```
# Importing the dependencies

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

# Data collection

dataset = pd.read_csv('ecomm.csv', encoding = 'unicode_escape')

dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 326401 entries, 0 to 326400
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   InvoiceNo    326401 non-null  object 
 1   StockCode    326401 non-null  object 
 2   Description  325231 non-null  object 
 3   Quantity     326400 non-null  float64
 4   InvoiceDate  326400 non-null  object 
 5   UnitPrice    326400 non-null  float64
 6   CustomerID   236490 non-null  float64
 7   Country      326400 non-null  object 
dtypes: float64(3), object(5)
memory usage: 19.9+ MB
```

We can observe that:

- InvoiceDate has to be converted into a date time Datatype.

```
dataset.head()
```

{"type":"dataframe","variable_name":"dataset"}

```
dataset.shape
```

```
(326401, 8)
```

```
dataset.isnull().sum()

# There are null-values in 2 columns (Description and Customer ID).
```

```
InvoiceNo          0
StockCode          0
Description     1170
Quantity           1
InvoiceDate        1
UnitPrice          1
CustomerID     89911
```

```
Country           1
dtype: int64
```

```
dataset.duplicated().sum()

# We can observe that there are 2533 duplicate entries in the dataset.
```

```
2533
```

```
# Analysing the quantitative values
dataset.describe()
```

```
{"summary":"{\n  \"name\": \"dataset\",\n  \"rows\": 8,\n  \"fields\":
[\n    {\n      \"column\": \"Quantity\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 122017.53221210137,\n
\"min\": -74215.0,\n        \"max\": 326400.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
9.65999387254902,\n          10.0,\n          326400.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"InvoiceDate\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"1970-01-01 00:00:00.000326400\",\n        \"max\": \"2011-09-05
12:00:00\",\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          \"326400\",\n          \"2011-04-18
12:21:39.092095744\",\n          \"2011-06-30 12:45:30\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"UnitPrice\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
114937.9858987506,\n        \"min\": -11062.06,\n        \"max\":
326400.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
4.886140140931371,\n          4.13,\n          326400.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"CustomerID\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
79054.00422379651,\n        \"min\": 1726.0329817152647,\n
\"max\": 236490.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          15278.241447841347,\n          16809.0,\n
236490.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

```
len(dataset['CustomerID'].unique())
```

```
3387
```

```
print(dataset['Country'].value_counts())
```

```
Country
United Kingdom          297934
Germany                   6047
France                    4772
EIRE                      4728
```

```
Spain                     1621
Netherlands               1539
Belgium                   1288
Switzerland               1192
Australia                 1013
Portugal                   775
Norway                     524
Channel Islands            509
Finland                    468
Italy                      419
Unspecified                393
Cyprus                     353
Sweden                     309
Japan                      298
Austria                    270
Poland                     253
Hong Kong                  249
Israel                     236
Denmark                    220
Singapore                  193
Canada                     151
Iceland                    124
Greece                     110
Malta                      104
United Arab Emirates        67
European Community          61
Lebanon                     45
Lithuania                   35
Brazil                      32
USA                         22
Bahrain                     19
Czech Republic              17
Saudi Arabia                10
Name: count, dtype: int64
```

Calculaing the Recency:

The days since the last purchase for each customers.

1. We need keep only the most recent date for each customer.
2. Rank each customer based on how recent their purchase was.
3. Assign a recency score.

```
# Converting the InvoiceDate into datetime Datatype.
dataset['InvoiceDate'] = pd.to_datetime(dataset['InvoiceDate'])

# Sorting the datset by CustomerID and Date
dataset.sort_values(['CustomerID', 'InvoiceDate'])

# Finding the most recent date for each customer and ranking each
```

```
customer based on how recent their purchase was.
dataset['rank'] = dataset.groupby(['CustomerID'])
['InvoiceDate'].rank(method = 'min')

