Retail Analytics

Review

Data Analysis

Enhancing Retail Performance through Transactional Data Analysis and RFM Analysis→

Dataset:

Customer Data→

```
customer =pd.read_csv('/content/Customer.csv')
customer
```

	customer_Id	DOB	Gender	city_code
0	268408	02-01-1970	M	4.0
1	269696	07-01-1970	F	8.0
2	268159	08-01-1970	F	8.0
3	270181	10-01-1970	F	2.0
4	268073	11-01-1970	M	1.0
...	...	...	...	...
5642	274474	19-12-1992	M	2.0
5643	267666	24-12-1992	M	6.0
5644	270476	25-12-1992	F	3.0
5645	269626	27-12-1992	F	5.0
5646	274308	29-12-1992	F	5.0

5647 rows × 4 columns

Product Hierarchy→

```python
prod_info= pd.read_csv('/content/prod_cat_info.csv')
prod_info
```

	prod_cat_code	prod_cat	prod_sub_cat_code	prod_subcat
0	1	Clothing	4	Mens
1	1	Clothing	1	Women
2	1	Clothing	3	Kids
3	2	Footwear	1	Mens
4	2	Footwear	3	Women
5	2	Footwear	4	Kids
6	3	Electronics	4	Mobiles
7	3	Electronics	5	Computers
8	3	Electronics	8	Personal Appliances
9	3	Electronics	9	Cameras
10	3	Electronics	10	Audio and video
11	4	Bags	1	Mens
12	4	Bags	4	Women
13	5	Books	7	Fiction
14	5	Books	12	Academic
15	5	Books	10	Non-Fiction
16	5	Books	11	Children
17	5	Books	3	Comics
18	5	Books	6	DIY
19	6	Home and kitchen	2	Furnishing
20	6	Home and kitchen	10	Kitchen

Transactions→

```python
transactions= pd.read_csv('/content/Transactions.csv')
transactions
```

	transaction_id	cust_id	tran_date	prod_subcat_code	prod_cat_code	Qty	Rate	Tax	total_amt	Store_type
0	80712190438	270351	28-02-2014	1	1	-5	-772	405.300	-4265.300	e-Shop
1	29258453508	270384	27-02-2014	5	3	-5	-1497	785.925	-8270.925	e-Shop
2	51750724947	273420	24-02-2014	6	5	-2	-791	166.110	-1748.110	TeleShop
3	93274880719	271509	24-02-2014	11	6	-3	-1363	429.345	-4518.345	e-Shop
4	51750724947	273420	23-02-2014	6	5	-2	-791	166.110	-1748.110	TeleShop
...	...	...	...	...	...	...	...	...	...	...
23048	94340757522	274550	25-01-2011	12	5	1	1264	132.720	1396.720	e-Shop
23049	89780862956	270022	25-01-2011	4	1	1	677	71.085	748.085	e-Shop
23050	85115299378	271020	25-01-2011	2	6	4	1052	441.840	4649.840	MBR
23051	72870271171	270911	25-01-2011	11	5	3	1142	359.730	3785.730	TeleShop
23052	77960931771	271961	25-01-2011	11	5	1	447	46.935	493.935	TeleShop

23053 rows × 10 columns

These datasets contain information about customers, product categories, and transaction details, respectively.

Data Analysis Techniques→

Data Integration and Merging:

- Merging multiple datasets (Customers, Product Hierarchy, Transactions) into a single comprehensive dataset (Customer_Final) using pandas merge operations.

```
customer_final= prod_concat.merge(customer,on='customer_Id',how='left')
customer_final
```

	transaction_id	customer_Id	tran_date	prod_subcat_code	prod_cat_code	Qty	Rate	Tax	total_amt	Store_type	prod_cat	prod_sub_cat_code	prod_subcat	DOB	Gender	city_code
0	80712190438	270351	28-02-2014	1	1	-5	-772	405.300	-4265.300	e-Shop	Clothing	1	Women	26-09-1981	M	5.0
1	29258453508	270384	27-02-2014	5	3	-5	-1497	785.925	-8270.925	e-Shop	Electronics	5	Computers	11-05-1973	F	8.0
2	51750724947	273420	24-02-2014	6	5	-2	-791	166.110	-1748.110	TeleShop	Books	6	DIY	27-07-1992	M	8.0
3	93274880719	271509	24-02-2014	11	6	-3	-1363	429.345	-4518.345	e-Shop	Home and kitchen	11	Bath	08-06-1981	M	3.0
4	51750724947	273420	23-02-2014	6	5	-2	-791	166.110	-1748.110	TeleShop	Books	6	DIY	27-07-1992	M	8.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
23048	94340757522	274550	25-01-2011	12	5	1	1264	132.720	1396.720	e-Shop	Books	12	Academic	21-02-1972	M	7.0
23049	89780862956	270022	25-01-2011	4	1	1	677	71.085	748.085	e-Shop	Clothing	4	Mens	27-04-1984	M	9.0
23050	85115299378	271020	25-01-2011	2	6	4	1052	441.840	4649.840	MBR	Home and kitchen	2	Furnishing	20-06-1976	M	8.0
23051	72870271171	270911	25-01-2011	11	5	3	1142	359.730	3785.730	TeleShop	Books	11	Children	22-05-1970	M	2.0
23052	77960931771	271961	25-01-2011	11	5	1	447	46.935	493.935	TeleShop	Books	11	Children	15-01-1982	M	1.0

23053 rows × 16 columns

```
customer_final.dtypes
```

```
transaction_id         int64
customer_Id            int64
tran_date             object
prod_subcat_code       int64
prod_cat_code          int64
Qty                    int64
Rate                   int64
Tax                  float64
total_amt            float64
Store_type            object
prod_cat              object
prod_sub_cat_code      int64
prod_subcat           object
DOB                   object
Gender                object
city_code            float64
dtype: object
```

Data Cleaning and Transformation:

- Renaming columns, converting data types (tran_date to datetime, DOB to datetime), handling missing values (city_code), and creating new derived features (age).

Renaming Columns-

```
[5] transactions.rename(columns = {'Prod_sub_cat_code': 'prod_subcat_code'},inplace = True)
    transactions.head()
```

	transaction_id	cust_id	tran_date	prod_subcat_code	prod_cat_code	Qty	Rate	Tax	total_amt	Store_type
0	80712190438	270351	28-02-2014	1	1	-5	-772	405.300	-4265.300	e-Shop
1	29258453508	270384	27-02-2014	5	3	-5	-1497	785.925	-8270.925	e-Shop
2	51750724947	273420	24-02-2014	6	5	-2	-791	166.110	-1748.110	TeleShop
3	93274880719	271509	24-02-2014	11	6	-3	-1363	429.345	-4518.345	e-Shop
4	51750724947	273420	23-02-2014	6	5	-2	-791	166.110	-1748.110	TeleShop

```
[6] prod_concat = transactions.merge(prod_info, left_on=['prod_cat_code', 'prod_subcat_code'], right_on=['prod_cat_code', 'prod_sub_cat_code'], how ='left')
    prod_concat.head()
```

	transaction_id	cust_id	tran_date	prod_subcat_code	prod_cat_code	Qty	Rate	Tax	total_amt	Store_type	prod_cat	prod_sub_cat_code	prod_subcat
0	80712190438	270351	28-02-2014	1	1	-5	-772	405.300	-4265.300	e-Shop	Clothing	1	Women
1	29258453508	270384	27-02-2014	5	3	-5	-1497	785.925	-8270.925	e-Shop	Electronics	5	Computers
2	51750724947	273420	24-02-2014	6	5	-2	-791	166.110	-1748.110	TeleShop	Books	6	DIY
3	93274880719	271509	24-02-2014	11	6	-3	-1363	429.345	-4518.345	e-Shop	Home and kitchen	11	Bath
4	51750724947	273420	23-02-2014	6	5	-2	-791	166.110	-1748.110	TeleShop	Books	6	DIY

```
[7] prod_concat.rename(columns={'cust_id':'customer_Id'},inplace=True)
```

