In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Exploratory Data Analysis

In [2]:
```
# Read Excel file into Spark DataFrame
file_path = "data\\online_retail.xlsx"

# Data from sheet 1
df_1 = pd.read_excel(file_path, sheet_name="Year 2009-2010")

df_1
```

Out[2]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	United Kingdom
...	...	...	...	...	...	...	...	...
525456	538171	22271	FELTCRAFT DOLL ROSIE	2	2010-12-09 20:01:00	2.95	17530.0	United Kingdom
525457	538171	22750	FELTCRAFT PRINCESS LOLA DOLL	1	2010-12-09 20:01:00	3.75	17530.0	United Kingdom
525458	538171	22751	FELTCRAFT PRINCESS OLIVIA DOLL	1	2010-12-09 20:01:00	3.75	17530.0	United Kingdom
525459	538171	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	2	2010-12-09 20:01:00	3.75	17530.0	United Kingdom
525460	538171	21931	JUMBO STORAGE BAG SUKI	2	2010-12-09 20:01:00	1.95	17530.0	United Kingdom

525461 rows × 8 columns

In [3]:
```
# Data from sheet 2
df_2 = pd.read_excel(file_path, sheet_name="Year 2010-2011")

df_2
```

Out[3]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	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
...	...	...	...	...	...	...	...	...
541905	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	12680.0	France
541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	12680.0	France
541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	12680.0	France
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	12680.0	France
541909	581587	POST	POSTAGE	1	2011-12-09 12:50:00	18.00	12680.0	France

541910 rows × 8 columns

In [4]:

```
# Merging data which contains record from 2009 until 2011
df = pd.concat([df_1, df_2], ignore_index=True)
print(df.info())
df
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1067371 entries, 0 to 1067370
Data columns (total 8 columns):
 #   Column       Non-Null Count    Dtype
---  ------       --------------    -----
 0   Invoice      1067371 non-null  object
 1   StockCode    1067371 non-null  object
 2   Description  1062989 non-null  object
 3   Quantity     1067371 non-null  int64
 4   InvoiceDate  1067371 non-null  datetime64[ns]
 5   Price        1067371 non-null  float64
 6   Customer ID  824364 non-null   float64
 7   Country      1067371 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 65.1+ MB
None
```

Out[4]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	United Kingdom
...	...	...	...	...	...	...	...	...
1067366	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	12680.0	France
1067367	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	12680.0	France
1067368	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	12680.0	France
1067369	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	12680.0	France
1067370	581587	POST	POSTAGE	1	2011-12-09 12:50:00	18.00	12680.0	France

1067371 rows × 8 columns

In [5]:

```
# Qunatity & Price has minus values which are not error. Basically, they are discount or giveaway package
# InvoiceNo: Invoice number. Nominal. A 6-digit integral number uniquely assigned to each transaction.
# If this code starts with the letter 'c', it indicates a cancellation.
df.describe()
```

Out[5]:

	Quantity	InvoiceDate	Price	Customer ID
count	1.067371e+06	1067371	1.067371e+06	824364.000000
mean	9.938898e+00	2011-01-02 21:13:55.394028544	4.649388e+00	15324.638504
min	-8.099500e+04	2009-12-01 07:45:00	-5.359436e+04	12346.000000
25%	1.000000e+00	2010-07-09 09:46:00	1.250000e+00	13975.000000
50%	3.000000e+00	2010-12-07 15:28:00	2.100000e+00	15255.000000
75%	1.000000e+01	2011-07-22 10:23:00	4.150000e+00	16797.000000
max	8.099500e+04	2011-12-09 12:50:00	3.897000e+04	18287.000000
std	1.727058e+02	NaN	1.235531e+02	1697.464450

In [6]:

```
# To see what possibilities we could do with this dataset
df.describe(include='O')
```

Out[6]:

	Invoice	StockCode	Description	Country
count	1067371	1067371	1062989	1067371
unique	53628	5305	5698	43
top	537434	85123A	WHITE HANGING HEART T-LIGHT HOLDER	United Kingdom
freq	1350	5829	5918	981330

In [7]:

```
# Remove duplicate rows based on all columns
df = df.drop_duplicates()
df
```

Out[7]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	United Kingdom
...	...	...	...	...	...	...	...	...
1067366	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	12680.0	France
1067367	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	12680.0	France
1067368	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	12680.0	France
1067369	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	12680.0	France
1067370	581587	POST	POSTAGE	1	2011-12-09 12:50:00	18.00	12680.0	France

1033036 rows × 8 columns

In [8]:

```
# Count the number of null values in each column
null_counts = df.isnull().sum()

# Print the count of null values per column
print("\nCount of null values per column:")
print(null_counts)
```

```
Count of null values per column:
Invoice              0
StockCode            0
Description       4275
Quantity             0
InvoiceDate          0
Price                0
Customer ID     235151
Country              0
dtype: int64
```

In [9]:

```
# according to the printing result, discounting is not a cause of non-value in Customer ID column

df_discount = df[df["Quantity"] < 0]
df_discount
```

Out[9]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
178	C489449	22087	PAPER BUNTING WHITE LACE	-12	2009-12-01 10:33:00	2.95	16321.0	Australia
179	C489449	85206A	CREAM FELT EASTER EGG BASKET	-6	2009-12-01 10:33:00	1.65	16321.0	Australia
180	C489449	21895	POTTING SHED SOW 'N' GROW SET	-4	2009-12-01 10:33:00	4.25	16321.0	Australia
181	C489449	21896	POTTING SHED TWINE	-6	2009-12-01 10:33:00	2.10	16321.0	Australia
182	C489449	22083	PAPER CHAIN KIT RETRO SPOT	-12	2009-12-01 10:33:00	2.95	16321.0	Australia
...	...	...	...	...	...	...	...	...
1065910	C581490	23144	ZINC T-LIGHT HOLDER STARS SMALL	-11	2011-12-09 09:57:00	0.83	14397.0	United Kingdom
1067002	C581499	M	Manual	-1	2011-12-09 10:28:00	224.69	15498.0	United Kingdom
1067176	C581568	21258	VICTORIAN SEWING BOX LARGE	-5	2011-12-09 11:57:00	10.95	15311.0	United Kingdom
1067177	C581569	84978	HANGING HEART JAR T-LIGHT HOLDER	-1	2011-12-09 11:58:00	1.25	17315.0	United Kingdom
1067178	C581569	20979	36 PENCILS TUBE RED RETROSPOT	-5	2011-12-09 11:58:00	1.25	17315.0	United Kingdom

22496 rows × 8 columns

In [10]:

```
df[df["Invoice"].str.match("^\\d{6}$") == False]
```

Out[10]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
178	C489449	22087	PAPER BUNTING WHITE LACE	-12	2009-12-01 10:33:00	2.95	16321.0	Australia
179	C489449	85206A	CREAM FELT EASTER EGG BASKET	-6	2009-12-01 10:33:00	1.65	16321.0	Australia
180	C489449	21895	POTTING SHED SOW 'N' GROW SET	-4	2009-12-01 10:33:00	4.25	16321.0	Australia
181	C489449	21896	POTTING SHED TWINE	-6	2009-12-01 10:33:00	2.10	16321.0	Australia
182	C489449	22083	PAPER CHAIN KIT RETRO SPOT	-12	2009-12-01 10:33:00	2.95	16321.0	Australia
...	...	...	...	...	...	...	...	...
1065910	C581490	23144	ZINC T-LIGHT HOLDER STARS SMALL	-11	2011-12-09 09:57:00	0.83	14397.0	United Kingdom
1067002	C581499	M	Manual	-1	2011-12-09 10:28:00	224.69	15498.0	United Kingdom
1067176	C581568	21258	VICTORIAN SEWING BOX LARGE	-5	2011-12-09 11:57:00	10.95	15311.0	United Kingdom
1067177	C581569	84978	HANGING HEART JAR T-LIGHT HOLDER	-1	2011-12-09 11:58:00	1.25	17315.0	United Kingdom
1067178	C581569	20979	36 PENCILS TUBE RED RETROSPOT	-5	2011-12-09 11:58:00	1.25	17315.0	United Kingdom

19110 rows × 8 columns

In [11]:

```
df["Invoice"].str.replace("[0-9]", "", regex=True).unique()
```

Out[11]:

```
array([nan, 'C', 'A'], dtype=object)
```

In [30]:

```
# It seems that these are records that spend a lot of money, but Customer ID are null so we need to filter them out w
df_filtered = df[df['Invoice'].astype(str).str.startswith('A')]
df_filtered
```

Out[30]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
179403	A506401	B	Adjust bad debt	1	2010-04-29 13:36:00	-53594.36	NaN	United Kingdom
276274	A516228	B	Adjust bad debt	1	2010-07-19 11:24:00	-44031.79	NaN	United Kingdom
403472	A528059	B	Adjust bad debt	1	2010-10-20 12:04:00	-38925.87	NaN	United Kingdom
825443	A563185	B	Adjust bad debt	1	2011-08-12 14:50:00	11062.06	NaN	United Kingdom
825444	A563186	B	Adjust bad debt	1	2011-08-12 14:51:00	-11062.06	NaN	United Kingdom
825445	A563187	B	Adjust bad debt	1	2011-08-12 14:52:00	-11062.06	NaN	United Kingdom

In [14]:

```
# There are too many unique values that don't follow column's rule and we don't have any information about them
# So we will see some of these unique values and keep them for future analysis
df[(df["StockCode"].str.match("^\\d{5}$") == False) & (df["StockCode"].str.match("^\\d{5}[a-zA-Z]+$") == False)]["Sto
```

Out[14]:

```
array(['POST', 'D', 'DCGS0058', 'DCGS0068', 'DOT', 'M', 'DCGS0004',
       'DCGS0076', 'C2', 'BANK CHARGES', 'DCGS0003', 'TEST001',
       'gift_0001_80', 'DCGS0072', 'gift_0001_20', 'DCGS0044', 'TEST002',
       'gift_0001_10', 'gift_0001_50', 'DCGS0066N', 'gift_0001_30',
       'PADS', 'ADJUST', 'gift_0001_40', 'gift_0001_60', 'gift_0001_70',
       'gift_0001_90', 'DCGSSGIRL', 'DCGS0006', 'DCGS0016', 'DCGS0027',
       'DCGS0036', 'DCGS0039', 'DCGS0060', 'DCGS0056', 'DCGS0059', 'GIFT',
       'DCGSLBOY', 'm', 'DCGS0053', 'DCGS0062', 'DCGS0037', 'DCGSSBOY',
       'DCGSLGIRL', 'S', 'DCGS0069', 'DCGS0070', 'DCGS0075', 'B',
       'DCGS0041', 'ADJUST2', '47503J ', 'C3', 'SP1002', 'AMAZONFEE',
       'DCGS0055', 'DCGS0074', 'DCGS0057', 'DCGS0073', 'DCGS0071',
       'DCGS0066P', 'DCGS0067', 'CRUK'], dtype=object)
```