# Recency score.
df_recency = dataset[dataset['rank'] == 1]
df_recency.head(10)
```

```
{"summary":"{\n  \"name\": \"df_recency\",\n  \"rows\": 72673,\n
\"fields\": [\n    {\n      \"column\": \"InvoiceNo\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3398,\n        \"samples\": [\n
\"537134\",\n          \"554632\",\n          \"549789\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"StockCode\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3175,\n        \"samples\": [\n
\"21278\",\n          \"17021\",\n          \"84857C\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Description\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3243,\n        \"samples\": [\n
\"SCALLOP SHELL SOAP DISH\",\n          \"SINGLE WIRE HOOK PINK
HEART\",\n          \"LOCAL CAFE MUG\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Quantity\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
280.3382080680222,\n        \"min\": -9360.0,\n        \"max\":
74215.0,\n        \"num_unique_values\": 148,\n        \"samples\": [\
n          2400.0,\n          52.0,\n          320.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"InvoiceDate\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2010-12-01 08:26:00\",\n        \"max\": \"2011-09-05 11:38:00\",\n
\"num_unique_values\": 3302,\n        \"samples\": [\n
\"2010-12-01 13:17:00\",\n          \"2010-12-13 15:34:00\",\n
\"2011-01-27 10:56:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"UnitPrice\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 18.57135957026946,\n
\"min\": 0.0,\n        \"max\": 4287.63,\n
\"num_unique_values\": 193,\n        \"samples\": [\n          165.0,\
n          5.75,\n          295.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"CustomerID\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1730.0256253358443,\n        \"min\": 12346.0,\n        \"max\":
18287.0,\n        \"num_unique_values\": 3386,\n        \"samples\":
[\n          17450.0,\n          13579.0,\n          13050.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Country\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 36,\n
\"samples\": [\n          \"Malta\",\n          \"Lithuania\",\n
\"United Arab Emirates\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"rank\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.0,\n        \"min\": 1.0,\n
\"max\": 1.0,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          1.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df_recency"}
```

We can see that more than one entry for a customer has the same rank, this is because the customer has done multiple purchases (different products) on the same day and `rank('min')` assigns the same rank for all these transactions done on the same date.

```python
# Assigning recency score to each customer by calculating the
difference between the most recent purchase and a reference date.

# Reference date - latest(most recent) transcation date
ref_date = dataset['InvoiceDate'].min()
print(ref_date)

df_recency['Recency'] = df_recency['InvoiceDate'] -  ref_date

# Converting the difference into days.
df_recency['Recency'] = df_recency['Recency'].dt.days

df_recency.head(10)
```

```
2010-12-01 08:26:00
```

```
<ipython-input-54-03906faa8b9a>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df_recency['Recency'] = df_recency['InvoiceDate'] -  ref_date
<ipython-input-54-03906faa8b9a>:10: 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
  df_recency['Recency'] = df_recency['Recency'].dt.days
```

```
{"summary":"{\n  \"name\": \"df_recency\",\n  \"rows\": 72673,\n
\"fields\": [\n    {\n      \"column\": \"InvoiceNo\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3398,\n        \"samples\": [\n
\"537134\",\n          \"554632\",\n          \"549789\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"StockCode\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3175,\n        \"samples\": [\n
\"21278\",\n          \"17021\",\n          \"84857C\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Description\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3243,\n        \"samples\": [\n
\"SCALLOP SHELL SOAP DISH\",\n          \"SINGLE WIRE HOOK PINK
HEART\",\n          \"LOCAL CAFE MUG\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Quantity\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
280.3382080680222,\n        \"min\": -9360.0,\n        \"max\":
74215.0,\n        \"num_unique_values\": 148,\n        \"samples\": [\
n          2400.0,\n          52.0,\n          320.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"InvoiceDate\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2010-12-01 08:26:00\",\n        \"max\": \"2011-09-05 11:38:00\",\n
\"num_unique_values\": 3302,\n        \"samples\": [\n
\"2010-12-01 13:17:00\",\n          \"2010-12-13 15:34:00\",\n
\"2011-01-27 10:56:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"UnitPrice\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 18.57135957026946,\n
\"min\": 0.0,\n        \"max\": 4287.63,\n
\"num_unique_values\": 193,\n        \"samples\": [\n          165.0,\
n          5.75,\n          295.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"CustomerID\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1730.0256253358443,\n        \"min\": 12346.0,\n        \"max\":
18287.0,\n        \"num_unique_values\": 3386,\n        \"samples\":
[\n          17450.0,\n          13579.0,\n          13050.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Country\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 36,\n
\"samples\": [\n          \"Malta\",\n          \"Lithuania\",\n
\"United Arab Emirates\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"rank\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.0,\n        \"min\": 1.0,\n
\"max\": 1.0,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          1.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Recency\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 80,\n        \"min\": 0,\n        \"max\": 278,\n
\"num_unique_values\": 223,\n        \"samples\": [\n          11\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"df_recency"}
```

Calculaing the Frequency:

The number of times each customer made a purchase on the platform. Helps identify loyal customers.

