CUSTOMER ANALYSIS AND SEGMENTATION USING K-Means

Analysing and Segmenting Customers of an UK Based Online Giftware Store Using Feature Engineering, K Means Clustering and Visualisation

```
# importing relevant libraries and setting up pre-requisites

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler

# Setting to make numbers easier to read on display
pd.options.display.float_format = '{:20.2f}'.format

# Show all columns on output
pd.set_option('display.max_columns', 999)
```

DATA EXPLORATION

```
# Data Source https://archive.ics.uci.edu/dataset/502/online+retail+ii

df = pd.read_excel('/content/Data/online_retail_II.xlsx', sheet_name=0)

# check data
df.head(10)
```

	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	13085.00	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.00	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.00	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.00	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.00	United Kingdom
5	489434	22064	PINK DOUGHNUT TRINKET POT	24	2009-12-01 07:45:00	1.65	13085.00	United Kingdom
6	489434	21871	SAVE THE PLANET MUG	24	2009-12-01 07:45:00	1.25	13085.00	United Kingdom
7	489434	21523	FANCY FONT HOME SWEET HOME DOORMAT	10	2009-12-01 07:45:00	5.95	13085.00	United Kingdom
8	489435	22350	CAT BOWL	12	2009-12-01 07:46:00	2.55	13085.00	United Kingdom
9	489435	22349	DOG BOWL , CHASING BALL DESIGN	12	2009-12-01 07:46:00	3.75	13085.00	United Kingdom

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 525461 entries, 0 to 525460
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   Invoice      525461 non-null  object
 1   StockCode    525461 non-null  object
 2   Description  522533 non-null  object
 3   Quantity     525461 non-null  int64
 4   InvoiceDate  525461 non-null  datetime64[ns]
 5   Price        525461 non-null  float64
 6   Customer ID  417534 non-null  float64
 7   Country      525461 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 32.1+ MB
```

In the above code we can see that there are null customer id values The datatypes seem to be workable and not in need of conversion

```
df.describe()
```

	Quantity	InvoiceDate	Price	Customer ID
count	525461.00	525461	525461.00	417534.00
mean	10.34	2010-06-28 11:37:36.845017856	4.69	15360.65
min	-9600.00	2009-12-01 07:45:00	-53594.36	12346.00
25%	1.00	2010-03-21 12:20:00	1.25	13983.00
50%	3.00	2010-07-06 09:51:00	2.10	15311.00
75%	10.00	2010-10-15 12:45:00	4.21	16799.00
max	19152.00	2010-12-09 20:01:00	25111.09	18287.00
std	107.42	NaN	146.13	1680.81

Here the first thing we can notice is how the minimum quantity is -9600 which given the context of the data being from a sales platoform needs to be investigated Price also seems to have a negative value which is suspicious And again Customer Id data is missing a considerable amount of values.

```
# Looking at object data

df.describe(include='O')
```

	Invoice	StockCode	Description	Country
count	525461	525461	522533	525461
unique	28816	4632	4681	40
top	537434	85123A	WHITE HANGING HEART T-LIGHT HOLDER	United Kingdom
freq	675	3516	3549	485852

Invoice is considered a string 28k unique invoices 4632 stock codes 4681 descriptions It's intersting that the number of unique stock codes and description doesn't align

```
# Looking at the missing Customer Ids

df[df["Customer ID"].isna()].head(10)
```

	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
578	489525	85227	SET/6 3D KIT CARDS FOR KIDS	1	2009-12-01 11:49:00	0.85	NaN	United Kingdom
1055	489548	22271	FELTCRAFT DOLL ROSIE	1	2009-12-01 12:32:00	2.95	NaN	United Kingdom
1056	489548	22254	FELT TOADSTOOL LARGE	12	2009-12-01 12:32:00	1.25	NaN	United Kingdom
1057	489548	22273	FELTCRAFT DOLL MOLLY	3	2009-12-01 12:32:00	2.95	NaN	United Kingdom
1058	489548	22195	LARGE HEART MEASURING SPOONS	1	2009-12-01 12:32:00	1.65	NaN	United Kingdom

Since there is no customer id we should drop this data even for the transactions that look legitmate without negative quantities and the 0 price.

```
# Looking at the Negative Quantities

df[df["Quantity"] < 0].head(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	16321.00	Australia
179	C489449	85206A	CREAM FELT EASTER EGG BASKET	-6	2009-12-01 10:33:00	1.65	16321.00	Australia
180	C489449	21895	POTTING SHED SOW 'N' GROW SET	-4	2009-12-01 10:33:00	4.25	16321.00	Australia
181	C489449	21896	POTTING SHED TWINE	-6	2009-12-01 10:33:00	2.10	16321.00	Australia
182	C489449	22083	PAPER CHAIN KIT RETRO SPOT	-12	2009-12-01 10:33:00	2.95	16321.00	Australia
183	C489449	21871	SAVE THE PLANET MUG	-12	2009-12-01 10:33:00	1.25	16321.00	Australia
184	C489449	84946	ANTIQUE SILVER TEA GLASS ETCHED	-12	2009-12-01 10:33:00	1.25	16321.00	Australia
185	C489449	84970S	HANGING HEART ZINC T-LIGHT HOLDER	-24	2009-12-01 10:33:00	0.85	16321.00	Australia
186	C489449	22090	PAPER BUNTING RETRO SPOTS	-12	2009-12-01 10:33:00	2.95	16321.00	Australia
196	C489459	90200A	PURPLE SWEETHEART BRACELET	-3	2009-12-01 10:44:00	4.25	17592.00	United Kingdom

We can see these are legitimate transactions though with negative quantities it has Customer Id attached to it as well The thing to notice would be the C infront of the invoice which according to the data source UC Irvince's website indicates a Cancellations.

```
# Exploring invoice data which doesn't have exactly just 6 digits


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

	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	16321.00	Australia
179	C489449	85206A	CREAM FELT EASTER EGG BASKET	-6	2009-12-01 10:33:00	1.65	16321.00	Australia
180	C489449	21895	POTTING SHED SOW 'N' GROW SET	-4	2009-12-01 10:33:00	4.25	16321.00	Australia
181	C489449	21896	POTTING SHED TWINE	-6	2009-12-01 10:33:00	2.10	16321.00	Australia
182	C489449	22083	PAPER CHAIN KIT RETRO SPOT	-12	2009-12-01 10:33:00	2.95	16321.00	Australia
...	...	...	...	...	...	...	...	...
524695	C538123	22956	36 FOIL HEART CAKE CASES	-2	2010-12-09 15:41:00	2.10	12605.00	Germany
524696	C538124	M	Manual	-4	2010-12-09 15:43:00	0.50	15329.00	United Kingdom
524697	C538124	22699	ROSES REGENCY TEACUP AND SAUCER	-1	2010-12-09 15:43:00	2.95	15329.00	United Kingdom
524698	C538124	22423	REGENCY CAKESTAND 3 TIER	-1	2010-12-09 15:43:00	12.75	15329.00	United Kingdom
525282	C538164	35004B	SET OF 3 BLACK FLYING DUCKS	-1	2010-12-09 17:32:00	1.95	14031.00	United Kingdom

10209 rows × 8 columns

