Customer Data Analysis Project

Customer Data Analysis for Business Insights

Business Problem Statement

A mid-sized Indian retail company wants to analyze its customer base to improve targeted marketing, retain valuable customers, and detect patterns in customer behavior.

You will work with multiple interlinked datasets — such as customers master data, transactions data, detect anomalies, segment customers, and understand risk exposure.

The emphasis is on applying **advanced DataFrame operations** like multi-key joins, group-based aggregations, resampling, reshaping, and conditional logic — all without using SQL or external libraries.

Project Objectives

This project is designed to strengthen your skills in working with complex, real-world DataFrame operations using Pandas. The main objectives are:

Your goal is to:

- Assess customer demographics and transaction behavior
- Clean and prepare the data for analysis
- Conduct RFM (Recency, Frequency, Monetary) analysis
- Visualize key patterns using charts

Key goal:

- Detect incomplete, inconsistent, or duplicate customer records and improve data quality, if applicable
- Segment customers based on demographics and purchase behavior
- Identify high-value customers using **Recency-Frequency-Monetary (RFM)** analysis
- Visualize customer distribution across geographies, age groups, or income brackets
- Generate actionable insights for the marketing department

Dataset Description

You are provided with a **synthetic dataset** of **23,050 retail transactions** made by **1,000 unique customers**.

There are **two key datasets**:

1. Customer Master Data

Column Name	Description
CustomerID	Unique identifier for each customer
Name	Customer's full name
Email	Customer's email ID
Gender	Male, Female, or Not Disclosed
Age	Customer's age (between 18 and 75)
City	City where the customer resides (Indian metro / tier-2 cities)
MaritalStatus	Single, Married, Divorced, Widowed
NumChildren	Number of children in the household
JoinDate	Date the customer first registered with the company

2. Transaction Data

Column Name	Description
CustomerID	Links back to the customer master dataset
TransactionDate	Date on which the transaction occurred
TransactionAmount	Amount spent by the customer in that transaction (₹)

RFM Analysis Instructions (Conceptual Explanation)

RFM (Recency, Frequency, Monetary) is a customer segmentation method used to identify high-value customers based on their purchasing behavior.

Metric	Description
Recency	How recently a customer made a purchase
Frequency	How often a customer makes purchases
Monetary	Total amount of money spent by the customer

Step 1: Load the Data

We will begin by loading the two core datasets:

- **Customers master data**: Contains demographic and registration details of each customer
- **Transactions data**: Contains transaction-level data with transaction amount

- Load both CSV files into Pandas DataFrames
- Check the shape and structure of the datasets
- Preview the data to understand column names and values

In [1]:
```
# Step 1: Import required libraries
import pandas as pd
```

```python
import numpy as np

# Step 2: Define file paths (assuming the datasets are in the 'data/' folder)
Customers_data=pd.read_csv(r"C:\Users\INDIA\Downloads\4453477-Dataset (1)\Dataset\Customer_Master_Data.csv")
Transactions_data=pd.read_csv(r"C:\Users\INDIA\Downloads\Dataset\Customer_Transactions.csv")
# Step 3: Preview the first few records of each dataset
print("\n Customers:")
display(Customers_data.head())

print("\n Transactions:")
display(Transactions_data.head())

# -------------------------------------------------------
# Display the dimensions (rows, columns) of each dataset
# -------------------------------------------------------
print("Dataset Dimensions")
print("Customers:    ", Customers_data.shape)
print("Transactions: ", Transactions_data.shape)

# -------------------------------------------------------
# Show column-level information including data types
# and count of non-null values
# -------------------------------------------------------
print("\nCustomer Dataset Info")
Customers_data.info()

print("\nTransaction Dataset Info")
Transactions_data.info()

# -------------------------------------------------------
```

```
Customers:
```

	CustomerID	Name	Email	Gender	Age	City	MaritalStatus	NumChildren	JoinDate
0	CUST10000	Onkar Bhargava	pkeer@yahoo.com	Male	54	Delhi	Divorced	0	2021-02-22
1	CUST10001	Divit Kohli	mkalita@sarin.com	Female	48	Kolkata	Married	0	2023-12-06
2	CUST10002	Kiara Behl	apteanay@hotmail.com	Male	75	Kolkata	Widowed	2	2023-08-23
3	CUST10003	Vaibhav Sankar	bseshadri@choudhry.info	Male	62	Pune	Divorced	2	2022-11-17
4	CUST10004	Shray D’Alia	bdhillon@toor-mall.com	Male	55	Delhi	Divorced	0	2022-12-04

```
Transactions:
```

	CustomerID	TransactionDate	TransactionAmount
0	CUST10771	7/31/23	2383.07
1	CUST10100	3/10/24	497.54
2	CUST10031	2/17/25	536.78
3	CUST10987	7/17/23	314.89
4	CUST10831	12/15/24	2543.19

```
Dataset Dimensions
Customers:     (1000, 9)
Transactions:  (23050, 3)

Customer Dataset Info
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   CustomerID     1000 non-null   object
 1   Name           1000 non-null   object
 2   Email          1000 non-null   object
 3   Gender         1000 non-null   object
 4   Age            1000 non-null   int64 
 5   City           1000 non-null   object
 6   MaritalStatus  1000 non-null   object
 7   NumChildren    1000 non-null   int64 
 8   JoinDate       1000 non-null   object
dtypes: int64(2), object(7)
memory usage: 70.4+ KB

Transaction Dataset Info
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23050 entries, 0 to 23049
Data columns (total 3 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   CustomerID         23050 non-null  object 
 1   TransactionDate    23050 non-null  object 
 2   TransactionAmount  23050 non-null  float64
dtypes: float64(1), object(2)
memory usage: 540.4+ KB
```