```
# Grouping by Customer ID and Counting the number of Transactions
frequency = dataset.groupby(['CustomerID'])['InvoiceDate'].count()
print(frequency)

CustomerID
12346.0      2
12347.0    124
12348.0     28
12350.0     17
12352.0     48
          ...
18280.0     10
18281.0      7
18282.0      8
18283.0    400
18287.0     29
Name: InvoiceDate, Length: 3386, dtype: int64

# Converting the frequencies into dataframes

df_frequency = pd.DataFrame(frequency).reset_index()
df_frequency.columns = ['CustomerID', 'Frequency']
df_frequency.head()
```

```
{"summary":"{\n  \"name\": \"df_frequency\",\n  \"rows\": 3386,\n
\"fields\": [\n    {\n      \"column\": \"CustomerID\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1727.210404878609,\n        \"min\": 12346.0,\n        \"max\":
18287.0,\n        \"num_unique_values\": 3386,\n        \"samples\":
[\n          13008.0,\n          12857.0,\n          13101.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Frequency\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 146,\n        \"min\": 1,\n
\"max\": 4387,\n        \"num_unique_values\": 362,\n
\"samples\": [\n          62,\n          89,\n          34\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df_frequency"}
```

```python
# Merging the frequency with the recency data
rec_freq = df_frequency.merge(df_recency, on = 'CustomerID')
rec_freq.head()
```

```
{"summary":"{\n  \"name\": \"rec_freq\",\n  \"rows\": 72673,\n
\"fields\": [\n    {\n      \"column\": \"CustomerID\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1730.0256253358166,\n        \"min\": 12346.0,\n        \"max\":
18287.0,\n        \"num_unique_values\": 3386,\n        \"samples\":
[\n          13008.0,\n          12857.0,\n          13101.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Frequency\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 149,\n        \"min\": 1,\n
\"max\": 4387,\n        \"num_unique_values\": 362,\n
\"samples\": [\n          62,\n          89,\n          34\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"InvoiceNo\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3398,\n
\"samples\": [\n          \"554654\",\n          \"547415\",\n
\"547098\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"StockCode\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3175,\n
\"samples\": [\n          \"84249A\",\n          \"22434\",\n
\"84874B\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Description\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3243,\n
\"samples\": [\n          \"ENGLISH ROSE NOTEBOOK A6 SIZE\",\n
\"ZINC WIRE KITCHEN ORGANISER\",\n          \"PARTY INVITES
DINOSAURS\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Quantity\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 280.3382080680207,\n        \"min\": -
9360.0,\n        \"max\": 74215.0,\n        \"num_unique_values\":
148,\n        \"samples\": [\n          1400.0,\n          600.0,\n
110.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"InvoiceDate\",\n      \"properties\": {\n        \"dtype\":
\"date\",\n        \"min\": \"2010-12-01 08:26:00\",\n        \"max\":
\"2011-09-05 11:38:00\",\n        \"num_unique_values\": 3302,\n
\"samples\": [\n          \"2011-06-30 12:06:00\",\n          \"2011-
08-09 12:20:00\",\n          \"2011-06-14 10:00:00\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"UnitPrice\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
18.57135957026995,\n        \"min\": 0.0,\n        \"max\": 4287.63,\n
\"num_unique_values\": 193,\n        \"samples\": [\n          0.21,\n
41.75,\n          65.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Country\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 36,\n
\"samples\": [\n          \"Malta\",\n          \"Australia\",\n
\"Netherlands\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"rank\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.0,\n        \"min\": 1.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          1.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Recency\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
80,\n        \"min\": 0,\n        \"max\": 278,\n
\"num_unique_values\": 223,\n        \"samples\": [\n          42\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"rec_freq"}
```

Calculaing the Monetary Value:

The total amount each customer has spent on the platform.

```
# Calculating the value of each transaction - Quantity * Unit
rec_freq['Value'] = rec_freq['Quantity'] * rec_freq['UnitPrice']

# Grouping by customers and summing the total amount spent by each
customer.
m = rec_freq.groupby(['CustomerID'])['Value'].sum()
print(m)

CustomerID
12346.0    77183.60
12347.0      711.79
12348.0      892.80
12350.0      334.40
12352.0      296.50
             ...
18280.0      180.60
18281.0       80.82
18282.0      100.21
18283.0      108.45
18287.0      765.28
Name: Value, Length: 3386, dtype: float64
```

```
# Converting this into a DataFrame
m = pd.DataFrame(m)

# Renaming the column names
m.columns = ['Monetary_value']
m.head()
```

```
{"summary":"{\n  \"name\": \"m\",\n  \"rows\": 3386,\n  \"fields\": [\
n    {\n      \"column\": \"CustomerID\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1727.210404878609,\n
\"min\": 12346.0,\n        \"max\": 18287.0,\n
\"num_unique_values\": 3386,\n        \"samples\": [\n
13008.0,\n          12857.0,\n          13101.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Monetary_value\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1436.2865317068997,\n        \"min\": -4287.63,\n        \"max\":
77183.6,\n        \"num_unique_values\": 3183,\n        \"samples\":
[\n          200.47,\n          409.93,\n          330.79\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"m"}
```

