'] = df_crm['InvoiceDate'].dt.to_period('M')

# Calculate monthly sales
monthly_sales = df_crm.groupby('YearMonth')['TotalPrice'].sum().reset_index()

# Convert the YearMonth period to string for plotting
monthly_sales['YearMonth'] = monthly_sales['YearMonth'].astype(str)

# Plot the Monthly Sales Trend
plt.figure(figsize=(10,6))
sns.lineplot(data=monthly_sales, x='YearMonth', y='TotalPrice', marker='o')
plt.title('Monthly Sales Trend')
plt.xticks(rotation=45)
plt.xlabel('Year-Month')
```

```
plt.ylabel('Total Sales')
plt.show()
```

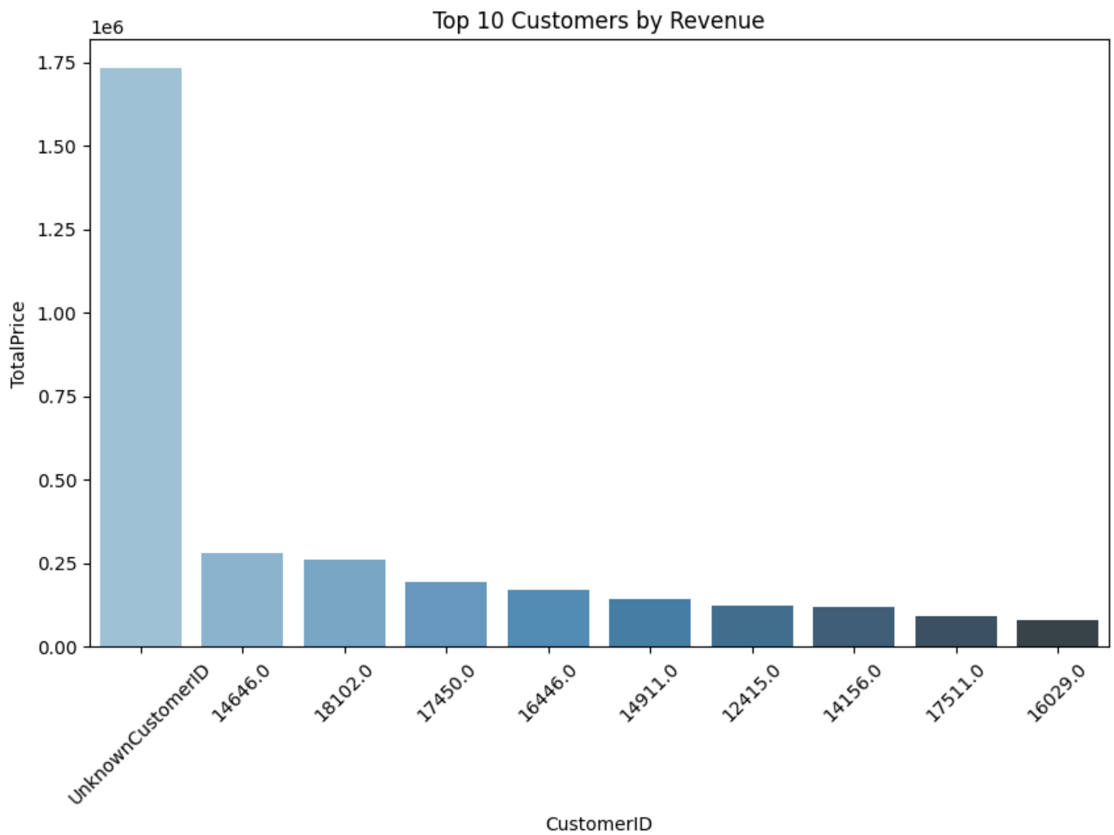


Customer Segmentation Insights

[401]:
```
# Total revenue by customer
customer_revenue = df_crm.groupby('CustomerID')['TotalPrice'].sum().
↪reset_index()
print(customer_revenue.head())

# Plotting the top 10 customers by revenuek
top_10_customers = customer_revenue.sort_values(by='TotalPrice',␣
↪ascending=False).head(10)

plt.figure(figsize=(10,6))
sns.barplot(data=top_10_customers, x='CustomerID', y='TotalPrice',␣
↪palette='Blues_d')
plt.title('Top 10 Customers by Revenue')
plt.xticks(rotation=45)
plt.show()
```

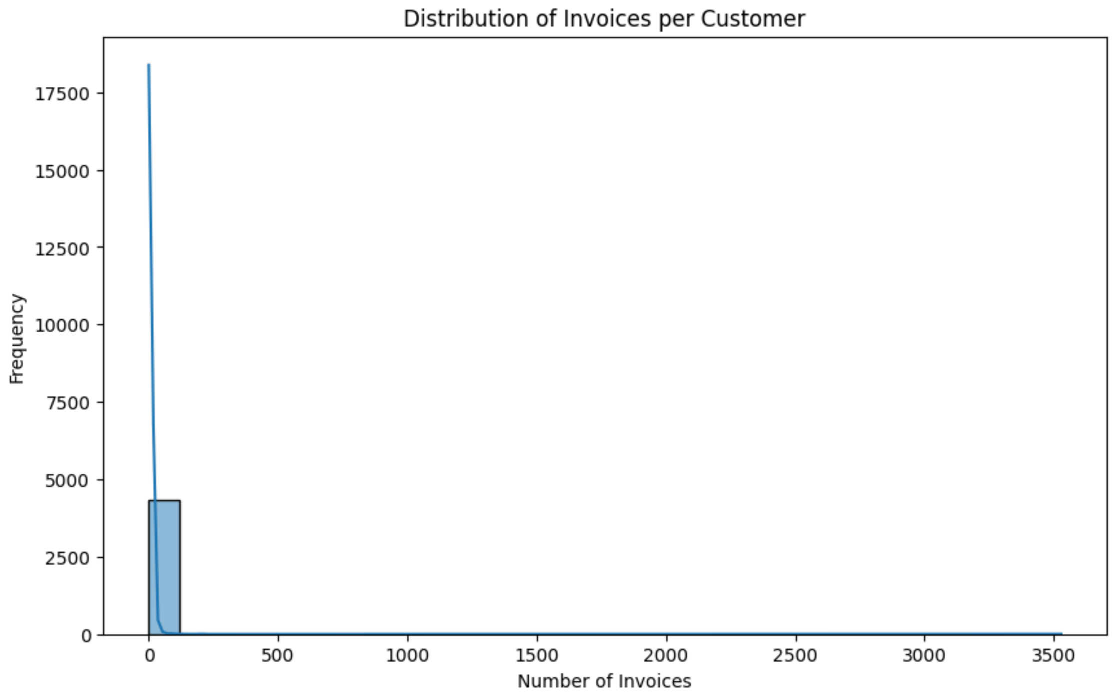
```
  CustomerID  TotalPrice
0    12346.0    77183.60
```