```
# Checking if C is the only character that appears in the invoice number

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

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

As seen we see Nothing, C and A. Let's investigate invoices that have A in their invoice number

```
# Invoices with A in them

df[df["Invoice"].str.startswith("A")]
```

	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

Just 3 records that seem to indicate bad debt that look like accounting type invoices It would be best to remove these while cleaning the data

```
# Exploring the stock code

df["StockCode"] = df["StockCode"].astype("str")
df[(df["StockCode"].str.match("^\\d{5}$") == False) & (df["StockCode"].str.match("^\\d{5}[a-zA-Z]+$") == False)]["StockCode"].unique()
```

```
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'],
      dtype=object)
```

The data source indicates that all stock codes are 5 digits however clearly there are other codes.

```
# Exploring if the Stock Codes that are not the expected 5 digit code

df[df["StockCode"].str.contains("^DOT")]
```

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
2379	489597	DOT	DOTCOM POSTAGE	1	2009-12-01 14:28:00	647.19	NaN	United Kingdom
2539	489600	DOT	DOTCOM POSTAGE	1	2009-12-01 14:43:00	55.96	NaN	United Kingdom
2551	489601	DOT	DOTCOM POSTAGE	1	2009-12-01 14:44:00	68.39	NaN	United Kingdom
2571	489602	DOT	DOTCOM POSTAGE	1	2009-12-01 14:45:00	59.35	NaN	United Kingdom
2619	489603	DOT	DOTCOM POSTAGE	1	2009-12-01 14:46:00	42.39	NaN	United Kingdom
...	...	...	...	...	...	...	...	...
524272	538071	DOT	DOTCOM POSTAGE	1	2010-12-09 14:09:00	885.94	NaN	United Kingdom
524887	538148	DOT	DOTCOM POSTAGE	1	2010-12-09 16:26:00	547.32	NaN	United Kingdom
525000	538149	DOT	DOTCOM POSTAGE	1	2010-12-09 16:27:00	620.68	NaN	United Kingdom
525126	538153	DOT	DOTCOM POSTAGE	1	2010-12-09 16:31:00	822.94	NaN	United Kingdom
525147	538154	DOT	DOTCOM POSTAGE	1	2010-12-09 16:35:00	85.79	NaN	United Kingdom

736 rows × 8 columns

```
# Looking at the data agian

df.head(10)
```

	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	13085.00	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.00	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.00	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.00	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.00	United Kingdom
5	489434	22064	PINK DOUGHNUT TRINKET POT	24	2009-12-01 07:45:00	1.65	13085.00	United Kingdom
6	489434	21871	SAVE THE PLANET MUG	24	2009-12-01 07:45:00	1.25	13085.00	United Kingdom
7	489434	21523	FANCY FONT HOME SWEET HOME DOORMAT	10	2009-12-01 07:45:00	5.95	13085.00	United Kingdom
8	489435	22350	CAT BOWL	12	2009-12-01 07:46:00	2.55	13085.00	United Kingdom
9	489435	22349	DOG BOWL , CHASING BALL DESIGN	12	2009-12-01 07:46:00	3.75	13085.00	United Kingdom

Notes

Stock Code StockCode is meant to follow the pattern [0-9]{5} but seems to have legit values for [0-9]{5}[a-zA-Z]+ Also contains other values:

Code	Description	Action
DCGS	Looks valid, some quantities are negative though and customer ID is null	Exclude from clustering
D	Looks valid, represents discount values	Exclude from clustering
DOT	Looks valid, represents postage charges	Exclude from clustering
M or m	Looks valid, represents manual transactions	Exclude from clustering
C2	Carriage transaction - not sure what this means	Exclude from clustering
C3	Not sure, only 1 transaction	Exclude
BANK CHARGES or B	Bank charges	Exclude from clustering
S	Samples sent to customer	Exclude from clustering
TESTXXX	Testing data, not valid	Exclude from clustering

Code	Description	Action
gift__XXX	Purchases with gift cards, might be interesting for another analysis, but no customer data	Exclude
PADS	Looks like a legit stock code for padding	Include
SP1002	Looks like a special request item, only 2 transactions, 3 look legit, 1 has 0 pricing	Exclude for now
AMAZONFEE	Looks like fees for Amazon shipping or something	Exclude for now
ADJUSTX	Looks like manual account adjustments by admins	Exclude for now

DATA CLEANING

```
# Creating a copy to preserve the orignal data

cleaned_df = df.copy()
```

```
# Keeping invoices that are strictly where there are 6 digits only


cleaned_df["Invoice"] = cleaned_df["Invoice"].astype("str")

mask = (
    cleaned_df["Invoice"].str.match("^\\d{6}$") == True
)

cleaned_df = cleaned_df[mask]

cleaned_df
```

	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	13085.00	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.00	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.00	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.00	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.00	United Kingdom
...	...	...	...	...	...	...	...	...
525456	538171	22271	FELTCRAFT DOLL ROSIE	2	2010-12-09 20:01:00	2.95	17530.00	United Kingdom
525457	538171	22750	FELTCRAFT PRINCESS LOLA DOLL	1	2010-12-09 20:01:00	3.75	17530.00	United Kingdom
525458	538171	22751	FELTCRAFT PRINCESS OLIVIA DOLL	1	2010-12-09 20:01:00	3.75	17530.00	United Kingdom
525459	538171	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	2	2010-12-09 20:01:00	3.75	17530.00	United Kingdom
525460	538171	21931	JUMBO STORAGE BAG SUKI	2	2010-12-09 20:01:00	1.95	17530.00	United Kingdom

515252 rows × 8 columns

```
# Cleaning with the previous conclusion we got from examining the stock codes

cleaned_df["StockCode"] = cleaned_df["StockCode"].astype("str")

mask = (
    (cleaned_df["StockCode"].str.match("^\\d{5}$") == True)
    | (cleaned_df["StockCode"].str.match("^\\d{5}[a-zA-Z]+$") == True)
    | (cleaned_df["StockCode"].str.match("^PADS$") == True)
)

cleaned_df = cleaned_df[mask]

cleaned_df
```

```
<ipython-input-16-abf8f5ed7ab1>:3: 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
  cleaned_df["StockCode"] = cleaned_df["StockCode"].astype("str")
```

	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	13085.00	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.00	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.00	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.00	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.00	United Kingdom
...	...	...	...	...	...	...	...	...
525456	538171	22271	FELTCRAFT DOLL ROSIE	2	2010-12-09 20:01:00	2.95	17530.00	United Kingdom
525457	538171	22750	FELTCRAFT PRINCESS LOLA DOLL	1	2010-12-09 20:01:00	3.75	17530.00	United Kingdom
525458	538171	22751	FELTCRAFT PRINCESS OLIVIA DOLL	1	2010-12-09 20:01:00	3.75	17530.00	United Kingdom
525459	538171	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	2	2010-12-09 20:01:00	3.75	17530.00	United Kingdom
525460	538171	21931	JUMBO STORAGE BAG SUKI	2	2010-12-09 20:01:00	1.95	17530.00	United Kingdom