```
# Merging the monetary value to recency and frequency

rfm = m.merge(rec_freq, on = 'CustomerID')

rfm.head()
```

```
{"summary":"{\n  \"name\": \"rfm\",\n  \"rows\": 72673,\n  \"fields\":
[\n    {\n      \"column\": \"CustomerID\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1730.0256253358166,\n
\"min\": 12346.0,\n        \"max\": 18287.0,\n
\"num_unique_values\": 3386,\n        \"samples\": [\n
13008.0,\n          12857.0,\n          13101.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Monetary_value\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
796.4817723751027,\n        \"min\": -4287.63,\n        \"max\":
77183.6,\n        \"num_unique_values\": 3183,\n        \"samples\":
[\n          200.47,\n          409.93,\n          330.79\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Frequency\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
149,\n        \"min\": 1,\n        \"max\": 4387,\n
\"num_unique_values\": 362,\n        \"samples\": [\n          62,\n
89,\n          34\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"InvoiceNo\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3398,\n
\"samples\": [\n          \"554654\",\n          \"547415\",\n
\"547098\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"StockCode\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3175,\n
\"samples\": [\n          \"84249A\",\n          \"22434\",\n
\"84874B\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Description\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3243,\n
\"samples\": [\n          \"ENGLISH ROSE NOTEBOOK A6 SIZE\",\n
\"ZINC WIRE KITCHEN ORGANISER\",\n          \"PARTY INVITES
DINOSAURS\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Quantity\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 280.3382080680207,\n        \"min\": -
9360.0,\n        \"max\": 74215.0,\n        \"num_unique_values\":
148,\n        \"samples\": [\n          1400.0,\n          600.0,\n
110.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"InvoiceDate\",\n      \"properties\": {\n        \"dtype\":
\"date\",\n        \"min\": \"2010-12-01 08:26:00\",\n        \"max\":
\"2011-09-05 11:38:00\",\n        \"num_unique_values\": 3302,\n
\"samples\": [\n          \"2011-06-30 12:06:00\",\n          \"2011-
08-09 12:20:00\",\n          \"2011-06-14 10:00:00\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"UnitPrice\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
18.57135957026995,\n        \"min\": 0.0,\n        \"max\": 4287.63,\n
\"num_unique_values\": 193,\n        \"samples\": [\n          0.21,\n
41.75,\n          65.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Country\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 36,\n
\"samples\": [\n          \"Malta\",\n          \"Australia\",\n
\"Netherlands\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"rank\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.0,\n        \"min\": 1.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          1.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Recency\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
80,\n        \"min\": 0,\n        \"max\": 278,\n
\"num_unique_values\": 223,\n        \"samples\": [\n          42\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Value\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
291.4874084731016,\n        \"min\": -4287.63,\n        \"max\":
77183.6,\n        \"num_unique_values\": 1285,\n        \"samples\":
```

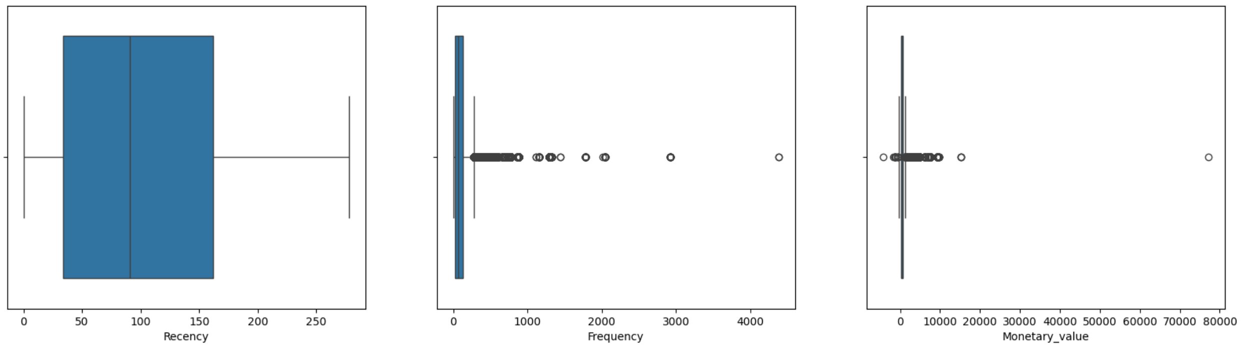
```
[\n          11.56\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"rfm"}
```