```
1     12347.0     4310.00
2     12348.0     1797.24
3     12349.0     1757.55
4     12350.0      334.40
```



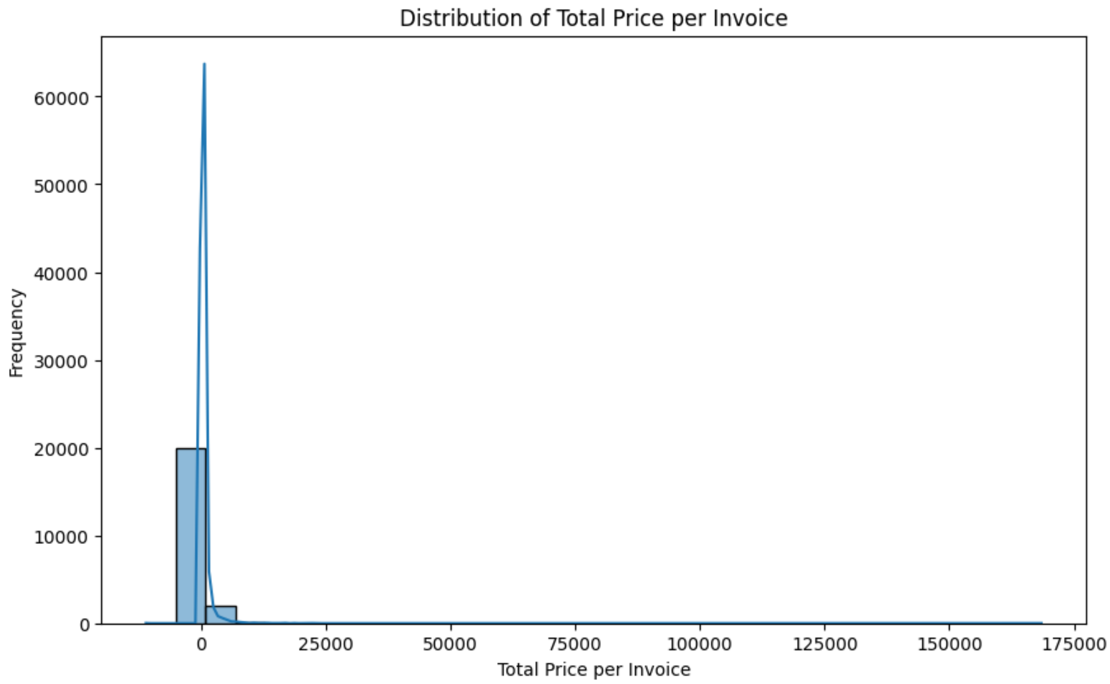
Distribution of Invoices per Customer This will help you understand how many invoices each customer has.

[402]:
```
plt.figure(figsize=(10,6))
sns.histplot(df_crm.groupby('CustomerID')['InvoiceNo'].nunique(), bins=30,␣
↪kde=True)
plt.title('Distribution of Invoices per Customer')
plt.xlabel('Number of Invoices')
plt.ylabel('Frequency')
plt.show()
```



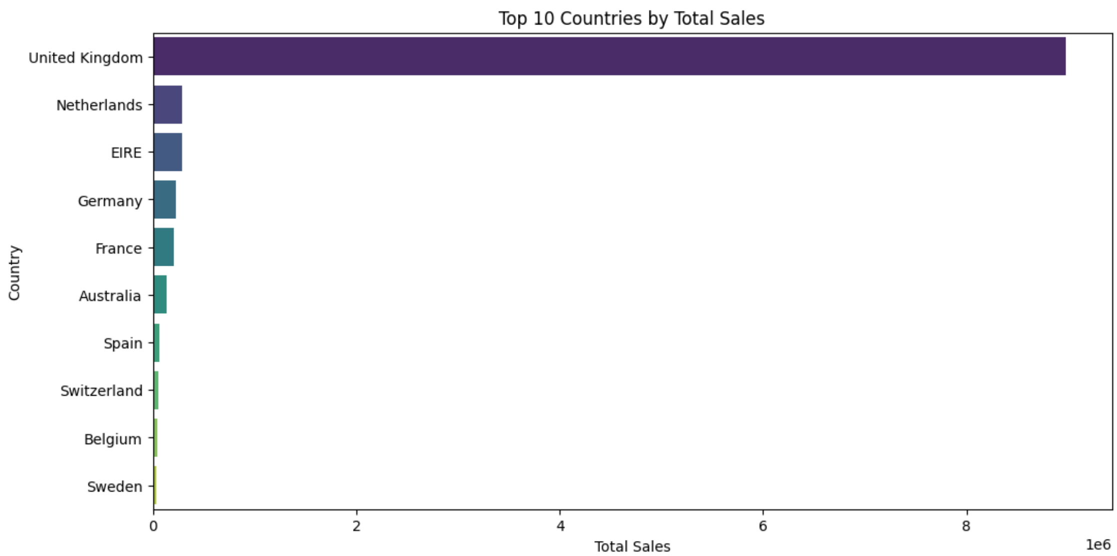
Distribution of Total Price per Invoice This plot shows the variation in total spending across different invoices.

[403]:
```
plt.figure(figsize=(10,6))
sns.histplot(df_crm.groupby('InvoiceNo')['TotalPrice'].sum(), bins=30, kde=True)
plt.title('Distribution of Total Price per Invoice')
plt.xlabel('Total Price per Invoice')
plt.ylabel('Frequency')
plt.show()
```



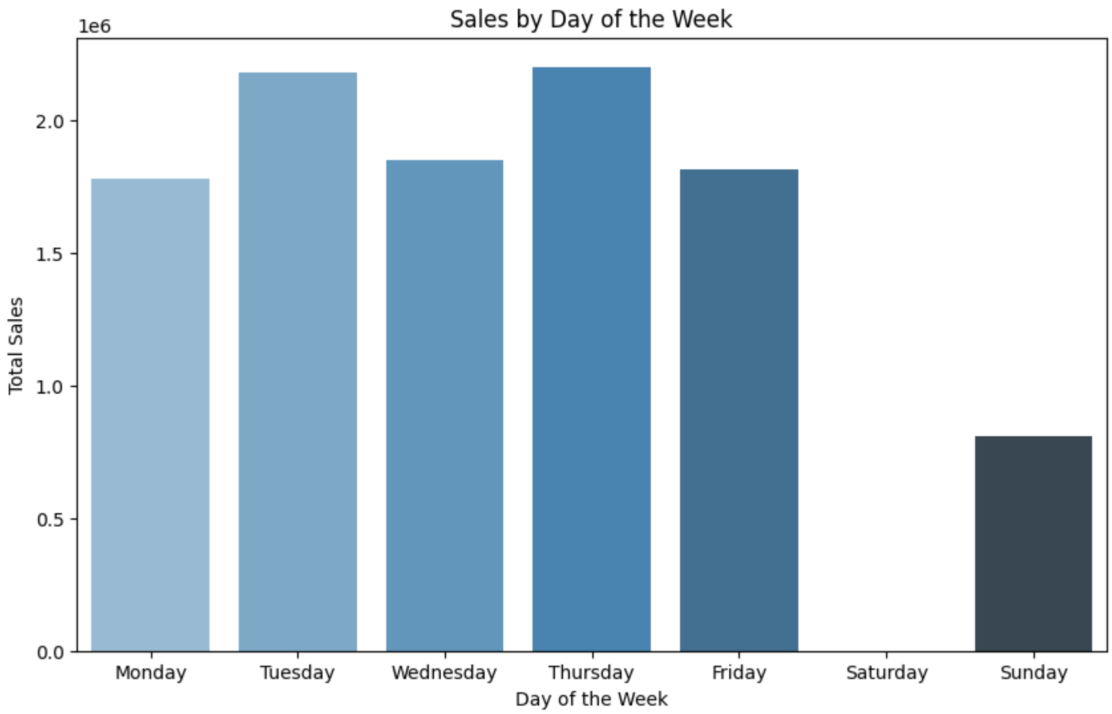
Total Sales by Country

[404]:
```
country_sales = df_crm.groupby('Country')['TotalPrice'].sum().reset_index().
    ↪sort_values(by='TotalPrice', ascending=False)
plt.figure(figsize=(12,6))
sns.barplot(data=country_sales.head(10), x='TotalPrice', y='Country',␣
    ↪palette='viridis')
plt.title('Top 10 Countries by Total Sales')
plt.xlabel('Total Sales')
plt.ylabel('Country')
plt.show()
```



Sales by Day of the Week

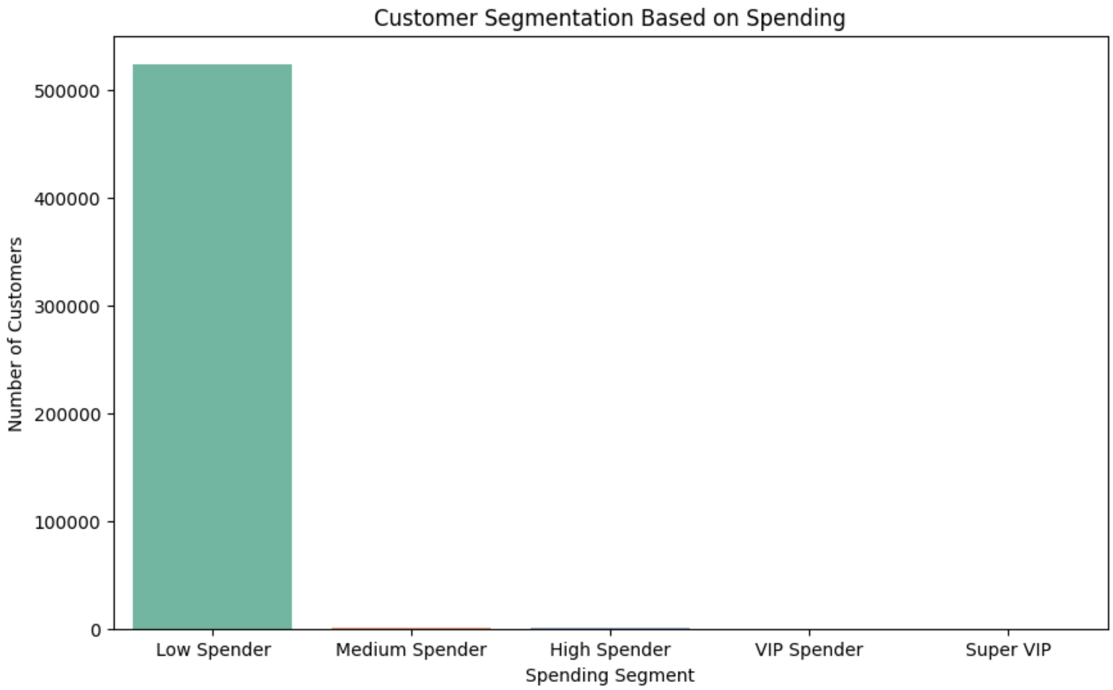
[405]:
```
# Extract day of the week
df_crm['DayOfWeek'] = df_crm['InvoiceDate'].dt.day_name()

sales_by_day = df_crm.groupby('DayOfWeek')['TotalPrice'].sum().reindex([
    'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'
])

plt.figure(figsize=(10,6))
sns.barplot(x=sales_by_day.index, y=sales_by_day.values, palette='Blues_d')
plt.title('Sales by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Total Sales')
plt.show()
```



Customer Segmentation by Total Spending

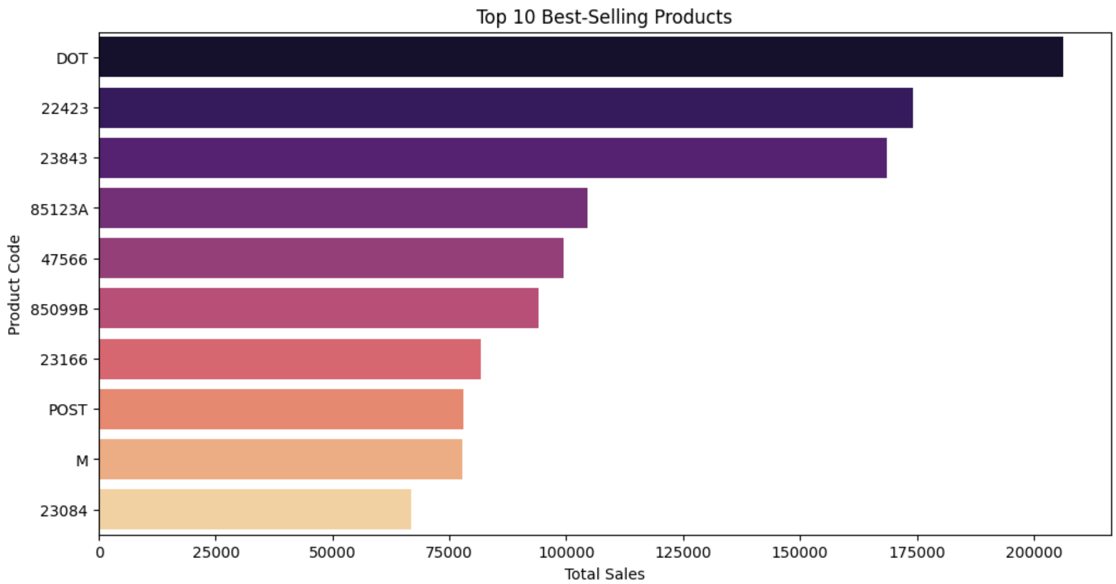
You can group customers into different spending tiers to understand who your top, medium, and low spenders are.

[406]:
```
# Segment customers based on their total spending
spending_bins = [0, 500, 1000, 5000, 10000, df_crm['TotalPrice'].max()]
spending_labels = ['Low Spender', 'Medium Spender', 'High Spender', 'VIP␣
↪Spender', 'Super VIP']

df_crm['Spending_Segment'] = pd.cut(df_crm['TotalPrice'], bins=spending_bins,␣
↪labels=spending_labels)