512796 rows × 8 columns

```
# Removing data rows where the customer id is null

cleaned_df.dropna(subset=["Customer ID"], inplace=True)
```

```
<ipython-input-17-4ba2fd6404f1>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_df.dropna(subset=["Customer ID"], inplace=True)
```

```
# Descibing the cleaned data set

cleaned_df.describe()
```

	Quantity	InvoiceDate	Price	Customer ID
count	406337.00	406337	406337.00	406337.00
mean	13.62	2010-07-01 10:11:06.543288320	2.99	15373.63
min	1.00	2009-12-01 07:45:00	0.00	12346.00
25%	2.00	2010-03-26 14:01:00	1.25	14004.00
50%	5.00	2010-07-09 15:48:00	1.95	15326.00
75%	12.00	2010-10-14 17:09:00	3.75	16814.00
max	19152.00	2010-12-09 20:01:00	295.00	18287.00
std	97.00	NaN	4.29	1677.37

```
# Checking if there are any price values of 0


len(cleaned_df[cleaned_df["Price"] == 0])
```

```
28
```

```
# Removing values that are not greater than 0

cleaned_df = cleaned_df[cleaned_df["Price"] > 0.0]
```

```
cleaned_df.describe()
```

	Quantity	InvoiceDate	Price	Customer ID
count	406309.00	406309	406309.00	406309.00
mean	13.62	2010-07-01 10:14:25.869572352	2.99	15373.72
min	1.00	2009-12-01 07:45:00	0.00	12346.00
25%	2.00	2010-03-26 14:01:00	1.25	14006.00
50%	5.00	2010-07-09 15:48:00	1.95	15326.00
75%	12.00	2010-10-14 17:09:00	3.75	16814.00
max	19152.00	2010-12-09 20:01:00	295.00	18287.00
std	97.00	NaN	4.29	1677.33

```
cleaned_df["Price"].min()
```

0.001

```
len(cleaned_df)/len(df)
```

0.7732429238325965

We have dropped about 23% of the original dataset while clearning

FEATURE ENGINEERING

Featues that will help us more about customers are frequency of purchase, recency of their transaction and how much they have spent (price).

```
# Creating a new column sales line total to better understand the customer

cleaned_df["SalesLineTotal"] = cleaned_df["Quantity"] * cleaned_df["Price"]

cleaned_df
```

<ipython-input-24-a6410fbd78f7>:3: 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
cleaned_df["SalesLineTotal"] = cleaned_df["Quantity"] * cleaned_df["Price"]

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country	SalesLineTotal
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.00	United Kingdom	83.40
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.00	United Kingdom	81.00
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.00	United Kingdom	81.00
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.00	United Kingdom	100.80
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.00	United Kingdom	30.00
...	...	...	...	...	...	...	...	...	...
525456	538171	22271	FELTCRAFT DOLL ROSIE	2	2010-12-09 20:01:00	2.95	17530.00	United Kingdom	5.90
525457	538171	22750	FELTCRAFT PRINCESS LOLA DOLL	1	2010-12-09	3.75	17530.00	United	3.75

```
# Aggregating the data according to customer id, and computing aggregates for recency, frequency and the sales line total (monetary value)

aggregated_df = cleaned_df.groupby(by="Customer ID", as_index=False) \
    .agg(
        MonetaryValue=("SalesLineTotal", "sum"),
        Frequency=("Invoice", "nunique"),
        LastInvoiceDate=("InvoiceDate", "max")
    )

aggregated_df
```

	Customer ID	MonetaryValue	Frequency	LastInvoiceDate
0	12346.00	169.36	2	2010-06-28 13:53:00
1	12347.00	1323.32	2	2010-12-07 14:57:00
2	12348.00	221.16	1	2010-09-27 14:59:00
3	12349.00	2221.14	2	2010-10-28 08:23:00
4	12351.00	300.93	1	2010-11-29 15:23:00
...	...	...	...	...
4280	18283.00	641.77	6	2010-11-22 15:30:00
4281	18284.00	411.68	1	2010-10-04 11:33:00
4282	18285.00	377.00	1	2010-02-17 10:24:00
4283	18286.00	1246.43	2	2010-08-20 11:57:00
4284	18287.00	2295.71	4	2010-11-22 11:51:00

4285 rows × 4 columns

Next steps: Generate code with aggregated_df | View recommended plots | New interactive sheet

```
# Recency based off of the most recent date in the dataframe

max_invoice_date = aggregated_df["LastInvoiceDate"].max()

aggregated_df["Recency"] = (max_invoice_date - aggregated_df["LastInvoiceDate"]).dt.days

aggregated_df
```

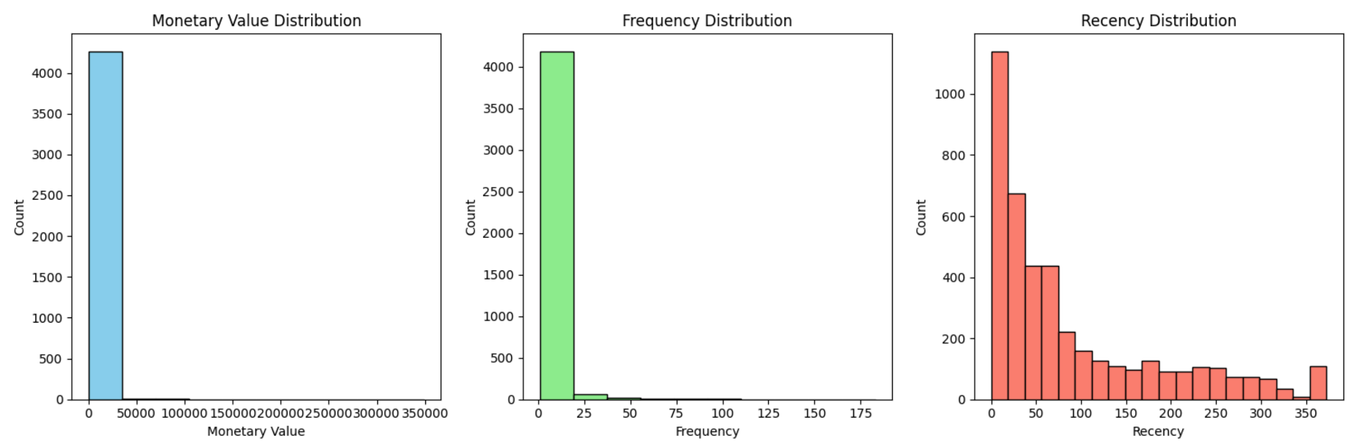
	Customer ID	MonetaryValue	Frequency	LastInvoiceDate	Recency
0	12346.00	169.36	2	2010-06-28 13:53:00	164
1	12347.00	1323.32	2	2010-12-07 14:57:00	2
2	12348.00	221.16	1	2010-09-27 14:59:00	73
3	12349.00	2221.14	2	2010-10-28 08:23:00	42
4	12351.00	300.93	1	2010-11-29 15:23:00	10
...	...	...	...	...	...
4280	18283.00	641.77	6	2010-11-22 15:30:00	17
4281	18284.00	411.68	1	2010-10-04 11:33:00	66
4282	18285.00	377.00	1	2010-02-17 10:24:00	295
4283	18286.00	1246.43	2	2010-08-20 11:57:00	111
4284	18287.00	2295.71	4	2010-11-22 11:51:00	17