```
final_dataset = rfm[['CustomerID', 'Recency', 'Frequency',
'Monetary_value']]

final_dataset.head()
```

```
{"summary":"{\n  \"name\": \"final_dataset\",\n  \"rows\": 72673,\n
\"fields\": [\n    {\n      \"column\": \"CustomerID\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1730.0256253358166,\n        \"min\": 12346.0,\n        \"max\":
18287.0,\n        \"num_unique_values\": 3386,\n        \"samples\":
[\n          13008.0,\n          12857.0,\n          13101.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Recency\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 80,\n        \"min\": 0,\n        \"max\": 278,\n
\"num_unique_values\": 223,\n        \"samples\": [\n          42,\n
100,\n          76\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Frequency\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 149,\n        \"min\": 1,\n
\"max\": 4387,\n        \"num_unique_values\": 362,\n
\"samples\": [\n          62,\n          89,\n          34\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Monetary_value\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 796.4817723751027,\n        \"min\": -
4287.63,\n        \"max\": 77183.6,\n        \"num_unique_values\":
3183,\n        \"samples\": [\n          200.47,\n          409.93,\n
330.79\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"final_dataset"}
```

Removing Outliers

```
ls = ['Recency', 'Frequency', 'Monetary_value']
fig, axes = plt.subplots(1, 3, figsize = (20, 5)) # Create one subplot
with all 3 boxplots

for i, ax in zip(ls, axes):
    sns.boxplot(x = final_dataset[i], ax = ax)
plt.show()
```



Observations:

1. Recency has no visible outliers.
2. Frequency and Monetary Value have many outliers which needs to be removed before using to build the model.

To identify outliers, we will compute Z-Score. Z-Scores tell us how far away from the mean a data point is.

```
* Z-Score = 0 → The data point is exactly at the mean.
* Z-Score = 1 → The data point is 1 standard deviation above the mean.
* Z-Score = -1 → The data point is 1 standard deviation below the
mean.

from scipy import stats

new_rfm = final_dataset[['Recency', 'Frequency', 'Monetary_value']]
z_score = stats.zscore(new_rfm) # Computing z-score for each sample

# Removing samples with z_score < 3
abs_z_score = np.abs(z_score)
filtered_entries = (abs_z_score < 3).all(axis = 1) # Checking for
samples with z-score less than 3
new_rfm = new_rfm[filtered_entries]

print(new_rfm)

       Recency  Frequency  Monetary_value
1            6        124          711.79
2            6        124          711.79
3            6        124          711.79
4            6        124          711.79
5            6        124          711.79
...        ...        ...             ...
72668      172         29          765.28
72669      172         29          765.28
72670      172         29          765.28
72671      172         29          765.28
72672      172         29          765.28
```

```
[70330 rows x 3 columns]
```

Pre-processing the data

Standardization scales the data to have a mean of 0 and a standard deviation of 1.

```
from sklearn.preprocessing import StandardScaler

# Dropping duplicate samples
new_rfm = new_rfm.drop_duplicates()

col_names = new_rfm.columns
features = new_rfm[col_names]