# Countplot to visualize customer segments
plt.figure(figsize=(10,6))
sns.countplot(data=df_crm, x='Spending_Segment', palette='Set2')
plt.title('Customer Segmentation Based on Spending')
plt.xlabel('Spending Segment')
plt.ylabel('Number of Customers')
plt.show()
```



Top 10 Best-Selling Products Identifying your top-selling products can help focus your efforts on high-demand items.

[407]:
```
top_products = df_crm.groupby('StockCode')['TotalPrice'].sum().
 ↪sort_values(ascending=False).head(10)
plt.figure(figsize=(12,6))
sns.barplot(x=top_products.values, y=top_products.index, palette='magma')
plt.title('Top 10 Best-Selling Products')
plt.xlabel('Total Sales')
plt.ylabel('Product Code')
plt.show()
```

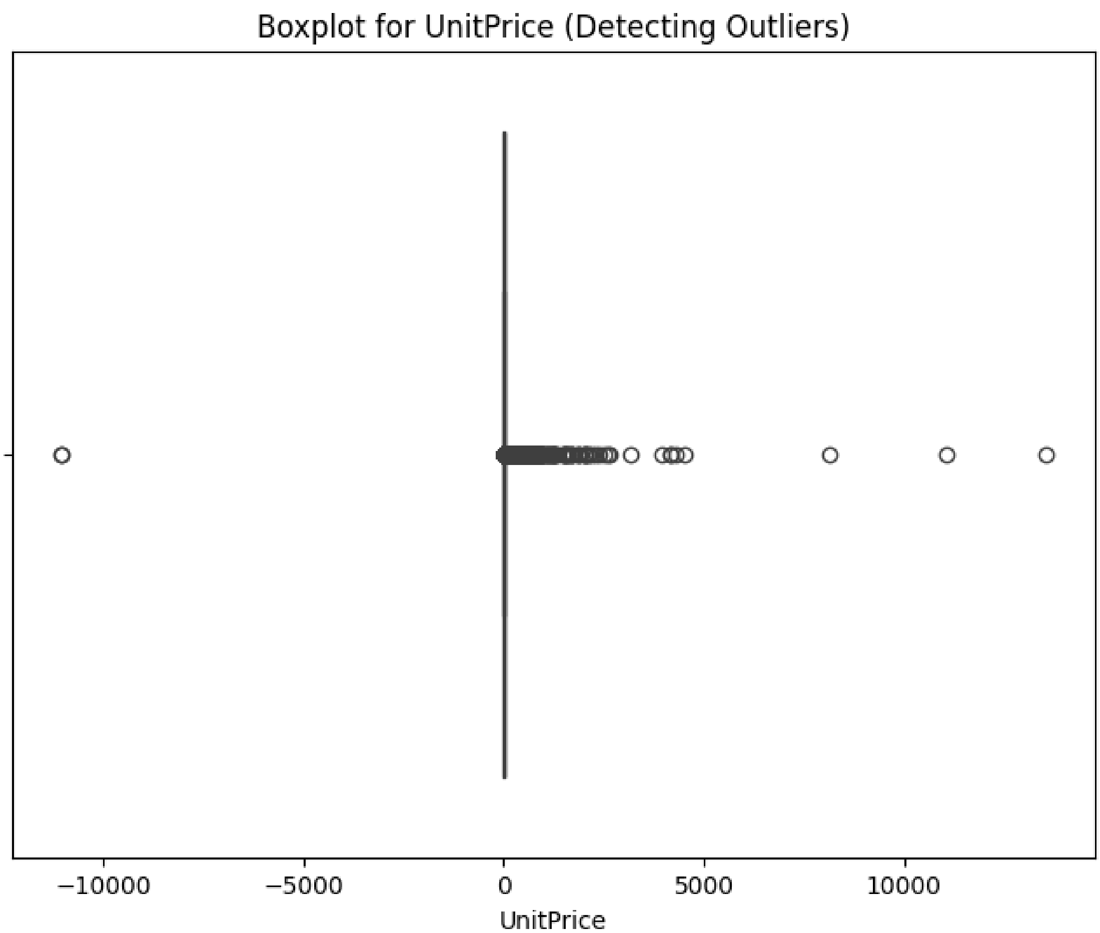


[408]:
```
outliers = detect_outliers_iqr(df_crm, 'UnitPrice')
print(f'Outliers detected: {len(outliers)}')
```

```
Outliers detected: 37829
```

Visualizing Outliers with Boxplot:

[409]:
```
# Boxplot to visualize outliers
plt.figure(figsize=(8, 6))
sns.boxplot(x=df_crm['UnitPrice'])
plt.title('Boxplot for UnitPrice (Detecting Outliers)')
plt.show()
```



[410]:
```
outliers = detect_outliers_iqr(df_crm, 'TotalPrice')
print(f'Outliers detected: {len(outliers)}')
```

```
Outliers detected: 42623
```

[411]:
```
#Impute Outliers
def impute_outliers(df_crm, column):
    Q1 = df_crm[column].quantile(0.25)
    Q3 = df_crm[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Impute outliers with median value
    median = df_crm[column].median()
    df_crm[column] = df_crm[column].apply(lambda x: median if x < lower_bound␣
 ↪or x > upper_bound else x)
    return df_crm
```

```python
# Impute outliers in 'UnitPrice'
df_crm_imputed = impute_outliers(df_crm, 'UnitPrice')
```

RFM Calculation RFM Analysis

RFM stands for Recency, Frequency, and Monetary value, each corresponding to some key customer trait. These RFM metrics are important indicators of a customer's behavior because frequency and monetary value affects a customer's lifetime value, and recency affects retention, a measure of engagement.

RFM factors illustrate these facts:

The more recent the purchase, the more responsive the customer is to promotions The more frequently the customer buys, the more engaged and satisfied they are Monetary value differentiates heavy spenders from low-value purchasers RFM Metrics

[412]:
```python
print(df_crm['InvoiceDate'].max())
```

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

[413]:
```python
# Calculate Recency, Frequency, and Monetary values
today_date = dt.datetime(2011, 12, 11)
df_crm['InvoiceDate'] = pd.to_datetime(df_crm['InvoiceDate'], errors='coerce')
rfm = df_crm.groupby('CustomerID').agg({
    'InvoiceDate': lambda x: (today_date - x.max()).days,  # Recency
    'InvoiceNo': lambda x: x.nunique(),  # Frequency (count of unique invoices)
    'TotalPrice': lambda x: x.sum()  # Monetary value (sum of transaction␣
 ↪amounts)
})

# Rename the columns
rfm.columns = ['recency', 'frequency', 'monetary']

# Filter out customers with 0 or negative monetary value
rfm = rfm[rfm['monetary'] > 0].reset_index()

# Check the RFM table
rfm.head()
```

[413]:

	CustomerID	recency	frequency	monetary
0	12346.0	326	1	77183.60
1	12347.0	3	7	4310.00
2	12348.0	76	4	1797.24
3	12349.0	19	1	1757.55
4	12350.0	311	1	334.40

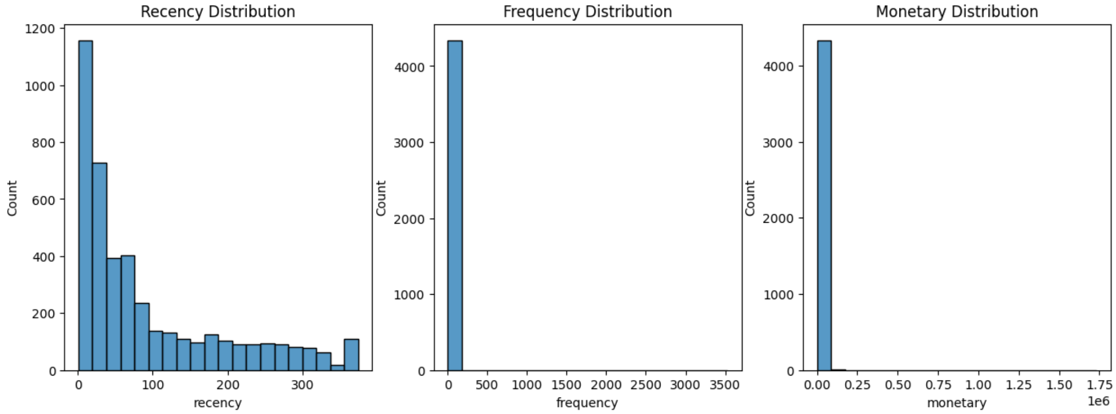
RFM Segmentation

```
[414]: # Define RFM score bins for each metric (convert the bins to string for later␣
        ↪concatenation)
       def get_rfm_scores(dataframe) -> pd.core.frame.DataFrame:

           df_crm = dataframe.copy()
           df_crm["recency_score"] = pd.qcut(df_crm["recency"], 5, labels=[5, 4, 3, 2,␣
        ↪1])
           df_crm["frequency_score"] = pd.qcut(
               df_crm["frequency"].rank(method="first"), 5, labels=[1, 2, 3, 4, 5]
           )
           df_crm["monetary_score"] = pd.qcut(df_crm["monetary"], 5, labels=[1, 2, 3,␣
        ↪4, 5])
           df_crm["RFM_SCORE"] = df_crm["recency_score"].astype(str) +␣
        ↪df_crm["frequency_score"].astype(
               str
           )

           return df_crm

       rfm = get_rfm_scores(rfm)
```

Visualization

```
[415]: # Plot the distribution of Recency, Frequency, and Monetary
       plt.figure(figsize=(15, 5))