Step 2: Clean the Data

- Convert **JoinDate** and **TransactionDate** columns to datetime format
- Ensure there are no null values or incorrect data types
- Validate the uniqueness of **CustomerID** in the customer master dataset

- Ensure all transaction **CustomerID** values exist in the master data

In [2]:
```
# Show total missing values in each column of each dataset
# -------------------------------------------------------
print("\nMissing Values in Customer Dataset")
print(Customers_data.isna().sum())
display(Customers_data)
print("\nMissing Values in Transaction Dataset")
print(Transactions_data.isna().sum())
display(Transactions_data)
# -------------------------------------------------------
# Convert date columns to datetime
Customers_data['JoinDate'] = pd.to_datetime(Customers_data['JoinDate'])
Transactions_data['TransactionDate'] = pd.to_datetime(Transactions_data['TransactionDate'])

# Check for null values
print("Null values in Customers Data:")
print(Customers_data.isnull().sum())

print("\nNull values in Transactions Data:")
print(Transactions_data.isnull().sum())

# Validate uniqueness of CustomerID in customer master data
unique_customers = Customers_data['CustomerID'].is_unique
print("\nIs CustomerID unique in customer master data?", unique_customers)

# Identify duplicate CustomerIDs if any
duplicate_customers = Customers_data[Customers_data.duplicated('CustomerID')]
print("\nDuplicate CustomerIDs (if any):")
print(duplicate_customers)

# Ensure all transaction CustomerIDs exist in customer master data
invalid_transactions = Transactions_data[Transactions_data['CustomerID'].isin(Customers_data['CustomerID'])]

print("\nTransactions with CustomerIDs not present in master data:")
print(invalid_transactions)
```

```
Missing Values in Customer Dataset
CustomerID       0
Name             0
Email            0
Gender           0
Age              0
City             0
MaritalStatus    0
NumChildren      0
JoinDate         0
dtype: int64
```

	CustomerID	Name	Email	Gender	Age	City	MaritalStatus	NumChildren	JoinDate
0	CUST10000	Onkar Bhargava	pkeer@yahoo.com	Male	54	Delhi	Divorced	0	2021-02-22
1	CUST10001	Divit Kohli	mkalita@sarin.com	Female	48	Kolkata	Married	0	2023-12-06
2	CUST10002	Kiara Behl	apteanay@hotmail.com	Male	75	Kolkata	Widowed	2	2023-08-23
3	CUST10003	Vaibhav Sankar	bseshadri@choudhry.info	Male	62	Pune	Divorced	2	2022-11-17
4	CUST10004	Shray D'Alia	bdhillon@toor-mall.com	Male	55	Delhi	Divorced	0	2022-12-04
...	...	...	...	...	...	...	...	...	...
995	CUST10995	Mehul Chada	hridaanagate@hotmail.com	Male	70	Hyderabad	Divorced	2	2020-07-29
996	CUST10996	Arhaan Tara	qwali@mand-sood.com	Male	35	Delhi	Single	1	2022-07-24
997	CUST10997	Mahika Uppal	vdalal@yahoo.com	Female	70	Ahmedabad	Married	3	2023-01-27
998	CUST10998	Bhamini Aggarwal	kartik15@bajaj-singhal.com	Male	37	Jaipur	Single	0	2022-07-22
999	CUST10999	Alia Sekhon	urvichadha@hotmail.com	Male	67	Hyderabad	Widowed	1	2021-09-07

1000 rows × 9 columns

```
Missing Values in Transaction Dataset
CustomerID           0
TransactionDate      0
TransactionAmount    0
dtype: int64
```

	CustomerID	TransactionDate	TransactionAmount
0	CUST10771	7/31/23	2383.07
1	CUST10100	3/10/24	497.54
2	CUST10031	2/17/25	536.78
3	CUST10987	7/17/23	314.89
4	CUST10831	12/15/24	2543.19
...	...	...	...
23045	CUST10710	3/11/24	931.09
23046	CUST10209	6/19/24	2659.35
23047	CUST10570	6/27/24	266.97
23048	CUST10075	12/26/23	1671.73
23049	CUST10234	7/7/24	981.32

23050 rows × 3 columns

```
C:\Users\INDIA\AppData\Local\Temp\ipykernel_21620\423367693.py:12: UserWarning: Could 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.
  Transactions_data['TransactionDate'] = pd.to_datetime(Transactions_data['TransactionDate'])
```

```
Null values in Customers Data:
CustomerID       0
Name             0
Email            0
Gender           0
Age              0
City             0
MaritalStatus    0
NumChildren      0
JoinDate         0
dtype: int64

Null values in Transactions Data:
CustomerID           0
TransactionDate      0
TransactionAmount    0
dtype: int64

Is CustomerID unique in customer master data? True

Duplicate CustomerIDs (if any):
Empty DataFrame
Columns: [CustomerID, Name, Email, Gender, Age, City, MaritalStatus, NumChildren, JoinDate]
Index: []

Transactions with CustomerIDs not present in master data:
      CustomerID TransactionDate  TransactionAmount
0      CUST10771      2023-07-31            2383.07
1      CUST10100      2024-03-10             497.54
2      CUST10031      2025-02-17             536.78
3      CUST10987      2023-07-17             314.89
4      CUST10831      2024-12-15            2543.19
...          ...             ...                ...
23045  CUST10710      2024-03-11             931.09
23046  CUST10209      2024-06-19            2659.35
23047  CUST10570      2024-06-27             266.97
23048  CUST10075      2023-12-26            1671.73
23049  CUST10234      2024-07-07             981.32

[23050 rows x 3 columns]
```

Step 3: Merging the Datasets for Unified Analysis

Our two datasets are interconnected via primary and foreign keys:

- `customer_id` links each transaction amount to its corresponding customer id

We will now:

- Join **transactions** with the enriched custmoers dataset to create a final dataframe: `full_df`

This merged dataframe will allow us to analyze customer and transaction behavior in a single unified view.