DATA CLEANING

1. dropping out all null records
2. Format InvoiceDate into dd/mm/yyyy
3. Type casting Customer ID from float => string
4. Filter out non-legit Invoice column
5. Filter out minus Price column
6. Filter out minus Quantity column

In [15]:

```
# We can't do anything to fill out null values of Cusotomer ID column and it would be bad to calculate without knowir
# So, we should drop them out for better result
df_customerID_isnull = df[df["Customer ID"].isnull()]
df_customerID_isnull
```

Out[15]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
263	489464	21733	85123a mixed	-96	2009-12-01 10:52:00	0.00	NaN	United Kingdom
283	489463	71477	short	-240	2009-12-01 10:52:00	0.00	NaN	United Kingdom
284	489467	85123A	21733 mixed	-192	2009-12-01 10:53:00	0.00	NaN	United Kingdom
470	489521	21646	NaN	-50	2009-12-01 11:44:00	0.00	NaN	United Kingdom
577	489525	85226C	BLUE PULL BACK RACING CAR	1	2009-12-01 11:49:00	0.55	NaN	United Kingdom
...	...	...	...	...	...	...	...	...
1066997	581498	85099B	JUMBO BAG RED RETROSPOT	5	2011-12-09 10:26:00	4.13	NaN	United Kingdom
1066998	581498	85099C	JUMBO BAG BAROQUE BLACK WHITE	4	2011-12-09 10:26:00	4.13	NaN	United Kingdom
1066999	581498	85150	LADIES & GENTLEMEN METAL SIGN	1	2011-12-09 10:26:00	4.96	NaN	United Kingdom
1067000	581498	85174	S/4 CACTI CANDLES	1	2011-12-09 10:26:00	10.79	NaN	United Kingdom
1067001	581498	DOT	DOTCOM POSTAGE	1	2011-12-09 10:26:00	1714.17	NaN	United Kingdom

235151 rows × 8 columns

In [16]:

```
# 1. dropping out all null records
df_noNull = df.dropna()

print("Number of records before dropping Null: ", len(df))
print("Number of records after dropping Null: ", len(df_noNull))
```

```
Number of records before dropping Null:  1033036
Number of records after dropping Null:  797885
```

In [25]:
```
# Double checking for null value
rows_with_null = df_noNull[df_noNull.isnull().any(axis=1)]

rows_with_null
```

Out[25]:

Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country

In [26]:
```
# 2. Format InvoiceDate into dd/mm/yyyy

# Convert 'InvoiceDate' from mm/dd/yyyy to datetime format
df_date_formatted = df_noNull.assign(InvoiceDate=pd.to_datetime(df_noNull['InvoiceDate'], format='%m/%d/%Y %H:%M'))

# Reformat 'InvoiceDate' to dd/mm/yyyy format (no time included)
df_date_formatted = df_date_formatted.assign(InvoiceDate=df_date_formatted['InvoiceDate'].dt.strftime('%d/%m/%Y'))


df_date_formatted
```

Out[26]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	01/12/2009	6.95	13085.0	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	01/12/2009	6.75	13085.0	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	01/12/2009	6.75	13085.0	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	01/12/2009	2.10	13085.0	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	01/12/2009	1.25	13085.0	United Kingdom
...	...	...	...	...	...	...	...	...
1067366	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	09/12/2011	2.10	12680.0	France
1067367	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	09/12/2011	4.15	12680.0	France
1067368	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	09/12/2011	4.15	12680.0	France
1067369	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	09/12/2011	4.95	12680.0	France
1067370	581587	POST	POSTAGE	1	09/12/2011	18.00	12680.0	France

797885 rows × 8 columns

In [27]:
```
# 3. Type casting Customer ID from float => string
# the data type is already changed to string(object) but still need to clean out decimal point with replace function

df_typecasted_customerID = df_date_formatted.assign(Customer_ID=df_date_formatted['Customer ID'].astype(str).str.repl
df_typecasted_customerID.drop(columns=['Customer ID'], inplace=True)

print(df_typecasted_customerID.dtypes)
df_typecasted_customerID
```

```
Invoice         object
StockCode       object
Description     object
Quantity         int64
InvoiceDate     object
Price          float64
Country         object
Customer_ID     object
dtype: object
```

Out[27]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Country	Customer_ID
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	01/12/2009	6.95	United Kingdom	13085
1	489434	79323P	PINK CHERRY LIGHTS	12	01/12/2009	6.75	United Kingdom	13085
2	489434	79323W	WHITE CHERRY LIGHTS	12	01/12/2009	6.75	United Kingdom	13085
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	01/12/2009	2.10	United Kingdom	13085
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	01/12/2009	1.25	United Kingdom	13085
...	...	...	...	...	...	...	...	...
1067366	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	09/12/2011	2.10	France	12680
1067367	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	09/12/2011	4.15	France	12680
1067368	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	09/12/2011	4.15	France	12680
1067369	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	09/12/2011	4.95	France	12680
1067370	581587	POST	POSTAGE	1	09/12/2011	18.00	France	12680

797885 rows × 8 columns

In [29]:

```
# 4. Filter out non-legit Invoice colum
# n
# Ensure the 'Invoice' column is treated as strings
df_typecasted_customerID['Invoice'] = df_typecasted_customerID['Invoice'].astype(str)
mask = (
    df_typecasted_customerID["Invoice"].str.match("^\\d{6}$") == True
)
df_cleaned_invoice = df_typecasted_customerID[mask]
df_cleaned_invoice
```

Out[29]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Country	Customer_ID
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	01/12/2009	6.95	United Kingdom	13085
1	489434	79323P	PINK CHERRY LIGHTS	12	01/12/2009	6.75	United Kingdom	13085
2	489434	79323W	WHITE CHERRY LIGHTS	12	01/12/2009	6.75	United Kingdom	13085
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	01/12/2009	2.10	United Kingdom	13085
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	01/12/2009	1.25	United Kingdom	13085
...	...	...	...	...	...	...	...	...
1067366	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	09/12/2011	2.10	France	12680
1067367	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	09/12/2011	4.15	France	12680
1067368	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	09/12/2011	4.15	France	12680
1067369	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	09/12/2011	4.95	France	12680
1067370	581587	POST	POSTAGE	1	09/12/2011	18.00	France	12680

779495 rows × 8 columns

In [31]:

```
# 5. Filter out minus Price column

df_cleaned_invoice[df_cleaned_invoice["Price"] < 0]
```

Out[31]:

Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Country	Customer_ID

In [32]:

```
# 6. Filter out minus Quantity column

df_cleaned_invoice[df_cleaned_invoice["Quantity"] < 0]
```

Out[32]:

Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Country	Customer_ID

In [33]:

```
cleaned_df = df_cleaned_invoice.copy()
drop_percentage = (((len(df) - len(cleaned_df)) / len(df))) * 100
print("Dropped about {}% of records during cleaning".format(drop_percentage))
```

```
Dropped about 24.5432879396265% of records during cleaning
```

Feature engineering

In [ ]:
```
"""
TODO
1. Daily Summary (daily transaction)
2. Customer Dataframe (per-customer summary)
3. Top 10 selling products
4. Top 10 customers with high spending and frequency of buying
5. K-mean clustering of customers

"""
```

1. Daily Summary (daily transaction)

In [34]:
```
transaction_df = cleaned_df.copy()

# Pre-calculate the 'TotalSales' column before the groupby operation
transaction_df['TotalSales'] = transaction_df['Quantity'] * transaction_df['Price']

# Group by and aggregate without using a lambda function
transaction_df = transaction_df.groupby(['InvoiceDate', 'StockCode', 'Description'], as_index=False).agg(
    ItemSold=('Quantity', 'sum'),
    TotalSales=('TotalSales', 'sum')
)

transaction_df
```

Out[34]:

	InvoiceDate	StockCode	Description	ItemSold	TotalSales
0	01/02/2010	10002	INFLATABLE POLITICAL GLOBE	3	2.55
1	01/02/2010	15036	ASSORTED COLOURS SILK FAN	12	7.80
2	01/02/2010	16012	FOOD/DRINK SPUNGE STICKERS	10	2.10
3	01/02/2010	16052	TEATIME PUSH DOWN RUBBER	24	10.08
4	01/02/2010	16235	RECYCLED PENCIL WITH RABBIT ERASER	24	5.04
...	...	...	...	...	...
440819	31/10/2011	90161C	ANT COPPER LIME BOUDICCA BRACELET	1	4.95
440820	31/10/2011	BANK CHARGES	Bank Charges	1	15.00
440821	31/10/2011	C2	CARRIAGE	1	50.00
440822	31/10/2011	DOT	DOTCOM POSTAGE	1	901.58
440823	31/10/2011	POST	POSTAGE	9	175.00

440824 rows × 5 columns

In [35]:
```
# Create new column called "InvoiceDate_forSorting" for sorting InvoiceDate column
# Because InvoiceDate is string and can't be sorted correctly compared to datetime
transaction_df = transaction_df.assign(InvoiceDate_forSorting=pd.to_datetime(transaction_df['InvoiceDate'], format='%

# Sort by 'InvoiceDate' in ascending order
transaction_df_sorted = transaction_df.sort_values(by='InvoiceDate_forSorting', ascending=True)

# Delete InvoiceDate_forSorting column since we don't need it in final result
transaction_df_sorted.drop(columns=['InvoiceDate_forSorting'], inplace=True)

# Copy aggregated and sorted dataframe into new variable
daily_sales_df = transaction_df_sorted.copy()

# save dataframe into a csv file
daily_sales_df.to_csv('daily_transaction.csv', index=False)

daily_sales_df
```

Out[35]:

	InvoiceDate	StockCode	Description	ItemSold	TotalSales
12249	01/12/2009	21186	WHITE DOVE HONEYCOMB PAPER GARLAND	2	3.30
12673	01/12/2009	22315	200 RED + WHITE BENDY STRAWS	3	3.75
12672	01/12/2009	22305	COFFEE MUG PINK PAISLEY DESIGN	3	7.65
12671	01/12/2009	22304	COFFEE MUG BLUE PAISLEY DESIGN	3	7.65
12670	01/12/2009	22303	COFFEE MUG APPLES DESIGN	3	7.65
...	...	...	...	...	...
135638	09/12/2011	22331	WOODLAND PARTY BAG + STICKER SET	8	13.20
135637	09/12/2011	22328	ROUND SNACK BOXES SET OF 4 FRUITS	12	35.40
135636	09/12/2011	22326	ROUND SNACK BOXES SET OF4 WOODLAND	18	53.10
135634	09/12/2011	22314	OFFICE MUG WARMER CHOC+BLUE	24	30.00
135584	09/12/2011	22059	CERAMIC STRAWBERRY DESIGN MUG	36	14.04