4285 rows × 5 columns

Next steps: Generate code with aggregated_df | View recommended plots | New interactive sheet

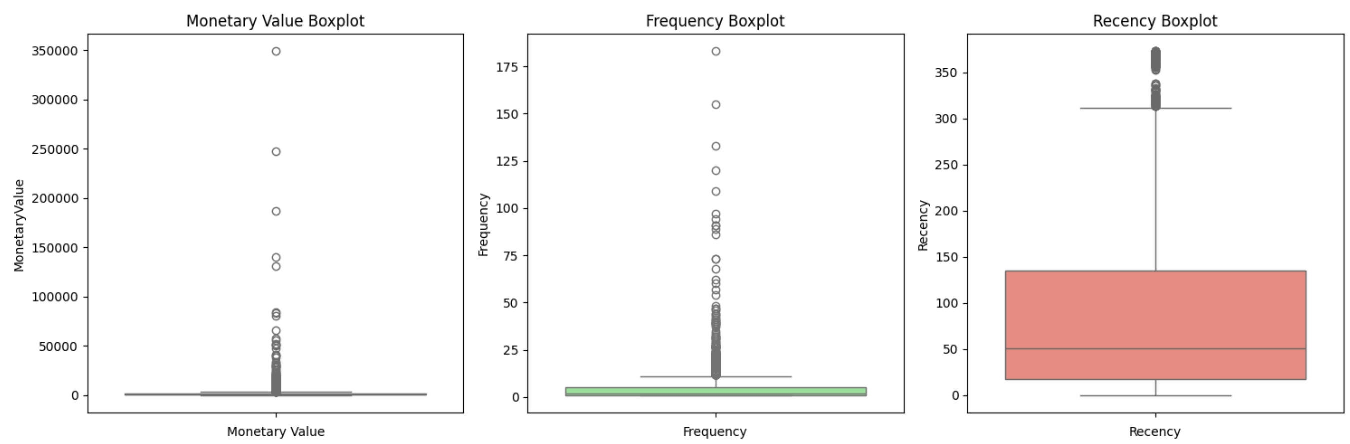
```
# Visualising the distribution of the featues in the aggregate data frame to discover any outliers
# Using Histograms

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

plt.subplot(1, 3, 1)
plt.hist(aggregated_df['MonetaryValue'], bins=10, color='skyblue', edgecolor='black')
plt.title('Monetary Value Distribution')
plt.xlabel('Monetary Value')
plt.ylabel('Count')

plt.subplot(1, 3, 2)
plt.hist(aggregated_df['Frequency'], bins=10, color='lightgreen', edgecolor='black')
plt.title('Frequency Distribution')
plt.xlabel('Frequency')
plt.ylabel('Count')

plt.subplot(1, 3, 3)
plt.hist(aggregated_df['Recency'], bins=20, color='salmon', edgecolor='black')
plt.title('Recency Distribution')
plt.xlabel('Recency')
plt.ylabel('Count')
```

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



The distributions of the monetary and frequency values looks quite skewed with some outliers while the recency values seems to be poisson distribution without as much outliers

```
# Looking to understand the outliers for each feature using boxplots

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

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

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

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

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



The Monetary value and Frequency values there are extreme and large amount of outliers The recency values do have some outliers

```
# Separating the outliers for Monetary Feature

M_Q1 = aggregated_df["MonetaryValue"].quantile(0.25)
M_Q3 = aggregated_df["MonetaryValue"].quantile(0.75)
M_IQR = M_Q3 - M_Q1

monetary_outliers_df = aggregated_df[(aggregated_df["MonetaryValue"] > (M_Q3 + 1.5 * M_IQR)) | (aggregated_df["MonetaryValue"] < (M_Q1 - 1.5 * M_IQR))].copy()
```

```
monetary_outliers_df.describe()
```

	Customer ID	MonetaryValue	Frequency	LastInvoiceDate	Recency
count	423.00	423.00	423.00	423	423.00
mean	15103.04	12188.10	17.17	2010-11-09 12:26:02.978723328	30.04
min	12357.00	3802.04	1.00	2009-12-10 18:03:00	0.00
25%	13622.00	4605.94	8.00	2010-11-08 13:17:30	3.00
50%	14961.00	6191.32	12.00	2010-11-26 12:19:00	13.00
75%	16692.00	10273.24	18.00	2010-12-06 10:34:30	31.00
max	18260.00	349164.35	183.00	2010-12-09 19:32:00	364.00
std	1728.66	25830.85	19.73	NaN	51.54

```
# Separating the outliers for Frequency Feature

F_Q1 = aggregated_df['Frequency'].quantile(0.25)
F_Q3 = aggregated_df['Frequency'].quantile(0.75)
F_IQR = F_Q3 - F_Q1

frequency_outliers_df = aggregated_df[(aggregated_df['Frequency'] > (F_Q3 + 1.5 * F_IQR)) | (aggregated_df['Frequency'] < (F_Q1 - 1.5 * F_IQR))].copy()

frequency_outliers_df.describe()
```

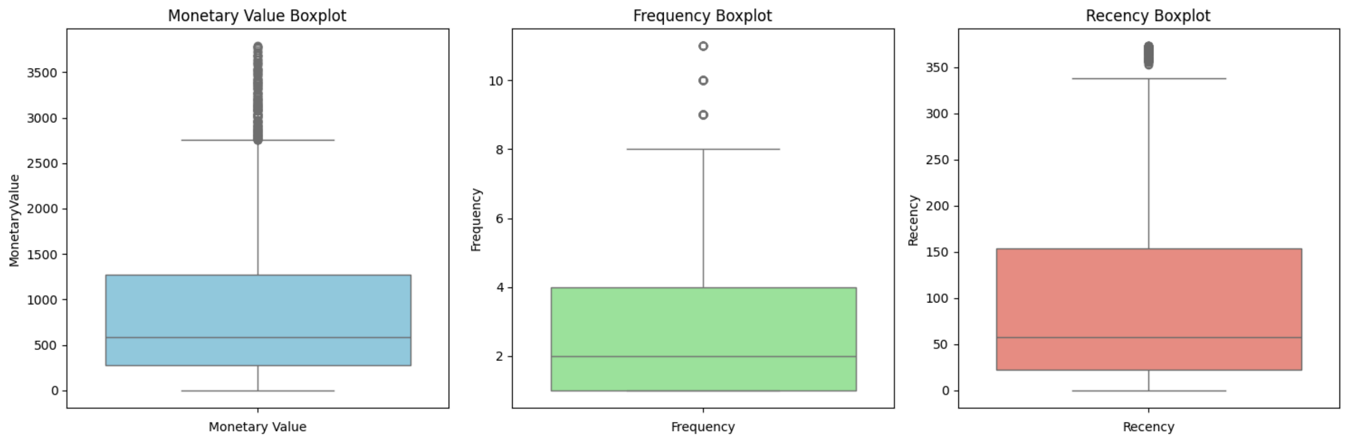
	Customer ID	MonetaryValue	Frequency	LastInvoiceDate	Recency
count	279.00	279.00	279.00	279	279.00
mean	15352.66	14409.71	23.81	2010-11-23 11:06:20.645161216	16.09
min	12437.00	1094.39	12.00	2010-05-12 16:51:00	0.00
25%	13800.00	4331.56	13.00	2010-11-20 13:14:30	2.00
50%	15465.00	6615.77	17.00	2010-12-02 10:46:00	7.00
75%	16828.50	11692.41	23.00	2010-12-07 11:08:30	19.00
max	18260.00	349164.35	183.00	2010-12-09 19:32:00	211.00
std	1748.43	31381.74	21.93	NaN	26.59