In [3]:
```
# joining the dataset: Customers and Tra
full_df = pd.merge(Customers_data,Transactions_data,on = "CustomerID",how = "inner")
display(full_df)

# data integrity
print("\n full_df information:")
print(full_df.info())
```

	CustomerID	Name	Email	Gender	Age	City	MaritalStatus	NumChildren	JoinDate	TransactionDate	Tr
0	CUST10000	Onkar Bhargava	pkeer@yahoo.com	Male	54	Delhi	Divorced	0	2021-02-22	2022-10-03	
1	CUST10000	Onkar Bhargava	pkeer@yahoo.com	Male	54	Delhi	Divorced	0	2021-02-22	2024-05-31	
2	CUST10000	Onkar Bhargava	pkeer@yahoo.com	Male	54	Delhi	Divorced	0	2021-02-22	2024-05-31	
3	CUST10000	Onkar Bhargava	pkeer@yahoo.com	Male	54	Delhi	Divorced	0	2021-02-22	2023-01-31	
4	CUST10000	Onkar Bhargava	pkeer@yahoo.com	Male	54	Delhi	Divorced	0	2021-02-22	2022-06-12	
...	...	...	...	...	...	...	...	...	...	...	
23045	CUST10999	Alia Sekhon	urvichadha@hotmail.com	Male	67	Hyderabad	Widowed	1	2021-09-07	2025-06-25	
23046	CUST10999	Alia Sekhon	urvichadha@hotmail.com	Male	67	Hyderabad	Widowed	1	2021-09-07	2025-02-12	
23047	CUST10999	Alia Sekhon	urvichadha@hotmail.com	Male	67	Hyderabad	Widowed	1	2021-09-07	2024-10-09	
23048	CUST10999	Alia Sekhon	urvichadha@hotmail.com	Male	67	Hyderabad	Widowed	1	2021-09-07	2023-03-15	
23049	CUST10999	Alia Sekhon	urvichadha@hotmail.com	Male	67	Hyderabad	Widowed	1	2021-09-07	2024-09-13	

23050 rows × 11 columns

```
 full_df information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23050 entries, 0 to 23049
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   CustomerID         23050 non-null  object
 1   Name               23050 non-null  object
 2   Email              23050 non-null  object
 3   Gender             23050 non-null  object
 4   Age                23050 non-null  int64
 5   City               23050 non-null  object
 6   MaritalStatus      23050 non-null  object
 7   NumChildren        23050 non-null  int64
 8   JoinDate           23050 non-null  datetime64[ns]
 9   TransactionDate    23050 non-null  datetime64[ns]
 10  TransactionAmount  23050 non-null  float64
dtypes: datetime64[ns](2), float64(1), int64(2), object(6)
memory usage: 1.9+ MB
None
```

Step 4: Perform RFM Calculation

RFM analysis is used to evaluate customer behavior based on **Recency**, **Frequency**, and **Monetary** values.

- Use **groupby** on **CustomerID**
- **max(TransactionDate)** → Recency
- **count(TransactionDate)** → Frequency
- **sum(TransactionAmount)** → Monetary
- Use a **reference date** to compute **Recency** in number of days
- Store the final result in a new DataFrame called **df_rfm**

In [4]:
```
# Sort the full dataset by CustomerID and TransactionDate
# Ensures transactions are in chronological order per customer
df_rfm = (
    full_df
    .sort_values(by=['CustomerID', 'TransactionDate'])
```

```python
    # Group transactions by each customer
    .groupby('CustomerID')

    # Aggregate RFM-related metrics
    .agg(
        # Most recent transaction date per customer
        LastTransactionDate=('TransactionDate', 'max'),

       # Second most recent transaction date per customer
       # nlargest(2) gets the two latest dates, iloc[-1] selects the older of the two
        SecondLastTransactionDate=('TransactionDate', lambda x: x.nlargest(2).iloc[-1]),

        # Frequency: total number of transactions per customer
        Frequency=('TransactionDate', 'count'),

        # Monetary: total amount spent by the customer
        Monetary=('TransactionAmount', 'sum')
    )
)

# Calculate Recency in days
# Difference between the last and second last transaction dates
df_rfm['RecencyDays'] = (
    df_rfm['LastTransactionDate'] - df_rfm['SecondLastTransactionDate']
).dt.days

# Display the resulting RFM dataframe
display(df_rfm)
```

	LastTransactionDate	SecondLastTransactionDate	Frequency	Monetary	RecencyDays
CustomerID					
CUST10000	2025-07-17	2025-06-28	23	21265.49	19
CUST10001	2025-06-25	2025-06-10	30	28654.31	15
CUST10002	2025-07-12	2025-06-22	24	23884.03	20
CUST10003	2025-05-10	2025-04-22	25	24206.03	18
CUST10004	2025-07-22	2025-07-20	19	25565.30	2
...	...	...	...	...	...
CUST10995	2024-06-23	2024-06-21	21	24325.19	2
CUST10996	2025-07-15	2025-07-07	21	21809.11	8
CUST10997	2025-06-28	2025-06-15	20	21120.48	13
CUST10998	2025-03-26	2025-02-02	25	29494.56	52
CUST10999	2025-07-25	2025-06-25	23	22028.01	30

1000 rows × 5 columns

Step 5: Score RFM

RFM scoring assigns numeric scores to customers based on **Recency**, **Frequency**, and **Monetary** values using quantile-based methods.