# Scaling the data
scaler = StandardScaler()
features = scaler.fit_transform(features.values)

# Converting into DataFrame
features = pd.DataFrame(features, columns = col_names)
features.head()
```

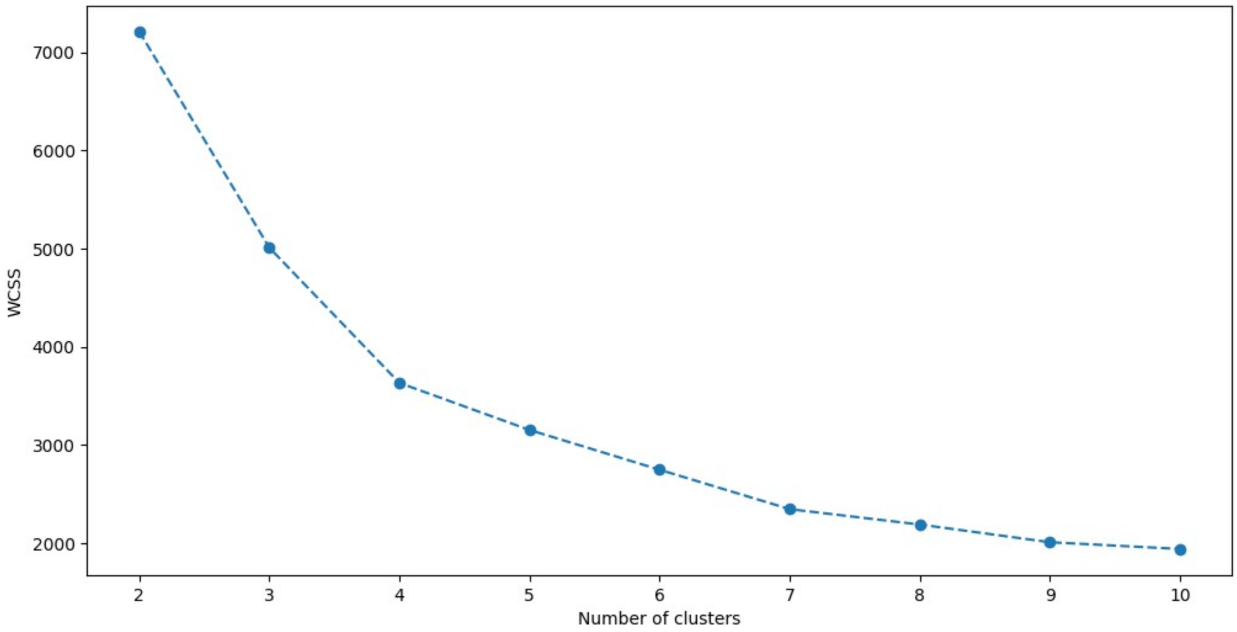
```
{"summary":"{\n  \"name\": \"features\",\n  \"rows\": 3341,\n
\"fields\": [\n    {\n      \"column\": \"Recency\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.0001496893953479,\n        \"min\": -1.224145316514637,\n
\"max\": 2.2816012353112196,\n        \"num_unique_values\": 223,\n
\"samples\": [\n          -0.6945001540085723,\n
1.8402302665561658,\n          1.2475321085136648\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Frequency\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.0001496893953432,\n        \"min\": -0.7831016291433658,\n
\"max\": 6.398088450657366,\n        \"num_unique_values\": 335,\n
\"samples\": [\n          0.4903452405836378,\n
0.8448098331880616,\n          -0.12668571691295155\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Monetary_value\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.0001496893953434,\n        \"min\": -5.619560772203546,\n
\"max\": 7.448607366190289,\n        \"num_unique_values\": 3132,\n
\"samples\": [\n          -0.14069073178788435,\n          -
1.4493032602207527,\n          0.05414430160523469\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"features"}
```

Building a Customer Segmentation Model using K-Means Clustering

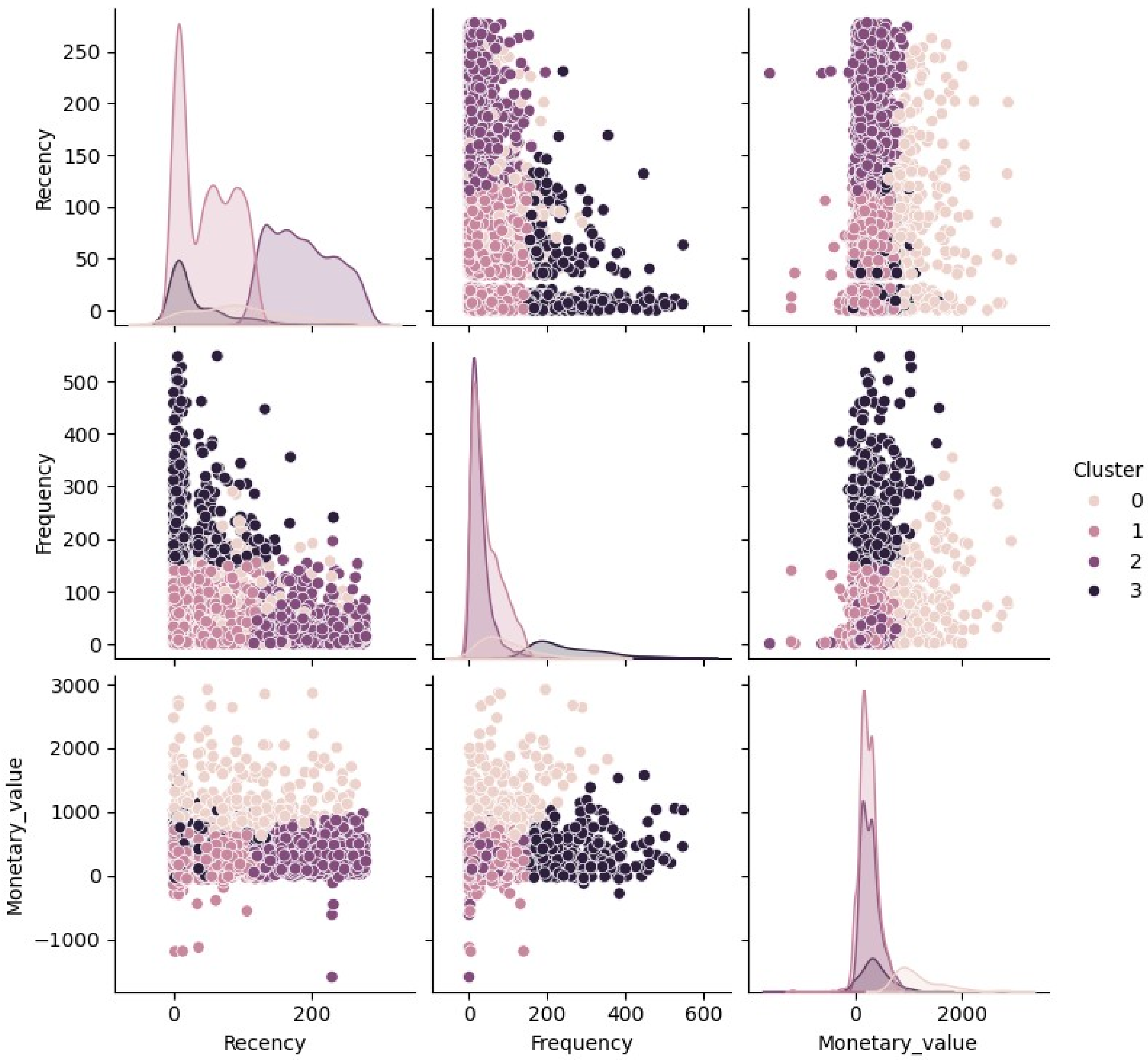
```python
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Finding the optimal number of clusters
WCSS = []
silhouette_scores = []
for clusters in range(2, 11):
    kmeans = KMeans(n_clusters = clusters, init = 'k-means++')
    kmeans.fit(features)
    WCSS.append(kmeans.inertia_)
    # Computing the sillouette score to evaluate the quality of the 
clustering
    score = silhouette_score(features, kmeans.labels_, 
metric='euclidean')
    silhouette_scores.append(score)