Converting Data types-

```
[21]

    # Convert the 'tran_date' column to datetime
    customer_final['tran_date'] = pd.to_datetime(customer_final['tran_date'], dayfirst=True, errors='coerce')

    # Display the result
    print(customer_final['tran_date'])
```

```
0        2014-02-28
1        2014-02-27
2        2014-02-24
3        2014-02-24
4        2014-02-23
            ...
23048    2011-01-25
23049    2011-01-25
23050    2011-01-25
23051    2011-01-25
23052    2011-01-25
Name: tran_date, Length: 23053, dtype: datetime64[ns]
```

Descriptive Statistics:

- Generating descriptive statistics including mean, standard deviation, quartiles, and counts for continuous variables (describe() function).

Summary Statistics-

```
[11] customer_final.describe()
```

	transaction_id	customer_Id	prod_subcat_code	prod_cat_code	Qty	Rate	Tax	total_amt	prod_sub_cat_code	city_code
count	2.305300e+04	23053.000000	23053.000000	23053.000000	23053.000000	23053.000000	23053.000000	23053.000000	23053.000000	23045.000000
mean	5.007348e+10	271021.746497	6.149091	3.763632	2.432395	636.369713	248.667192	2107.308002	6.149091	5.482534
std	2.898194e+10	2431.692059	3.726372	1.677016	2.268406	622.363498	187.177773	2507.561264	3.726372	2.863499
min	3.268991e+06	266783.000000	1.000000	1.000000	-5.000000	-1499.000000	7.350000	-8270.925000	1.000000	1.000000
25%	2.493864e+10	268935.000000	3.000000	2.000000	1.000000	312.000000	98.280000	762.450000	3.000000	3.000000
50%	5.009313e+10	270980.000000	5.000000	4.000000	3.000000	710.000000	199.080000	1754.740000	5.000000	5.000000
75%	7.533000e+10	273114.000000	10.000000	5.000000	4.000000	1109.000000	365.715000	3569.150000	10.000000	8.000000
max	9.998755e+10	275265.000000	12.000000	6.000000	5.000000	1500.000000	787.500000	8287.500000	12.000000	10.000000

Interpretation→

 Transaction and Customer Details:

- **Count**: There are 23,053 transactions recorded in the dataset, indicating the size of the sample.
- **Customer_ID**: The average customer ID is approximately 271,022. This doesn't necessarily provide actionable insights but confirms the typical range of customer IDs in the dataset.
- **prod_subcat_code and prod_cat_code**: The mean values for these codes suggest the most common subcategories and categories that customers purchase from.
- **Qty (Quantity)**: On average, customers purchase approximately 2.43 items per transaction. This average quantity gives a sense of typical purchase behavior.
- **Rate**: The average rate at which items are priced in transactions is around 636.37 units. This average rate indicates the typical pricing level of items sold.
- **Tax**: The average tax amount added to transactions is 248.67 units. This figure shows the average additional cost due to taxes in each transaction.
- **total_amt (Total Amount)**: The mean total amount spent per transaction is approximately 2,107.31 units. This average total amount indicates the typical expenditure per transaction.

 Data Distribution:

- **Standard Deviation**: The high standard deviations for Qty, Rate, and total_amt suggest that these variables have considerable variability across transactions. This variability implies that transactions vary widely in terms of quantity purchased, item rates, and total expenditure.

- **Min and Max Values**: The minimum and maximum values provide insights into transaction extremes:
  - The minimum quantity is -5, suggesting the presence of returns or refunds in the dataset.
  - The maximum rate and total amount are 1,500 and 8,287.50 units, respectively, indicating the highest prices and transaction amounts observed in the dataset.

 Percentiles:

- **25th, 50th (Median), and 75th Percentiles**: These values provide insights into the distribution of transaction attributes:
  - For example, the median quantity (50th percentile) is 3, indicating that half of the transactions involve purchasing 3 items or fewer.
  - The median total amount spent (50th percentile) is 1,754.74 units, suggesting that half of the transactions involve spending this amount or less.

 Business Insights:

- **Pricing Strategy**: The average rate and distribution of rates can help businesses assess their pricing strategy and competitiveness.
- **Inventory Management**: Understanding the average quantity purchased and variability in quantities helps in optimizing inventory levels and predicting demand.
- **Customer Spending Behavior**: The average total amount spent per transaction and its distribution provide insights into customer spending patterns and preferences.
- **Risk Assessment**: Identifying outliers and understanding variability through standard deviations helps in assessing financial risks associated with transactions.

Frequency tables for all the categorical variables-

```python
customer_final.loc[:,customer_final.dtypes=="object"].describe()
```

	tran_date	Store_type	prod_cat	prod_subcat	DOB	Gender
count	23053	23053	23053	23053	23053	23044
unique	1129	4	6	18	3987	2
top	13-07-2011	e-Shop	Books	Women	27-12-1988	M
freq	35	9311	6069	3048	32	11811

Interpretation→

 tran_date:

- **Count**: There are 23,053 transactions recorded.
- **Unique**: Transactions span across 1,129 unique dates.
- **Top**: The most frequent transaction date is 13-07-2011, appearing 35 times.
- **Frequency**: This date was the most common date for transactions in your dataset.

 Store_type:

- **Count**: There are 23,053 entries, each indicating the type of store where the transaction occurred.
- **Unique**: There are 4 unique store types.
- **Top**: The most common store type is "e-Shop", appearing 9,311 times.
- **Frequency**: "e-Shop" is the predominant store type in your dataset.

 prod_cat:

- **Count**: There are 23,053 entries categorizing the products into different categories.
- **Unique**: There are 6 unique product categories.
- **Top**: The most frequent product category is "Books", appearing 6,069 times.
- **Frequency**: "Books" is the most purchased category among the listed categories.

 prod_subcat:

- **Count**: There are 23,053 entries categorizing the products into subcategories.

- **Unique**: There are 18 unique product subcategories.
- **Top**: The most frequent product subcategory is "Women", appearing 3,048 times.
- **Frequency**: Products categorized under "Women" subcategory are the most frequently purchased.

⍰ **DOB** (Date of Birth):

- **Count**: There are 23,044 entries for date of birth.
- **Unique**: There are 3,987 unique dates of birth recorded.
- **Top**: The most common date of birth recorded is 27-12-1988, appearing 32 times.
- **Frequency**: People born on 27-12-1988 are the most frequent customers in terms of their date of birth.

⍰ **Gender**:

- **Count**: There are 23,053 entries for gender.
- **Unique**: There are 2 unique genders (assuming Male and Female).
- **Top**: The most frequent gender recorded is Male (M), appearing 11,811 times.
- **Frequency**: Males constitute a larger portion of the customer base in your dataset compared to females.