- Use **quantile-based scoring** using `pd.qcut()` or `rank()` with `cut()`
- Create three new score columns: **R_Score, F_Score, M_Score**

In [5]:
```
# calculating Recency Freqauncy and Mnonetry
# Using quntiles
df_rfm["R_Score"] = pd.qcut(df_rfm["RecencyDays"],q=5,labels=[5, 4, 3, 2, 1])
#___________________________________________________________________________________________________
df_rfm["F_Score"]=pd.qcut(df_rfm["Frequency"],q=5,labels=[1,2,3,4,5])
```

```
#____________________________________________________________________________
df_rfm["M_Score"]=pd.qcut(df_rfm["Monetary"],q=5,labels=[1,2,3,4,5])
#_____________________________________________________________________________
display(df_rfm)
```

CustomerID	LastTransactionDate	SecondLastTransactionDate	Frequency	Monetary	RecencyDays	R_Score	F_Score	M_Score
CUST10000	2025-07-17	2025-06-28	23	21265.49	19	4	3	2
CUST10001	2025-06-25	2025-06-10	30	28654.31	15	4	5	5
CUST10002	2025-07-12	2025-06-22	24	23884.03	20	4	3	3
CUST10003	2025-05-10	2025-04-22	25	24206.03	18	4	4	3
CUST10004	2025-07-22	2025-07-20	19	25565.30	2	5	1	4
...	...	...	...	...	...	...	...	...
CUST10995	2024-06-23	2024-06-21	21	24325.19	2	5	2	3
CUST10996	2025-07-15	2025-07-07	21	21809.11	8	5	2	3
CUST10997	2025-06-28	2025-06-15	20	21120.48	13	4	2	2
CUST10998	2025-03-26	2025-02-02	25	29494.56	52	2	4	5
CUST10999	2025-07-25	2025-06-25	23	22028.01	30	3	3	3

1000 rows × 8 columns

Step 6: Creating Combined RFM Segment

In this step, individual RFM scores are combined to form a single RFM segment code for each customer.

- Concatenate **R_Score**, **F_Score**, and **M_Score**
- Create a string-based segment such as **"555"**, **"432"**, etc.

The new column **RFM_Segment** represents each customer's overall value pattern and is used for customer segmentation and marketing strategy decisions.

In [6]:
```
#nCreating a new column 'RFM_Segment' by combining R, F, and M scores
# Converting each score to a string so they can be concatenated
df_rfm["RFM_Segment"] = (
    df_rfm["R_Score"].astype(str) +  # Recency score as string
    df_rfm["F_Score"].astype(str) +  # Frequency score as string
    df_rfm["M_Score"].astype(str)    # Monetary score as string
)

# Display the dataframe with the new RFM segment column
display(df_rfm)
```

CustomerID	LastTransactionDate	SecondLastTransactionDate	Frequency	Monetary	RecencyDays	R_Score	F_Score	M_Score	RFM_Segmen
CUST10000	2025-07-17	2025-06-28	23	21265.49	19	4	3	2	432
CUST10001	2025-06-25	2025-06-10	30	28654.31	15	4	5	5	455
CUST10002	2025-07-12	2025-06-22	24	23884.03	20	4	3	3	433
CUST10003	2025-05-10	2025-04-22	25	24206.03	18	4	4	3	443
CUST10004	2025-07-22	2025-07-20	19	25565.30	2	5	1	4	514
...	...	...	...	...	...	...	...	...	...
CUST10995	2024-06-23	2024-06-21	21	24325.19	2	5	2	3	523
CUST10996	2025-07-15	2025-07-07	21	21809.11	8	5	2	3	523
CUST10997	2025-06-28	2025-06-15	20	21120.48	13	4	2	2	422
CUST10998	2025-03-26	2025-02-02	25	29494.56	52	2	4	5	245
CUST10999	2025-07-25	2025-06-25	23	22028.01	30	3	3	3	333

1000 rows × 9 columns

Step 7: Assign Segment Labels

In this step, business rules are applied to translate combined **RFM segments** into meaningful customer labels.

- Use predefined **business rules** to assign segment labels
- Map selected **RFM score combinations** to customer types
- Refer to the RFM explanation above for score interpretation

The **Customer_Segment** column classifies customers into actionable business groups such as Champions, Loyal Customers, Potential Loyalists, and Lost customers.

In [7]:
```
def assign_segment(row):
    r = int(row['R_Score'])
    f = int(row['F_Score'])
    m = int(row['M_Score'])

    # Champions: Best customers
    if r >= 4 and f >= 4 and m >= 4:
        return "Champions"

    # Loyal Customers: Repeat buyers with strong frequency
    elif f >= 4 and r >= 3:
        return "Loyal Customers"

    # Potential Loyalist: Recently active, building loyalty
    elif r >= 4 and f >= 2:
        return "Potential Loyalist"

    # Big Spenders: Spend more money
    elif m >= 4 and f >= 2:
        return "Big Spenders"

    # At Risk: good history but poor recent activity
    elif r <= 2 and f >= 3:
        return "At Risk"

    # Lost: No longer buying, low value
    elif r == 1 and f <= 2 and m <= 2:
        return "Lost"

    # Others: Remaining customers not fitting rules above
    else:
        return "Regular Customer"

df_rfm["Customer_Segment"] = df_rfm.apply(assign_segment, axis=1)
display(df_rfm)
```