440824 rows × 5 columns

2. Customer Dataframe (per-customer summary)

In [37]:

```
customer_df = cleaned_df.copy()

# Reusing of preprocessed dataframe
customer_df = customer_df.assign(InvoiceDate=pd.to_datetime(customer_df['InvoiceDate'], format='%d/%m/%Y'))

# Create TotalSpending column for aggregating
customer_df['TotalSpending'] = customer_df['Quantity'] * customer_df['Price']

# Find maximun date for Recency calculation
max_date = customer_df['InvoiceDate'].max()

# Aggregate data using columns {CustomerID, sum(TotalSpending), count(CustomerID), max(LastPurchase)}
customer_df = customer_df.groupby(['Customer_ID'], as_index=False).agg(
    TotalSpending=('TotalSpending', 'sum'),
    Frequency=('Customer_ID', 'count'),
    LastPurchase=('InvoiceDate', 'max')
)

# Convert LastPurchase column into datetime for Recency calculation
customer_df = customer_df.assign(LastPurchase = pd.to_datetime(customer_df['LastPurchase']))

# Recency calculation
customer_df['Recency'] = (max_date - customer_df['LastPurchase']).dt.days

# Delete LastPurchase column since we don't need it in final result
customer_df.drop(columns=['LastPurchase'], inplace=True)

# Reordering column to match instruction
reorder_columns = ['Customer_ID', 'TotalSpending', 'Recency', 'Frequency']
customer_df = customer_df.reindex(columns=reorder_columns)

# Copy aggregated dataframe into new variable
customer_summary_df = customer_df.copy()

# save dataframe into a csv file
customer_summary_df.to_csv('customer_data.csv', index=False)


customer_summary_df
```

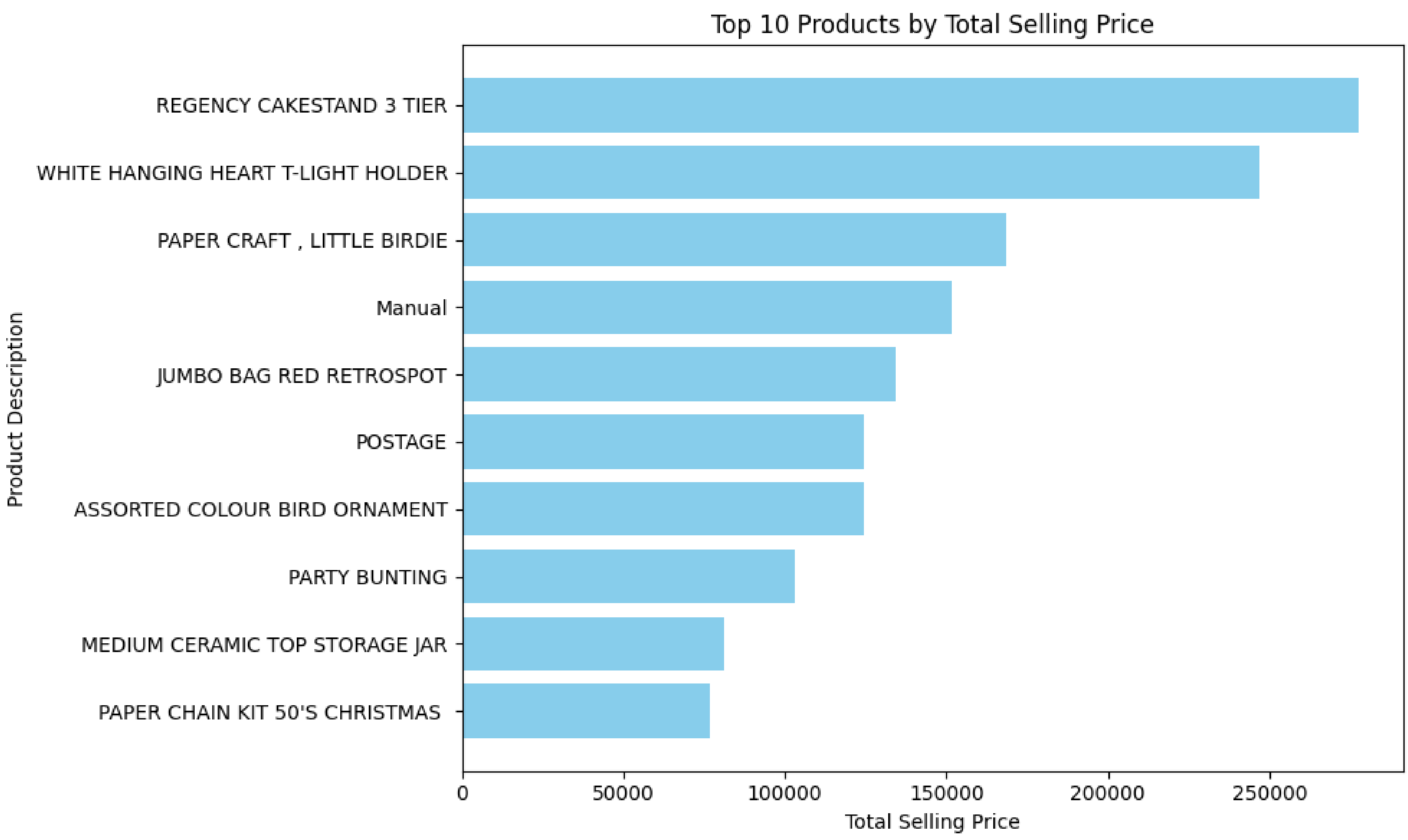
Out[37]:

	Customer_ID	TotalSpending	Recency	Frequency
0	12346	77556.46	325	34
1	12347	4921.53	2	222
2	12348	2019.40	75	51
3	12349	4428.69	18	175
4	12350	334.40	310	17
...	...	...	...	...
5876	18283	2664.90	3	938
5877	18284	461.68	431	28
5878	18285	427.00	660	12
5879	18286	1296.43	476	67
5880	18287	4182.99	42	155

5881 rows × 4 columns

3. Top 10 selling products

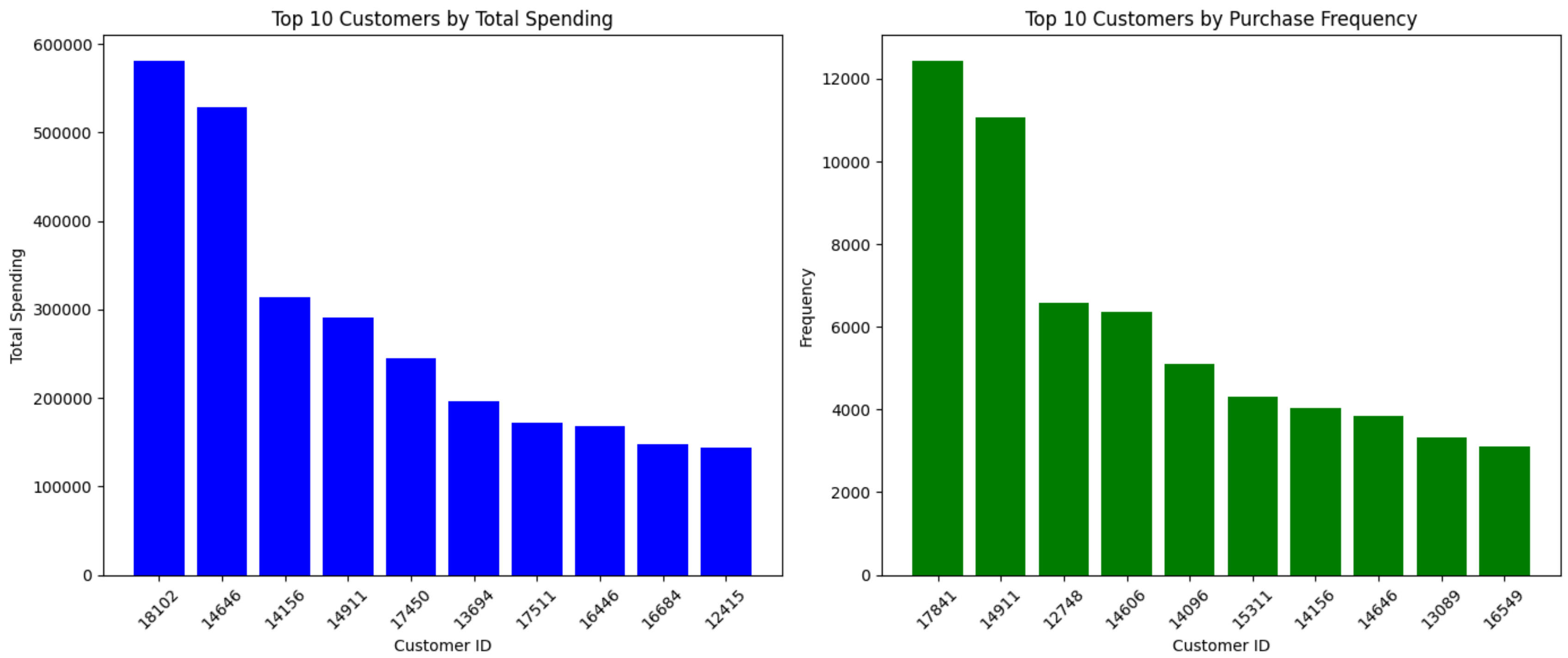
In [83]:
```
cleaned_df['TotalSales'] = cleaned_df['Quantity'] * cleaned_df['Price']

top_products_by_price = cleaned_df.groupby(['StockCode', 'Description'], as_index=False).agg({'TotalSales': 'sum'})

top_10_products_by_price = top_products_by_price.sort_values(by='TotalSales', ascending=False).head(10)



plt.figure(figsize=(10, 6))
plt.barh(top_10_products_by_price['Description'], top_10_products_by_price['TotalSales'], color='skyblue')
plt.xlabel('Total Selling Price')
plt.ylabel('Product Description')
plt.title('Top 10 Products by Total Selling Price')

# Invert the y-axis to have the highest selling product at the top
plt.gca().invert_yaxis()

# Show the plot
plt.tight_layout()
plt.show()
```



4. Top 10 customers

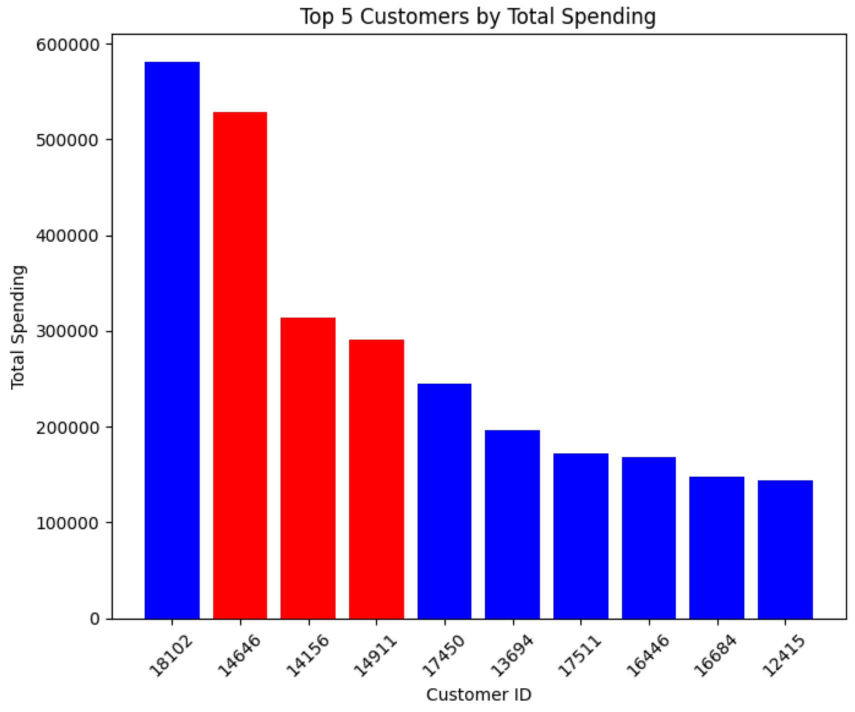
In [84]:
```
# Top 10 customers with the most total spending
top_10_spending = customer_summary_df.nlargest(10, 'TotalSpending')

# Top 10 customers with the most frequency
top_10_frequency = customer_summary_df.nlargest(10, 'Frequency')
```