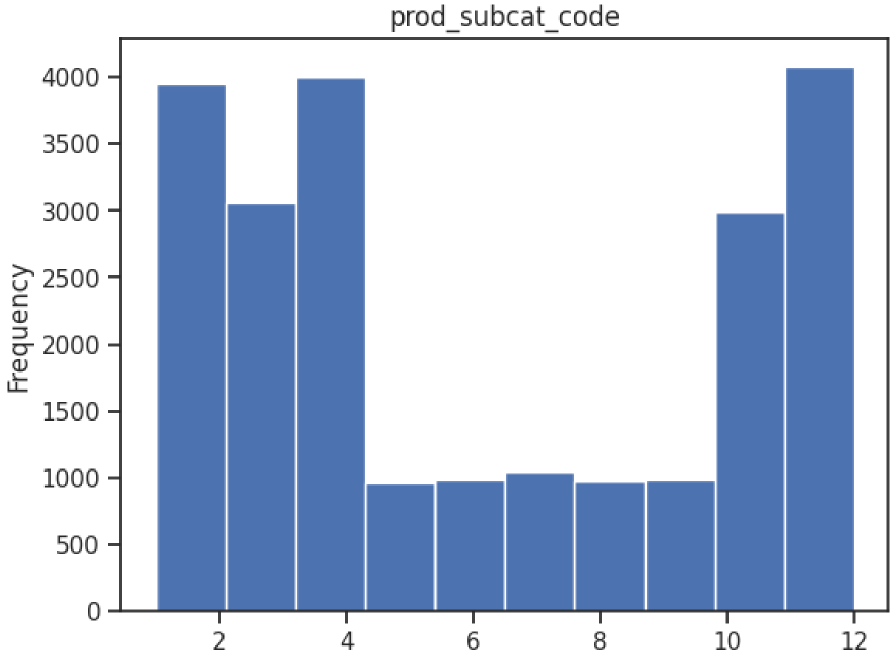
Visualization:

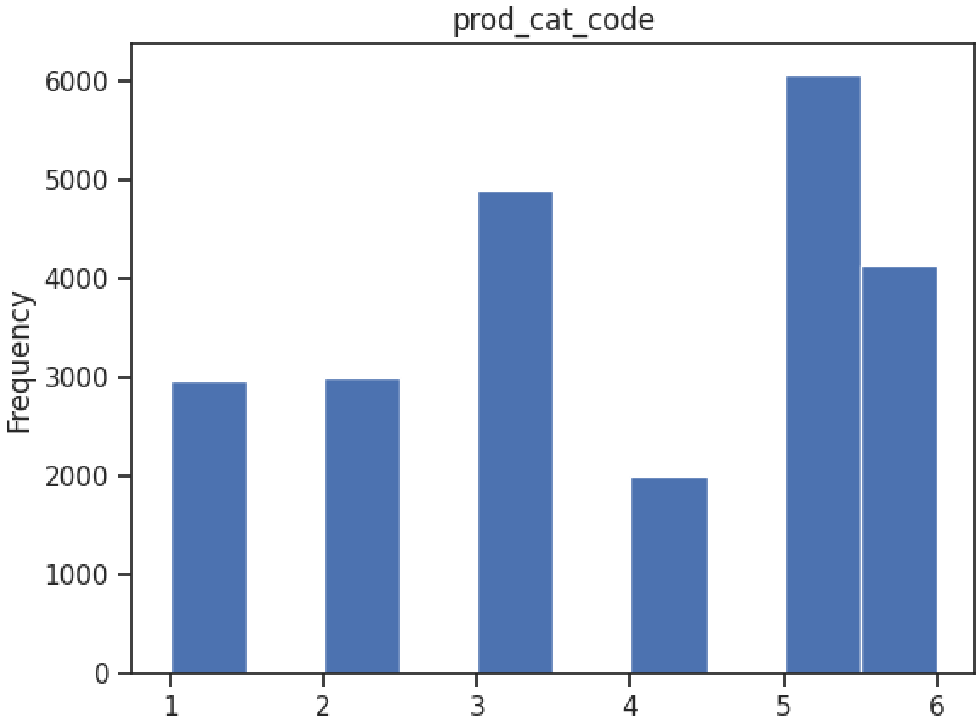
- Creating histograms for continuous variables to visualize their distributions.

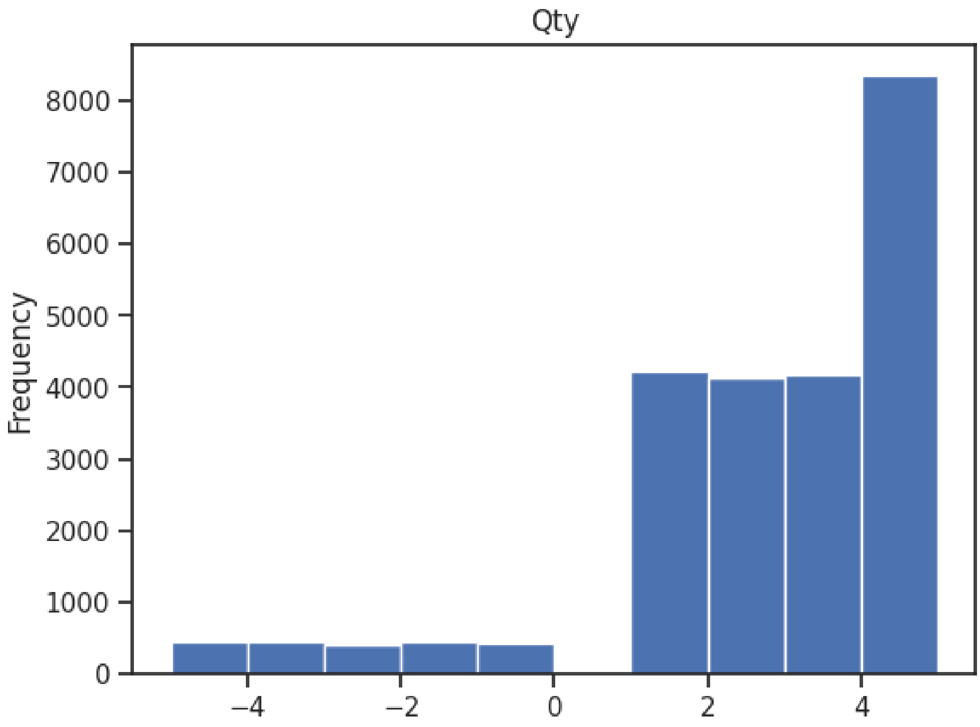
```
conti_customer = customer_final.loc[:,['prod_subcat_code','prod_cat_code', 'Qty', 'Rate', 'Tax', 'total_amt']]
conti_customer.columns
```