	LastTransactionDate	SecondLastTransactionDate	Frequency	Monetary	RecencyDays	R_Score	F_Score	M_Score	RFM_Segmen
CustomerID									
CUST10000	2025-07-17	2025-06-28	23	21265.49	19	4	3	2	432
CUST10001	2025-06-25	2025-06-10	30	28654.31	15	4	5	5	455
CUST10002	2025-07-12	2025-06-22	24	23884.03	20	4	3	3	433
CUST10003	2025-05-10	2025-04-22	25	24206.03	18	4	4	3	443
CUST10004	2025-07-22	2025-07-20	19	25565.30	2	5	1	4	514
...	...	...	...	...	...	...	...	...	..
CUST10995	2024-06-23	2024-06-21	21	24325.19	2	5	2	3	523
CUST10996	2025-07-15	2025-07-07	21	21809.11	8	5	2	3	523
CUST10997	2025-06-28	2025-06-15	20	21120.48	13	4	2	2	422
CUST10998	2025-03-26	2025-02-02	25	29494.56	52	2	4	5	245
CUST10999	2025-07-25	2025-06-25	23	22028.01	30	3	3	3	333

1000 rows × 10 columns

Steps 8: Visualization

Visualizations help interpret RFM analysis results by showing customer distribution across different segments and scores.

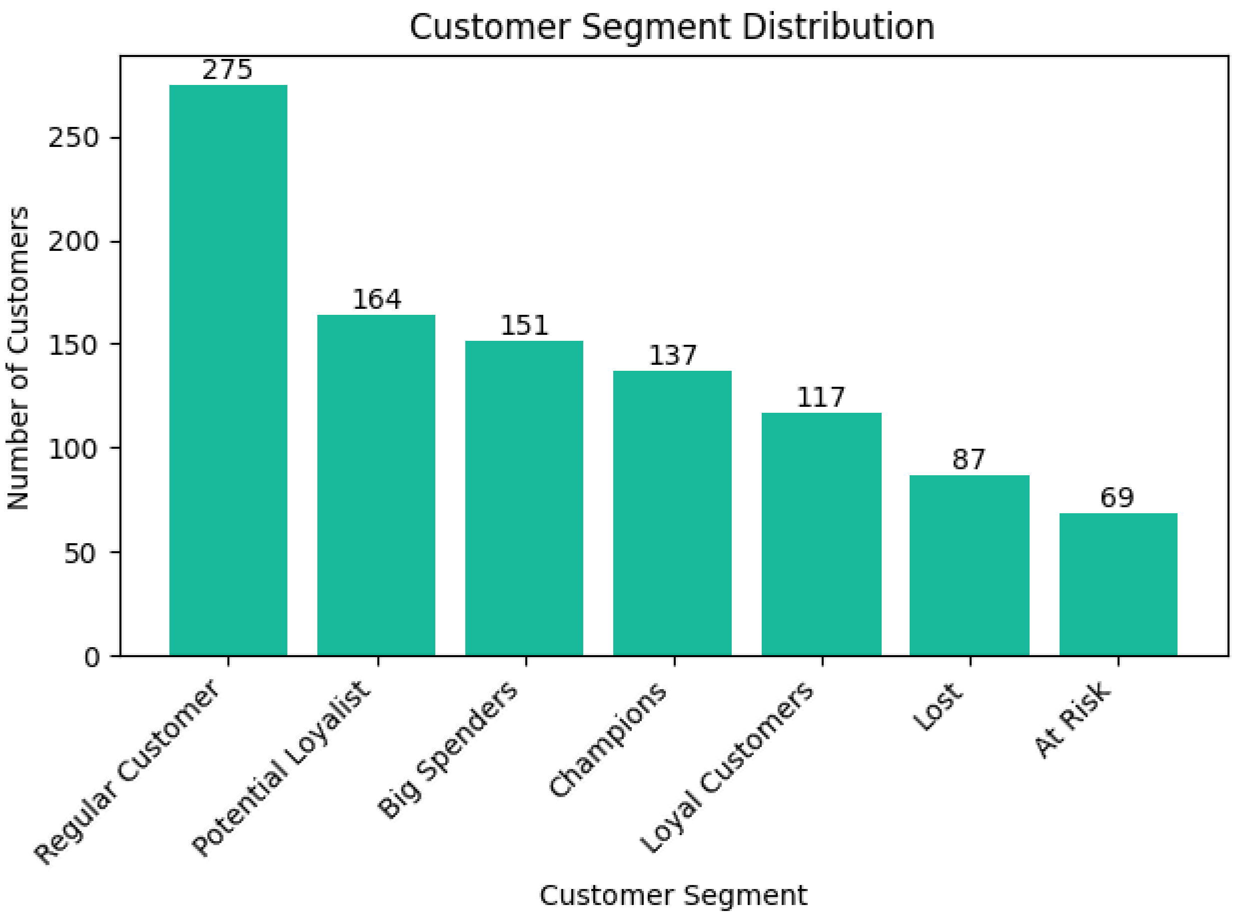
- Distribution of customers by RFM Segment
- Frequency of each Customer Segment
- Comparison of R, F, and M scores

These visualizations support business decisions by highlighting high-value customers, at-risk groups, and overall purchasing behavior.