       # Recency Distribution
       plt.subplot(1, 3, 1)
       sns.histplot(rfm['recency'], bins=20)
       plt.title('Recency Distribution')

       # Frequency Distribution
       plt.subplot(1, 3, 2)
       sns.histplot(rfm['frequency'], bins=20)
       plt.title('Frequency Distribution')

       # Monetary Distribution
       plt.subplot(1, 3, 3)
       sns.histplot(rfm['monetary'], bins=20)
       plt.title('Monetary Distribution')
```

```
[415]: Text(0.5, 1.0, 'Monetary Distribution')
```



[416]:
```
seg_map = {r'[1-2][1-2]': 'hibernating',
           r'[1-2][3-4]': 'at_Risk',
           r'[1-2]5': 'cant_loose',
           r'3[1-2]': 'about_to_sleep',
           r'33': 'need_attention',
           r'[3-4][4-5]': 'loyal_customers',
           r'41': 'promising',
           r'51': 'new_customers',
           r'[4-5][2-3]': 'potential_loyalists',
           r'5[4-5]': 'champions'}

rfm['segment'] =  rfm["RFM_SCORE"] .replace(seg_map, regex = True)

rfm.head()
```

[416]:
```
   CustomerID  recency  frequency  monetary recency_score frequency_score  \
0     12346.0      326          1  77183.60             1               1
1     12347.0        3          7   4310.00             5               5
2     12348.0       76          4   1797.24             2               4
3     12349.0       19          1   1757.55             4               1
4     12350.0      311          1    334.40             1               1

  monetary_score RFM_SCORE      segment
0              5        11  hibernating
1              5        55    champions
2              4        24      at_Risk
3              4        41    promising
4              2        11  hibernating
```

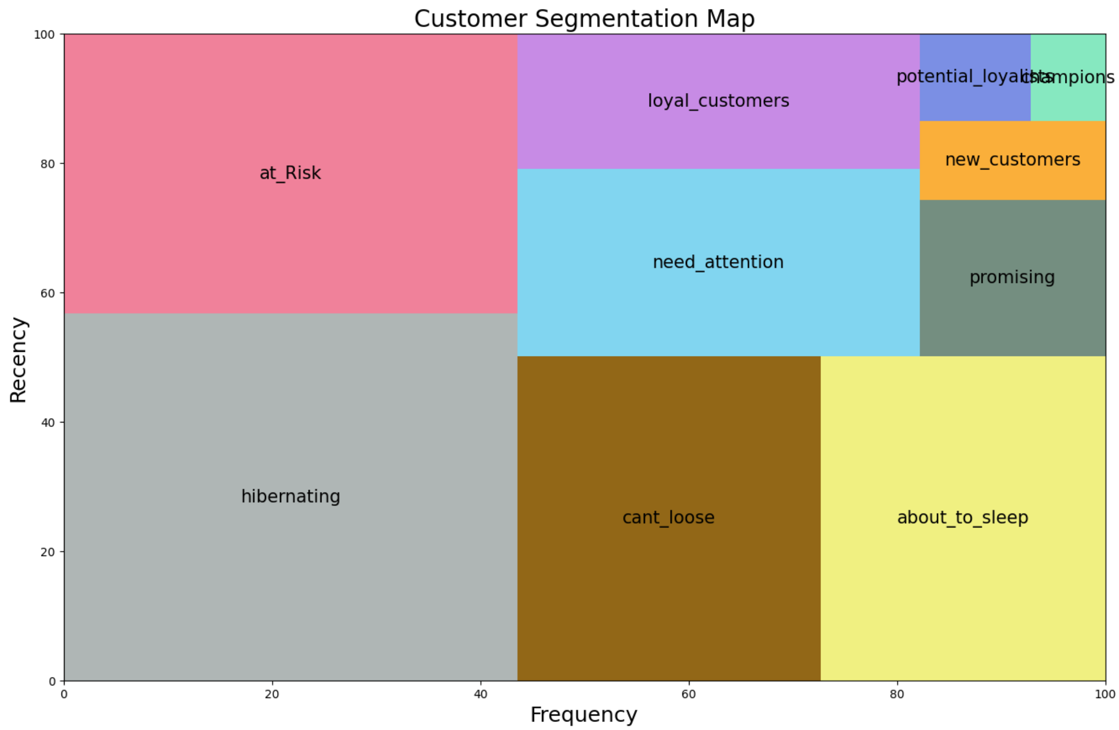
[417]:
```
print(rfm['segment'].nunique())
print(rfm['segment'].unique())
```

```
10
```

```
['hibernating' 'champions' 'at_Risk' 'promising' 'loyal_customers'
 'potential_loyalists' 'about_to_sleep' 'need_attention' 'new_customers'
 'cant_loose']
```

Segmentation Map

[418]:
```
segments = rfm['segment'].value_counts().sort_values(ascending=False)
fig = plt.gcf()
ax = fig.add_subplot()
fig.set_size_inches(16, 10)
squarify.plot(
    sizes=segments,
    label=[label for label in seg_map.values()],
    color=[
        "#AFB6B5",
        "#F0819A",
        "#926717",
        "#F0F081",
        "#81D5F0",
        "#C78BE5",
        "#748E80",
        "#FAAF3A",
        "#7B8FE4",
        "#86E8C0",
    ],
    pad=False,
    bar_kwargs={"alpha": 1},
    text_kwargs={"fontsize": 15},
)
plt.title("Customer Segmentation Map", fontsize=20)
plt.xlabel("Frequency", fontsize=18)
plt.ylabel("Recency", fontsize=18)
plt.show()
```



Let's choose 3 segments that we find important. These three segments

[419]:
```
rfm[["segment", "recency", "frequency", "monetary"]].groupby("segment").
 ↪agg(["mean", "count","max"])
```

[419]:

	recency			frequency			\
	mean	count	max	mean	count	max	
segment							
about_to_sleep	53.312500	352	72	1.161932	352	2	
at_Risk	153.785835	593	374	2.878583	593	6	
cant_loose	132.968254	63	373	8.380952	63	34	
champions	6.353312	634	13	17.962145	634	3528	
hibernating	217.605042	1071	374	1.101774	1071	2	
loyal_customers	33.608059	819	72	6.479853	819	63	
need_attention	52.427807	187	72	2.326203	187	3	
new_customers	7.428571	42	13	1.000000	42	1	
potential_loyalists	17.398760	484	33	2.010331	484	3	
promising	23.510638	94	33	1.000000	94	1	

	monetary		
	mean	count	max
segment			
about_to_sleep	469.893437	352	6207.67
at_Risk	1080.920373	593	44534.30