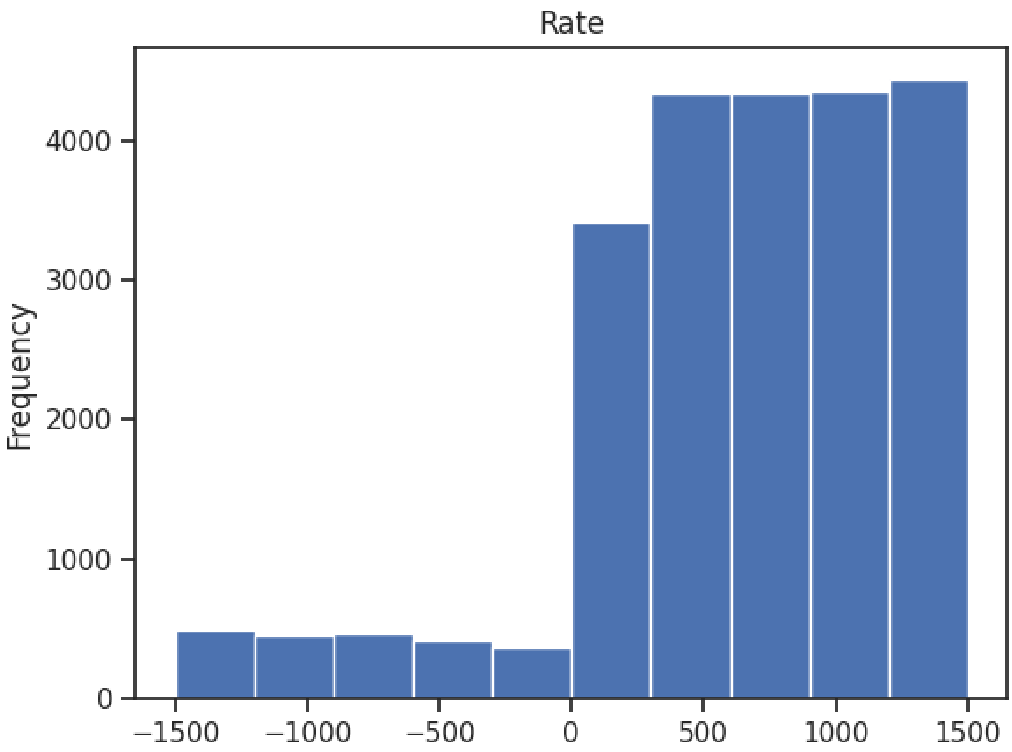
```
Index(['prod_subcat_code', 'prod_cat_code', 'Qty', 'Rate', 'Tax', 'total_amt'], dtype='object')
```

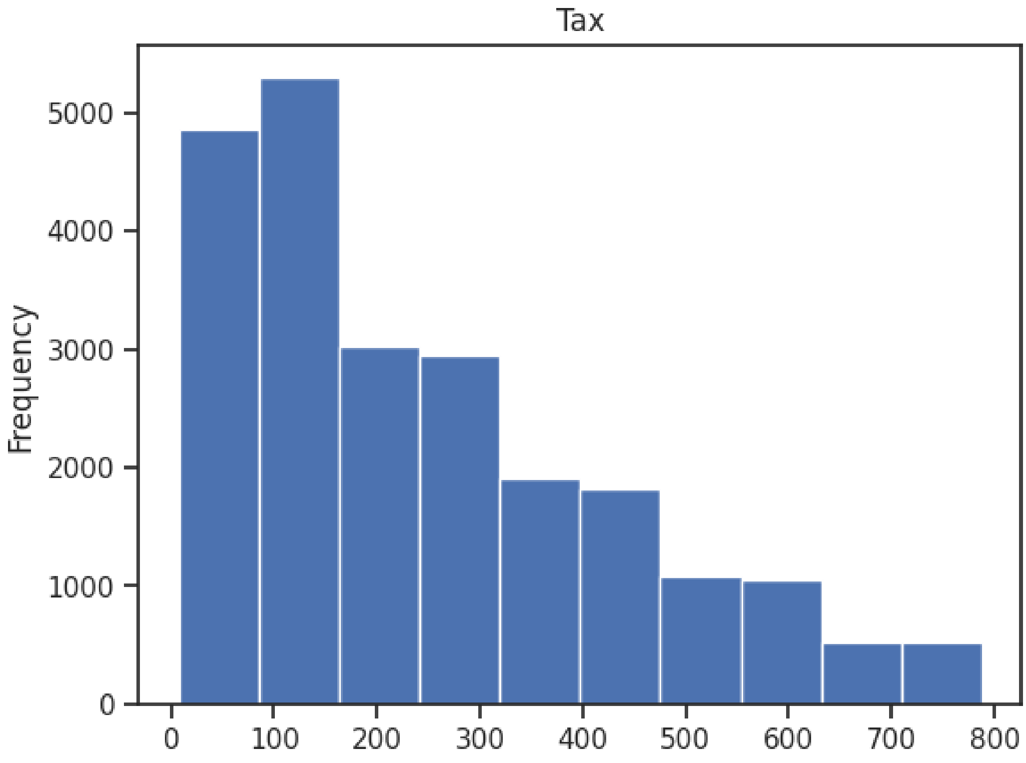
[14]
```
for i in conti_customer.columns:
    conti_customer[i].plot(kind='hist')
    plt.title(i)
    plt.show()
```