```
# Filtering out the outliers out ogf the aggregate data

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

non_outliers_df.describe()
```

	Customer ID	MonetaryValue	Frequency	LastInvoiceDate	Recency
count	3809.00	3809.00	3809.00	3809	3809.00
mean	15376.48	885.50	2.86	2010-09-03 11:16:46.516146176	97.08
min	12346.00	1.55	1.00	2009-12-01 10:49:00	0.00
25%	13912.00	279.91	1.00	2010-07-08 14:48:00	22.00
50%	15389.00	588.05	2.00	2010-10-12 16:25:00	58.00
75%	16854.00	1269.05	4.00	2010-11-17 13:14:00	154.00
max	18287.00	3788.21	11.00	2010-12-09 20:01:00	373.00
std	1693.20	817.67	2.24	NaN	98.11

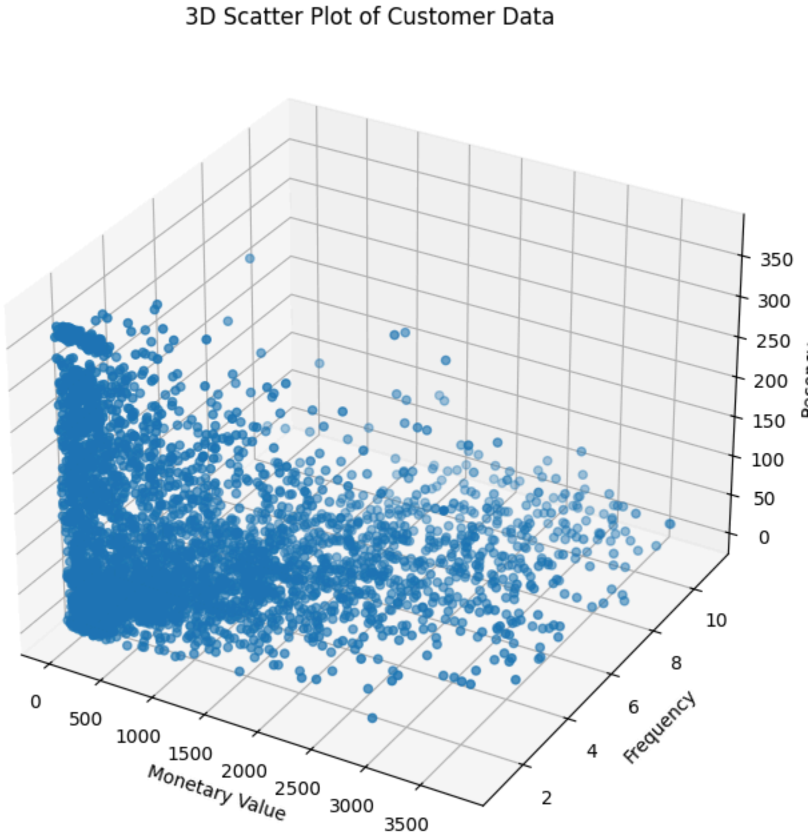
```
# Visualising the Non Outlier Data Frame on a Box Plot

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

plt.subplot(1, 3, 1)
sns.boxplot(data=non_outliers_df['MonetaryValue'], 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')
plt.xlabel('Recency')

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



We can observe that this is a less skewed. There are still outliers but it is better than before

```
# Plotting the Data

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

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

scatter = ax.scatter(non_outliers_df["MonetaryValue"], 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()
```



Standard Scaling

```
# Using a scalar for regularisation of the values so the K-Means algorithm since it is as distance based algorithms

scaler = StandardScaler()

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

scaled_data
```

```
array([[-0.87594534, -0.38488934,  0.68214853],
       [ 0.5355144 , -0.38488934, -0.96925093],
       [-0.81258645, -0.83063076, -0.24548944],
       ...,
       [-0.62197163, -0.83063076,  2.01753946],
       [ 0.44146683, -0.38488934,  0.14187587],
       [ 1.72488781,  0.50659348, -0.81634357]])
```

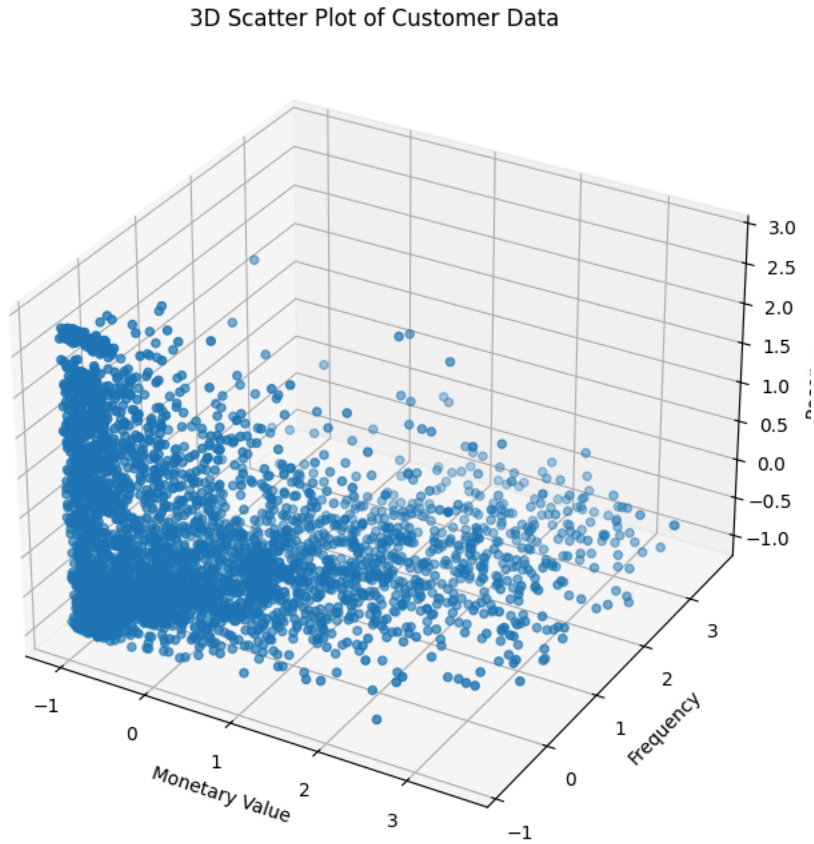
```
# Creating a data frame with the scaled values

scaled_data_df = pd.DataFrame(scaled_data, index=non_outliers_df.index, columns=("MonetaryValue", "Frequency", "Recency"))

scaled_data_df
```

	MonetaryValue	Frequency	Recency
0	-0.88	-0.38	0.68
1	0.54	-0.38	-0.97
2	-0.81	-0.83	-0.25
3	1.63	-0.38	-0.56
4	-0.72	-0.83	-0.89
...	...	...	...
4280	-0.30	1.40	-0.82
4281	-0.58	-0.83	-0.32
4282	-0.62	-0.83	2.02
4283	0.44	-0.38	0.14
4284	1.72	0.51	-0.82

3809 rows × 3 columns

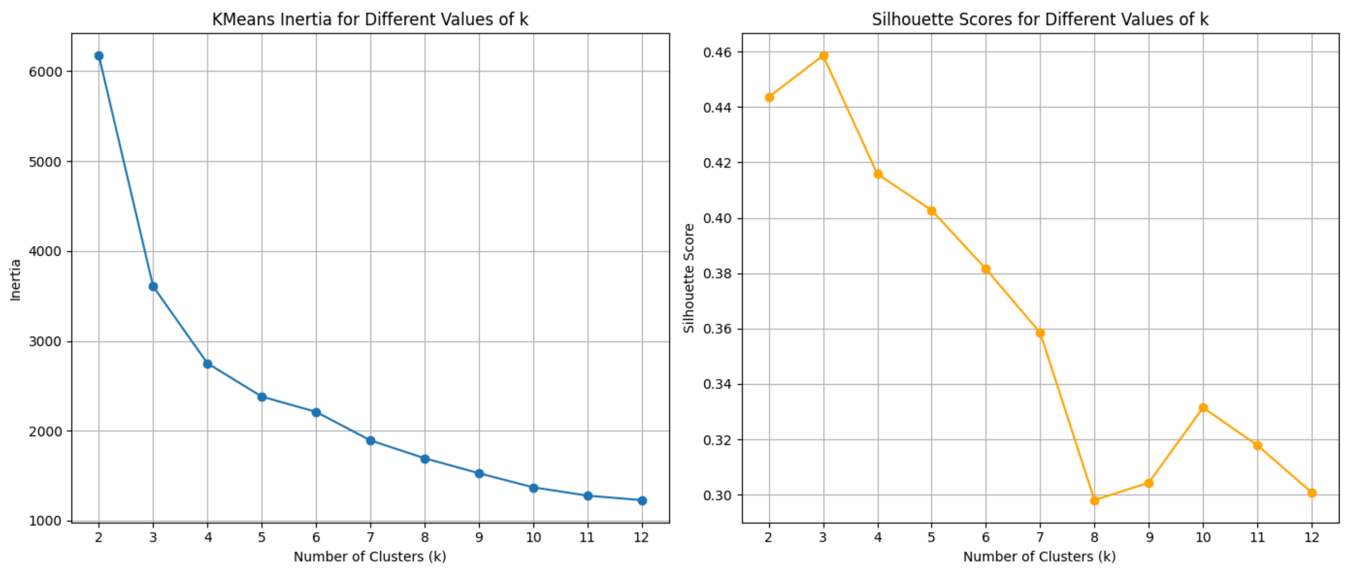
Next steps: Generate code with scaled_data_df View recommended plots New interactive sheet

```
# Plotting a 3 D plot of the data again

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

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

scatter = ax.scatter(scaled_data_df["MonetaryValue"], 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()
```



KMeans CLUSTERING

```
# Applying K-Means to the Scaled Data and Plotting Intertia and Silhouttee Score against the number of Clusters

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()
```



From the Intertia it can be gathered that around 4 to 5 clusters might be ideal, Anaylsing the Silhoutte Score it can be gathered that the between 4 and 5 clusters there seems to be a very small difference between the scores between 4 and 5 with 4 being higher and thus I am choosing to take 4 clusters

```
# Getting Cluster Labels for the scaled data
# Number of Clusters = 4

kmeans = KMeans(n_clusters=4, random_state=42, max_iter=1000)

cluster_labels = kmeans.fit_predict(scaled_data_df)

cluster_labels
```

array([1, 0, 2, ..., 1, 0, 0], dtype=int32)

```
# Adding a Cluster Label Column

non_outliers_df["Cluster"] = cluster_labels

non_outliers_df
```

<ipython-input-39-36399fc58836>:3: 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

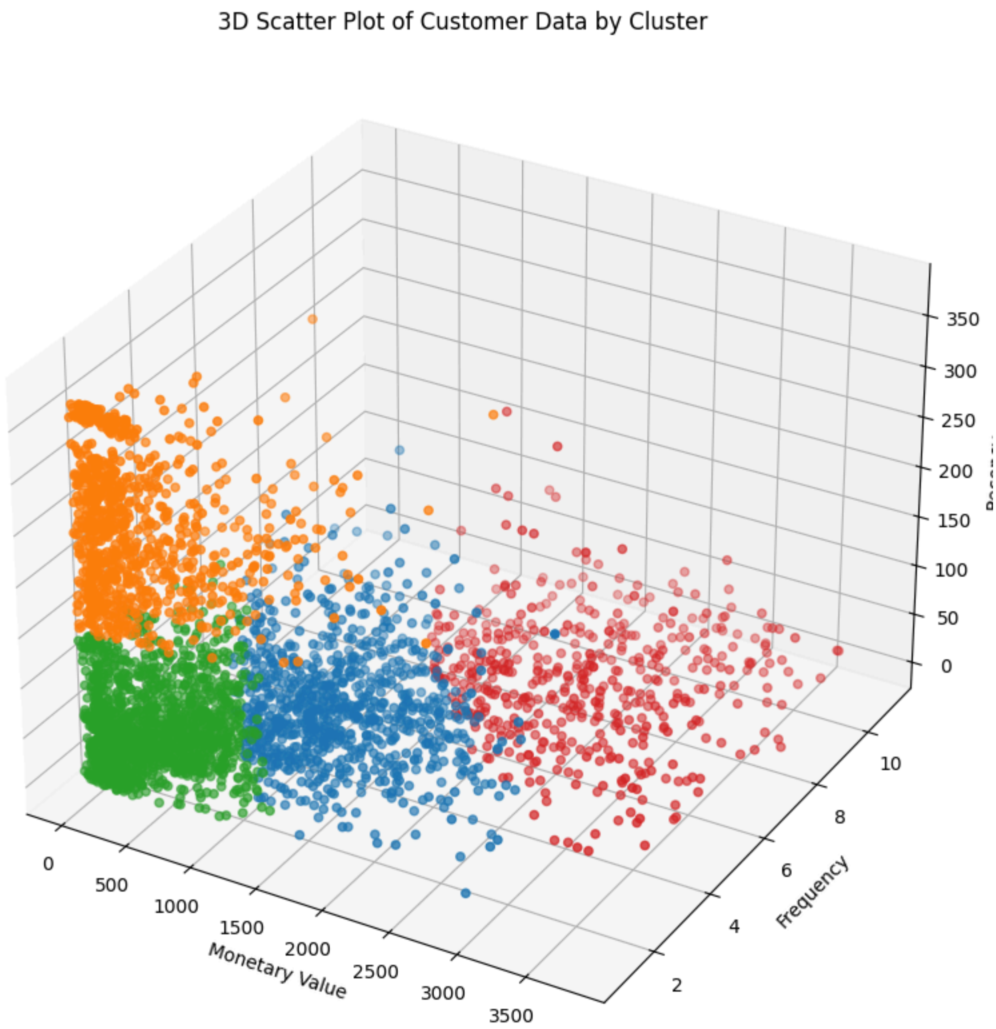
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
non_outliers_df["Cluster"] = cluster_labels