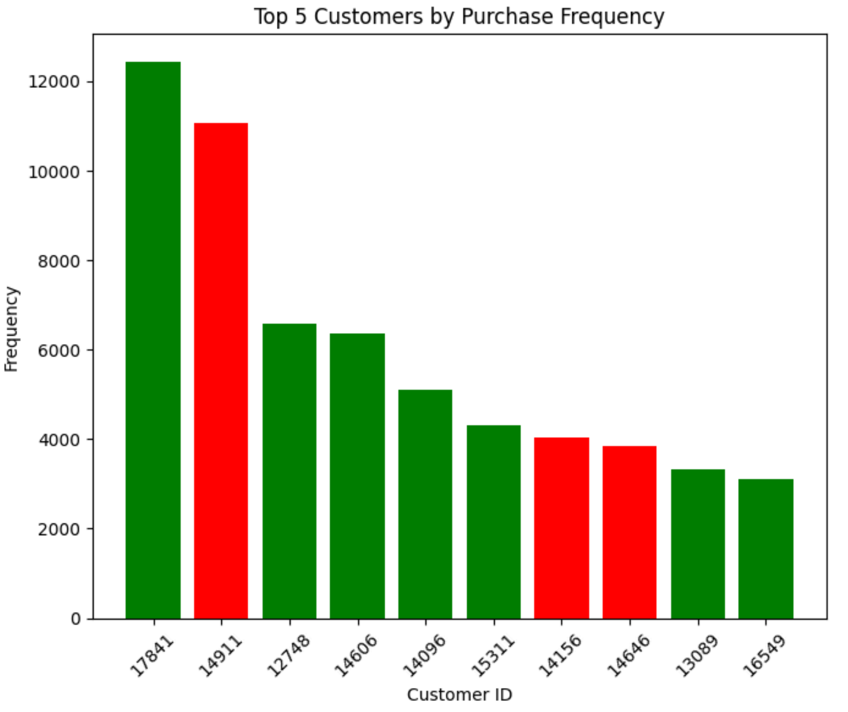
```python
# Plotting the data
plt.figure(figsize=(14, 6))

# Subplot for top 5 total spending
plt.subplot(1, 2, 1)
plt.bar(top_10_spending['Customer_ID'].astype(str), top_10_spending['TotalSpending'], color='blue')
plt.title('Top 10 Customers by Total Spending')
plt.xlabel('Customer ID')
plt.ylabel('Total Spending')
plt.xticks(rotation=45)

# Subplot for top 5 frequency
plt.subplot(1, 2, 2)
plt.bar(top_10_frequency['Customer_ID'].astype(str), top_10_frequency['Frequency'], color='green')
plt.title('Top 10 Customers by Purchase Frequency')
plt.xlabel('Customer ID')
plt.ylabel('Frequency')
plt.xticks(rotation=45)

# Adjust layout
plt.tight_layout()

# Display the plot
plt.show()
```



In [85]:

```python
# Simply identify customers who have both high spending and frequency of buying

common_customers = set(top_10_spending['Customer_ID']).intersection(set(top_10_frequency['Customer_ID']))

# Plotting the data
plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
colors_spending = ['red' if customer in common_customers else 'blue' for customer in top_10_spending['Customer_ID']]
plt.bar(top_10_spending['Customer_ID'].astype(str), top_10_spending['TotalSpending'], color=colors_spending)
plt.title('Top 5 Customers by Total Spending')
plt.xlabel('Customer ID')
plt.ylabel('Total Spending')
plt.xticks(rotation=45)

# Subplot for top 5 frequency
plt.subplot(1, 2, 2)
colors_frequency = ['red' if customer in common_customers else 'green' for customer in top_10_frequency['Customer_ID
plt.bar(top_10_frequency['Customer_ID'].astype(str), top_10_frequency['Frequency'], color=colors_frequency)
plt.title('Top 5 Customers by Purchase Frequency')
plt.xlabel('Customer ID')
plt.ylabel('Frequency')
plt.xticks(rotation=45)

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()
```

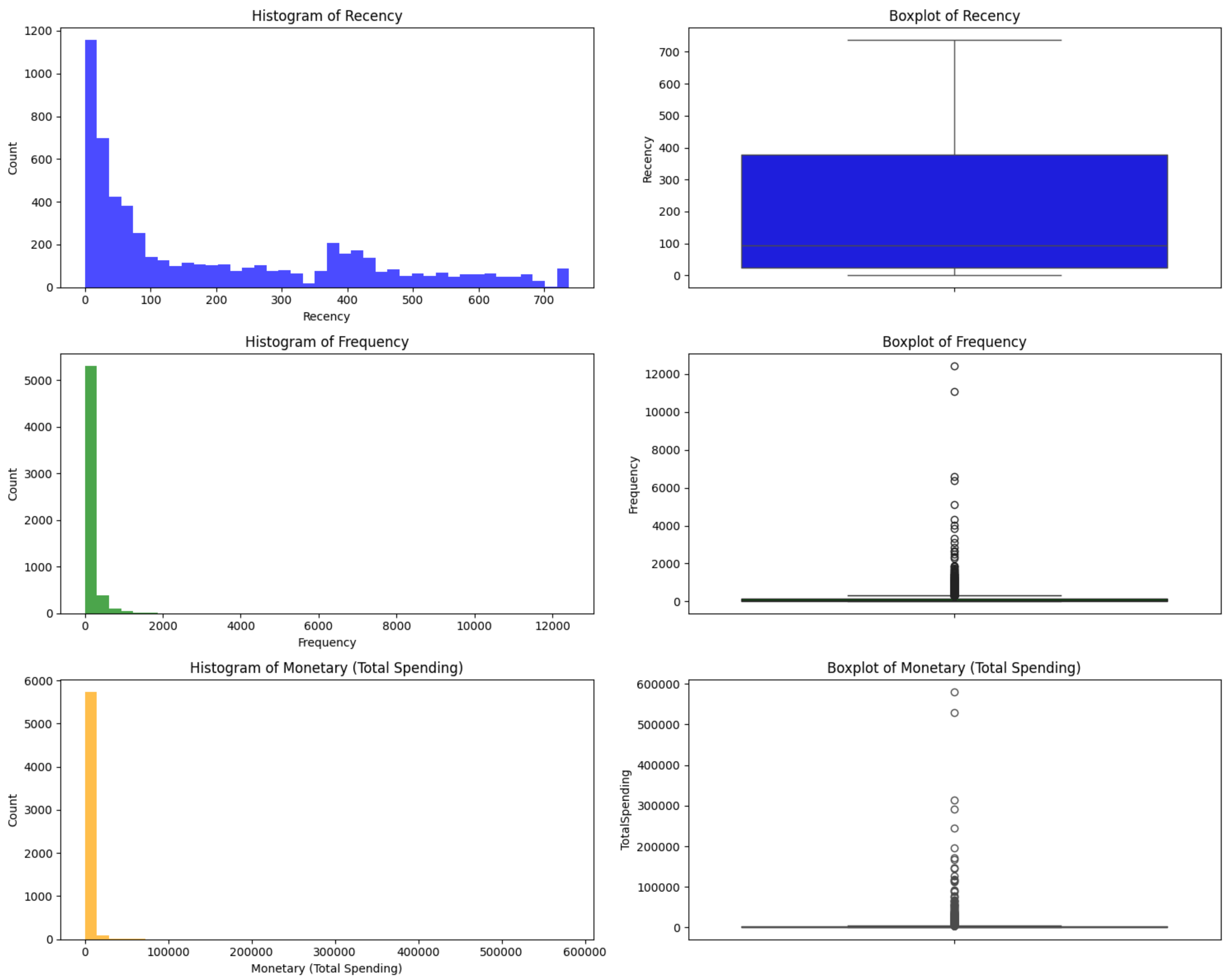




5. K-mean clustering of customers

In [44]:

```
fig, axes = plt.subplots(3, 2, figsize=(15, 12))

# Recency
# Histogram for Recency
axes[0, 0].hist(customer_summary_df['Recency'], bins=40, color='blue', alpha=0.7)
axes[0, 0].set_title('Histogram of Recency')
axes[0, 0].set_xlabel('Recency')
axes[0, 0].set_ylabel('Count')

# Boxplot for Recency
sns.boxplot(data=customer_summary_df, y='Recency', ax=axes[0, 1], color='blue')
axes[0, 1].set_title('Boxplot of Recency')

# Frequency
# Histogram for Frequency
axes[1, 0].hist(customer_summary_df['Frequency'], bins=40, color='green', alpha=0.7)
axes[1, 0].set_title('Histogram of Frequency')
axes[1, 0].set_xlabel('Frequency')
axes[1, 0].set_ylabel('Count')

# Boxplot for Frequency
sns.boxplot(data=customer_summary_df, y='Frequency', ax=axes[1, 1], color='green')
axes[1, 1].set_title('Boxplot of Frequency')

# Monetary (Total Spending)
# Histogram for Monetary
axes[2, 0].hist(customer_summary_df['TotalSpending'], bins=40, color='orange', alpha=0.7)
axes[2, 0].set_title('Histogram of Monetary (Total Spending)')
axes[2, 0].set_xlabel('Monetary (Total Spending)')
axes[2, 0].set_ylabel('Count')

# Boxplot for Monetary
sns.boxplot(data=customer_summary_df, y='TotalSpending', ax=axes[2, 1], color='orange')
axes[2, 1].set_title('Boxplot of Monetary (Total Spending)')

# Adjust layout to avoid overlap
plt.tight_layout()
plt.show()
```



In [ ]:

```
"""
From the histogram and box plot charts of RFM, we can see that the histogram of Recency is heavily right-skewed
which means the data is heavily distributed on the left-hand side of the chart.
On the other hands, Frequency and Monetary has the same behavior that the data only stick together at low values,
meaning there are many outliers.

However, we can't just delete these outliers from the dataset since those are valuable customers which spend with us
So, a stradegy here is to categorize into 2 groups: normal customers and outlier customers
"""
```