In [8]:
```
# importing visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns
# ------------------------------------
```

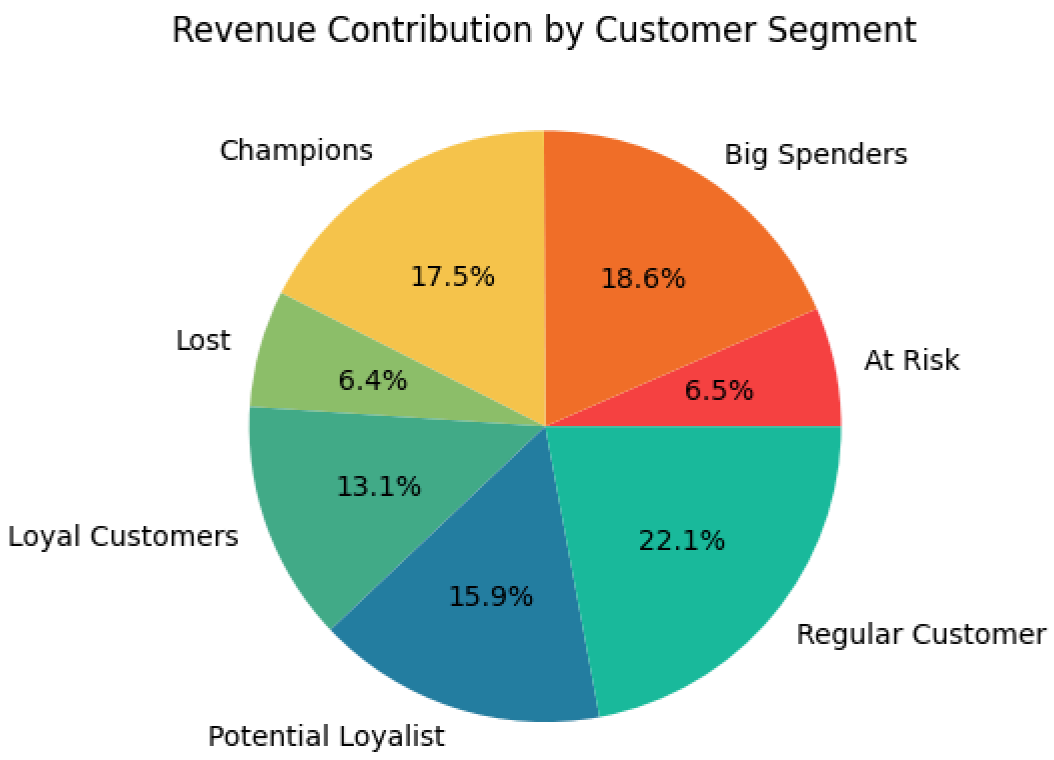
In [9]:
```
# Count the number of customers in each segment
segment_counts = df_rfm['Customer_Segment'].value_counts().reset_index()

# Rename columns for clarity
segment_counts.columns = ['Customer_Segment', 'count']

# Creating a new figure
plt.figure()

# Creating bar chart
bars = plt.bar(
    segment_counts['Customer_Segment'],
    segment_counts['count'],
    color="#1ABC9C"
)

# Rotate x-axis labels
plt.xticks(rotation=45, ha='right')

# Axis labels and title
plt.xlabel('Customer Segment')
plt.ylabel('Number of Customers')
plt.title('Customer Segment Distribution')

# Adding data labels on top of each bar
for bar in bars:
    height = bar.get_height()
    plt.text(
        bar.get_x() + bar.get_width() / 2,  # X position
        height,                              # Y position
        int(height),                         # Label text
        ha='center',
        va='bottom'
    )
```

```
# Adjusting layout
plt.tight_layout()

# Show plot
plt.show()
```



