Sales Data Analysis And Reporting For A Retail Chain

Project Plan

the aim of this project is to analyze sales data and generate meaningful reports for a retail chain.

Data Info

•TransactionID: A unique identifier for each transaction

•TransactionTime: The time the transaction took place

•ItemCode: The code of the item purchased

•ItemDescription: A description of the item purchased

•NumberOfItemsPurchased: The number of items purchased in the transaction

•CostPerItem: The cost per item

•Country: The country where the transaction took place

Data Preparation

In [5]:
```
import pandas as pd
```

In [6]:
```
trans= pd.read_csv('Retail_Data_transactions.csv')
```

In [7]:
```
trans
```

Out[7]:

	customer_id	trans_date	tran_amount
0	CS5295	11-Feb-13	35
1	CS4768	15-Mar-15	39
2	CS2122	26-Feb-13	52
3	CS1217	16-Nov-11	99
4	CS1850	20-Nov-13	78
...	...	...	...
124995	CS8433	26-Jun-11	64
124996	CS7232	19-Aug-14	38
124997	CS8731	28-Nov-14	42
124998	CS8133	14-Dec-13	13
124999	CS7996	13-Dec-14	36

125000 rows × 3 columns

In [8]:
```
resp= pd.read_csv('Retail_Data_Response.csv')
resp
```

Out[8]:

	customer_id	response
0	CS1112	0
1	CS1113	0
2	CS1114	1
3	CS1115	1
4	CS1116	1
...	...	...
6879	CS8996	0
6880	CS8997	0
6881	CS8998	0
6882	CS8999	0
6883	CS9000	0

6884 rows × 2 columns

Merging the data

In [9]:
```
data=trans.merge(resp, on='customer_id', how='left')
```

In [10]:
```
data
```

Out[10]:

	customer_id	trans_date	tran_amount	response
0	CS5295	11-Feb-13	35	1.0
1	CS4768	15-Mar-15	39	1.0
2	CS2122	26-Feb-13	52	0.0
3	CS1217	16-Nov-11	99	0.0
4	CS1850	20-Nov-13	78	0.0
...	...	...	...	...
124995	CS8433	26-Jun-11	64	0.0
124996	CS7232	19-Aug-14	38	0.0
124997	CS8731	28-Nov-14	42	0.0
124998	CS8133	14-Dec-13	13	0.0
124999	CS7996	13-Dec-14	36	0.0

125000 rows × 4 columns

In [11]:
```
data.dtypes
data.info()
data.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 125000 entries, 0 to 124999
Data columns (total 4 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   customer_id  125000 non-null  object 
 1   trans_date   125000 non-null  object 
 2   tran_amount  125000 non-null  int64  
 3   response     124969 non-null  float64
dtypes: float64(1), int64(1), object(2)
memory usage: 3.8+ MB
```

Out[11]:

	tran_amount	response
count	125000.000000	124969.000000
mean	64.991912	0.110763
std	22.860006	0.313840
min	10.000000	0.000000
25%	47.000000	0.000000
50%	65.000000	0.000000
75%	83.000000	0.000000
max	105.000000	1.000000

In [12]:
```
data.isnull().sum()
```

Out[12]:
```
customer_id     0
trans_date      0
tran_amount     0
response       31
dtype: int64
```

In [13]:
```
data=data.dropna()
```

In [14]:
```
data
```

Out[14]:

	customer_id	trans_date	tran_amount	response
0	CS5295	11-Feb-13	35	1.0
1	CS4768	15-Mar-15	39	1.0
2	CS2122	26-Feb-13	52	0.0
3	CS1217	16-Nov-11	99	0.0
4	CS1850	20-Nov-13	78	0.0
...	...	...	...	...
124995	CS8433	26-Jun-11	64	0.0
124996	CS7232	19-Aug-14	38	0.0
124997	CS8731	28-Nov-14	42	0.0
124998	CS8133	14-Dec-13	13	0.0
124999	CS7996	13-Dec-14	36	0.0

124969 rows × 4 columns

In [15]:
```
data['trans_date']= pd.to_datetime(data['trans_date'])
data['response']= data['response'].astype('int64')
```

```
C:\Users\raiya\AppData\Local\Temp\ipykernel_28128\1432683886.py:1: UserWarning: C
ould not infer format, so each element will be parsed individually, falling back
to `dateutil`. To ensure parsing is consistent and as-expected, please specify a
format.
  data['trans_date']= pd.to_datetime(data['trans_date'])
C:\Users\raiya\AppData\Local\Temp\ipykernel_28128\1432683886.py:1: SettingWithCop
yWarning:
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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['trans_date']= pd.to_datetime(data['trans_date'])
C:\Users\raiya\AppData\Local\Temp\ipykernel_28128\1432683886.py:2: SettingWithCop
yWarning:
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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['response']= data['response'].astype('int64')
```

In [16]:
```
data
```

Out[16]:

	customer_id	trans_date	tran_amount	response
0	CS5295	2013-02-11	35	1
1	CS4768	2015-03-15	39	1
2	CS2122	2013-02-26	52	0
3	CS1217	2011-11-16	99	0
4	CS1850	2013-11-20	78	0
...	...	...	...	...
124995	CS8433	2011-06-26	64	0
124996	CS7232	2014-08-19	38	0
124997	CS8731	2014-11-28	42	0
124998	CS8133	2013-12-14	13	0
124999	CS7996	2014-12-13	36	0