In [52]:

```
# Monetary outliers

M_Q1 = customer_summary_df["TotalSpending"].quantile(0.25)
M_Q3 = customer_summary_df["TotalSpending"].quantile(0.75)
M_IQR = M_Q3 - M_Q1

outlier_lowerbound = (M_Q1 - 1.5 * M_IQR)
outlier_upperbound = (M_Q3 + 1.5 * M_IQR)

monetary_outliers_df = customer_summary_df[(customer_summary_df["TotalSpending"] < outlier_lowerbound) | (customer_su

monetary_outliers_df.describe()
```

Out[52]:

	TotalSpending	Recency	Frequency
count	633.000000	633.000000	633.000000
mean	17922.198038	52.491311	564.679305
std	40949.203472	103.891148	888.695677
min	5114.230000	0.000000	1.000000
25%	6427.660000	5.000000	232.000000
50%	8839.670000	17.000000	363.000000
75%	14093.710000	50.000000	651.000000
max	580987.040000	691.000000	12435.000000

In [53]:

```
# Frequency outliers

M_Q1 = customer_summary_df["Frequency"].quantile(0.25)
M_Q3 = customer_summary_df["Frequency"].quantile(0.75)
M_IQR = M_Q3 - M_Q1

outlier_lowerbound = (M_Q1 - 1.5 * M_IQR)
outlier_upperbound = (M_Q3 + 1.5 * M_IQR)
```

```
frequency_outliers_df = customer_summary_df[(customer_summary_df["Frequency"] < outlier_lowerbound) | (customer_summa

frequency_outliers_df.describe()
```

Out[53]:

	TotalSpending	Recency	Frequency
count	567.000000	567.000000	567.000000
mean	15364.407647	41.446208	706.179894
std	42394.600968	78.737971	895.703630
min	1027.030000	0.000000	316.000000
25%	3899.180000	4.000000	377.000000
50%	7217.750000	15.000000	487.000000
75%	12090.920000	38.500000	737.500000
max	580987.040000	551.000000	12435.000000

In [54]:

```
# Recency outliers

M_Q1 = customer_summary_df["Recency"].quantile(0.25)
M_Q3 = customer_summary_df["Recency"].quantile(0.75)
M_IQR = M_Q3 - M_Q1

outlier_lowerbound = (M_Q1 - 1.5 * M_IQR)
outlier_upperbound = (M_Q3 + 1.5 * M_IQR)

recency_outliers_df = customer_summary_df[(customer_summary_df["Recency"] < outlier_lowerbound) | (customer_summary_c

recency_outliers_df.describe()
```

Out[54]:

	TotalSpending	Recency	Frequency
count	0.0	0.0	0.0
mean	NaN	NaN	NaN
std	NaN	NaN	NaN
min	NaN	NaN	NaN
25%	NaN	NaN	NaN
50%	NaN	NaN	NaN
75%	NaN	NaN	NaN
max	NaN	NaN	NaN

In [55]:

```
non_outliers_df = customer_summary_df[(~customer_summary_df.index.isin(monetary_outliers_df.index))
                                      & (~customer_summary_df.index.isin(frequency_outliers_df.index))]

non_outliers_df.describe()
```

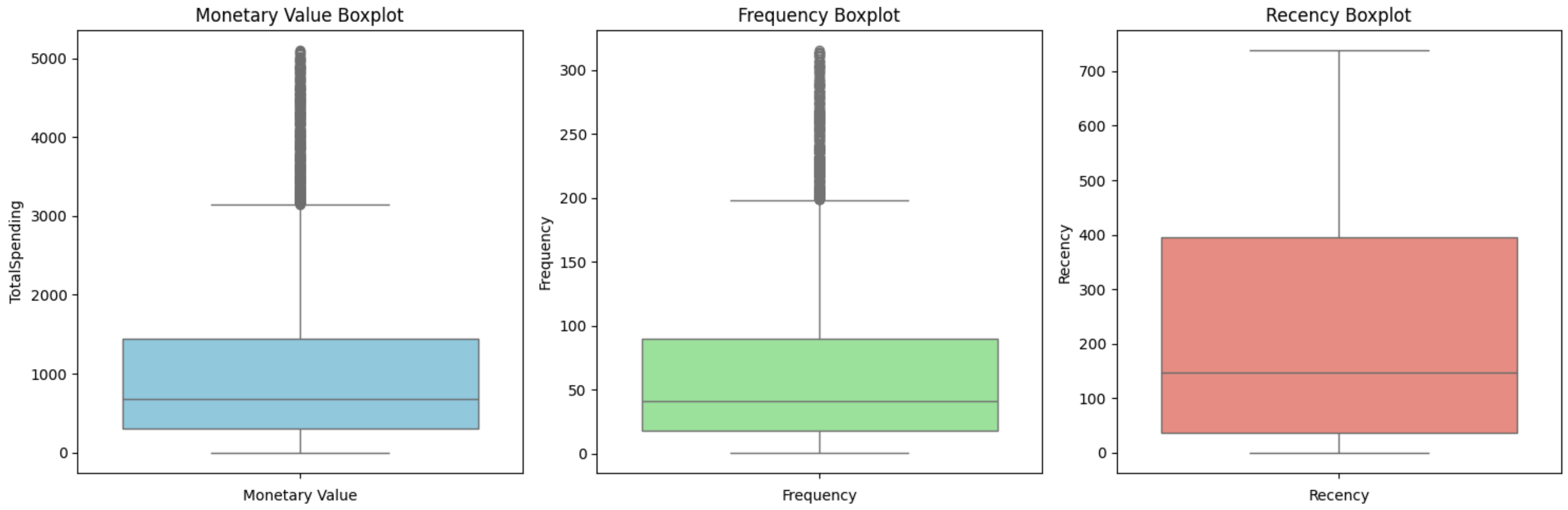
Out[55]:

	TotalSpending	Recency	Frequency
count	5052.000000	5052.000000	5052.000000
mean	1073.364768	224.936263	65.479810
std	1082.455936	212.750822	66.018747
min	0.000000	0.000000	1.000000
25%	304.247500	36.000000	18.000000
50%	675.620000	147.000000	41.000000
75%	1442.662500	396.000000	90.250000
max	5097.180000	738.000000	315.000000

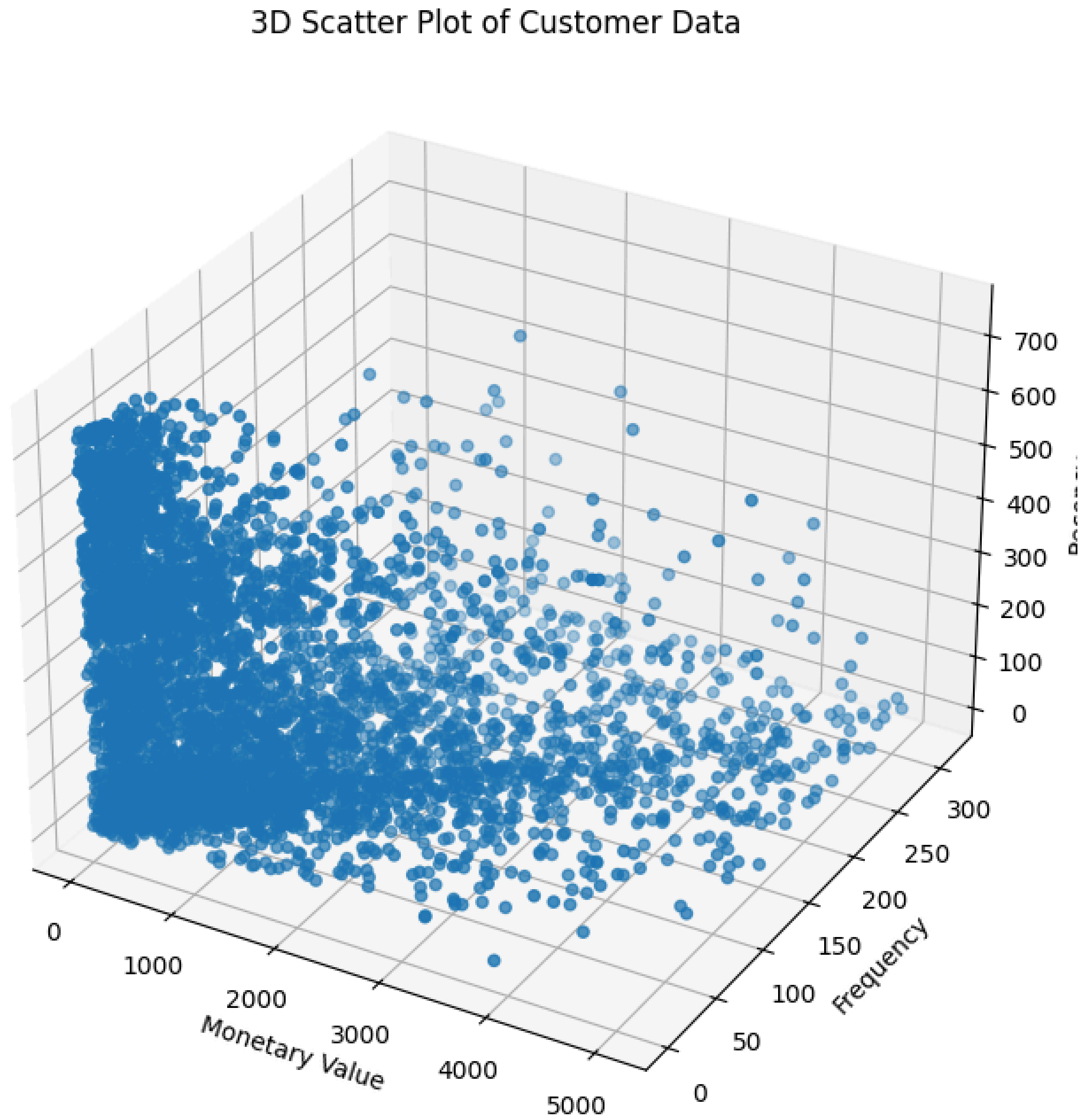
In [56]:

```
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
sns.boxplot(data=non_outliers_df['TotalSpending'], color='skyblue')
plt.title('Monetary Value Boxplot')
plt.xlabel('Monetary Value')

plt.subplot(1, 3, 2)
sns.boxplot(data=non_outliers_df['Frequency'], color='lightgreen')
plt.title('Frequency Boxplot')
plt.xlabel('Frequency')

plt.subplot(1, 3, 3)
sns.boxplot(data=non_outliers_df['Recency'], color='salmon')
plt.title('Recency Boxplot')
```

```python
plt.xlabel('Recency')

plt.tight_layout()
plt.show()
```



In [57]:
```python
fig = plt.figure(figsize=(8, 8))

ax = fig.add_subplot(projection="3d")

scatter = ax.scatter(non_outliers_df["TotalSpending"], non_outliers_df["Frequency"], non_outliers_df["Recency"])

ax.set_xlabel('Monetary Value')
ax.set_ylabel('Frequency')
ax.set_zlabel('Recency')

ax.set_title('3D Scatter Plot of Customer Data')

plt.show()
```



In [ ]:
```python
"""
We can see that the scales of RFM is very different which make it harder for clustering.
So, we will use standard scaling to scale the data with [0, 1] scale
"""
```

In [58]:
```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaled_data = scaler.fit_transform(non_outliers_df[["TotalSpending", "Frequency", "Recency"]])

scaled_data
```

Out[58]:
```
array([[ 3.55538366,  2.37107929, -1.0479787 ],
       [ 0.87405763, -0.21935047, -0.70482033],
       [ 3.10004061,  1.65908982, -0.97276591],
       ...,
       [-0.59718713, -0.81015024,  2.04514746],
       [ 0.20609367,  0.02302892,  1.18020032],
       [ 2.87303431,  1.35611558, -0.85994672]])
```

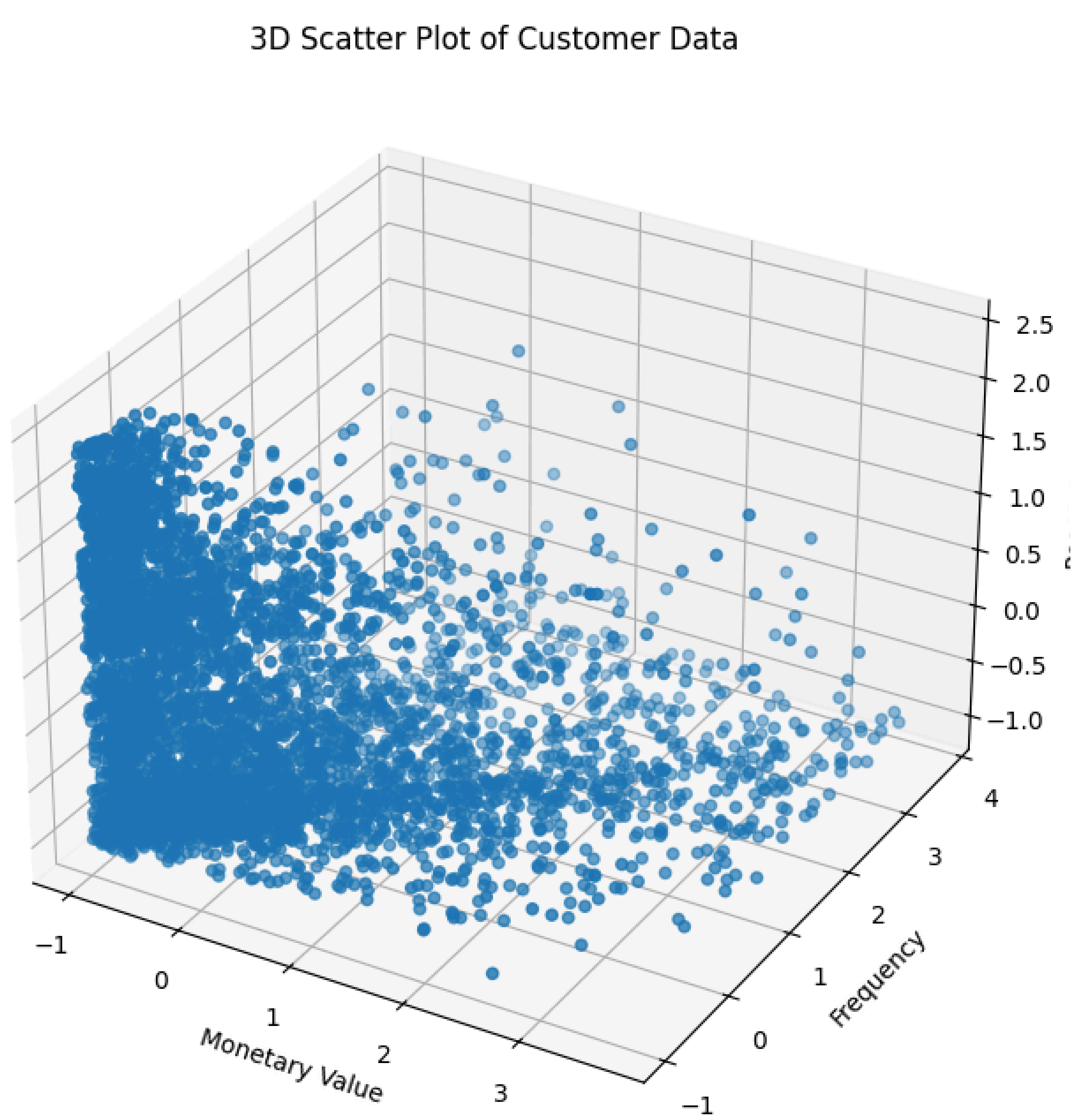
In [59]:
```
scaled_data_df = pd.DataFrame(scaled_data, index=non_outliers_df.index, columns=("TotalSpending", "Frequency", "Recen

scaled_data_df
```

Out[59]:

	TotalSpending	Frequency	Recency
1	3.555384	2.371079	-1.047979
2	0.874058	-0.219350	-0.704820
3	3.100041	1.659090	-0.972766
4	-0.682742	-0.734407	0.399868
5	-0.713665	-0.673812	0.705420
...	...	...	...
5875	-0.827196	-0.810150	-1.024475
5877	-0.565146	-0.567771	0.968664
5878	-0.597187	-0.810150	2.045147
5879	0.206094	0.023029	1.180200
5880	2.873034	1.356116	-0.859947

5052 rows × 3 columns

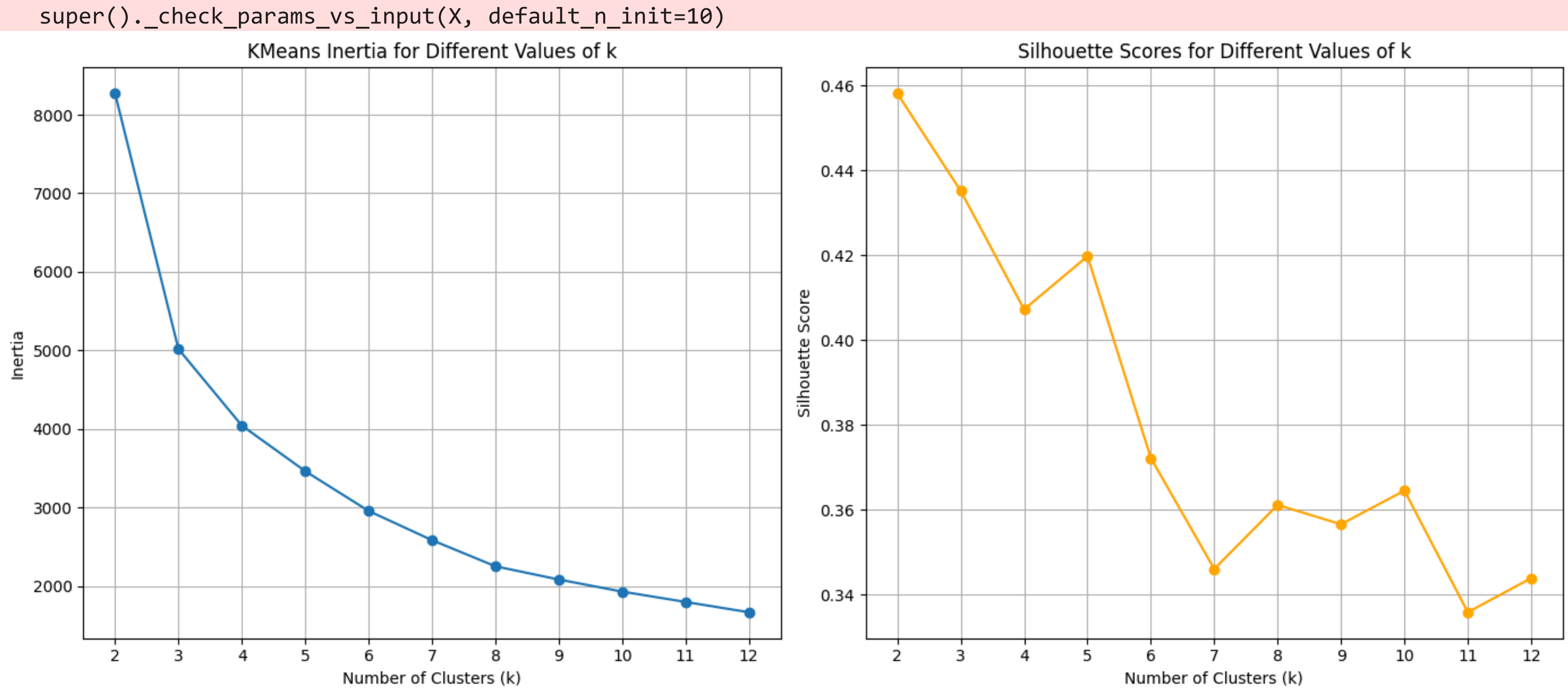
In [60]:
```
fig = plt.figure(figsize=(8, 8))

ax = fig.add_subplot(projection="3d")

scatter = ax.scatter(scaled_data_df["TotalSpending"], scaled_data_df["Frequency"], scaled_data_df["Recency"])

ax.set_xlabel('Monetary Value')
ax.set_ylabel('Frequency')
ax.set_zlabel('Recency')

ax.set_title('3D Scatter Plot of Customer Data')

plt.show()
```



In [ ]:
```
"""
find the optimal number of clusters by plotting the Within-Cluster Sum of Squared Errors (WCSS) or Inertia
Inertia is basically the average distance between each data point and its centroid

The Silhouette Score measures how similar a point is to its own cluster compared to other clusters.
The score ranges from -1 to 1, with higher values indicating that the clusters are well separated and compact.
A silhouette score close to 1 indicates that the clusters are dense and well-separated
"""
```

In [61]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

max_k = 12

inertia = []
silhoutte_scores = []
k_values = range(2, max_k + 1)

for k in k_values:

    kmeans = KMeans(n_clusters=k, random_state=42, max_iter=1000)

    cluster_labels = kmeans.fit_predict(scaled_data_df)

    sil_score = silhouette_score(scaled_data_df, cluster_labels)

    silhoutte_scores.append(sil_score)

    inertia.append(kmeans.inertia_)

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.plot(k_values, inertia, marker='o')
plt.title('KMeans Inertia for Different Values of k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.xticks(k_values)
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(k_values, silhoutte_scores, marker='o', color='orange')
plt.title('Silhouette Scores for Different Values of k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette Score')
plt.xticks(k_values)
plt.grid(True)

plt.tight_layout()
plt.show()
```

```
c:\Users\User\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarnin
g: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\User\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarnin
g: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\User\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarnin
g: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\User\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarnin
g: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\User\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarnin
g: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\User\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarnin
g: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\User\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarnin
g: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\User\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarnin
g: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\User\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarnin
g: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\User\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarnin
g: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\User\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarnin
g: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss the warning
  super()._check_params_vs_input(X, default_n_init=10)
```



In [ ]:
```
"""
From the charts, we can see that at k=4 the line slowly decrease
and sihouette score of k=2, 3, 4 is not that much different.
So, from these information we will assume that k=4 is the optimal number of cluster
"""
```

In [62]:
```
kmeans = KMeans(n_clusters=4, random_state=42, max_iter=1000)

cluster_labels = kmeans.fit_predict(scaled_data_df)

cluster_labels
```

```
c:\Users\User\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarnin
g: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppre
ss the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