```
cant_loose           2790.101429    63    10254.18
champions            9565.454890   634  1732777.79
hibernating           487.628909  1071    77183.60
loyal_customers      2855.791173   819   124914.53
need_attention        892.505936   187    12601.83
new_customers         385.022381    42     3861.00
potential_loyalists  1036.483099   484   168472.50
promising             292.050213    94     1757.55
```

Champions About to Sleep Can't Loose

Champions There are 905 people in this group. They do not shop for an average of 8.47 days. They shopped an average of 10.28 times. They earned an average of 5631.84 units of money.

Action Decision That Can Be Taken for Champions: The best customers are the last, the most frequent purchasers and the ones who contribute the most to the company because finding a new customer is always the hardest. We can categorize them as a premium customer and give prime offers such as free shipping.

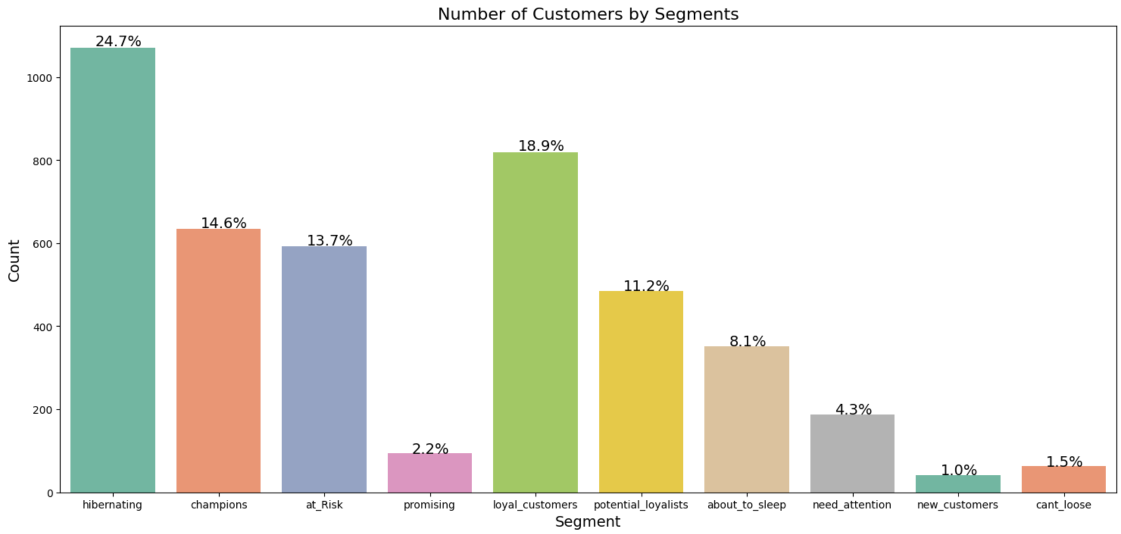
About to Sleep There are 355 people in this segment, They do not shop for an average of 74.20 days. They shopped an average of 1.00 times. They earned an average of 451.74 units of money.

Action Decision That Can Be Taken for About to Sleep: We can give them discount checks which will encourage them to purchase from our company.

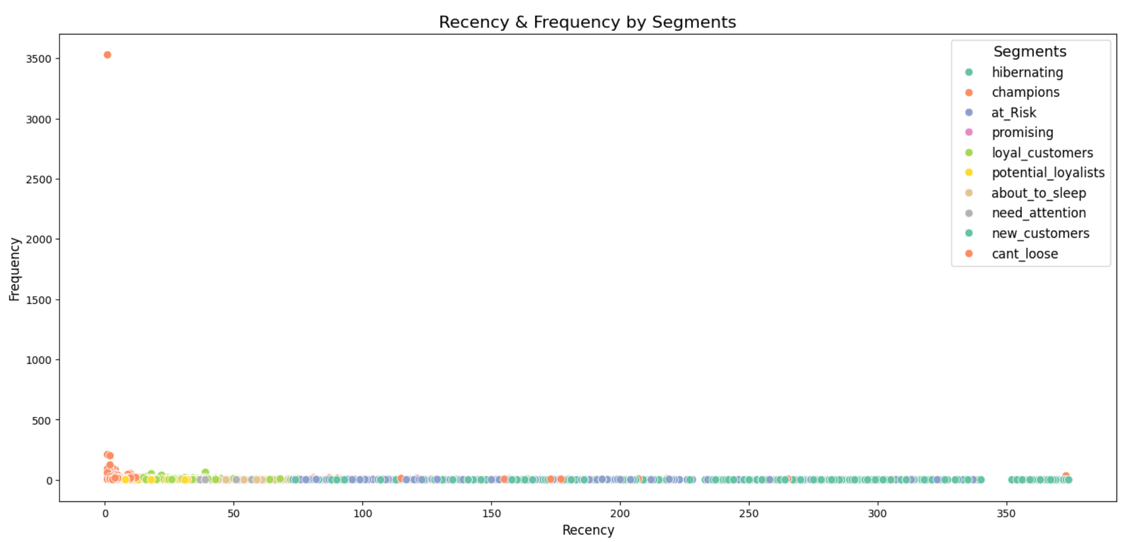
Can't Loose There are 48 people in this segment, They do not shop for an average of 185.77 days. They shopped an average of 7.0 times. They earned an average of 2054.09 units of money.

Action Decision That Can Be Taken for Can't Loose: It is one of the segments that should be given importance. These people categorized as can't lose might become a champions. We can send notifacations about privilege of premium customers and why they should become one of them

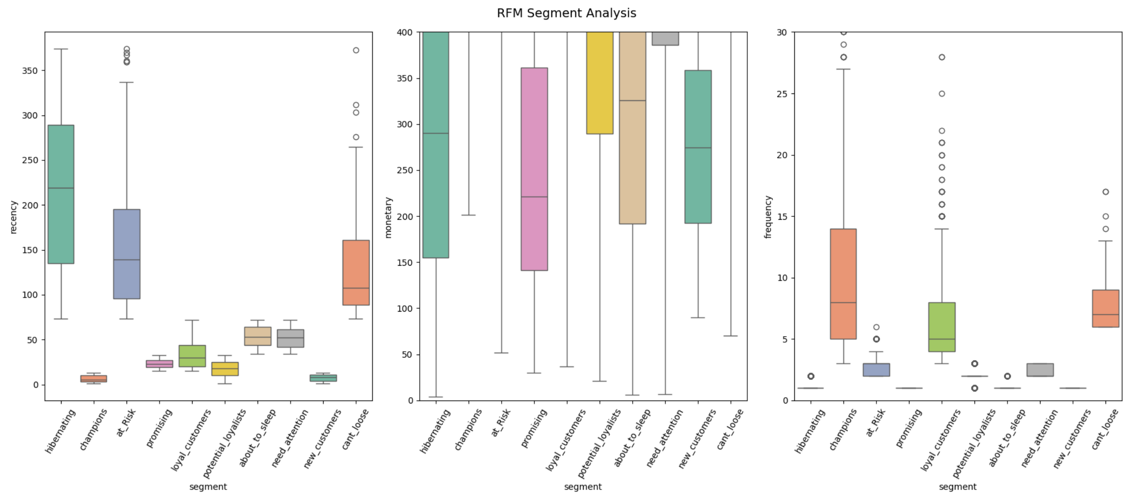
[420]:
```
palette = sns.color_palette("Set2")
plt.figure(figsize = (18, 8))
ax = sns.countplot(data = rfm,
                   x = 'segment',
                   palette = palette)
total = len(rfm.segment)
for patch in ax.patches:
    percentage = '{:.1f}%'.format(100 * patch.get_height()/total)
    x = patch.get_x() + patch.get_width() / 2 - 0.17
    y = patch.get_y() + patch.get_height() * 1.005
    ax.annotate(percentage, (x, y), size = 14)
plt.title('Number of Customers by Segments', size = 16)
plt.xlabel('Segment', size = 14)
plt.ylabel('Count', size = 14)
plt.xticks(size = 10)
plt.yticks(size = 10)
plt.show()
```



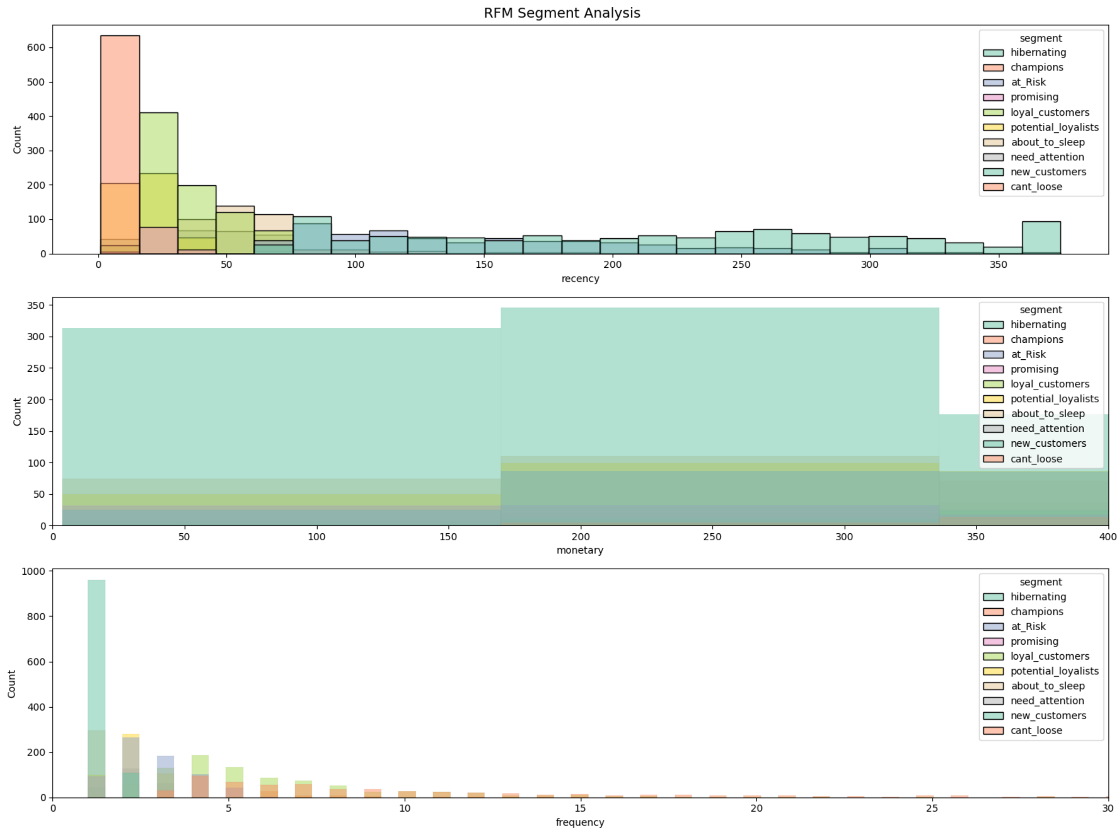
[421]:
```
plt.figure(figsize=(18, 8))
sns.scatterplot(
    data=rfm, x="recency", y="frequency", hue="segment", palette=palette, s=60
)
plt.title("Recency & Frequency by Segments", size=16)
plt.xlabel("Recency", size=12)
plt.ylabel("Frequency", size=12)
plt.xticks(size=10)
plt.yticks(size=10)
plt.legend(loc="best", fontsize=12, title="Segments", title_fontsize=14)
plt.show()
```



[422]:
```
fig, axes = plt.subplots(1, 3, figsize=(18, 8))
fig.suptitle("RFM Segment Analysis", size=14)
feature_list = ["recency", "monetary", "frequency"]
for idx, col in enumerate(feature_list):
    sns.boxplot(
        ax=axes[idx], data=rfm, x="segment", y=feature_list[idx],␣
 ↪palette=palette
    )
    axes[idx].set_xticklabels(axes[idx].get_xticklabels(), rotation=60)
    if idx == 1:
        axes[idx].set_ylim([0, 400])
    if idx == 2:
        axes[idx].set_ylim([0, 30])
plt.tight_layout()
plt.show()
```



[423]:
```
fig, axes = plt.subplots(3, 1, figsize=(16, 12))
fig.suptitle('RFM Segment Analysis', size = 14)
feature_list = ['recency', 'monetary', 'frequency']
for idx, col in enumerate(feature_list):
    sns.histplot(ax = axes[idx], data = rfm,
                 hue = 'segment', x = feature_list[idx],
                 palette= palette)
    if idx == 1:
        axes[idx].set_xlim([0, 400])
    if idx == 2:
        axes[idx].set_xlim([0, 30])
plt.tight_layout()
plt.show()
```



1. Descriptive Statistics

[424]:
```python
# Descriptive Statistics for frequency and monetary
desc_stats = rfm[['frequency', 'monetary']].describe()
print("Descriptive Statistics:")
print(desc_stats)

# Total revenue
total_revenue = rfm['monetary'].sum()
print(f"Total Revenue: {total_revenue}")

# Average customer purchase frequency and spending
average_frequency = rfm['frequency'].mean()
average_spending = rfm['monetary'].mean()
print(f"Average Frequency: {average_frequency}")
print(f"Average Spending: {average_spending}")
```

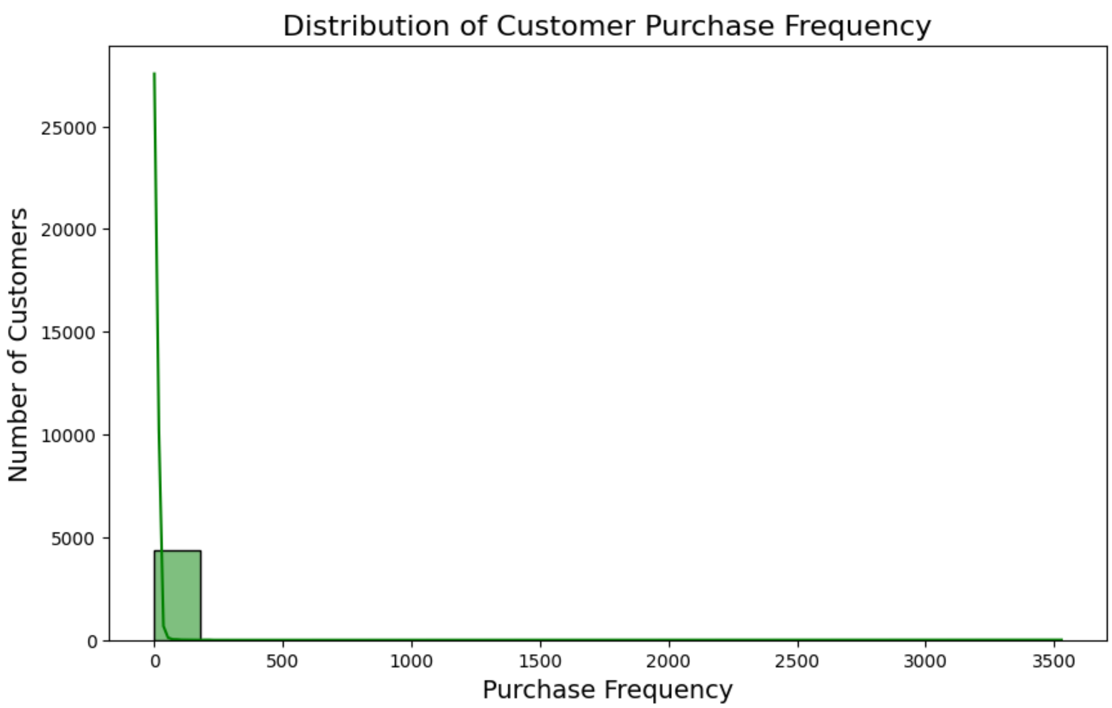
```
Descriptive Statistics:
         frequency      monetary
count  4339.000000  4.339000e+03
mean      5.084812  2.447565e+03
std      54.046385  2.776805e+04
```

```
min        1.000000  3.750000e+00
25%        1.000000  3.065050e+02
50%        2.000000  6.685800e+02
75%        5.000000  1.660890e+03
max     3528.000000  1.732778e+06
Total Revenue: 10619986.684
Average Frequency: 5.084812168702466
Average Spending: 2447.5654952754094
```

2. Visualizations

a. Histogram of Customer Purchases (Frequency)

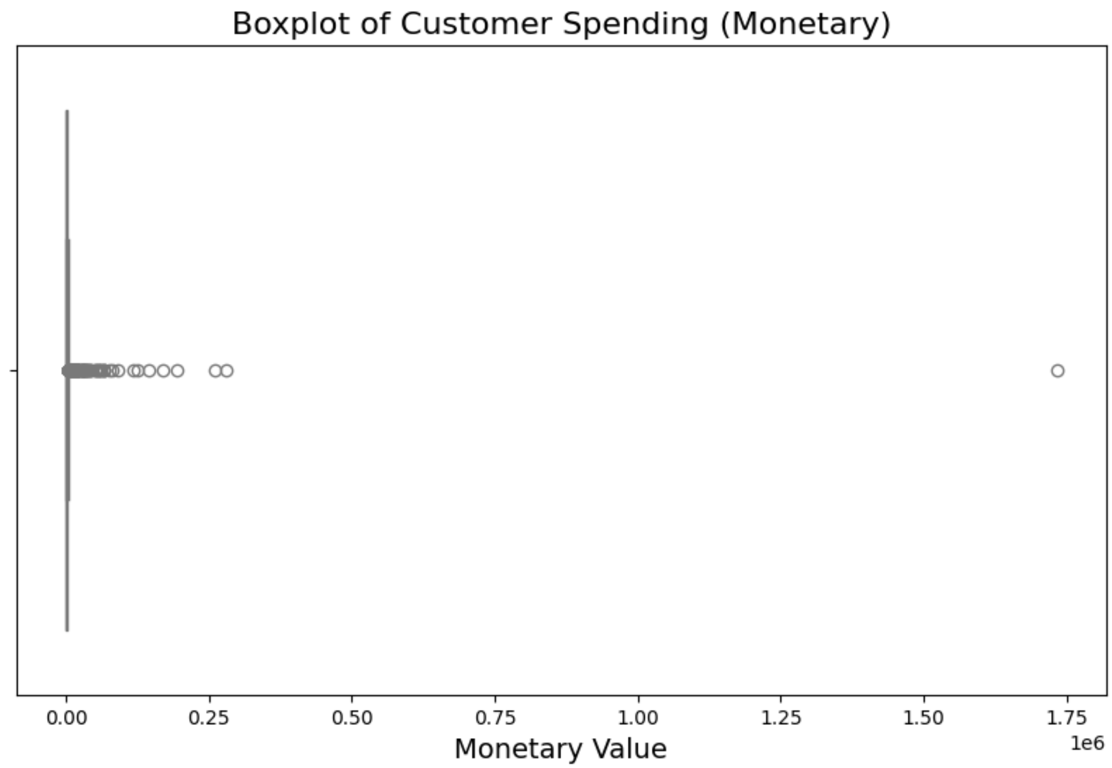
[425]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Histogram for customer purchase frequency
plt.figure(figsize=(10, 6))
sns.histplot(rfm['frequency'], bins=20, kde=True, color='green')
plt.title('Distribution of Customer Purchase Frequency', fontsize=16)
plt.xlabel('Purchase Frequency', fontsize=14)
plt.ylabel('Number of Customers', fontsize=14)
plt.show()
```



b. Boxplot for Customer Spending (Monetary)

[426]:

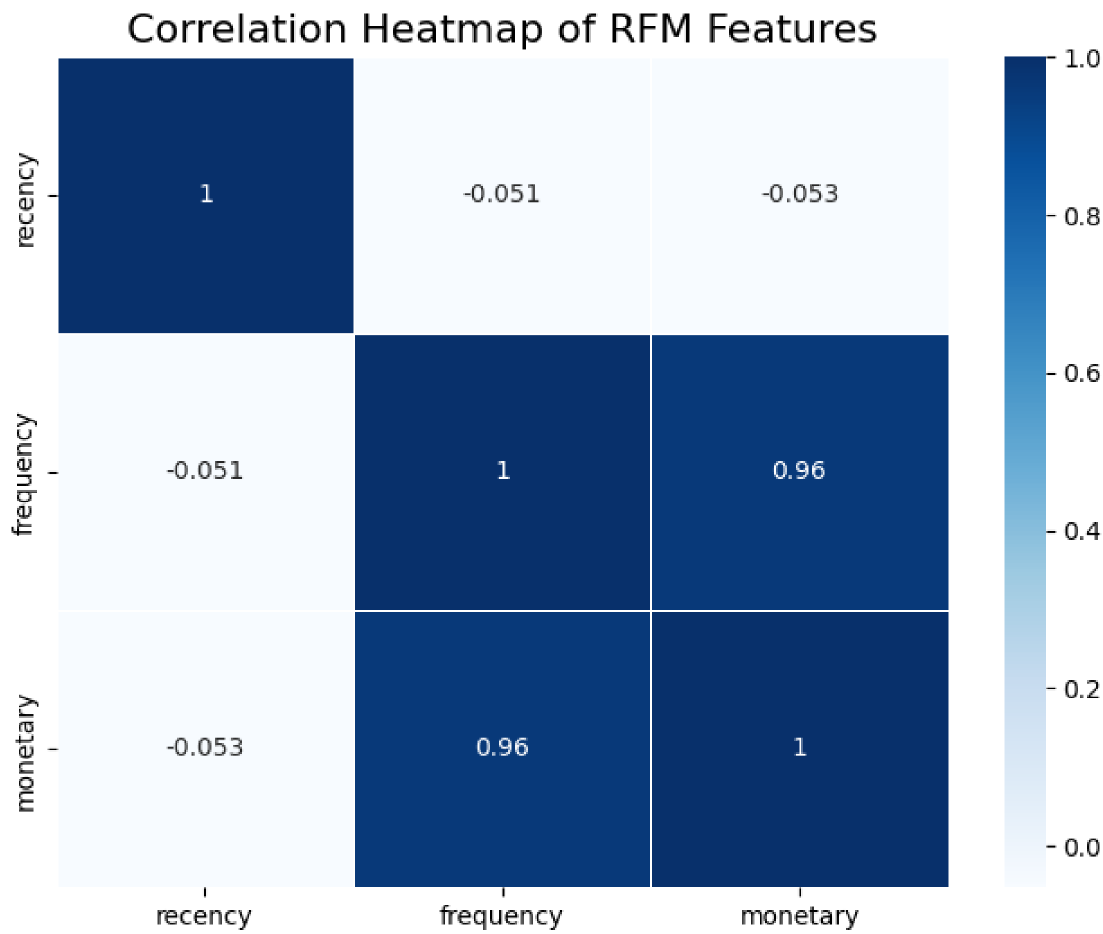
```
plt.figure(figsize=(10, 6))
sns.boxplot(x='monetary', data=rfm, color='lightblue')
plt.title('Boxplot of Customer Spending (Monetary)', fontsize=16)
plt.xlabel('Monetary Value', fontsize=14)
plt.show()
```



c. Correlation Heatmap of RFM Features The heatmap helps visualize the correlation between Recency, Frequency, and Monetary values to identify any strong relationships between them.

[427]:

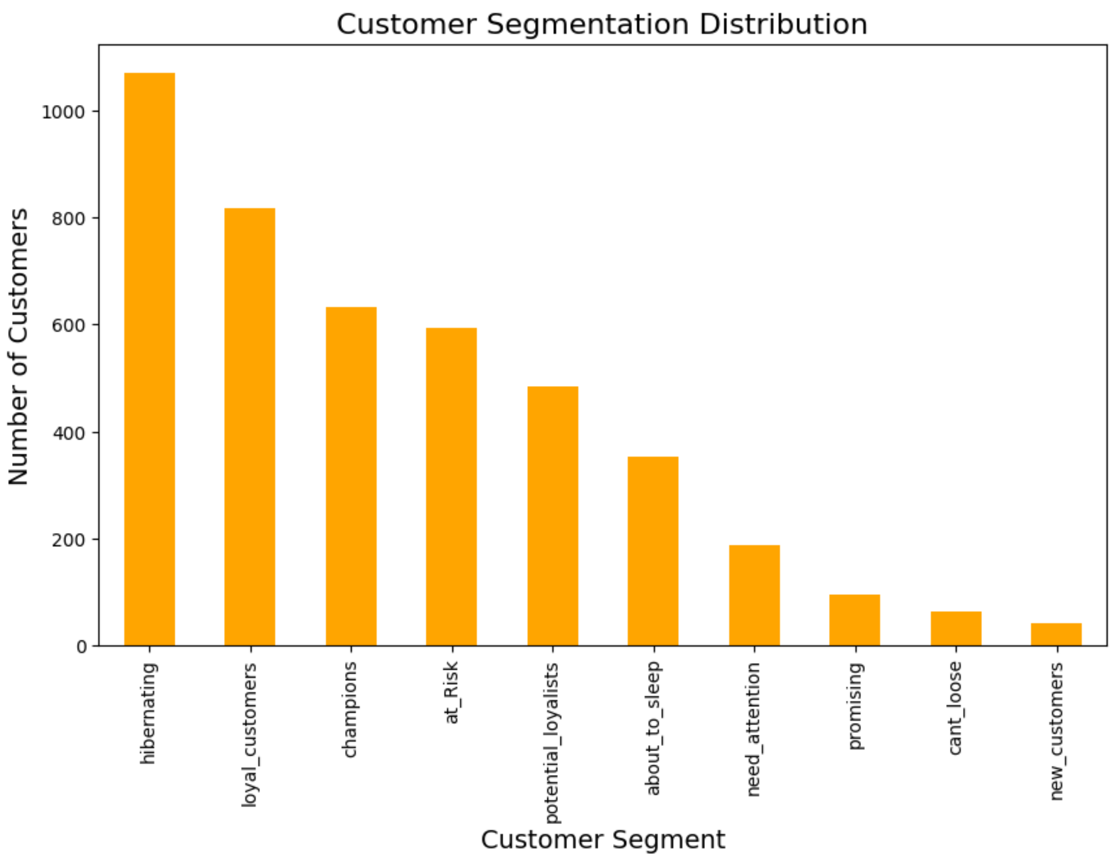
```
# Correlation heatmap of Recency, Frequency, and Monetary
plt.figure(figsize=(8, 6))
corr_matrix = rfm[['recency', 'frequency', 'monetary']].corr()
sns.heatmap(corr_matrix, annot=True, cmap='Blues', linewidths=0.5)
plt.title('Correlation Heatmap of RFM Features', fontsize=16)
plt.show()
```



3. Customer Segmentation

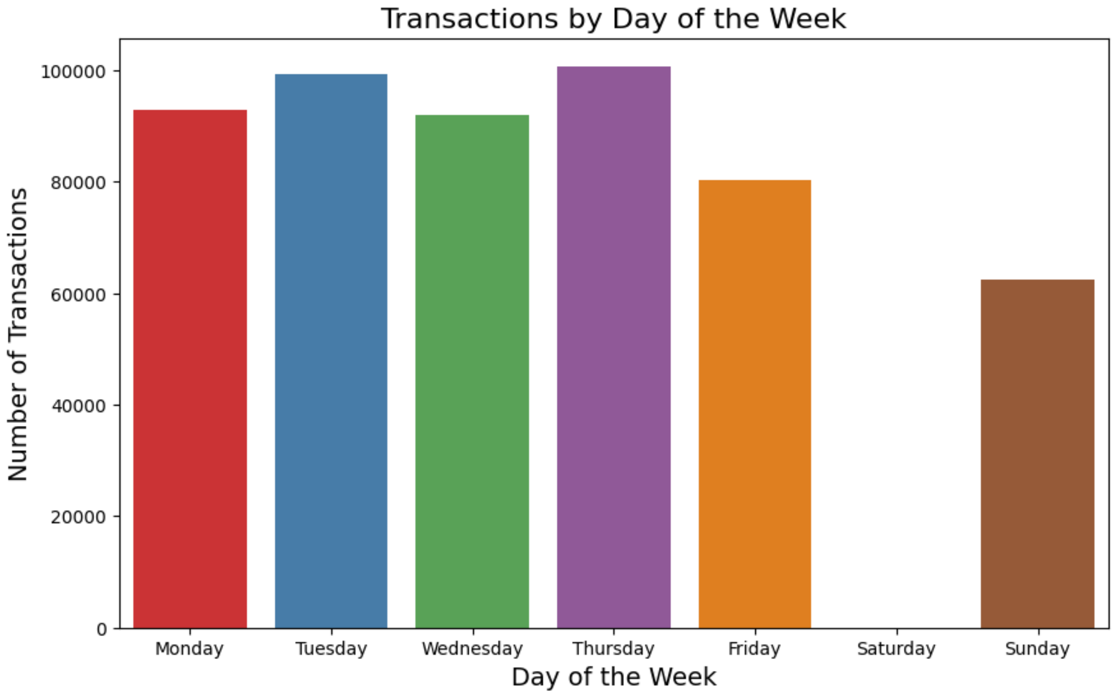
a. Distribution of Customer Segments You can group customers based on their RFM score and visualize the distribution of different segments.

[428]:
```
segment_distribution = rfm['segment'].value_counts()

# Bar plot for segment distribution
plt.figure(figsize=(10, 6))
segment_distribution.plot(kind='bar', color='orange')
plt.title('Customer Segmentation Distribution', fontsize=16)
plt.xlabel('Customer Segment', fontsize=14)
plt.ylabel('Number of Customers', fontsize=14)
plt.show()
```



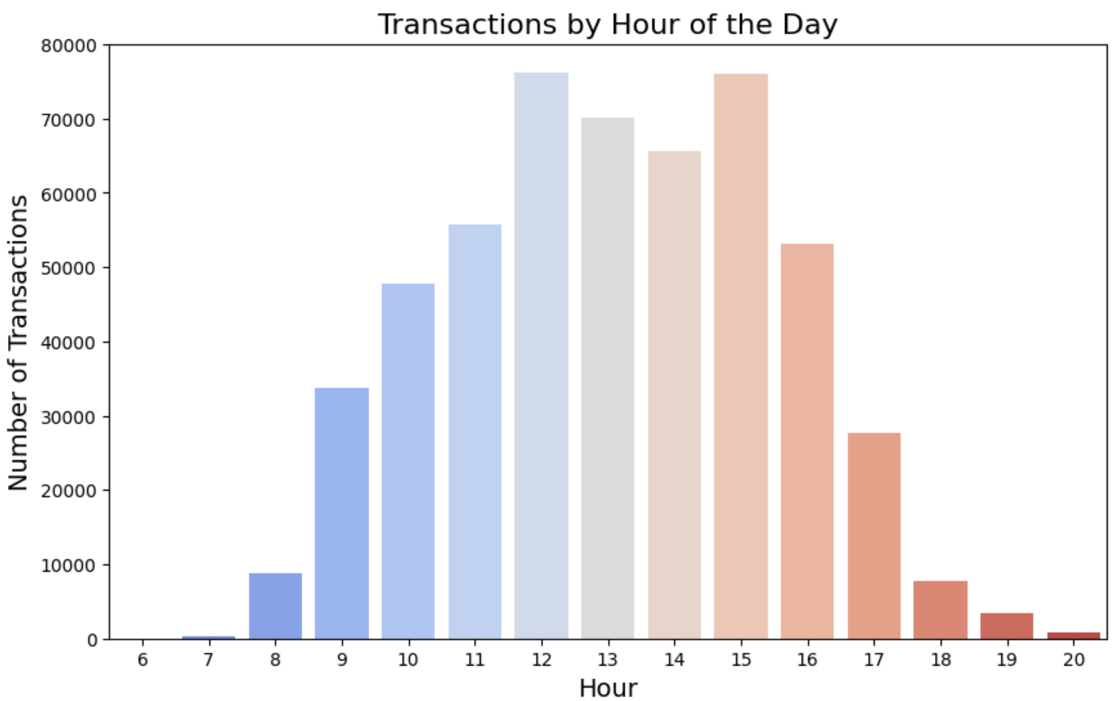
b. Transaction Trends by Day of the Week To explore if certain days of the week have higher customer activity.

[429]:
```
df_crm['DayOfWeek'] = df_crm['InvoiceDate'].dt.day_name()

# Plot the frequency of transactions by day of the week
plt.figure(figsize=(10, 6))
sns.countplot(x='DayOfWeek', data=df_crm, order=['Monday', 'Tuesday',␣
↪'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'], palette='Set1')
plt.title('Transactions by Day of the Week', fontsize=16)
plt.xlabel('Day of the Week', fontsize=14)
plt.ylabel('Number of Transactions', fontsize=14)
plt.show()
```



c. Transaction Trends by Hour

[430]:
```
df_crm['Hour'] = df_crm['InvoiceDate'].dt.hour

# Plot transaction frequency by hour of the day
plt.figure(figsize=(10, 6))
sns.countplot(x='Hour', data=df_crm, palette='coolwarm')
plt.title('Transactions by Hour of the Day', fontsize=16)
plt.xlabel('Hour', fontsize=14)
plt.ylabel('Number of Transactions', fontsize=14)
plt.show()
```



[431]:
```python
# 1. Average Days Between Purchases (for each customer)
# Sort by 'CustomerID' and 'InvoiceDate' to calculate time difference
df_crm = df_crm.sort_values(by=['CustomerID', 'InvoiceDate'])
df_crm['Prev_InvoiceDate'] = df_crm.groupby('CustomerID')['InvoiceDate'].
 ↪shift(1)

# Calculate the difference in days between successive purchases
df_crm['Days_Between_Purchases'] = (df_crm['InvoiceDate'] -␣
 ↪df_crm['Prev_InvoiceDate']).dt.days

# Calculate the average days between purchases for each customer
df_avg_days = df_crm.groupby('CustomerID')['Days_Between_Purchases'].mean().
 ↪reset_index()
df_avg_days.columns = ['CustomerID', 'Avg_Days_Between_Purchases']

# Merge back to the main dataframe
df_crm = pd.merge(df_crm, df_avg_days, on='CustomerID', how='left')

# 2. Preferred Shopping Days (day of the week customers shop most)
df_crm['Day_of_Week'] = df_crm['InvoiceDate'].dt.day_name()

# Calculate the preferred shopping day for each customer
df_pref_day = df_crm.groupby('CustomerID')['Day_of_Week'].agg(lambda x: x.
 ↪value_counts().idxmax()).reset_index()
```

```
df_pref_day.columns = ['CustomerID', 'Preferred_Shopping_Day']

# Merge back to the main dataframe
df_crm = pd.merge(df_crm, df_pref_day, on='CustomerID', how='left')

# 3. Peak Shopping Hours (hour of the day when customers shop most)
df_crm['Hour_of_Day'] = df_crm['InvoiceDate'].dt.hour

# Calculate the peak shopping hour for each customer
df_peak_hour = df_crm.groupby('CustomerID')['Hour_of_Day'].agg(lambda x: x.
 ↪value_counts().idxmax()).reset_index()
df_peak_hour.columns = ['CustomerID', 'Peak_Shopping_Hour']

# Merge back to the main dataframe
df_crm = pd.merge(df_crm, df_peak_hour, on='CustomerID', how='left')

# Show the engineered features
print(df_crm[['CustomerID', 'Avg_Days_Between_Purchases',␣
 ↪'Preferred_Shopping_Day', 'Peak_Shopping_Hour']].head())
```

```
  CustomerID  Avg_Days_Between_Purchases Preferred_Shopping_Day  \
0    12346.0                         NaN                Tuesday
1    12347.0                         2.0                Tuesday
2    12347.0                         2.0                Tuesday
3    12347.0                         2.0                Tuesday
4    12347.0                         2.0                Tuesday

   Peak_Shopping_Hour
0                  10
1                  14
2                  14
3                  14
4                  14
```

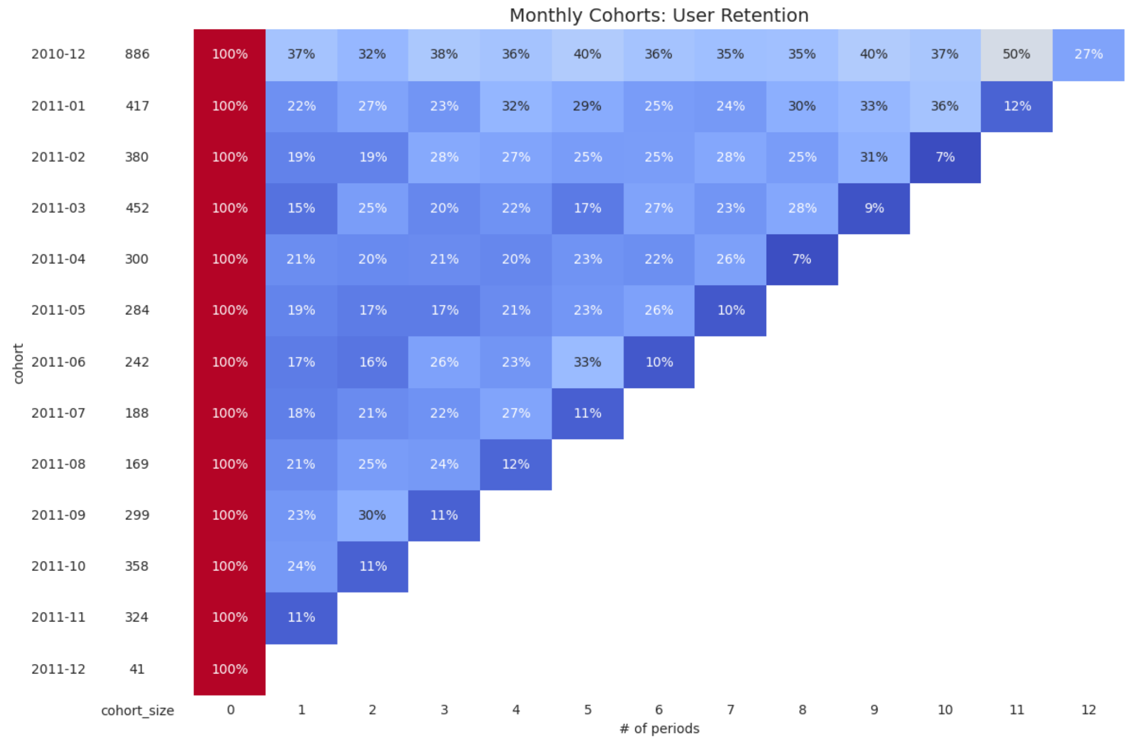
Cohort Analysis

A cohort is a group of people sharing something in common, such as the sign-up date to an app, the month of the first purchase, geographical location, acquisition channel (organic users, coming from performance marketing, etc.) and so on. In Cohort Analysis, we track these groups of users over time, to identify some common patterns or behaviors.

[437]:
```
import matplotlib.colors as mcolors
def CohortAnalysis(dataframe):

    data = dataframe.copy()
    data = data[["CustomerID", "InvoiceNo", "InvoiceDate"]].drop_duplicates()
    data["order_month"] = data["InvoiceDate"].dt.to_period("M")
    data["cohort"] = (
```

```python
        data.groupby("CustomerID")["InvoiceDate"].transform("min").dt.
↪to_period("M")
    )
    cohort_data = (
        data.groupby(["cohort", "order_month"])
        .agg(n_customers=("CustomerID", "nunique"))
        .reset_index(drop=False)
    )
    # Calculate period number without attrgetter
    cohort_data["period_number"] = (cohort_data.order_month - cohort_data.
↪cohort).apply(lambda x: x.n)

    cohort_pivot = cohort_data.pivot_table(
        index="cohort", columns="period_number", values="n_customers"
    )
    cohort_size = cohort_pivot.iloc[:, 0]
    retention_matrix = cohort_pivot.divide(cohort_size, axis=0)

    with sns.axes_style("white"):
        fig, ax = plt.subplots(
            1, 2, figsize=(12, 8), sharey=True, gridspec_kw={"width_ratios":␣
↪[1, 11]}
        )
        sns.heatmap(
            retention_matrix,
            mask=retention_matrix.isnull(),
            annot=True,
        cbar=False,
            fmt=".0%",
            cmap="coolwarm",
            ax=ax[1],
        )
        ax[1].set_title("Monthly Cohorts: User Retention", fontsize=14)
        ax[1].set(xlabel="# of periods", ylabel="")

        white_cmap = mcolors.ListedColormap(["white"])
        sns.heatmap(
            pd.DataFrame(cohort_size).rename(columns={0: "cohort_size"}),
            annot=True,
            cbar=False,
            fmt="g",
            cmap=white_cmap,
            ax=ax[0],
        )
        fig.tight_layout()

# Call the function using your dataframe
```

```
CohortAnalysis(df_crm)
```



Insights: Customer Segmentation:

Champions: This segment comprises 905 customers, who purchase frequently and generate high revenue, with an average spending of $5,631. These are your top customers, shopping within an average of 8.47 days since their last purchase. Hibernating: 1,604 customers have not shopped recently, with an average recency of 263.46 days and only 1 purchase on average. These customers have low engagement. At Risk: 650 customers have not shopped for an extended period (average 179 days), but have higher frequency (1.96 purchases) and monetary value. Can't Lose: 48 customers show signs of disengagement, with a high average of 7 purchases but have not shopped for about 185 days. Their average spending is $2,054, and they have high potential to be re-engaged. Spending Distribution:

The data shows a clear distinction between customer spending tiers. Many customers are classified as "Low Spenders," while "Champions" and "VIP Spenders" make up a smaller but highly valuable portion of your customer base. Outliers:

Significant outliers were detected in UnitPrice and TotalPrice with the most expensive transactions being much higher than the average. Correcting or addressing these could prevent skewing of overall results. Product Insights:

The top-selling products like "WORLD WAR 2 GLIDERS ASSTD DESIGNS" and "JUMBO BAG RED RETROSPOT" are the biggest revenue generators. Focusing on such products and ensuring stock availability can maximize profits. Customer Behavior by Time:

Day of the Week: Fridays and Thursdays have the highest sales, while weekends tend to have fewer transactions. Time of Day: The peak shopping hours are between 10 AM and 4 PM, offering opportunities to optimize promotions and marketing around this time.

Recommendations: Retention Strategy for Key Segments:

Champions: Provide exclusive offers like loyalty programs, priority customer support, or personalized discounts. These customers are already highly engaged, so maintaining their loyalty is crucial. At Risk & Can't Lose: Implement targeted re-engagement campaigns with discounts or offers designed to incentivize purchases. Highlight the benefits of returning, such as "last chance to avail loyalty points" or "we miss you" messages. Hibernating: Since these customers are largely inactive, a win-back strategy offering substantial discounts or freebies could help re-engage them. Product Strategy:

Focus on stocking and promoting your top-performing products to meet demand. Implement cross-selling and upselling techniques for customers purchasing these products. Analyze why certain products (like "Damaged" or "Unsaleable" items) show up as negative contributors and address those issues. Outlier Management:

Review transactions with extreme values for possible data entry errors or special cases. Outliers could distort your overall insights if left unchecked. Optimize for Peak Shopping Times:

Schedule email campaigns or ads around peak shopping hours (10 AM - 4 PM) and on popular shopping days (Thursdays and Fridays). Consider offering flash sales or special deals during these times to further increase engagement. Customer Lifecycle Management:

Utilize the Average Days Between Purchases and Preferred Shopping Day insights to better predict when a customer might next shop and target them with personalized recommendations. Create automated marketing campaigns to send reminders when customers approach their typical purchase cycle. By implementing these strategies, we can enhance customer retention, optimize sales, and improve overall customer satisfaction.