124969 rows × 4 columns

In [17]:
```
data.dtypes
```

Out[17]:
```
customer_id            object
trans_date     datetime64[ns]
tran_amount             int64
response                int64
dtype: object
```

In [18]:
```
from scipy import stats
import numpy as np

#calc z score
z_score = np.abs(stats.zscore(data['tran_amount']))

#set threshold
threshold = 3

outliers= z_score>threshold
print(data[outliers])
```

```
Empty DataFrame
Columns: [customer_id, trans_date, tran_amount, response]
Index: []
```

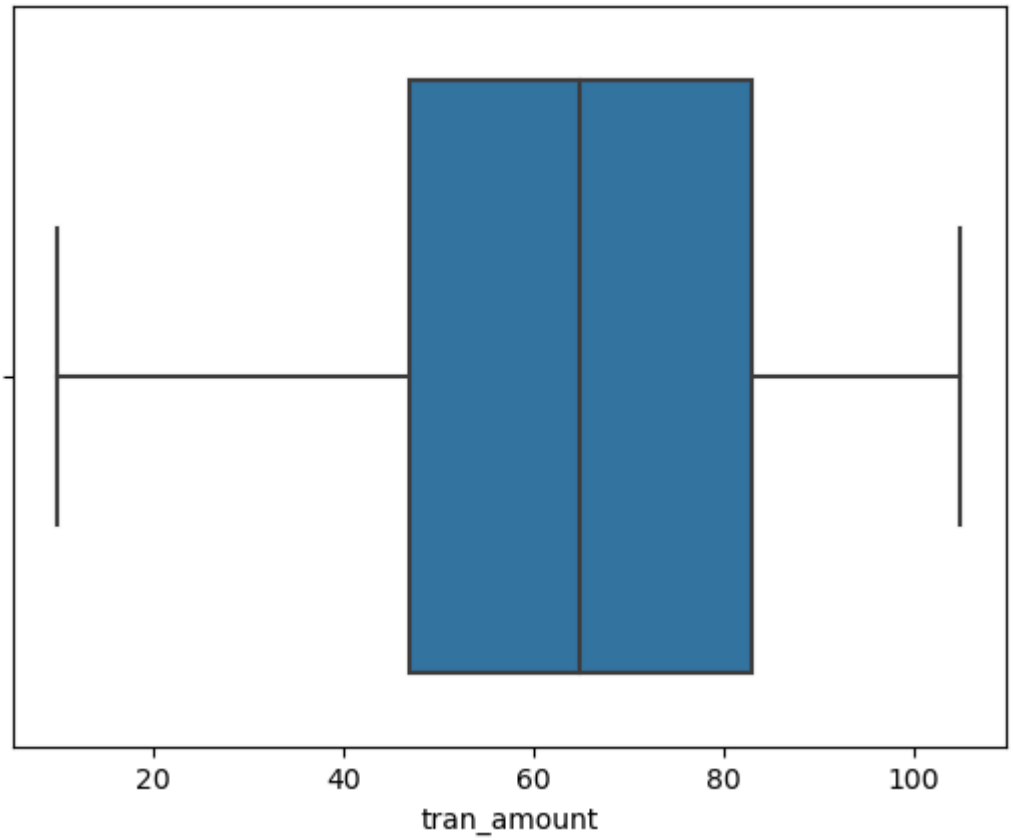
Data Exploration

In [19]:
```
from scipy import stats
import numpy as np

z_score= np.abs(stats.zscore(data['response']))
```

```
threshold= 3
outliers= z_score>threshold
print(data[outliers])
```

```
Empty DataFrame
Columns: [customer_id, trans_date, tran_amount, response]
Index: []
```

In [20]:
```
import seaborn as sns
import matplotlib.pyplot as plt

#making Boxplot
sns.boxplot(x=data['tran_amount'])
plt.show()
```



In [21]:
```
data['month']= data['trans_date'].dt.month
```

```
C:\Users\raiya\AppData\Local\Temp\ipykernel_28128\2336839975.py:1: 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
  data['month']= data['trans_date'].dt.month
```

In [22]:
```
data
```

Out[22]:

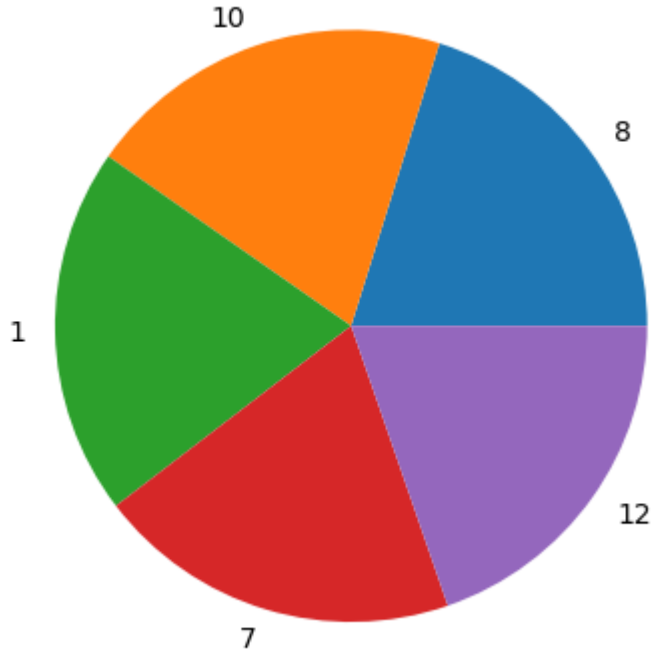
	customer_id	trans_date	tran_amount	response	month
0	CS5295	2013-02-11	35	1	2
1	CS4768	2015-03-15	39	1	3
2	CS2122	2013-02-26	52	0	2
3	CS1217	2011-11-16	99	0	11
4	CS1850	2013-11-20	78	0	11
...	...	...	...	...	...
124995	CS8433	2011-06-26	64	0	6
124996	CS7232	2014-08-19	38	0	8
124997	CS8731	2014-11-28	42	0	11
124998	CS8133	2013-12-14	13	0	12
124999	CS7996	2014-12-13	36	0	12

124969 rows × 5 columns

In [23]:
```
# Top 5 month which had highest transaction amounts
import matplotlib.pyplot as plt

monthly_sales = data.groupby('month')['tran_amount'].sum().reset_index()
monthly_sales = monthly_sales.sort_values('tran_amount', ascending=False).head(5
print(monthly_sales)

# making pie chart
plt.pie('tran_amount',labels='month',data=monthly_sales)
```

```
    month  tran_amount
7       8       726775
9      10       725058
0       1       724089
6       7       717011
11     12       709795
```

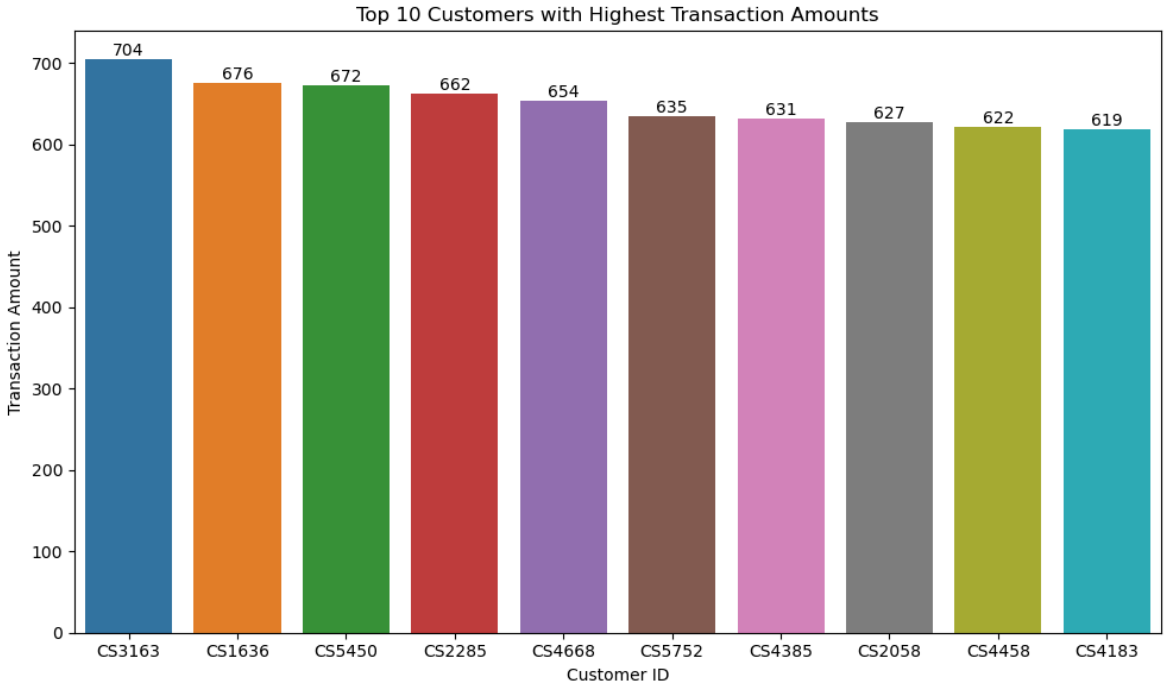
Out[23]:
```
([<matplotlib.patches.Wedge at 0x22ad8930b50>,
  <matplotlib.patches.Wedge at 0x22ad8a3ead0>,
  <matplotlib.patches.Wedge at 0x22ad8a3ff50>,
  <matplotlib.patches.Wedge at 0x22ad9a21690>,
  <matplotlib.patches.Wedge at 0x22ad9a22ad0>],
 [Text(0.8863934171038461, 0.6513882944258111, '8'),
  Text(-0.3553629640096963, 1.0410173696006437, '10'),
  Text(-1.0997380621320874, -0.024004055864823116, '1'),
  Text(-0.31700043024369035, -1.0533331511090473, '7'),
  Text(0.895940750210529, -0.6381928956923562, '12')])
```



In this Piechart , It shows Top 5 month which had highest transaction amounts From this insight it shows , On August the transaction amount was $726,775 which is the highest recorded total transaction amount of that year

In [24]:
```
# customer highest  have highest Transaction Amount in that month

customer_sales = data.groupby(['customer_id', 'month'])['tran_amount'].sum().res
customer_sales = customer_sales.sort_values('tran_amount', ascending=False).rese
print(customer_sales.head(10))

# Top 10 customers( making Bar Graph)
top_10= customer_sales.sort_values(by='tran_amount', ascending=False).head(10)
plt.figure(figsize=(10,6))
ax=sns.barplot(x='customer_id',y='tran_amount',data=top_10)
plt.xlabel('Customer ID')
plt.ylabel('Transaction Amount')
plt.title('Top 10 Customers with Highest Transaction Amounts')
plt.tight_layout()
for i in ax.containers:
    ax.bar_label(i,)
plt.show()
```

	customer_id	month	tran_amount
0	CS3163	11	704
1	CS1636	10	676
2	CS5450	7	672
3	CS2285	9	662
4	CS4668	9	654
5	CS5752	12	635
6	CS4385	5	631
7	CS2058	3	627
8	CS4458	12	622
9	CS4183	2	619



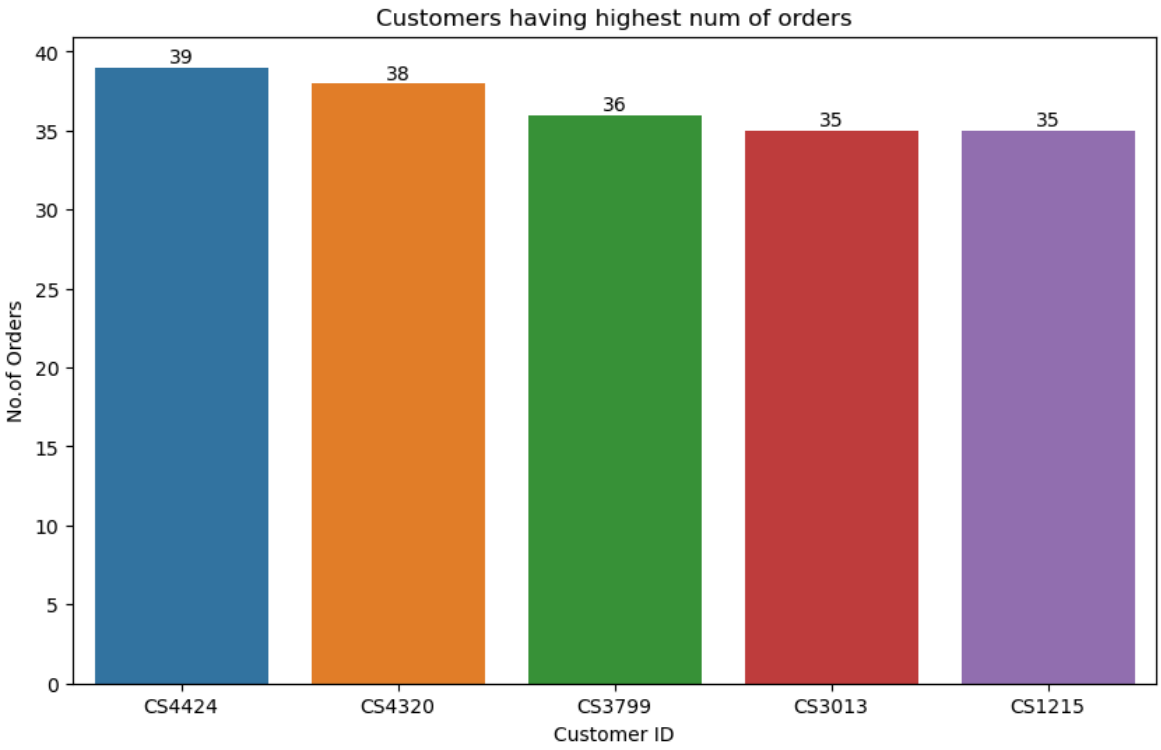
1. Highest Transaction Amount: Customer ID CS3163 has the highest transaction amount, totaling 704, making them the top contributor among these customers.

2. Close Competition: The differences between transaction amounts for these customers are relatively small, indicating a tight range among the top spenders. The second-highest customer (CS1636) has a transaction amount of 676, followed closely by others, down to CS4183 with 619.

3. Potential for Targeted Marketing: These top 10 customers could be valuable for targeted marketing campaigns, loyalty programs, or special offers, as they already demonstrate high spending patterns.

In [25]:
```
#Importing the libraries
import seaborn as sns
import matplotlib.pyplot as plt

# To compare transaction amounts between customers with different responses
plt.figure(figsize=(10, 6))
sns.barplot(x='response', y='tran_amount', data=data)
plt.title('Transaction Amounts by Customer Response')
plt.xlabel('Customer Response (0 = No, 1 = Yes)')
plt.ylabel('Transaction Amount')
plt.show()
```

In [26]:
```
#Importing the libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Customers having highest num of orders

customer_counts= data['customer_id'].value_counts().reset_index()
customer_counts.columns=['customer_id','count']

# sort

top_5= customer_counts.sort_values(by='count', ascending=False).head(5)
top_5
#Creating Bar Chart
plt.figure(figsize=(10,6))
dp=sns.barplot(x='customer_id',y='count',data=top_5)
plt.title('Customers having highest num of orders')
plt.xlabel('Customer ID')
plt.ylabel('No.of Orders')
for i in dp.containers:
    dp.bar_label(i,)
plt.show()
```



1.Customer "CS4424" has the highest number of orders with a count of 39.

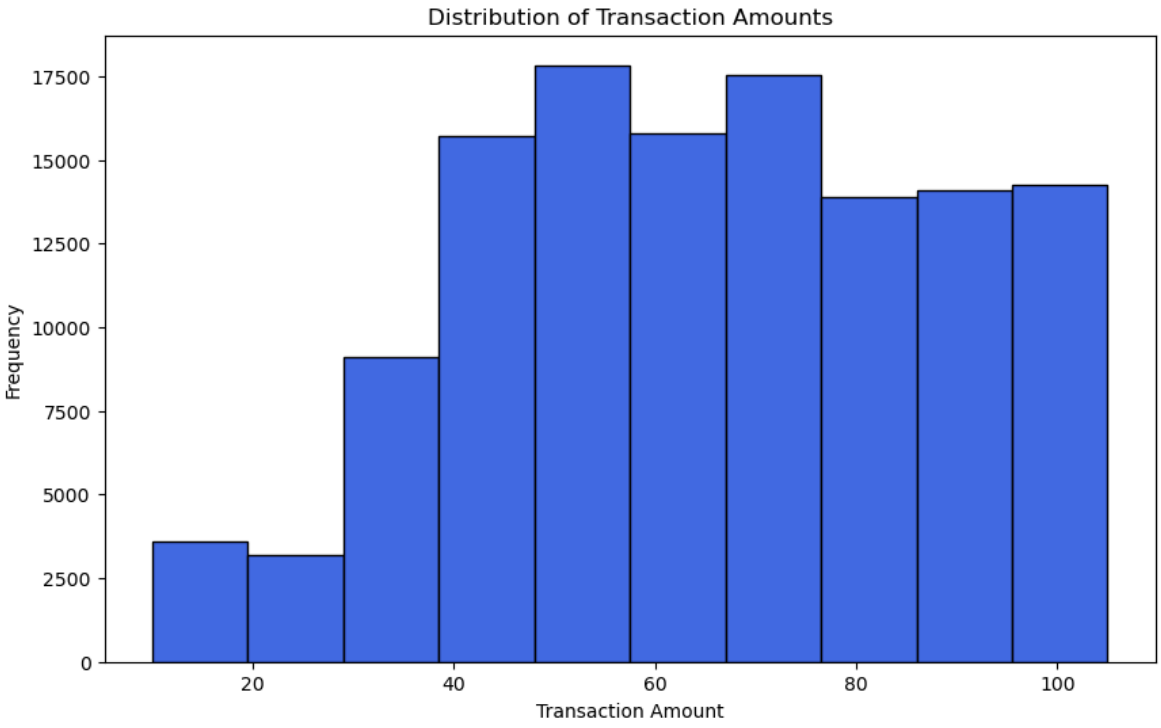
2.Customer "CS4320" follows closely with 38 orders.

3.Customer "CS3799" has the third-highest count with 36 orders.

4.Customers "CS3013" and "CS1215" both have 35 orders, making them the fourth and fifth in the ranking.

This data might help identify the most active and profitable clients for your business.

In [27]:
```
# Plotting a histogram to visualize the distribution of transaction amounts
import matplotlib.pyplot as plt

# Create histogram for 'tran_amount'
plt.figure(figsize=(10,6))
plt.hist(data['tran_amount'],color='royalblue', edgecolor='black')
plt.title('Distribution of Transaction Amounts')
plt.xlabel('Transaction Amount')
plt.ylabel('Frequency')


plt.show()
```

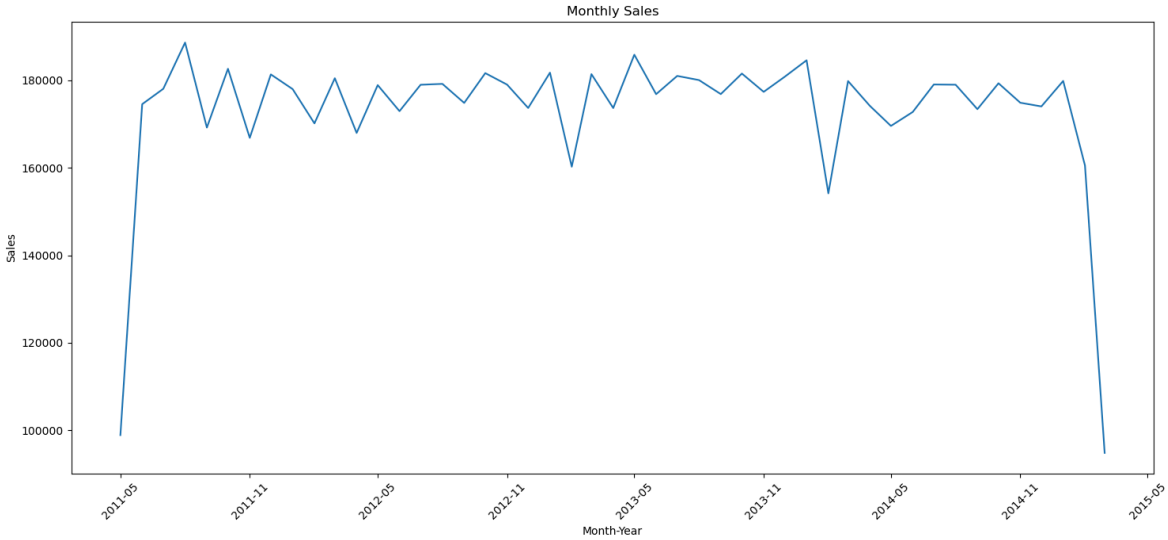


ALGORITHM ANALYSIS

In [28]:
```
# TIME SERIES ANALYSIS
```

In [29]:
```
bimport matplotlib.pyplot as plt
import matplotlib.dates as mdates

data['month_year'] = data['trans_date'].dt.to_period('M')
monthly_sales = data.groupby('month_year')['tran_amount'].sum()

# Convert the PeriodIndex to DateTimeIndex
monthly_sales.index = monthly_sales.index.to_timestamp()

plt.figure(figsize=(15,7))
plt.plot(monthly_sales.index, monthly_sales.values)
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=6))
plt.xlabel('Month-Year')
plt.ylabel('Sales')
plt.title('Monthly Sales')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
C:\Users\raiya\AppData\Local\Temp\ipykernel_28128\291286090.py:4: SettingWithCopy
Warning:
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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['month_year'] = data['trans_date'].dt.to_period('M')
```



1. The highest monthly sales occurred in August 2011, with a total of $188,605.

2. There's a noticeable seasonality in sales, with peaks around the end of the year.

3. Sales show a declining trend from early 2013 to early 2014.

4. Sales started to recover in early 2014 and remained relatively stable afterward.

In [30]:
```
#COHORT SEGMENTATION
```

In [31]:
```
import pandas as pd
import numpy as np

#Convert 'Trans_date' to datetime format
data['trans_date']= pd.to_datetime(data['trans_date'])


# Calculate RFM metrics
recency = data.groupby('customer_id')['trans_date'].max()  # Last transaction da
frequency = data.groupby('customer_id')['trans_date'].count()  # Number of trans
monetary = data.groupby('customer_id')['tran_amount'].sum()  # Total spending pe

#Combine RFM metrics into a DataFrame
rfm = pd.DataFrame({'recency': recency, 'frequency': frequency, 'monetary': mone

#Convert recency to number of days since last purchase
# Assuming today's date is '2024-09-18'
today_date = pd.to_datetime('2024-09-18')
rfm['recency'] = (today_date - rfm['recency']).dt.days

#Create RFM quartiles
r_labels = range(4, 0, -1)  # Higher recency score means more recent purchase
f_labels = range(1, 5)  # Higher frequency score means more transactions
m_labels = range(1, 5)  # Higher monetary score means higher spending

r_quartiles = pd.qcut(rfm['recency'], q=4, labels=r_labels)
f_quartiles = pd.qcut(rfm['frequency'], q=4, labels=f_labels)
m_quartiles = pd.qcut(rfm['monetary'], q=4, labels=m_labels)

rfm['R'] = r_quartiles
```

```
rfm['F'] = f_quartiles
rfm['M'] = m_quartiles

#Calculate RFM Score
rfm['RFM_Score'] = rfm['R'].astype(str) + rfm['F'].astype(str) + rfm['M'].astype

#Define segmentation function
def segment_customers(row):
    if row['RFM_Score'] in ['444', '434', '443', '433']:
        return 'Best Customers'
    elif row['R'] == 4:
        return 'Lost Customers'
    elif row['F'] == 4 and row['M'] == 4:
        return 'Loyal Customers'
    elif row['R'] >= 3 and row['F'] >= 3 and row['M'] >= 3:
        return 'Promising Customers'
    elif row['R'] >= 3 and row['F'] <= 2 and row['M'] <= 2:
        return 'Recent Customers'
    elif row['R'] <= 2 and row['F'] <= 2 and row['M'] <= 2:
        return 'Customers Needing Attention'
    else:
        return 'Other'

#Apply segmentation
rfm['Customer_Segment'] = rfm.apply(segment_customers, axis=1)

#Analyze segments
segment_counts = rfm['Customer_Segment'].value_counts()
segment_percentages = segment_counts / len(rfm) * 100

#Calculate average RFM values for each segment
segment_avg_values = rfm.groupby('Customer_Segment').agg({
    'recency': 'mean',
    'frequency': 'mean',
    'monetary': 'mean'
}).round(2)

#Display results
print("Customer Segment Counts:")
print(segment_counts)
print("\nCustomer Segment Percentages:")
print(segment_percentages)
print("\nAverage RFM Values per Segment:")
print(segment_avg_values)
```

```
C:\Users\raiya\AppData\Local\Temp\ipykernel_28128\1003957298.py:5: SettingWithCop
yWarning:
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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['trans_date']= pd.to_datetime(data['trans_date'])
```

```
Customer Segment Counts:
Customer_Segment
Customers Needing Attention    1864
Other                          1260
Best Customers                  932
Loyal Customers                 874
Lost Customers                  849
Recent Customers                663
Promising Customers             442
Name: count, dtype: int64

Customer Segment Percentages:
Customer_Segment
Customers Needing Attention    27.077281
Other                          18.303312
Best Customers                 13.538640
Loyal Customers                12.696107
Lost Customers                 12.332946
Recent Customers                9.631028
Promising Customers             6.420686
Name: count, dtype: float64

Average RFM Values per Segment:
                             recency  frequency  monetary
Customer_Segment
Best Customers               3484.05      23.11   1625.38
Customers Needing Attention  3629.38      13.48    758.16
Lost Customers               3484.83      15.07    890.43
Loyal Customers              3548.60      25.53   1801.73
Other                        3582.27      19.69   1344.95
Promising Customers          3510.49      20.63   1450.32
Recent Customers             3511.63      13.99    796.30
```

In [32]:

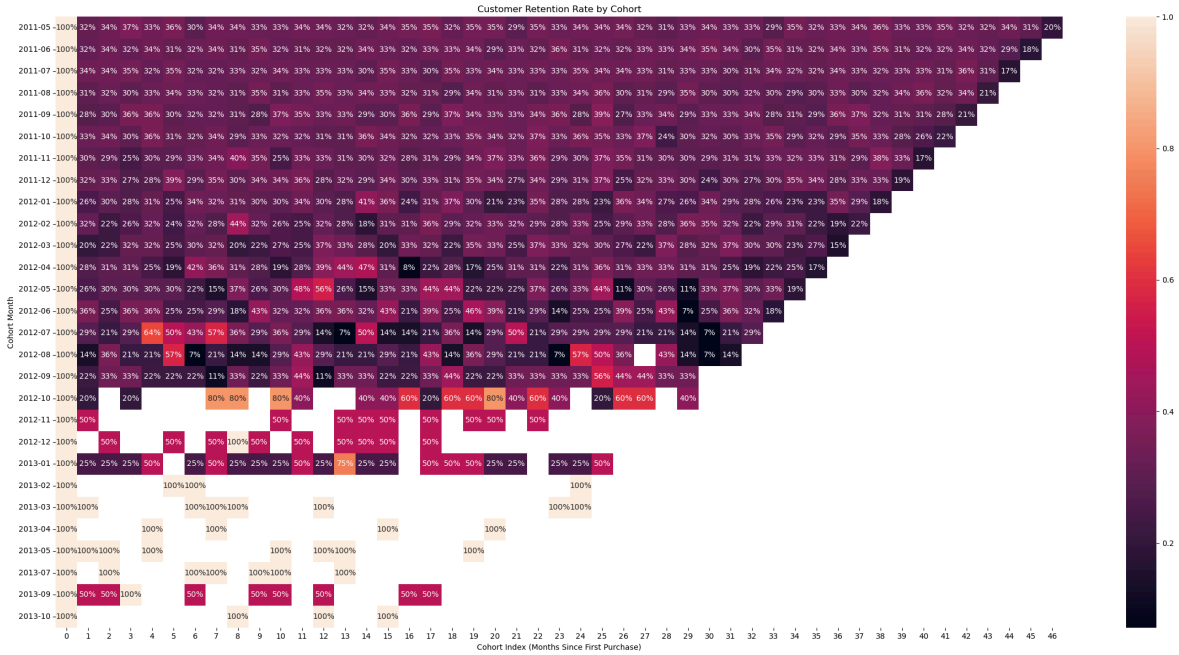
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# RFM Analysis
recency = data.groupby('customer_id')['trans_date'].max()
frequency = data.groupby('customer_id')['trans_date'].count()
monetary = data.groupby('customer_id')['tran_amount'].sum()

rfm = pd.DataFrame({'recency': recency, 'frequency': frequency, 'monetary': mone

# Customer Segmentation
def segment_customer(row):
    if row['recency'].year >= 2012 and row['frequency'] >= 15 and row['monetary'
        return 'P0'
    elif (2011 <= row['recency'].year < 2012) and (10 < row['frequency'] <= 15)
        return 'P1'
    else:
        return 'P2'

rfm['segment'] = rfm.apply(segment_customer, axis=1)
print(rfm)

# Cohort Analysis
# Note: You need to create cohort_pivot before this step.
# Here's a basic example of how you might create it:
```

```
# Convert trans_date to datetime if it's not already
data['trans_date'] = pd.to_datetime(data['trans_date'])

# Create cohort month
data['cohort_month'] = data.groupby('customer_id')['trans_date'].transform('min'

# Create cohort index
data['cohort_index'] = (data['trans_date'].dt.to_period('M') -
                        data.groupby('customer_id')['trans_date'].transform('min')

# Count unique customers for each cohort and month
cohort_data = data.groupby(['cohort_month', 'cohort_index'])['customer_id'].nuni

# Create the pivot table
cohort_pivot = cohort_data.pivot(index='cohort_month', columns='cohort_index', v

# Calculate retention rate
cohort_size = cohort_pivot.iloc[:, 0]
cohort_pivot = cohort_pivot.divide(cohort_size, axis=0)

# Visualization
plt.figure(figsize=(24,12))
sns.heatmap(cohort_pivot, annot=True, fmt=".0%")
plt.title('Customer Retention Rate by Cohort')
plt.xlabel('Cohort Index (Months Since First Purchase)')
plt.ylabel('Cohort Month')
plt.tight_layout()
plt.show()
```

customer_id	recency	frequency	monetary	segment
CS1112	2015-01-14	15	1012	P0
CS1113	2015-02-09	20	1490	P0
CS1114	2015-02-12	19	1432	P0
CS1115	2015-03-05	22	1659	P0
CS1116	2014-08-25	13	857	P2
...	...	...	...	...
CS8996	2014-12-09	13	582	P2
CS8997	2014-06-28	14	543	P2
CS8998	2014-12-22	13	624	P2
CS8999	2014-07-02	12	383	P2
CS9000	2015-02-28	13	533	P2

[6884 rows x 4 columns]

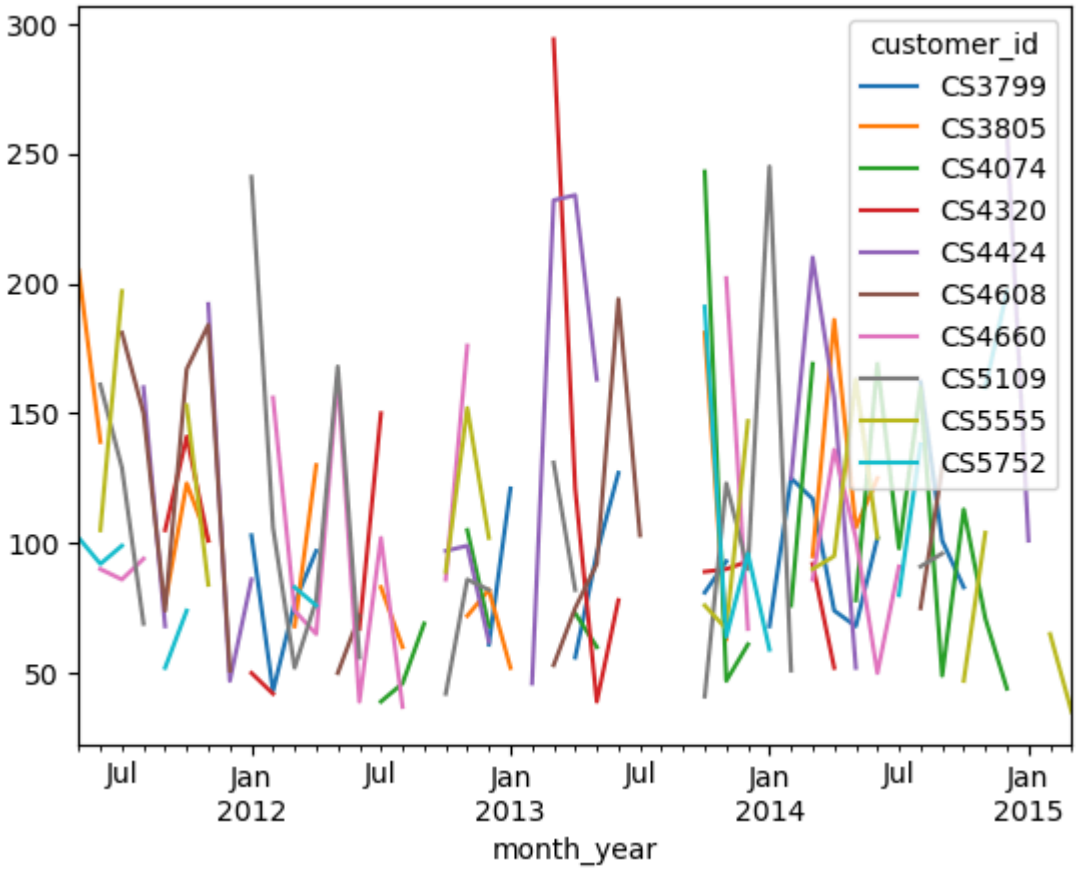
```
C:\Users\raiya\AppData\Local\Temp\ipykernel_28128\804826618.py:29: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['trans_date'] = pd.to_datetime(data['trans_date'])
C:\Users\raiya\AppData\Local\Temp\ipykernel_28128\804826618.py:32: 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
  data['cohort_month'] = data.groupby('customer_id')['trans_date'].transform('min').dt.to_period('M')
C:\Users\raiya\AppData\Local\Temp\ipykernel_28128\804826618.py:35: 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
  data['cohort_index'] = (data['trans_date'].dt.to_period('M') -
```



In [33]:
```
#ANALYZING TOP CUSTOMERS
```

In [34]:
```
#Top 10 Customers
top_10_customers = monetary.sort_values(ascending=False).head(10).index

# Filter The Transactions of The Top 10 Customers
top_customers_data = data[data['customer_id'].isin(top_10_customers)]

#Plotting the Plot
top_customers_sales = top_customers_data.groupby(['customer_id', 'month_year'])[
top_customers_sales.plot(kind='line')
```

Out[34]: <Axes: xlabel='month_year'>



In [36]:
```
data.to_csv('maindata.csv')
```

In [37]:
```
rfm.to_csv('AddAnalysis.csv')
```