Out[62]:
```
array([2, 3, 2, ..., 1, 1, 2])
```

In [63]:
```
# Assign cluster's name to each record
non_outliers_df["Cluster"] = cluster_labels

non_outliers_df
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_35496\2187940272.py:2: 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
  non_outliers_df["Cluster"] = cluster_labels
```

Out[63]:

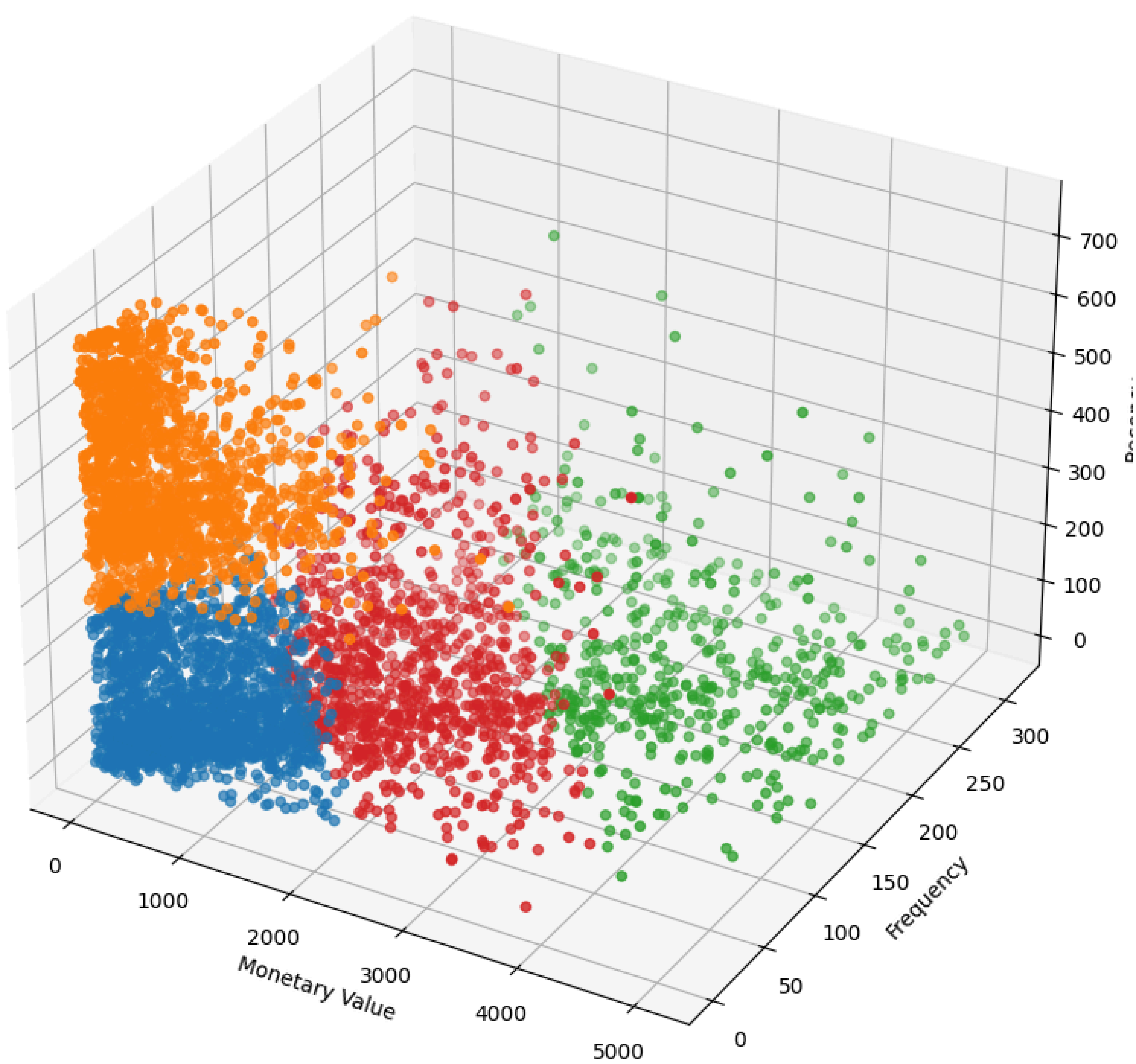
	Customer_ID	TotalSpending	Recency	Frequency	Cluster
1	12347	4921.53	2	222	2
2	12348	2019.40	75	51	3
3	12349	4428.69	18	175	2
4	12350	334.40	310	17	1
5	12351	300.93	375	21	1
...	...	...	...	...	...
5875	18282	178.05	7	12	0
5877	18284	461.68	431	28	1
5878	18285	427.00	660	12	1
5879	18286	1296.43	476	67	1
5880	18287	4182.99	42	155	2

5052 rows × 5 columns

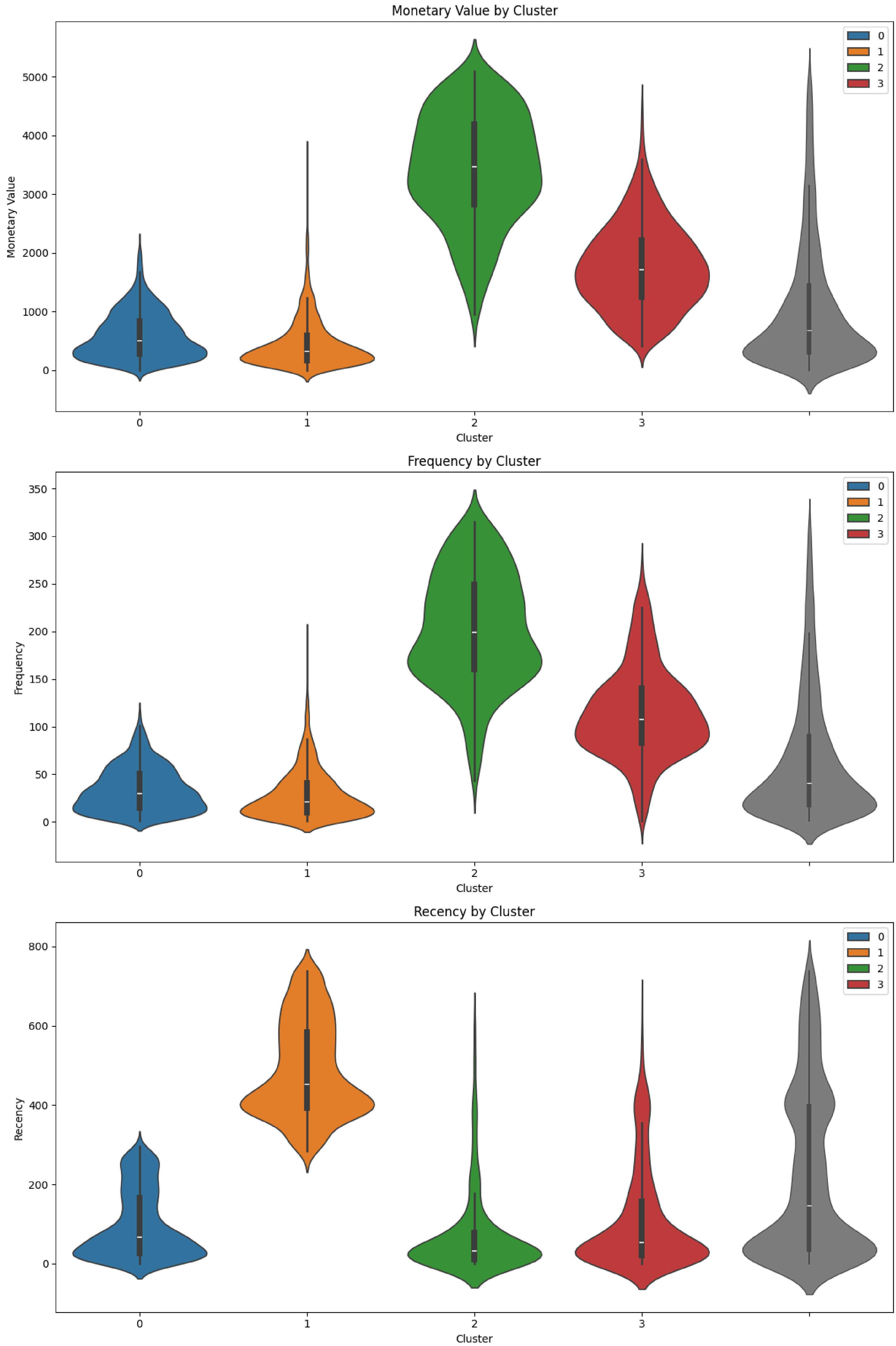
In [65]:

```
cluster_colors = {0: '#1f77b4',  # Blue
                  1: '#ff7f0e',  # Orange
                  2: '#2ca02c',  # Green
                  3: '#d62728'}  # Red

colors = non_outliers_df['Cluster'].map(cluster_colors)

fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(projection='3d')

scatter = ax.scatter(non_outliers_df['TotalSpending'],
                     non_outliers_df['Frequency'],
                     non_outliers_df['Recency'],
                     c=colors,  # Use mapped solid colors
                     marker='o')

ax.set_xlabel('Monetary Value')
ax.set_ylabel('Frequency')
ax.set_zlabel('Recency')

ax.set_title('3D Scatter Plot of Customer Data by Cluster')

plt.show()
```

3D Scatter Plot of Customer Data by Cluster



In [66]:
```
plt.figure(figsize=(12, 18))

plt.subplot(3, 1, 1)
sns.violinplot(x=non_outliers_df['Cluster'], y=non_outliers_df['TotalSpending'], palette=cluster_colors, hue=non_outl
sns.violinplot(y=non_outliers_df['TotalSpending'], color='gray', linewidth=1.0)
plt.title('Monetary Value by Cluster')
plt.ylabel('Monetary Value')

plt.subplot(3, 1, 2)
sns.violinplot(x=non_outliers_df['Cluster'], y=non_outliers_df['Frequency'], palette=cluster_colors, hue=non_outliers
sns.violinplot(y=non_outliers_df['Frequency'], color='gray', linewidth=1.0)
plt.title('Frequency by Cluster')
plt.ylabel('Frequency')


plt.subplot(3, 1, 3)
sns.violinplot(x=non_outliers_df['Cluster'], y=non_outliers_df['Recency'], palette=cluster_colors, hue=non_outliers_
sns.violinplot(y=non_outliers_df['Recency'], color='gray', linewidth=1.0)
plt.title('Recency by Cluster')
plt.ylabel('Recency')

plt.tight_layout()
plt.show()
```



Non-outlier customer clusters

Cluster 0 : Potential Loyalist

- Violin plot analysis => their monetary and frequency are almost equal to median of total data and recency is very low
- Characteristic => They are customer who didn't make much purchase but spend quite high and they just recenty spent something with us.
- Suggestion => We should make a haste to make these customers feel impressed and confident in our brand