	Customer ID	MonetaryValue	Frequency	LastInvoiceDate	Recency	Cluster
0	12346.00	169.36	2	2010-06-28 13:53:00	164	1
1	12347.00	1323.32	2	2010-12-07 14:57:00	2	0
2	12348.00	221.16	1	2010-09-27 14:59:00	73	2
3	12349.00	2221.14	2	2010-10-28 08:23:00	42	0
4	12351.00	300.93	1	2010-11-29 15:23:00	10	2
...	...	...	...	...	...	...
4280	18283.00	641.77	6	2010-11-22 15:30:00	17	0
4281	18284.00	411.68	1	2010-10-04 11:33:00	66	2
4282	18285.00	377.00	1	2010-02-17 10:24:00	295	1
4283	18286.00	1246.43	2	2010-08-20 11:57:00	111	0
4284	18287.00	2295.71	4	2010-11-22 11:51:00	17	0

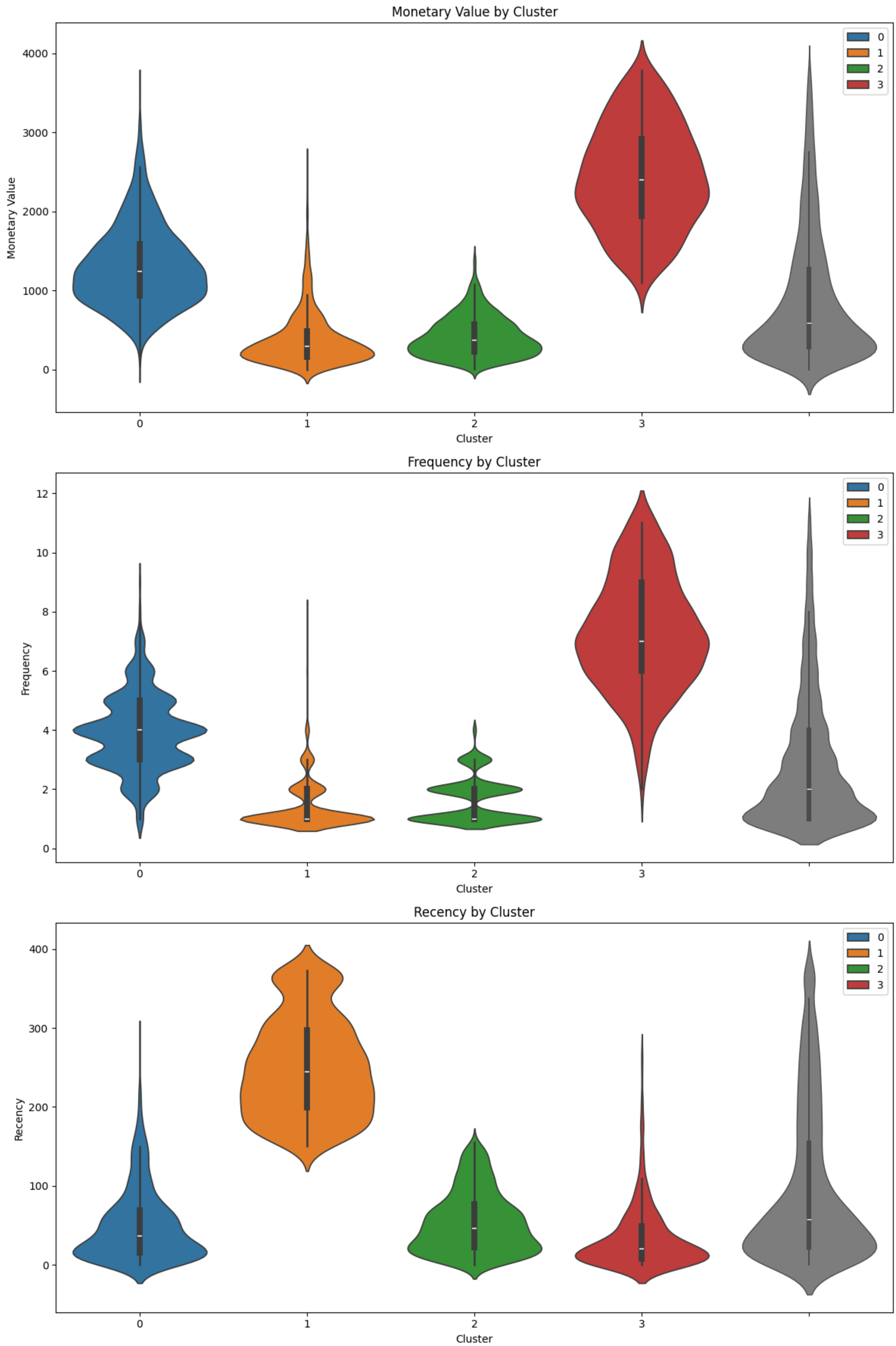
3809 rows × 6 columns

Next steps: Generate code with non_outliers_df | View recommended plots | New interactive sheet

```
# Plotting a Scatter Plot of Customer Data by Cluster

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['MonetaryValue'],
                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()
```




```
# Analysing the Clusters

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

plt.subplot(3, 1, 1)
sns.violinplot(x=non_outliers_df['Cluster'], y=non_outliers_df['MonetaryValue'], palette=cluster_colors, hue=non_outliers_df["Cluster"])
sns.violinplot(y=non_outliers_df['MonetaryValue'], 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_df["Cluster"])
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_df["Cluster"])
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()
```


Monetary Value by Cluster
Monetary Value
4000
3000
2000
1000
0
0
1
2
3
Cluster
Frequency by Cluster
Frequency
12
10
8
6
4
2
0
0
1
2
3
Cluster
Recency by Cluster
Recency
400
300
200
100
0
0
1
2
3
Cluster

Analysis of Non-Outlier Data

1. Cluster 0 [RETAIN]: Has Mid to High Monetary and Frequency Values but lower Recency Values. The data seems to be pretty centered thus not very variable. With this rationale this cluster represents High Value Customers who purchase regularly and recently so the focus on a customer-buisness relationship would be to retain them.
2. Cluster 1: Has Low Monetary and Frequency Values but high Recency Values. With this information we can conclude that the cluster represents customers who no longer engage.
3. Cluster 2: Has Low Monetary, Frequency and Recency values. This is the group of customers who have started engaging with the platform.
4. Cluster 3: Has very high Monetary and Frequency values and very low recenc values. These represent valuable customers who regularly engage and spend a lot on the platform.