print(WCSS)
print(silhouette_scores)
```

```
[7212.682811509438, 5010.012822069706, 3631.9970919935427, 
3178.721429528645, 2730.227603867467, 2345.879024652273, 
2156.459892152007, 2027.0821758254745, 1890.329838229845]
[0.3085576796424791, 0.36060102322799437, 0.39283936219457005, 
0.39046300712818244, 0.3124417437919354, 0.3244519114675101, 
0.32949067006387356, 0.30802378594708674, 0.29655879849465044]
```

```python
# PLotting an elbow graph to visuaise the optimal number of clusters
plt.figure(figsize = (12,6))
plt.plot(range(2, 11), WCSS, marker = 'o', linestyle = '--')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



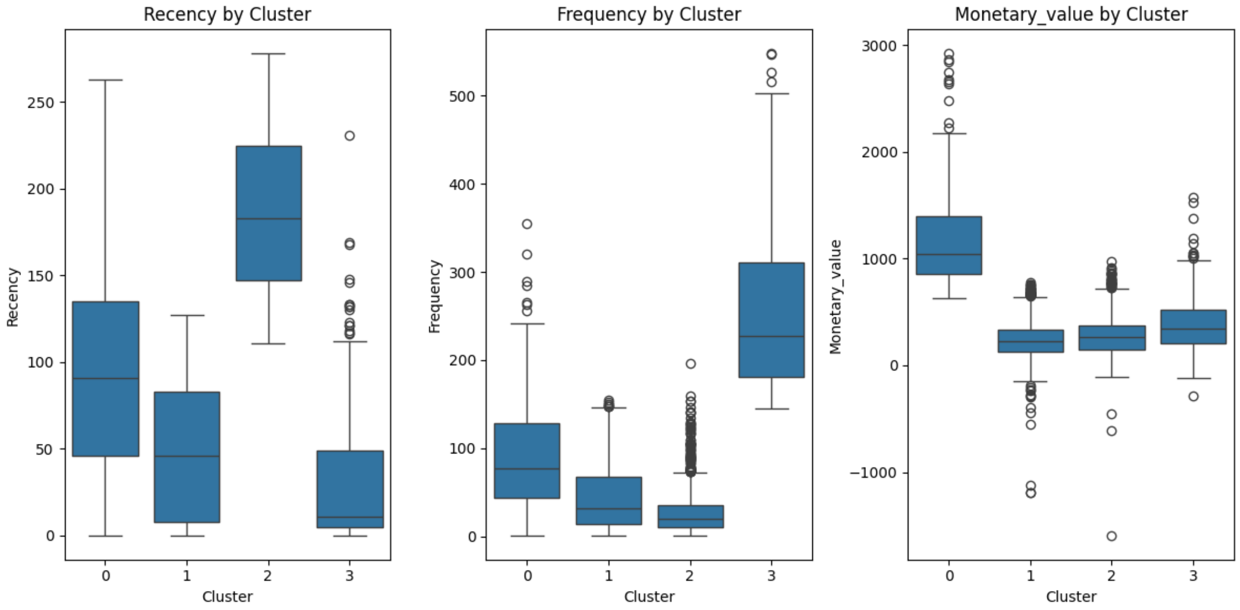
```
# Training the K-Means algorithm with 4 clusters

kmeans = KMeans(n_clusters = 4, init = 'k-means++')
y = kmeans.fit_predict(features)

new_rfm['Cluster'] = y
print(new_rfm)

       Recency  Frequency  Monetary_value  Cluster
1            6        124          711.79        1
32          15         28          892.80        0
49          63         17          334.40        1
66          77         48          296.50        1
81         169          4           89.00        2
...        ...        ...             ...      ...
72563       96         10          180.60        1
72573      193          7           80.82        2
72580      247          8          100.21        2
72587       36        400          108.45        3
72644      172         29          765.28        2

[3341 rows x 4 columns]

# Relationships between features and their cluster distribution.
sns.pairplot(new_rfm, hue = 'Cluster', vars=['Recency', 'Frequency',
'Monetary_value'])
plt.show()
```



Observations

1. Customers in Cluster 3 have a high frequency and Monetary value but less recency. Recent buyers, loyal and high spenders.
2. Custermers in Cluster 2 have high recency and moderate monetary value but low frequency Less engaged customers
3. Customers in Cluster 1 and 0 are intermediates.

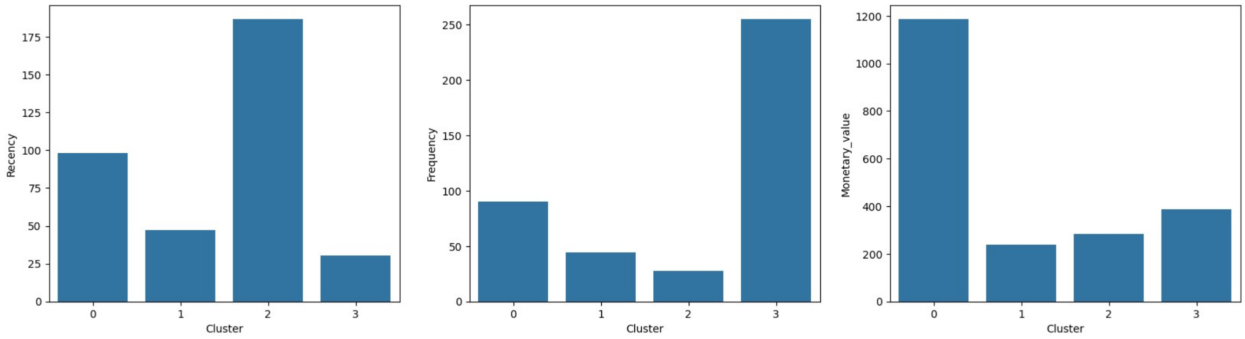
```python
import seaborn as sns

plt.figure(figsize=(12, 6))
for i, col in enumerate(['Recency', 'Frequency', 'Monetary_value']):
    plt.subplot(1, 3, i+1)
    sns.boxplot(x='Cluster', y=col, data=new_rfm)
    plt.title(f'{col} by Cluster')
plt.tight_layout()
plt.show()
```



Observations

1. Customers in Cluster 3 have low median **recency** and Cluster 2 have the highest median recency (at-risk).
2. Cluster 3 has the highest **frequency** and Cluster 2 has the lowest frequency.
3. Cluster 0 has the highest **monetary value** (high spenders) and cluster 1 has the lowest spenders.

```
avg_df = new_rfm.groupby(['Cluster'], as_index=False).mean()

fig, axes = plt.subplots(1, 3, figsize = (20, 5))
for i, ax in zip(ls, axes):
    sns.barplot(x = avg_df['Cluster'], y = avg_df[i], ax = ax)
plt.show()
```



Insights:

1. Cluster 3 has the most values customers - Focus on retention stratergies and offers.

2. Clusters 2 has the leasted engaged customers - Campaigns and offers to win them back
3. Cluster 0 - Moderate customers - Focus on converting them into High valued customers since they have a monetary value of moderate to high, and a moderate recency and frequency.