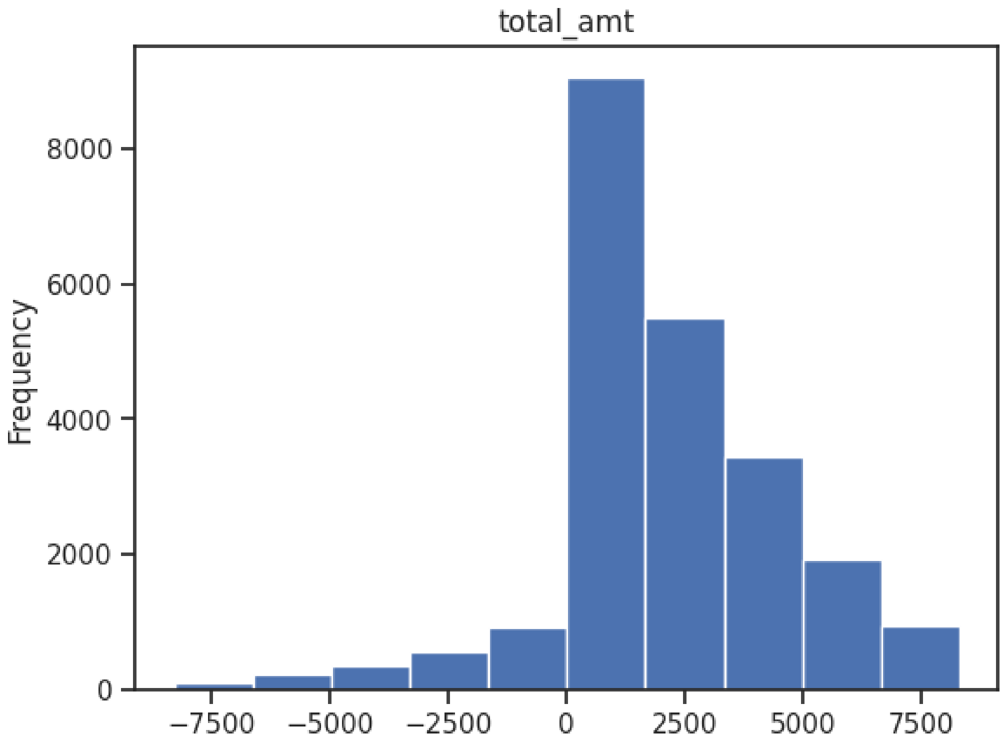





Qty
Frequency
8000
7000
6000
5000
4000
3000
2000
1000
0
−4
−2
0
2
4


Rate
Frequency
4000
3000
2000
1000
0
−1500
−1000
−500
0
500
1000
1500



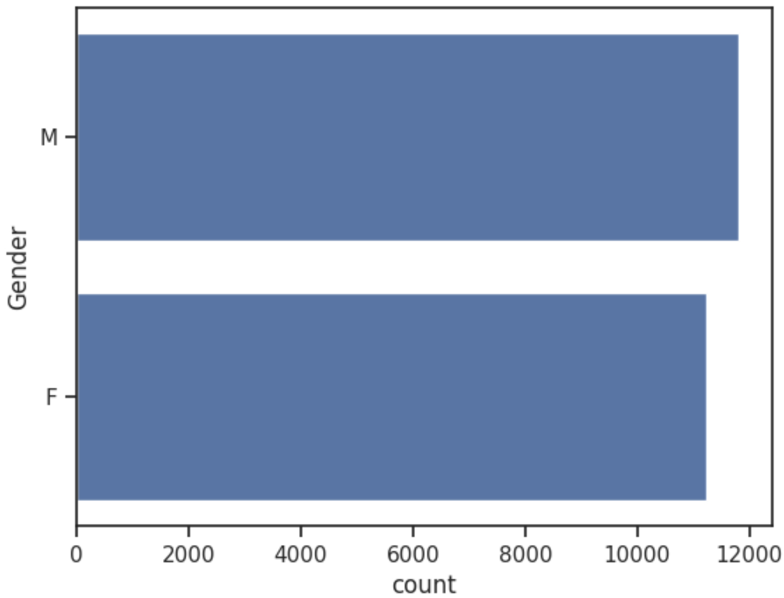


- Using seaborn for categorical data visualizations such as count plots and bar plots to show frequency distributions across different categories (Store_type, prod_cat, prod_subcat, Gender).

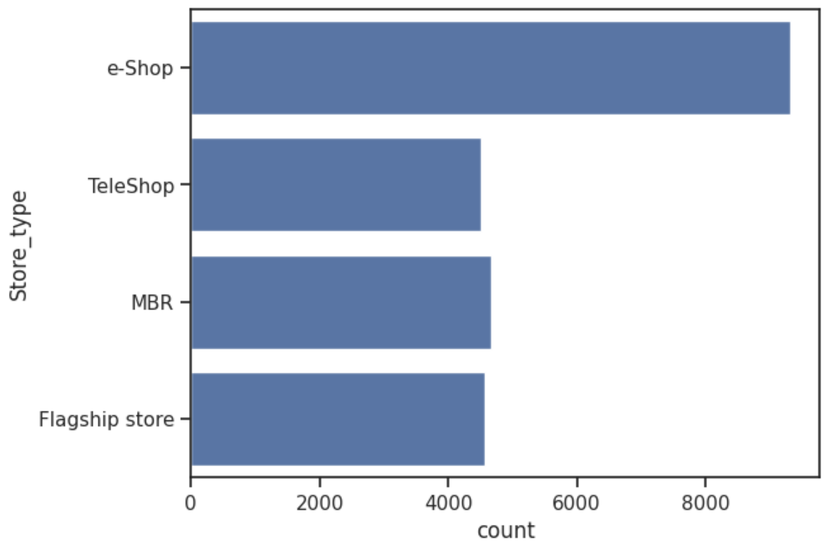
```
[15] cat = customer_final.loc[:, (customer_final.dtypes=='object')]
     cat.columns
```

```
Index(['tran_date', 'Store_type', 'prod_cat', 'prod_subcat', 'DOB', 'Gender'], dtype='object')
```

```
[16] sns.countplot(customer_final['Gender'])
     plt.show()
```

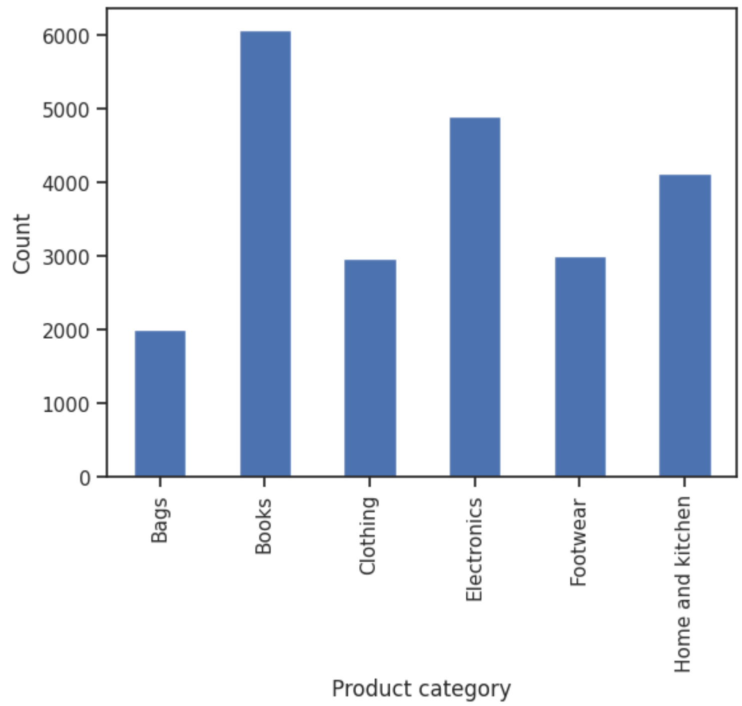



```
sns.countplot(customer_final['Store_type'])
plt.show()
```



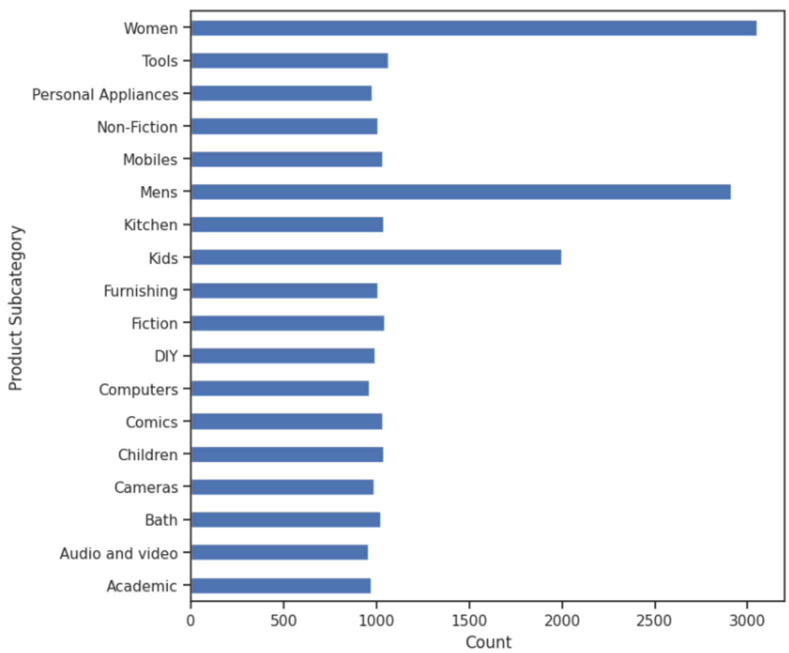
Here, we can see that e-shop is the most sought out Store_type.

```python
customer_final.groupby('prod_cat')['prod_cat'].count().plot(kind='bar')
plt.xlabel('Product category')
plt.ylabel('Count')
plt.show()
```



Here, we can see that Books are the most sought out Product Category.

```python
plt.figure(figsize=(8,8))
customer_final.groupby('prod_subcat')['prod_subcat'].count().plot(kind='barh')
plt.xlabel('Count')
plt.ylabel('Product Subcategory')
plt.show()
```



Here, we can see that women and men are the most used Product Subcategory.

Time Series Analysis:

- Analyzing the time period of the available transaction data (tran_date), finding the earliest and latest dates, and determining the duration of the dataset.
- Time period of the available transaction data

Outputs and Interpretation:

```
[21]
# Convert the 'tran_date' column to datetime
customer_final['tran_date'] = pd.to_datetime(customer_final['tran_date'], dayfirst=True, errors='coerce')

# Display the result
print(customer_final['tran_date'])
```

```
0        2014-02-28
1        2014-02-27
2        2014-02-24
3        2014-02-24
4        2014-02-23
            ...
23048    2011-01-25
23049    2011-01-25
23050    2011-01-25
23051    2011-01-25
23052    2011-01-25
Name: tran_date, Length: 23053, dtype: datetime64[ns]
```

```
[23] min_date = customer_final["tran_date"].min()
     min_date
```

```
Timestamp('2011-01-25 00:00:00')
```

```
[24] max_date = customer_final['tran_date'].max()
     max_date
```

```
Timestamp('2014-02-28 00:00:00')
```

Duration-

```
[25] print('The time period is from '+ pd.Timestamp.strftime(min_date,format='%d-%m-%Y') +' to ' + pd.Timestamp.strftime(max_date,format='%d-%m-%Y'))
     print()
```

```
The time period is from 25-01-2011 to 28-02-2014
```

Grouping and Aggregation & Comparative Analysis:

- Grouping data by categorical variables (prod_cat, Gender, Store_type, city_code) and performing aggregation functions (count, sum) to derive insights into transaction volumes and total amounts.

- Comparing product category preferences between genders (prod_cat vs Gender), identifying popular categories, and determining differences in purchasing behavior.

Outputs and Interpretation:

```
count_cat = customer_final.groupby(['prod_cat', 'Gender'])['transaction_id'].count().reset_index()
count_cat.rename(columns = {'transaction_id':'count'}, inplace = True)
count_cat
```

	prod_cat	Gender	count
0	Bags	F	994
1	Bags	M	1004
2	Books	F	2949
3	Books	M	3116
4	Clothing	F	1439
5	Clothing	M	1518
6	Electronics	F	2328
7	Electronics	M	2570
8	Footwear	F	1529
9	Footwear	M	1469
10	Home and kitchen	F	1994
11	Home and kitchen	M	2134

Interpretation→

 Gender Comparison: This table allows you to compare transaction counts between male and female customers across different product categories. For instance, in most categories, males tend to have slightly higher transaction counts than females, except for Clothing and Footwear where females have slightly higher counts.

 Product Category Insights: You can see which product categories are more popular among male and female customers. For example, Books and Electronics show relatively higher transaction volumes overall, with Electronics being more popular among males.

□ **Marketing and Inventory**: These insights can guide marketing strategies (targeting specific genders for certain categories) and inventory management (ensuring sufficient stock for popular categories among each gender).

```
max_tran = count_cat.groupby('prod_cat')['count'].max()
max_gender = count_cat.merge(max_tran, on='count', how='right')
max_gender
```

	prod_cat	Gender	count
0	Bags	M	1004
1	Books	M	3116
2	Clothing	M	1518
3	Electronics	M	2570
4	Footwear	F	1529
5	Home and kitchen	M	2134

Interpretation:

- **Gender Dominance**: In most categories, the highest transaction counts are associated with males, except for Footwear where females have the highest count.
- **Category Popularity**: This table reinforces which product categories are most popular among each gender based on transaction volumes.
- **Marketing Insights**: Knowing which gender dominates in each category can guide targeted marketing efforts and product promotions aimed at maximizing sales and customer engagement.

Which City code has the maximum customers and what was the percentage of customers from that city:

```
city_count = customer_final.groupby('city_code')['transaction_id'].count().reset_index()
city_count
```

	city_code	transaction_id
0	1.0	2258
1	2.0	2270
2	3.0	2411
3	4.0	2422
4	5.0	2360
5	6.0	2127
6	7.0	2356
7	8.0	2330
8	9.0	2178
9	10.0	2333

```
city_count[city_count.transaction_id == city_count.transaction_id.max()]
```

	city_code	transaction_id
3	4.0	2422

Amount wise maximum product:

```
#amount wise maximum product
value = customer_final.groupby('Store_type')['total_amt'].sum().reset_index()
value
value[value.total_amt == value.total_amt.max()]
```

	Store_type	total_amt
3	e-Shop	19824816.05

What was the total amount earned from the "Electronics" and "Clothing" categories from Flagship Stores:

```
category = customer_final.groupby(['Store_type', 'prod_cat'])['total_amt'].sum().reset_index()
flagship = category[category.Store_type == 'Flagship store']
Electronic_clothes = flagship[(flagship.prod_cat == 'Electronics') | (flagship.prod_cat == 'Clothing')]
Electronic_clothes
```

	Store_type	prod_cat	total_amt
2	Flagship store	Clothing	1194423.23
3	Flagship store	Electronics	2215136.04

Conditional Filtering & Customer Segmentation:

- Filtering data based on conditions such as transaction amount (negative vs positive), customer age groups (25-35 years), and specific date ranges (tran_date).

- Segmenting customers based on age groups (age), analyzing spending patterns across different categories (prod_cat), and calculating total amounts spent within specific date ranges (1st Jan 2014 to 1st Mar 2014).

Outputs and Interpretation:

```
postive_trans = customer_final[customer_final.total_amt>0].reset_index(drop=True)
postive_trans
```

	transaction_id	customer_Id	tran_date	prod_subcat_code	prod_cat_code	Qty	Rate	Tax	total_amt	Store_type	prod_cat	prod_sub_cat_code	prod_subcat	DOB	Gender	city_code
0	29258453508	270384	2014-02-20	5	3	5	1497	785.925	8270.925	e-Shop	Electronics	5	Computers	11-05-1973	F	8.0
1	25455265351	267750	2014-02-20	12	6	3	1360	428.400	4508.400	e-Shop	Home and kitchen	12	Tools	13-10-1986	M	1.0
2	1571002198	275023	2014-02-20	6	5	4	587	246.540	2594.540	e-Shop	Books	6	DIY	09-03-1971	M	6.0
3	36554696014	269345	2014-02-20	3	5	3	1253	394.695	4153.695	e-Shop	Books	3	Comics	26-06-1970	F	10.0
4	56814940239	268799	2014-02-20	7	5	5	368	193.200	2033.200	e-Shop	Books	7	Fiction	27-06-1979	M	9.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
20871	94340757522	274550	2011-01-25	12	5	1	1264	132.720	1396.720	e-Shop	Books	12	Academic	21-02-1972	M	7.0
20872	89780862956	270022	2011-01-25	4	1	1	677	71.085	748.085	e-Shop	Clothing	4	Mens	27-04-1984	M	9.0
20873	85115299378	271020	2011-01-25	2	6	4	1052	441.840	4649.840	MBR	Home and kitchen	2	Furnishing	20-06-1976	M	8.0
20874	72870271171	270911	2011-01-25	11	5	3	1142	359.730	3785.730	TeleShop	Books	11	Children	22-05-1970	M	2.0
20875	77960931771	271961	2011-01-25	11	5	1	447	46.935	493.935	TeleShop	Books	11	Children	15-01-1982	M	1.0

20876 rows × 16 columns

The postive_trans dataframe filters out transactions where the total_amt is greater than 0, indicating only transactions with a positive amount. Here's an overview of the dataframe structure and what it implies:

- **Structure**: The dataframe has 20,876 rows and 16 columns.

- **Columns**: The columns include transaction-related details such as transaction_id, customer_Id, tran_date, prod_subcat_code, prod_cat_code, Qty, Rate, Tax, total_amt, Store_type, prod_cat, prod_sub_cat_code, prod_subcat, DOB, Gender, and city_code.

Implications:

1. **Positive Transactions**: These are transactions where customers have made purchases resulting in a positive monetary value (total_amt > 0).

2. **Filtered Data**: By focusing on positive transactions, we exclude scenarios such as returns or refunds, providing a clearer picture of revenue-generating activities.

3. **Analysis Scope**: This dataset is ideal for analyzing sales trends, customer preferences based on purchases, seasonal variations in buying behavior, and more.

4. **Insights Generation**: Through this filtered dataset, businesses can derive insights into profitable product categories, popular store types, demographic purchasing patterns (based on Gender and DOB), and geographical preferences (city_code).

Negative Transactions-

```
negative_trans = customer_final.loc[customer_final["total_amt"] < 0,"transaction_id"].count()
print('Total negative transacatios:',negative_trans)
```

```
Total negative transacatios: 2177
```

For all customers aged between 25 – 35:

```
# Converting DOB to datetime

customer_final.DOB = pd.to_datetime(customer_final.DOB, format = "%d-%m-%Y")
DOB = customer_final.DOB

# Finding age of each individual
customer_final['age'] = DOB.apply(lambda x: pd.to_datetime('today').year-x.year)

# Clipping age betwen 25-35
age_barred = customer_final[(customer_final.age>=25) & (customer_final.age<=35)].reset_index(drop=True)
age_barred.head()
```

	transaction_id	customer_Id	tran_date	prod_subcat_code	prod_cat_code	Qty	Rate	Tax	total_amt	Store_type	prod_cat	prod_sub_cat_code	prod_subcat	DOB	Gender	city_code	age
0	51750724947	273420	2014-02-24	6	5	-2	-791	166.110	-1748.110	TeleShop	Books	6	DIY	1992-07-27	M	8.0	32
1	51750724947	273420	2014-02-23	6	5	-2	-791	166.110	-1748.110	TeleShop	Books	6	DIY	1992-07-27	M	8.0	32
2	91116291703	268509	2014-02-20	1	2	4	1243	522.060	5494.060	MBR	Footwear	1	Mens	1989-08-17	M	10.0	35
3	88853694830	268444	2014-02-20	4	4	-3	-80	25.200	-265.200	MBR	Bags	4	Women	1992-01-02	F	6.0	32
4	31384765864	267058	2014-02-19	3	2	1	793	83.265	876.265	e-Shop	Footwear	3	Women	1992-02-06	F	10.0	32

What was the total amount spent for “Electronics” and “Books” product categories?

```
[ ] age_catg = age_barred.groupby('prod_cat')['total_amt'].sum().reset_index()
    age_catg
```

	prod_cat	total_amt
0	Bags	696442.825
1	Books	2109168.750
2	Clothing	1179706.840
3	Electronics	1819301.835
4	Footwear	1100413.145
5	Home and kitchen	1493283.740

```
[ ] age_catg[(age_catg.prod_cat=='Electronics') | (age_catg.prod_cat=='Books')]
```

	prod_cat	total_amt
1	Books	2109168.750
3	Electronics	1819301.835

What was the total amount spent by these customers between 1st Jan, 2014 to 1st Mar, 2014?

```
age_barred.tran_date = pd.to_datetime(age_barred.tran_date, format='%d-%m-%Y')
date_barred = age_barred[(age_barred.tran_date > pd.to_datetime('01-01-2014', format='%d-%m-%Y')) & (age_barred.tran_date<pd.to_datetime('01-03-2014', format='%d-%m-%Y'))].reset_index(drop=True)
date_barred
```

	transaction_id	customer_Id	tran_date	prod_subcat_code	prod_cat_code	Qty	Rate	Tax	total_amt	Store_type	prod_cat	prod_sub_cat_code	prod_subcat	DOB	Gender	city_code	age
0	51750724947	273420	2014-02-24	6	5	-2	-791	166.110	-1748.110	TeleShop	Books	6	DIY	1992-07-27	M	8.0	32
1	51750724947	273420	2014-02-23	6	5	-2	-791	166.110	-1748.110	TeleShop	Books	6	DIY	1992-07-27	M	8.0	32
2	91116291703	268509	2014-02-20	1	2	4	1243	522.060	5494.060	MBR	Footwear	1	Mens	1989-08-17	M	10.0	35
3	88853694830	268444	2014-02-20	4	4	-3	-80	25.200	-265.200	MBR	Bags	4	Women	1992-01-02	F	6.0	32
4	31384765864	267058	2014-02-19	3	2	1	793	83.265	876.265	e-Shop	Footwear	3	Women	1992-02-06	F	10.0	32
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
173	77232232822	273564	2014-01-03	3	5	3	436	137.340	1445.340	e-Shop	Books	3	Comics	1992-03-11	F	3.0	32
174	32899104170	270064	2014-01-02	2	6	3	802	252.630	2658.630	TeleShop	Home and kitchen	2	Furnishing	1990-02-22	F	3.0	34
175	14302941720	273058	2014-01-02	1	2	1	674	70.770	744.770	e-Shop	Footwear	1	Mens	1989-08-11	F	10.0	35
176	32125935023	272067	2014-01-02	10	3	5	1211	635.775	6690.775	e-Shop	Electronics	10	Audio and video	1989-05-14	M	5.0	35
177	91039584326	272165	2014-01-02	10	5	1	1268	133.140	1401.140	e-Shop	Books	10	Non-Fiction	1990-12-01	M	4.0	34

178 rows × 17 columns

```
date_barred.total_amt.sum()
```

```
416304.32999999996
```

RFM Analysis:

RFM analysis involves evaluating three key aspects of customer behavior:

- **Recency (R):** How recently did the customer make a purchase?
- **Frequency (F):** How often does the customer make a purchase?
- **Monetary Value (M):** How much does the customer spend on each purchase?

These three metrics are used to segment customers into different groups, allowing businesses to target them with appropriate marketing strategies and personalized experiences.

```
# Convert the 'tran_date' column to datetime
transactions['tran_date'] = pd.to_datetime(transactions['tran_date'], dayfirst=True, errors='coerce')

# Verify the conversion
print(transactions['tran_date'])
```

```
0       2014-02-28
1       2014-02-27
2       2014-02-24
3       2014-02-24
4       2014-02-23
           ...
23048   2011-01-25
23049   2011-01-25
23050   2011-01-25
23051   2011-01-25
23052   2011-01-25
Name: tran_date, Length: 23053, dtype: datetime64[ns]
```

Converts the 'tran_date' column in the transactions dataframe to datetime format, ensuring that dates are interpreted correctly where the day is specified first (European date format).

```
[98] # Current date to calculate recency
     current_date = transactions['tran_date'].max() + pd.DateOffset(1)
```

```
[99] # RFM Calculation
     rfm = transactions.groupby('cust_id').agg({
         'tran_date': lambda x: (current_date - x.max()).days,  # Recency
         'transaction_id': 'count',  # Frequency
         'total_amt': 'sum'  # Monetary
     }).reset_index()
```

```
[100] # Rename the columns
     rfm.columns = ['cust_id', 'Recency', 'Frequency', 'Monetary']
```

Here, rfm dataframe is created by grouping transactions by 'cust_id' and calculating:

- **Recency:** Days since the last transaction (lambda x: (current_date - x.max()).days)
- **Frequency:** Count of transactions per customer ('transaction_id': 'count')
- **Monetary:** Total amount spent by each customer ('total_amt': 'sum')

```
[101] # Print the RFM scores
      print(rfm)
```

```
      cust_id  Recency  Frequency   Monetary
0      266783      374          5   3113.890
1      266784      452          3   5694.065
2      266785      212          8  21613.800
3      266788      382          4   6092.970
4      266794       17         12  27981.915
...       ...      ...        ...        ...
5501   275257      179          5  12574.900
5502   275261      147          5    442.000
5503   275262      731          2   5078.580
5504   275264      875          2   3815.565
5505   275265      332          3   3252.015

[5506 rows x 4 columns]
```

Based on the RFM analysis output provided:

1. **Recency (R):**
   - Recency indicates how recently a customer made a purchase. For example:
     - Customer ID 266794 made a purchase 17 days ago, indicating a very recent transaction.
     - Customer ID 275264, on the other hand, made a purchase 875 days ago, indicating a less recent transaction.
2. **Frequency (F):**
   - Frequency shows how often a customer makes purchases. For example:

- Customer ID 266785 made 8 purchases, indicating a high frequency of transactions.

- Customer ID 275262 made only 2 purchases, indicating a lower frequency.

3. **Monetary (M):**

   - Monetary value represents the total amount spent by each customer. For example:

     - Customer ID 266785 spent a total of 21,613.80 units of currency, indicating high monetary value.

     - Customer ID 275261 spent only 442.00 units of currency, indicating lower monetary value.

```
[102] # Assign RFM scores
      rfm['R_score'] = pd.qcut(rfm['Recency'], 4, labels=[4, 3, 2, 1])
      rfm['F_score'] = pd.qcut(rfm['Frequency'].rank(method='first'), 4, labels=[1, 2, 3, 4])
      rfm['M_score'] = pd.qcut(rfm['Monetary'], 4, labels=[1, 2, 3, 4])
```

```
[102] Start coding or generate with AI.
```

```
# Calculate RFM segment and RFM score
rfm['RFM_Segment'] = rfm['R_score'].astype(str) + rfm['F_score'].astype(str) + rfm['M_score'].astype(str)
rfm['RFM_Score'] = rfm[['R_score', 'F_score', 'M_score']].sum(axis=1)

# Print the RFM segmentation results
print(rfm)
```

```
      cust_id  Recency  Frequency   Monetary R_score F_score M_score  \
0      266783      374          5   3113.890       2       3       1
1      266784      452          3   5694.065       1       1       2
2      266785      212          8  21613.800       3       4       4
3      266788      382          4   6092.970       2       2       2
4      266794       17         12  27981.915       4       4       4
...       ...      ...        ...        ...     ...     ...     ...
5501   275257      179          5  12574.900       3       3       4
5502   275261      147          5    442.000       3       3       1
5503   275262      731          2   5078.580       1       1       2
5504   275264      875          2   3815.565       1       1       1
5505   275265      332          3   3252.015       2       2       1

     RFM_Segment  RFM_Score
0            231          6
1            112          4
2            344         11
3            222          6
4            444         12
...          ...        ...
5501         334         10
5502         331          7
5503         112          4
5504         111          3
5505         221          5

[5506 rows x 9 columns]
```

Interpretation→

Interpreting the Columns:

- **cust_id:** Unique identifier for each customer.
- **Recency:** Number of days since the customer's last purchase. Lower values suggest more recent activity.
- **Frequency:** Total number of transactions made by the customer.
- **Monetary:** Total amount spent by the customer across all transactions.
- **R_score:** Recency score categorized into quartiles (1 to 4). Higher score (e.g., 4) indicates more recent activity.
- **F_score:** Frequency score categorized into quartiles (1 to 4). Higher score (e.g., 4) indicates more frequent purchases.
- **M_score:** Monetary score categorized into quartiles (1 to 4). Higher score (e.g., 4) indicates higher spending.
- **RFM_Segment:** Concatenation of R_score, F_score, and M_score into a segment label (e.g., '231' means R=2, F=3, M=1).
- **RFM_Score:** Sum of R_score, F_score, and M_score. Provides an overall score that can be used for ranking or segmentation purposes.

Example Interpretation:

- Customer with cust_id 266783 has a Recency of 374 days (R_score=2), made 5 purchases (F_score=3), and spent 3113.89 (M_score=1). Their RFM_Segment is '231' and RFM_Score is 6.
- Customer with cust_id 266785 has a Recency of 212 days (R_score=3), made 8 purchases (F_score=4), and spent 21613.80 (M_score=4). Their RFM_Segment is '344' and RFM_Score is 11.

- **R_score:** Assigns quartile labels based on recency (pd.qcut divides into quartiles with labels [4, 3, 2, 1]).

- **F_score:** Ranks frequencies and assigns quartile labels (rank(method='first') ensures no ties, then quartile labels [1, 2, 3, 4]).
- **M_score:** Assigns quartile labels based on monetary value.

Based on the RFM scores derived from the analysis, here are some suggested strategies for different segments:

High RFM Score (e.g., 444)

- **Segment Characteristics:** These customers are recent, frequent buyers who spend the most.
- **Strategies:**
    - **Loyalty Programs:** Offer exclusive rewards, VIP treatment, or membership perks to reinforce their loyalty.
    - **Cross-Selling:** Recommend complementary products or services based on their past purchases to increase average order value.
    - **Feedback Channels:** Seek their opinions and feedback to further personalize their experience and strengthen loyalty.
    - **Early Access:** Provide early access to new products or services to maintain their interest and satisfaction.

High Recency, Lower Frequency and Monetary (e.g., 431)

- **Segment Characteristics:** Recent buyers who spend moderately but infrequently.
- **Strategies:**
    - **Reactivation Campaigns:** Engage with personalized emails offering discounts or incentives to encourage more frequent purchases.
    - **Product Recommendations:** Use personalized recommendations based on their past purchases to increase engagement.

  - **Survey and Feedback:** Understand reasons for low frequency and find ways to address any barriers preventing more frequent purchases.
  - **Limited-Time Offers:** Create urgency with time-limited offers or promotions to prompt immediate action.

Balanced RFM Scores (e.g., 322)

- **Segment Characteristics:** Moderately recent, moderately frequent, and moderate spenders.
- **Strategies:**
  - **Segment-Specific Campaigns:** Develop targeted campaigns based on their behavior, focusing on enhancing their purchase frequency or increasing order values.
  - **Upselling Opportunities:** Identify opportunities to upsell higher-value items based on their purchasing patterns.
  - **Customer Reviews:** Encourage them to leave reviews or testimonials to build social proof and influence potential customers.
  - **Personalized Communication:** Tailor communications with relevant content and offers based on their transactional history and preferences.

Lower RFM Scores (e.g., 111)

- **Segment Characteristics:** Customers who haven't purchased recently, make few purchases, and spend less.
- **Strategies:**
  - **Reactivation Efforts:** Implement win-back campaigns with compelling offers or discounts to reignite their interest.
  - **Segment-Specific Promotions:** Offer special discounts or incentives to incentivize repeat purchases.
  - **Engagement Initiatives:** Engage through educational content, tips, or product usage ideas to showcase value and benefits.

- **Surveys and Feedback:** Gather insights into their challenges or reasons for disengagement to tailor reactivation efforts effectively.

General Strategies for All Segments:

- **Personalization:** Use customer data to personalize interactions, offers, and communications.
- **Customer Service Excellence:** Focus on delivering exceptional customer service to enhance overall satisfaction and loyalty.
- **Feedback Loop:** Continuously gather feedback to understand changing preferences and improve service delivery.
- **Omni-channel Experience:** Ensure a seamless experience across all channels to accommodate customer preferences and behaviors.