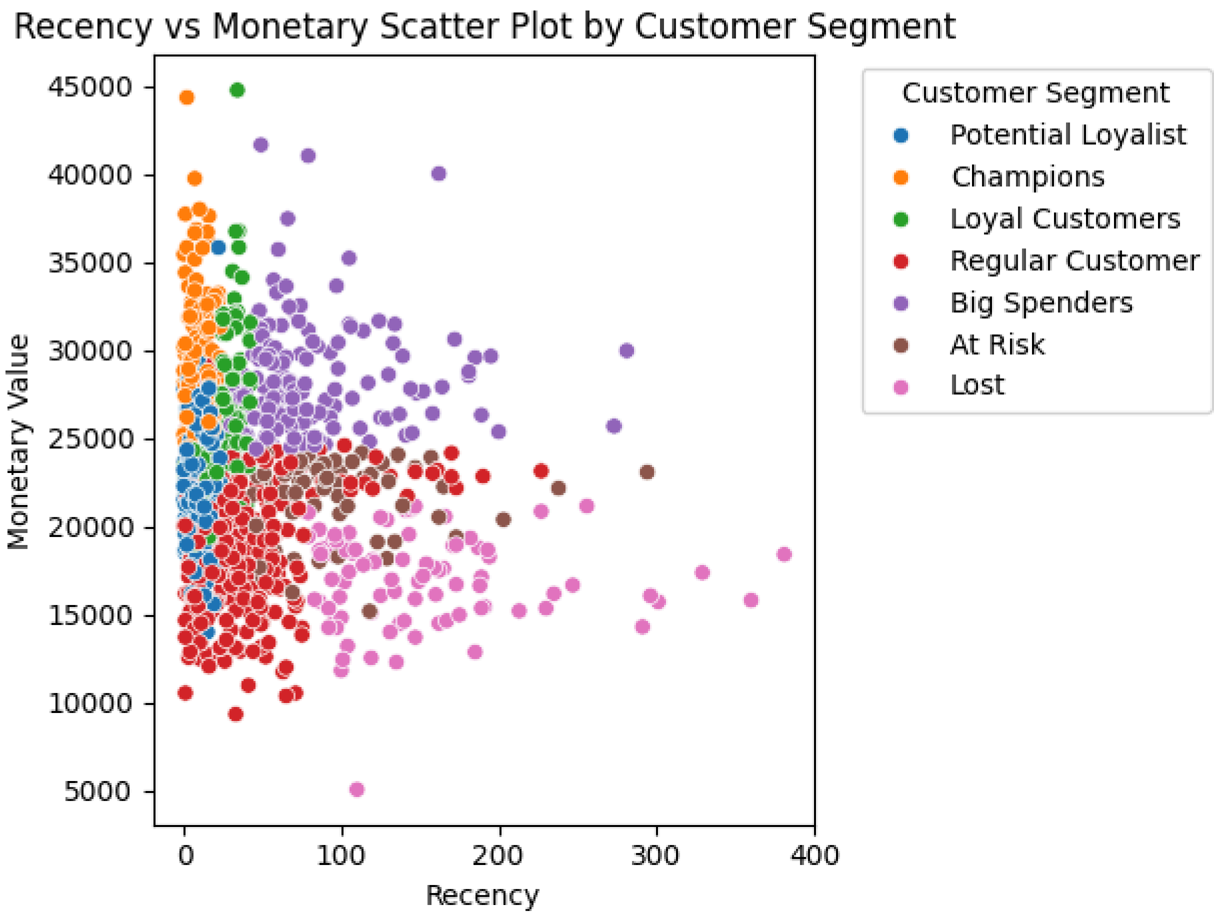
In [10]:
```
# Revenue contribution per segment
# Calculating total revenue (Monetary) for each customer segment
revenue_contribution = (
    df_rfm
    .groupby("Customer_Segment")["Monetary"]  # Group by customer segment
    .sum()                                    # Sum revenue per segment
```

```
    .reset_index()                            # Convert to DataFrame
)

# Rename columns for better readability
revenue_contribution.columns = ["Customer_Segment", "Revenue"]

# Creating a pie chart to show revenue contribution
plt.figure()
plt.pie(
    revenue_contribution["Revenue"],                # Values for pie slices
    labels=revenue_contribution["Customer_Segment"],# Labels for each slice
    autopct="%1.1f%%",                               # Show percentage values
    colors = ['#F94144', '#F3722C', '#F9C74F', '#90BE6D', '#43AA8B', '#277DA1', '#1ABC9C']


              # Custom colors
)

# Add a title to the chart
plt.title("Revenue Contribution by Customer Segment")

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



In [11]:
```
# Creating scatter plot
plt.figure()
sns.scatterplot(
    data=df_rfm,
    x="RecencyDays",                 # X-axis: Recency
    y="Monetary",                # Y-axis: Monetary value
    hue="Customer_Segment",      # Color points by segment
    palette="tab10"              # Color palette
)

# Add labels and title
plt.xlabel("Recency")
plt.ylabel("Monetary Value")
plt.title("Recency vs Monetary Scatter Plot by Customer Segment")
```

```
# Adjust legend position
plt.legend(title="Customer Segment", bbox_to_anchor=(1.05, 1), loc="upper left")

# Adjust layout
plt.tight_layout()

# Show plot
plt.show()
```

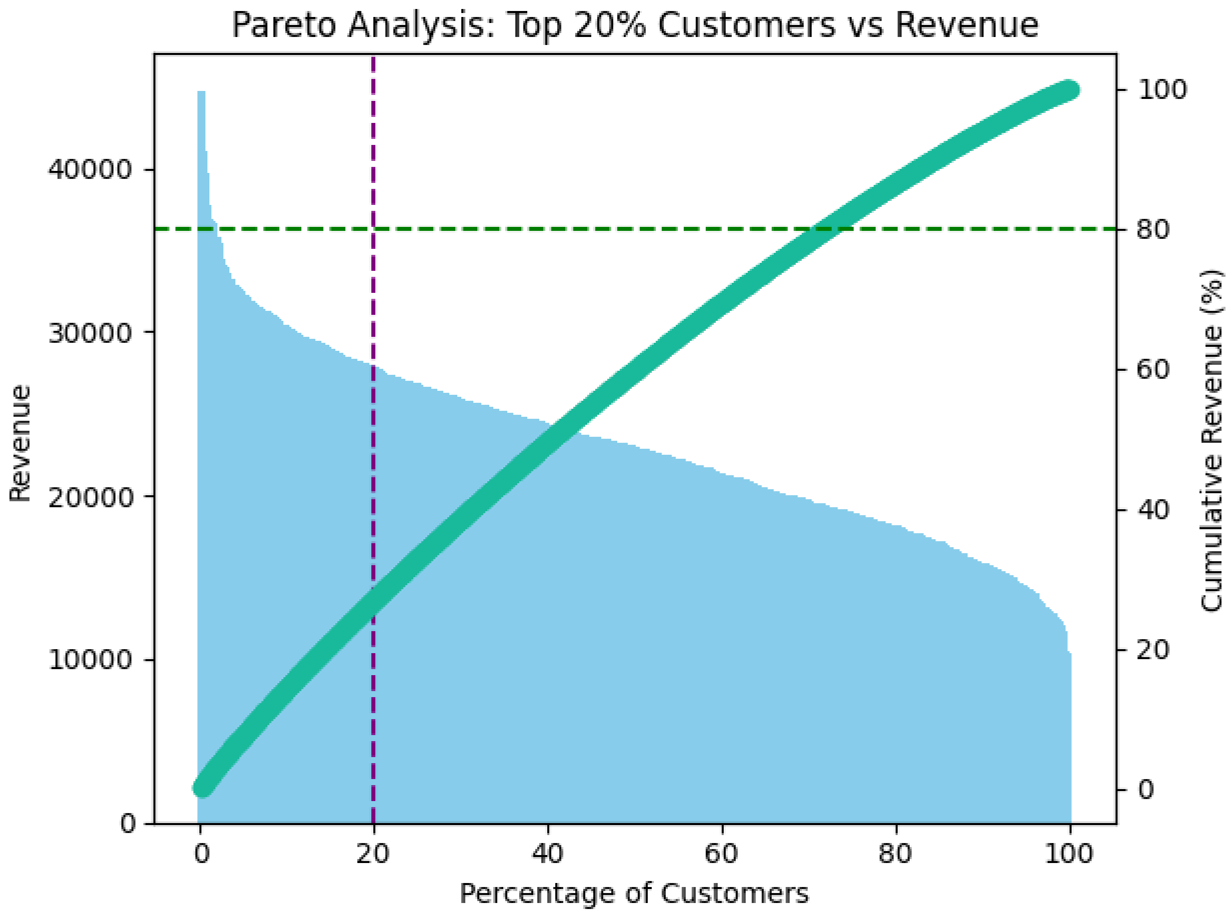


This scatter plot shows the relationship between **Recency** (how recently a customer made a purchase) and **Monetary Value** (how much money the customer spends), categorized by different customer segments. Each color represents a specific customer group based on purchasing behavior.

Customers with low recency and high monetary value are classified as **Champions** or **Big Spenders**, indicating highly valuable and active customers. Customers with higher recency and lower spending fall into segments such as **At Risk** or **Lost**, showing reduced engagement. This visualization helps businesses understand customer behavior, identify valuable customers, and design targeted retention and marketing strategies.

In [12]:
```
# Aggregate revenue per customer (assuming CustomerID exists)
customer_revenue = (
    df_rfm
    .groupby("CustomerID")["Monetary"]
    .sum()
    .reset_index()
)