Cluster 0 (Blue): "Retain"

Rationale: This cluster represents high-value customers who purchase regularly, though not always very recently. The focus should be on retention efforts to maintain their loyalty and spending levels. **Action:** Implement loyalty programs, personalized offers, and regular engagement to ensure they remain active.

Cluster 1 (Orange): "Re-Engage"

Rationale: This group includes lower-value, infrequent buyers who haven't purchased recently. The focus should be on re-engagement to bring them back into active purchasing behavior. **Action:** Use targeted marketing campaigns, special discounts, or reminders to encourage them to return and purchase again.

Cluster 2 (Green): "Nurture"

Rationale: This cluster represents the least active and lowest-value customers, but they have made recent purchases. These customers may be new or need nurturing to increase their engagement and spending. **Action:** Focus on building relationships, providing excellent customer service, and offering incentives to encourage more frequent purchases.

Cluster 3 (Red): "Reward"

Rationale: This cluster includes high-value, very frequent buyers, many of whom are still actively purchasing. They are your most loyal customers, and rewarding their loyalty is key to maintaining their engagement. **Action:** Implement a robust loyalty program, provide exclusive offers, and recognize their loyalty to keep them engaged and satisfied.

Summary of Cluster Names:

1. Cluster 0 (Blue): "Retain"
2. Cluster 1 (Orange): "Re-Engage"
3. Cluster 2 (Green): "Nurture"
4. Cluster 3 (Red): "Reward"

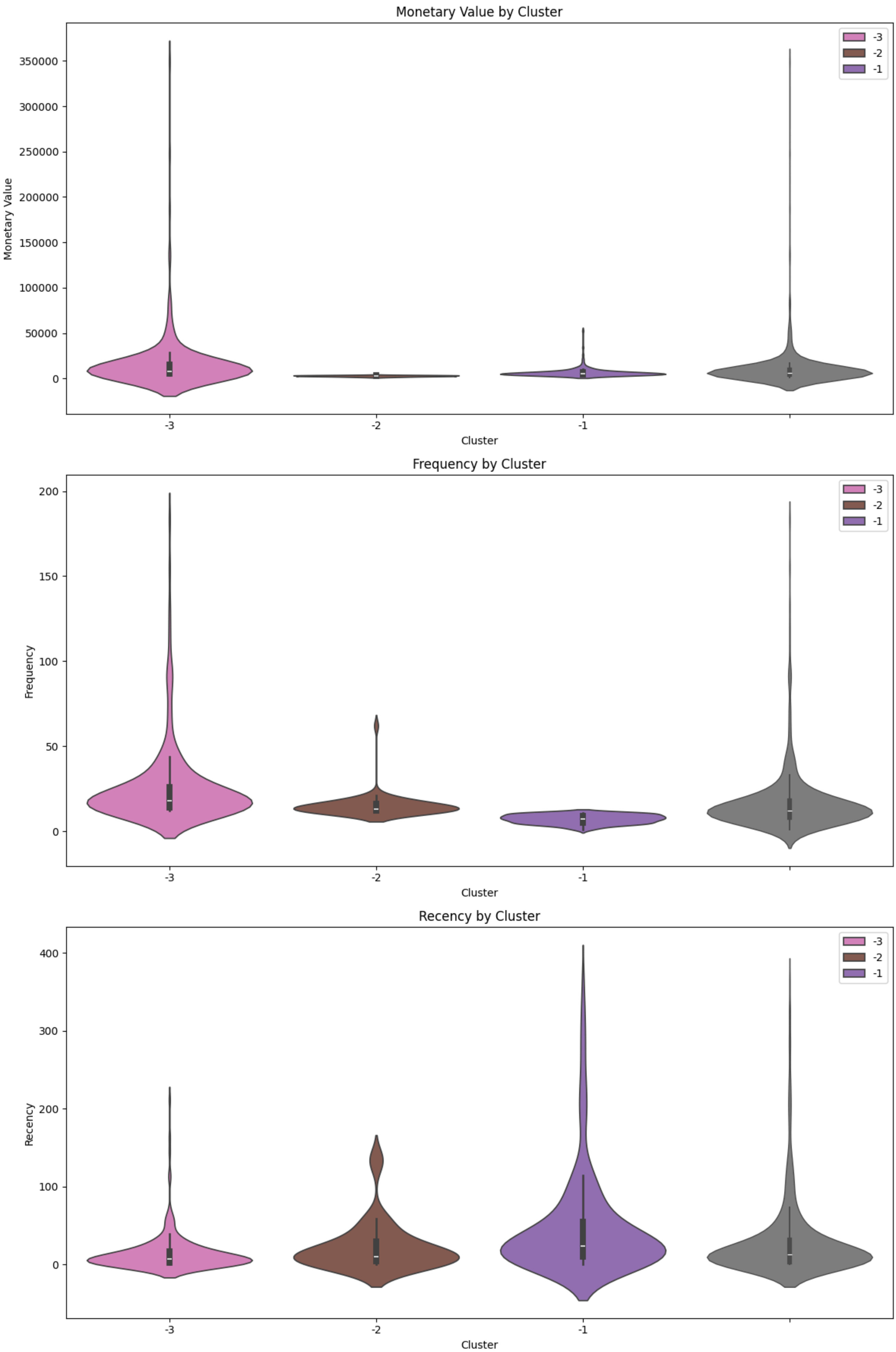
```
# We are now going to analyse the Outliers

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
```

	Customer ID	MonetaryValue	Frequency	LastInvoiceDate	Recency	Cluster
9	12357.00	11229.99	1	2010-11-16 10:05:00	23	-1
25	12380.00	4782.84	4	2010-08-31 14:54:00	100	-1
42	12409.00	12346.62	4	2010-10-15 10:24:00	55	-1
48	12415.00	19468.84	4	2010-11-29 15:07:00	10	-1
61	12431.00	4145.52	11	2010-12-01 10:03:00	8	-1
...	...	...	...	...	...	...
4235	18223.00	7516.31	12	2010-11-17 12:20:00	22	-3
4236	18225.00	7545.14	15	2010-12-09 15:46:00	0	-3
4237	18226.00	6650.83	15	2010-11-26 15:51:00	13	-3
4241	18231.00	4791.80	23	2010-10-29 14:17:00	41	-3
4262	18260.00	7318.91	17	2010-11-30 12:25:00	9	-3

476 rows × 6 columns

Next steps: Generate code with outlier_clusters_df | View recommended plots | New interactive sheet

```
# Plotting the Outliers on a Violin Plot

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['MonetaryValue'], palette=cluster_colors, hue=outlier_clusters_df["Cluster"])
sns.violinplot(y=outlier_clusters_df['MonetaryValue'], 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=outlier_clusters_df["Cluster"])
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=outlier_clusters_df["Cluster"])
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()
```


Monetary Value by Cluster
Monetary Value
350000
300000
250000
200000
150000
100000
50000
0
-3
-2
-1
Cluster
Frequency by Cluster
Frequency
200
150
100
50
0
Recency by Cluster
Recency
400
300
200
100
0