Cluster 1 : Hibernating

- Violin plot analysis => their monetary and frequency are quite low compared to all data but recency is very high
- Characteristic => They have average payment and frequency but it's been a long since their last purchase
- Suggestion => They are still worth to bring them back by doing some campaigns or promotions

Cluster 2 : Champion

- Violin plot analysis => their monetary and frequency are very high compared to which of all data and recency is also very low
- Characteristic => The most important customer which frequently buy and spent a lot of money recently.
- Suggestion => We need to keep them for the best and don't let them go no matter what

Cluster 3 : Loyal Customers

- Violin plot analysis => they have similiar RFM's behavior with cluster 2 but tend to have much lower value except for recency which is a little higher
- Characteristic => They are valueable customers with high spending and frequency of buying even they didn't make any order recenly.
- Suggestion => We should do research on what they like and the reasons they buy our products to deliver straightforward services

In [67]:

```
"""
Now, we will try to visualize the outlier data to see its clustering
"""

overlap_indices = monetary_outliers_df.index.intersection(frequency_outliers_df.index)

monetary_only_outliers = monetary_outliers_df.drop(overlap_indices)
frequency_only_outliers = frequency_outliers_df.drop(overlap_indices)
monetary_and_frequency_outliers = monetary_outliers_df.loc[overlap_indices]

monetary_only_outliers["Cluster"] = -1
frequency_only_outliers["Cluster"] = -2
monetary_and_frequency_outliers["Cluster"] = -3

outlier_clusters_df = pd.concat([monetary_only_outliers, frequency_only_outliers, monetary_and_frequency_outliers])

outlier_clusters_df
```

Out[67]:

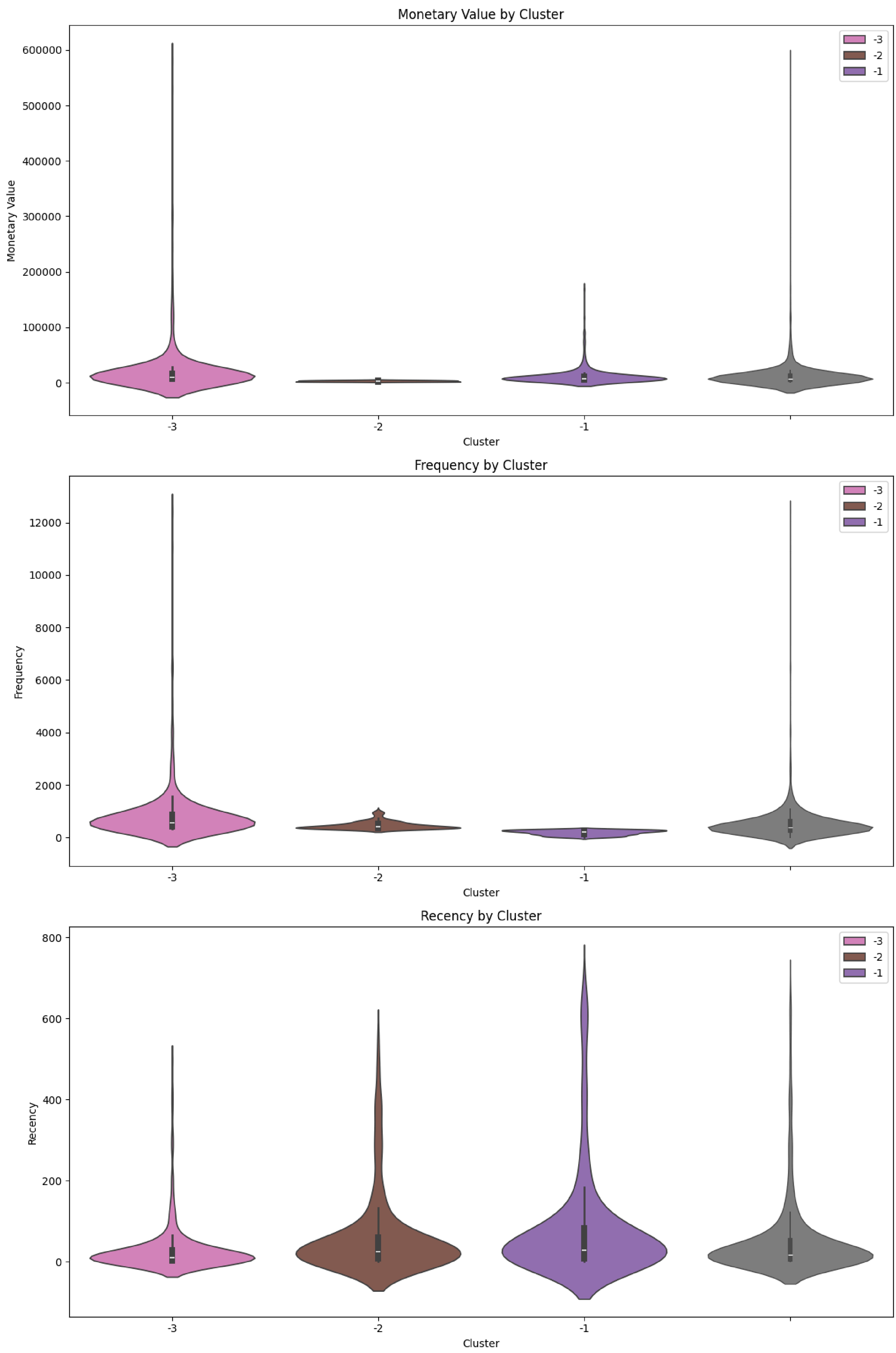
	Customer_ID	TotalSpending	Recency	Frequency	Cluster
0	12346	77556.46	325	34	-1
10	12356	6371.73	22	142	-1
11	12357	18287.66	33	296	-1
16	12362	5356.23	3	267	-1
32	12378	5416.32	129	301	-1
...	...	...	...	...	...
5818	18225	13014.80	3	572	-3
5819	18226	11878.88	44	528	-3
5824	18231	6854.17	192	383	-3
5838	18245	6324.98	7	458	-3
5853	18260	9947.26	172	410	-3

829 rows × 5 columns

In [68]:

```
cluster_colors = {-1: '#9467bd',
                  -2: '#8c564b',
                  -3: '#e377c2'}

plt.figure(figsize=(12, 18))

plt.subplot(3, 1, 1)
sns.violinplot(x=outlier_clusters_df['Cluster'], y=outlier_clusters_df['TotalSpending'], palette=cluster_colors, hue=
sns.violinplot(y=outlier_clusters_df['TotalSpending'], color='gray', linewidth=1.0)
plt.title('Monetary Value by Cluster')
plt.ylabel('Monetary Value')

plt.subplot(3, 1, 2)
sns.violinplot(x=outlier_clusters_df['Cluster'], y=outlier_clusters_df['Frequency'], palette=cluster_colors, hue=outl
sns.violinplot(y=outlier_clusters_df['Frequency'], color='gray', linewidth=1.0)
```

```
plt.title('Frequency by Cluster')
plt.ylabel('Frequency')

plt.subplot(3, 1, 3)
sns.violinplot(x=outlier_clusters_df['Cluster'], y=outlier_clusters_df['Recency'], palette=cluster_colors, hue=outlie
sns.violinplot(y=outlier_clusters_df['Recency'], color='gray', linewidth=1.0)
plt.title('Recency by Cluster')
plt.ylabel('Recency')

plt.tight_layout()
plt.show()
```



Outlier customer clusters

Cluster -1 (Monetary Outliers)

- Characteristic => High spenders but not necessarily frequent buyers. Their purchases are large but infrequent
- Suggestion => Focus on maintaining their loyalty with personalized offers that align with their amount of spending

Cluster -2 (Frequency Outliers):

- Characteristic => Frequent buyers who spend less per purchase. These customers are consistently engaged but might benefit from upselling opportunities
- Suggestion => Implement loyalty programs or bundle deals to encourage higher spending per visit

Cluster -3 (Monetary & Frequency Outliers):

- Characteristic => extreme spending and frequent purchases. They are likely your top-tier customers who require special attention
- Suggestion => Develop VIP programs or exclusive offers to maintain their loyalty and encourage continued engagement

In [69]:
```
cluster_labels = {
    0: "Potential Loyalist",
    1: "Hibernating",
    2: "Champion",
    3: "Loyal Customers",
    -1: "Monetary Outliers",
    -2: "Frequency Outliers",
    -3: "Monetary & Frequency Outliers"
}
```

In [70]:
```
full_clustering_df = pd.concat([non_outliers_df, outlier_clusters_df])

full_clustering_df
```

Out[70]:

	Customer_ID	TotalSpending	Recency	Frequency	Cluster
1	12347	4921.53	2	222	2
2	12348	2019.40	75	51	3
3	12349	4428.69	18	175	2
4	12350	334.40	310	17	1
5	12351	300.93	375	21	1
...	...	...	...	...	...
5818	18225	13014.80	3	572	-3
5819	18226	11878.88	44	528	-3
5824	18231	6854.17	192	383	-3
5838	18245	6324.98	7	458	-3
5853	18260	9947.26	172	410	-3

5881 rows × 5 columns

In [71]:
```
full_clustering_df["ClusterLabel"] = full_clustering_df["Cluster"].map(cluster_labels)

full_clustering_df
```

Out[71]:

	Customer_ID	TotalSpending	Recency	Frequency	Cluster	ClusterLabel
1	12347	4921.53	2	222	2	Champion
2	12348	2019.40	75	51	3	Loyal Customers
3	12349	4428.69	18	175	2	Champion
4	12350	334.40	310	17	1	Hibernating
5	12351	300.93	375	21	1	Hibernating
...	...	...	...	...	...	...
5818	18225	13014.80	3	572	-3	Monetary & Frequency Outliers
5819	18226	11878.88	44	528	-3	Monetary & Frequency Outliers
5824	18231	6854.17	192	383	-3	Monetary & Frequency Outliers
5838	18245	6324.98	7	458	-3	Monetary & Frequency Outliers
5853	18260	9947.26	172	410	-3	Monetary & Frequency Outliers

5881 rows × 6 columns

Summary Visualization

In [77]:

```
cluster_counts = full_clustering_df['ClusterLabel'].value_counts()
full_clustering_df["MonetaryValue per 100 pounds"] = full_clustering_df["TotalSpending"] / 100.00
feature_means = full_clustering_df.groupby('ClusterLabel')[['Recency', 'Frequency', 'MonetaryValue per 100 pounds']]

fig, ax1 = plt.subplots(figsize=(18, 8))

sns.barplot(x=cluster_counts.index, y=cluster_counts.values, ax=ax1, palette='viridis', hue=cluster_counts.index)
ax1.set_ylabel('Number of Customers', color='b')
ax1.set_title('Cluster Distribution with Average Feature Values')

ax2 = ax1.twinx()

sns.lineplot(data=feature_means, ax=ax2, palette='Set2', marker='o')
ax2.set_ylabel('Average Value', color='g')

plt.show()
```