# Sort customers by revenue (descending)
customer_revenue = customer_revenue.sort_values(
    by="Monetary", ascending=False
).reset_index(drop=True)

# Calculate cumulative revenue percentage
customer_revenue["Cumulative_Revenue"] = customer_revenue["Monetary"].cumsum()
customer_revenue["Cumulative_Revenue_Percent"] = (
    customer_revenue["Cumulative_Revenue"] /
    customer_revenue["Monetary"].sum()
) * 100

# Calculate customer percentage
customer_revenue["Customer_Percent"] = (
    (customer_revenue.index + 1) / len(customer_revenue)
) * 100

# Create Pareto chart
fig, ax1 = plt.subplots()

# Bar chart: individual customer revenue
ax1.bar(
    customer_revenue["Customer_Percent"],
    customer_revenue["Monetary"],
```

```
    color="skyblue"
)
ax1.set_xlabel("Percentage of Customers")
ax1.set_ylabel("Revenue")
ax1.set_title("Pareto Analysis: Top 20% Customers vs Revenue")

# Line chart: cumulative revenue percentage
ax2 = ax1.twinx()
ax2.plot(
    customer_revenue["Customer_Percent"],
    customer_revenue["Cumulative_Revenue_Percent"],
    color="#1ABC9C",
    marker="o"
)
ax2.set_ylabel("Cumulative Revenue (%)")

# Add reference lines for 80% revenue and 20% customers
ax2.axhline(80, color="green", linestyle="--")
ax1.axvline(20, color="purple", linestyle="--")

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



This Pareto chart represents the relationship between customers and total revenue. Customers are arranged from highest to lowest based on their spending contribution. The blue area shows individual customer revenue, while the green line represents the cumulative percentage of total revenue.

The chart demonstrates the Pareto Principle (80–20 rule), indicating that a small percentage of customers (approximately 20%) contributes to the majority (around 80%) of the total revenue. This analysis helps businesses identify high-value customers and focus their strategies on improving customer retention, targeted marketing, and revenue optimization.

Business Insights from RFM Analysis

- **Champions Segment:** Customers classified as **Champions** show the highest recency, frequency, and monetary values. This segment contributes the maximum revenue and represents the most engaged and loyal customers. These customers should be rewarded with exclusive offers, early access to products, and loyalty benefits.
- **Loyal Customers:** Loyal customers purchase frequently and have strong engagement but slightly lower monetary value compared to Champions. Retention strategies such as membership programs and personalized recommendations can help convert them into Champions.
- **Potential Loyalists:** This segment consists of customers who have purchased recently but not very frequently. With targeted marketing and timely follow-ups, these customers have high potential to become loyal customers.
- **At Risk Customers:** Customers in the At Risk segment were valuable in the past but have not made recent purchases. This group shows a decline in recency and needs immediate re-engagement through discounts, reminders, or personalized communication to prevent churn.
- **Lost Customers:** Lost customers have low recency, frequency, and monetary values. These customers are largely inactive and contribute minimal revenue. Marketing spend on this group should be limited unless reactivation campaigns are cost-effective.
- **Revenue Concentration:** The cumulative revenue analysis indicates that a relatively small portion of customers contributes a large share of total revenue. This highlights the importance of focusing marketing and retention efforts on high-value customer segments.

Conclusion

This project successfully demonstrates how RFM (Recency, Frequency, Monetary) analysis can be used to segment customers based on purchasing behavior. By analyzing transaction history and customer activity, distinct customer groups were identified, enabling better understanding of customer value.

The insights derived from this analysis allow businesses to focus on retaining high-value customers, re-engage customers at risk of churn, and optimize marketing strategies for different customer segments. Overall, RFM analysis provides a strong foundation for data-driven decision-making and improved business performance.

In [ ]: