🛍️ Customer Data Analysis for Business Insights

Using Python, Pandas & Data Visualization

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import timedelta
```

🔧 Step 1: Load the Data

In [2]:
```
df_master = pd.read_csv("Customer_Master_Data.csv")
df_trans = pd.read_csv("Customer_Transactions.csv")
```

In [3]:
```
df_master.head(10)
```

Out[3]:

	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
5	CUST10005	Fateh Sharaf	qkulkarni@gmail.com	Male	59	Jaipur	Single	3	2021-05-13
6	CUST10006	Khushi Wadhwa	craja@yahoo.com	Female	61	Hyderabad	Widowed	2	2021-11-12
7	CUST10007	Zeeshan Salvi	ira51@saini-kumar.com	Not Disclosed	32	Pune	Widowed	1	2021-08-10
8	CUST10008	Elakshi Trivedi	ayesha07@gmail.com	Female	32	Hyderabad	Widowed	4	2023-11-20
9	CUST10009	Neelofar Chada	abramsolanki@madan.com	Male	44	Jaipur	Single	0	2021-11-20

In [4]:
```
df_trans.head(10)
```

Out[4]:

	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
5	CUST10404	2/28/25	432.22
6	CUST10488	6/7/25	2178.25
7	CUST10988	3/25/25	85.46
8	CUST10657	9/10/23	1800.32
9	CUST10007	12/15/23	305.90

In [5]:
```
print(f"Total Customer records in the master data: {df_master.shape[0]}")
print(f"Total Transaction data: {df_trans.shape[0]}")
```

```
Total Customer records in the master data: 1000
Total Transaction data: 23050
```

In [6]:
```
df_master.info()
df_trans.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
<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

In [7]:
```
print(df_master["JoinDate"].dtype)
print(df_trans["TransactionDate"].dtype)
```

```
object
object
```

In [9]:
```
#converting all the date columns to date time format :

df_master["JoinDate"] = pd.to_datetime(df_master["JoinDate"])
df_trans["TransactionDate"] = pd.to_datetime(df_trans["TransactionDate"])
```

In [10]:
```
print("Duplicate CustomerIDs in master:", df_master["CustomerID"].duplicated().sum())
print("Invalid transaction IDs:", (~df_trans["CustomerID"].isin(df_master["CustomerID"])).sum())
```

```
Duplicate CustomerIDs in master: 0
Invalid transaction IDs: 0
```

In [11]:
```
missing_values_master = df_master.isnull().sum()
missing_values_txn = df_trans.isnull().sum()

print("Missing Values in Master dataset:")
print(missing_values_master)

print("\nMissing Values in Transaction dataset:")
print(missing_values_txn)
```

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

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

In [12]:
```
print(df_master.dtypes)
print(df_trans.dtypes)
```

```
CustomerID              object
Name                    object
Email                   object
Gender                  object
Age                      int64
City                    object
MaritalStatus           object
NumChildren              int64
JoinDate         datetime64[ns]
dtype: object
CustomerID                  object
TransactionDate      datetime64[ns]
TransactionAmount           float64
dtype: object
```

In [13]:
```
master_dup = df_master["CustomerID"].duplicated().sum()
print("Duplicate CustomerIDs in master dataset:", master_dup)
```

```
Duplicate CustomerIDs in master dataset: 0
```

In [14]:
```
invalid_ids = ~df_trans["CustomerID"].isin(df_master["CustomerID"])
print("Invalid transaction CustomerIDs:", invalid_ids.sum())
```

```
Invalid transaction CustomerIDs: 0
```

🔗 Step 3: Merge Data

In [15]:
```
merged_df = df_trans.merge(df_master, on="CustomerID", how="left")
print("Merged Data Shape:", merged_df.shape)
merged_df.head()
```

```
Merged Data Shape: (23050, 11)
```

Out[15]:

	CustomerID	TransactionDate	TransactionAmount	Name	Email	Gender	Age	City	MaritalStatus	Num
0	CUST10771	2023-07-31	2383.07	Lakshay Dhillon	dharmajantara@gmail.com	Female	27	Ahmedabad	Widowed	
1	CUST10100	2024-03-10	497.54	Aniruddh Borah	jivikabhavsar@gmail.com	Female	53	Lucknow	Divorced	
2	CUST10031	2025-02-17	536.78	Ritvik Ahuja	jhaverifarhan@chandra.org	Male	40	Ahmedabad	Single	
3	CUST10987	2023-07-17	314.89	Jayan Wagle	ojas82@gmail.com	Not Disclosed	75	Bangalore	Widowed	
4	CUST10831	2024-12-15	2543.19	Ishita Agarwal	vbalay@yahoo.com	Not Disclosed	27	Jaipur	Divorced	

📊 Step 4: RFM Calculation

In [16]:
```
print(f"Total customer recods in the master data:",df_master.shape[0])
print(f"Total Transaction data:",df_trans.shape[0])
```

```
Total customer recods in the master data: 1000
Total Transaction data: 23050
```

In [17]:
```
# Filter transaction records to include only those customers
# that exist in the master dataset.
# This removes invalid or unknown CustomerIDs.

df_txn = df_trans[df_trans["CustomerID"].isin(df_master["CustomerID"])].copy()
```

In [18]:
```
# Reference date is set as 1 day after the most recent transaction date.

ref_date = df_trans["TransactionDate"].max() + timedelta(days=1)
print(ref_date)
```

```
2025-07-30 00:00:00
```

In [19]:
```
# Group transaction data by CustomerID to calculate:
# 1. LastTxDate  → Most recent transaction date
# 2. Frequency   → Number of transactions
# 3. Monetary    → Total spend (sum of transaction amounts)

rfm = (
    df_trans.groupby("CustomerID")
    .agg(
        LastTxDate=("TransactionDate", "max"),
        Frequency=("TransactionDate", "count"),
        Monetary=("TransactionAmount", "sum")
    )
    .reset_index()
```

```
)
rfm["Recency"] = (ref_date - rfm["LastTxDate"]).dt.days
```

In [20]:
```
# Reorder the RFM dataframe columns to the standard format:
# CustomerID → Recency → Frequency → Monetary

rfm = rfm[["CustomerID","Recency","Frequency","Monetary"]]
```

In [21]:
```
rfm.head(10)
```

Out[21]:

	CustomerID	Recency	Frequency	Monetary
0	CUST10000	13	23	21265.49
1	CUST10001	35	30	28654.31
2	CUST10002	18	24	23884.03
3	CUST10003	81	25	24206.03
4	CUST10004	8	19	25565.30
5	CUST10005	24	29	29459.82
6	CUST10006	11	28	27922.36
7	CUST10007	86	15	14957.06
8	CUST10008	3	19	19479.25
9	CUST10009	7	25	22832.83

🏆 Step 5: Score RFM

In [22]:
```
#Defining the scores for R/F/M

#Recency (lower is better)
#<=30 days: 5
# <=60 days: 4
# <=120 days: 3
# <= 240 days: 2
#  >240:  1

r_bins = [0,30,60,120,240,float("inf")]
r_labels = [5, 4, 3, 2, 1] # Reverse because lower recency is better
rfm["R_Score"]= pd.cut(rfm["Recency"],bins = r_bins, labels = r_labels, include_lowest=True, right=True).astype(int)

#Frequency (Higher is Better)
# <=07: 1
# <=14: 2
# <=21 : 3
# <=28: 4
#  >28: 5

f_bins = [0,7,14,21,28,float("inf")]
f_labels = [1, 2, 3, 4, 5]
rfm["F_Score"] = pd.cut(rfm["Frequency"], bins=f_bins, labels = f_labels, include_lowest = True, right= True).astype(int)

#:: Monetary:: (Higher is better):
# <= 10000: 1
# <= 20000: 2
# <= 30000: 3
# <= 40000: 4
#  > 40000: 5

m_bins = [0, 10000, 20000, 30000, 40000, float("inf")]
m_labels = [1, 2, 3, 4, 5]
rfm["M_Score"] = pd.cut(rfm["Monetary"], bins=m_bins, labels = m_labels, include_lowest = True, right= True).astype(int)
rfm.head(5)
```

Out[22]:

	CustomerID	Recency	Frequency	Monetary	R_Score	F_Score	M_Score
0	CUST10000	13	23	21265.49	5	4	3
1	CUST10001	35	30	28654.31	4	5	3
2	CUST10002	18	24	23884.03	5	4	3
3	CUST10003	81	25	24206.03	3	4	3
4	CUST10004	8	19	25565.30	5	3	3

🧬 Step 6: Create RFM Segment Code

In [23]:
```
#RFM Score string
rfm["RFM_Score"] = (rfm["R_Score"].astype(str) + rfm["F_Score"].astype(str) + rfm["M_Score"].astype(str))
rfm.head(5)
```

Out[23]:

	CustomerID	Recency	Frequency	Monetary	R_Score	F_Score	M_Score	RFM_Score
0	CUST10000	13	23	21265.49	5	4	3	543
1	CUST10001	35	30	28654.31	4	5	3	453
2	CUST10002	18	24	23884.03	5	4	3	543
3	CUST10003	81	25	24206.03	3	4	3	343
4	CUST10004	8	19	25565.30	5	3	3	533

🏷️ Step 7: Assign Segment Labels

In [24]:
```
# Naming the segment

def categorise_seg(r,f,m):
    if (r>=4)  and (f>= 4)  and (m>=4):
        return "Champions"
    elif (f>=4) and (r>=2):
        return "Loyal"
    elif (r>=4) and (2<=f<=3):
        return "Potential Loyalist"
    elif (r<=2) and (f>=3):
        return "At Risk"
    elif (m>=4) and (2<=f<=3) and (r>=3):
        return "Big spenders"
    elif (r==1)and (f<=2) and (m<=2):
        return "Lost"
    else:
        return "Others"

rfm["Segment"] =[ categorise_seg(r,f,m) for r,f,m, in zip(rfm["R_Score"], rfm["F_Score"], rfm["M_Score"])]
```

In [25]:
```
rfm.head(10)
```

Out[25]:

	CustomerID	Recency	Frequency	Monetary	R_Score	F_Score	M_Score	RFM_Score	Segment
0	CUST10000	13	23	21265.49	5	4	3	543	Loyal
1	CUST10001	35	30	28654.31	4	5	3	453	Loyal
2	CUST10002	18	24	23884.03	5	4	3	543	Loyal
3	CUST10003	81	25	24206.03	3	4	3	343	Loyal
4	CUST10004	8	19	25565.30	5	3	3	533	Potential Loyalist
5	CUST10005	24	29	29459.82	5	5	3	553	Loyal
6	CUST10006	11	28	27922.36	5	4	3	543	Loyal
7	CUST10007	86	15	14957.06	3	3	2	332	Others
8	CUST10008	3	19	19479.25	5	3	2	532	Potential Loyalist
9	CUST10009	7	25	22832.83	5	4	3	543	Loyal

In [26]:
```
rfm["Segment"].value_counts()
```

Out[26]:
```
Segment
Loyal                 542
Potential Loyalist    248
Champions              84
Others                 77
At Risk                47
Lost                    2
Name: count, dtype: int64
```

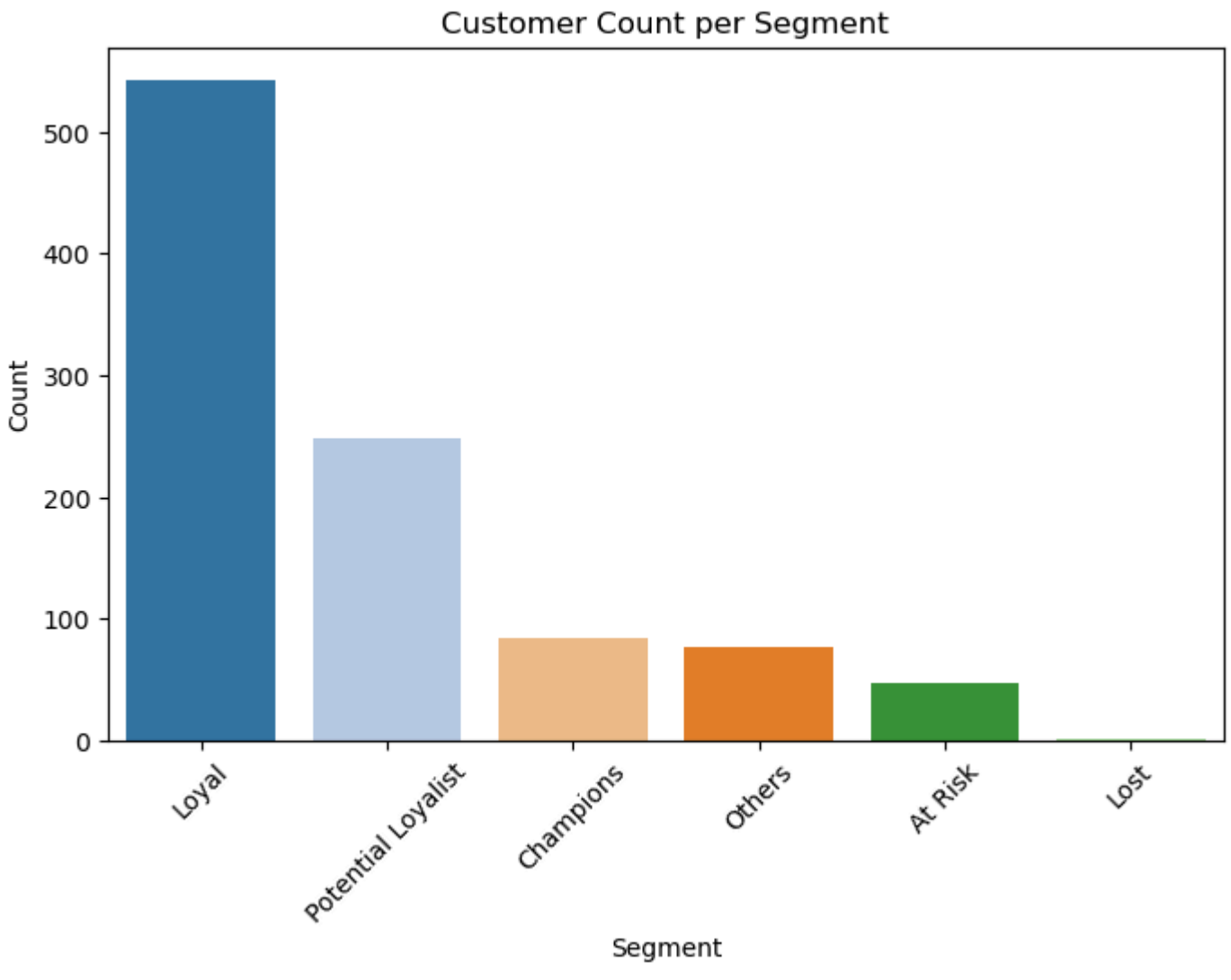
📈 Step 8: Visualize Results

In [27]:
```
# Simple business Analysis
print("\n=============== SIMPLE BUSINESS ANALYSIS =============")

total_customers = rfm["CustomerID"].nunique()
total_revenue = rfm["Monetary"].sum()
avg_revenue_per_customer = rfm["Monetary"].mean()

print(f"\nTotal Customers: {total_customers}")
print(f"Total Revenue: Rs. {total_revenue:,.2f}")
print(f"Average Revenue per Customer: Rs. {avg_revenue_per_customer:,.2f}")
```

```
print("\n=============== SEGMENT WISE SUMMARY ===============\n")

segment_customers = rfm["Segment"].value_counts()
print("Number of Customers in Each Segment:")
print(segment_customers)

revenue_segment = (
    rfm.groupby("Segment")["Monetary"]
    .sum()
    .reset_index()
    .sort_values(by="Monetary", ascending=False)
)

print("\nRevenue Contribution by Each Segment:")
print(revenue_segment)
```

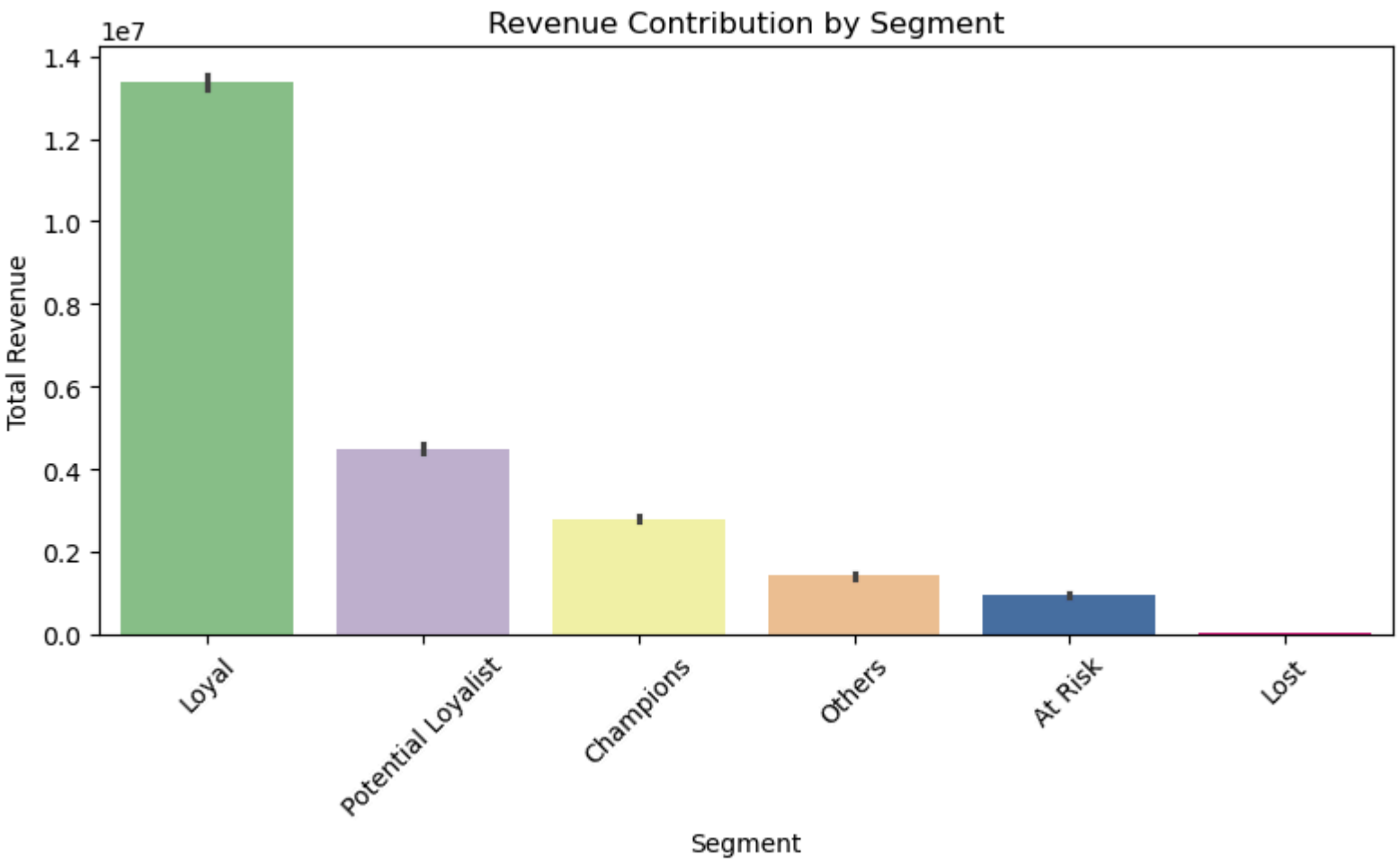
```
=============== SIMPLE BUSINESS ANALYSIS =============

Total Customers: 1000
Total Revenue: Rs. 23,053,199.66
Average Revenue per Customer: Rs. 23,053.20

=============== SEGMENT WISE SUMMARY ===============

Number of Customers in Each Segment:
Segment
Loyal                542
Potential Loyalist   248
Champions             84
Others                77
At Risk               47
Lost                   2
Name: count, dtype: int64

Revenue Contribution by Each Segment:
              Segment     Monetary
3               Loyal  13381926.89
5  Potential Loyalist   4495435.99
1           Champions   2791699.03
4              Others   1416376.98
0             At Risk    938539.38
2                Lost     29221.39
```

In [28]:
```
plt.figure(figsize=(8,5))
sns.countplot(data=rfm,x="Segment",hue="Segment",order=rfm["Segment"].value_counts().index,palette="tab20",dodge=False)
plt.title("Customer Count per Segment")
plt.xlabel("Segment")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.legend([], [], frameon=False)
plt.show()
```

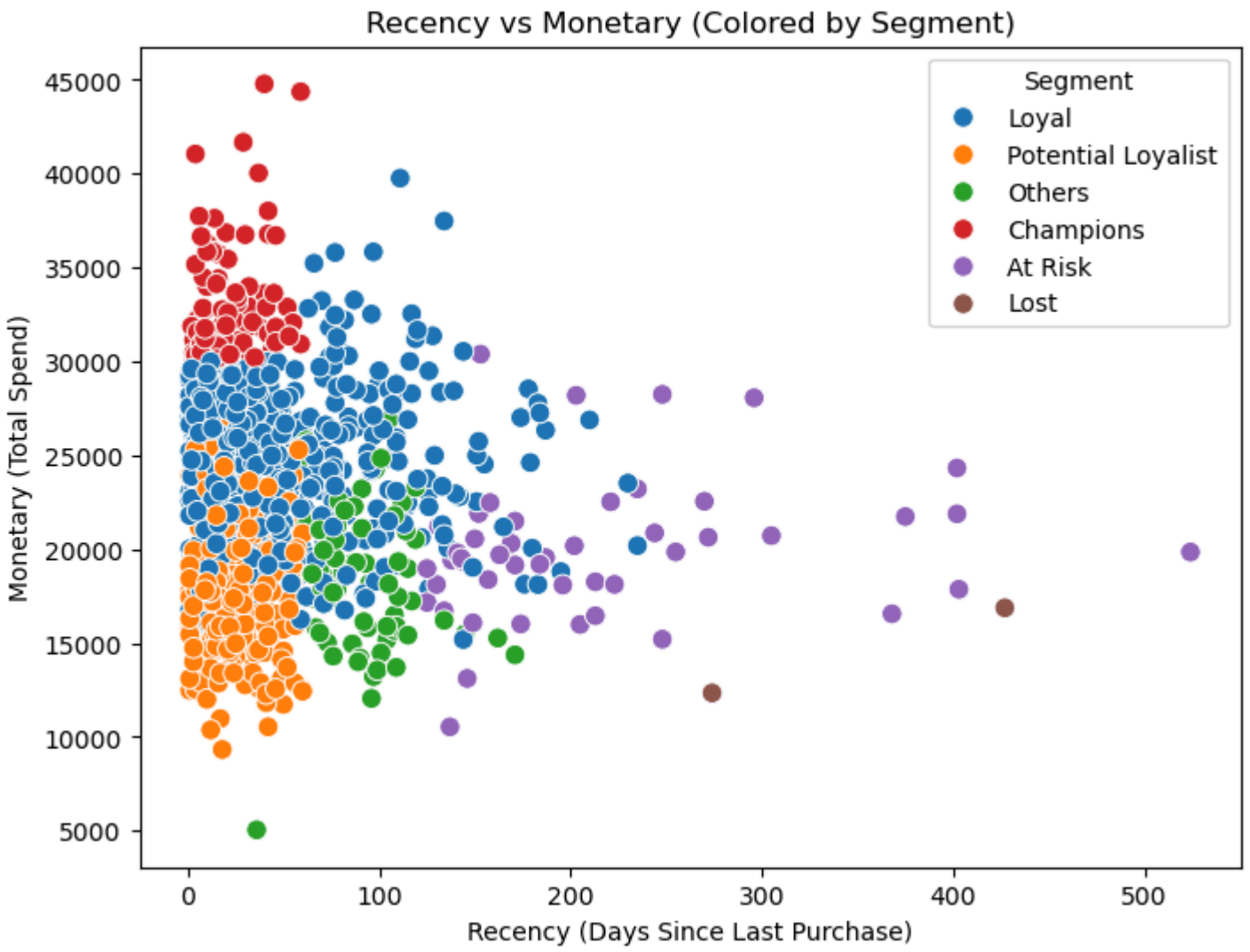


In [29]:
```
plt.figure(figsize=(8,5))
sns.barplot(
```

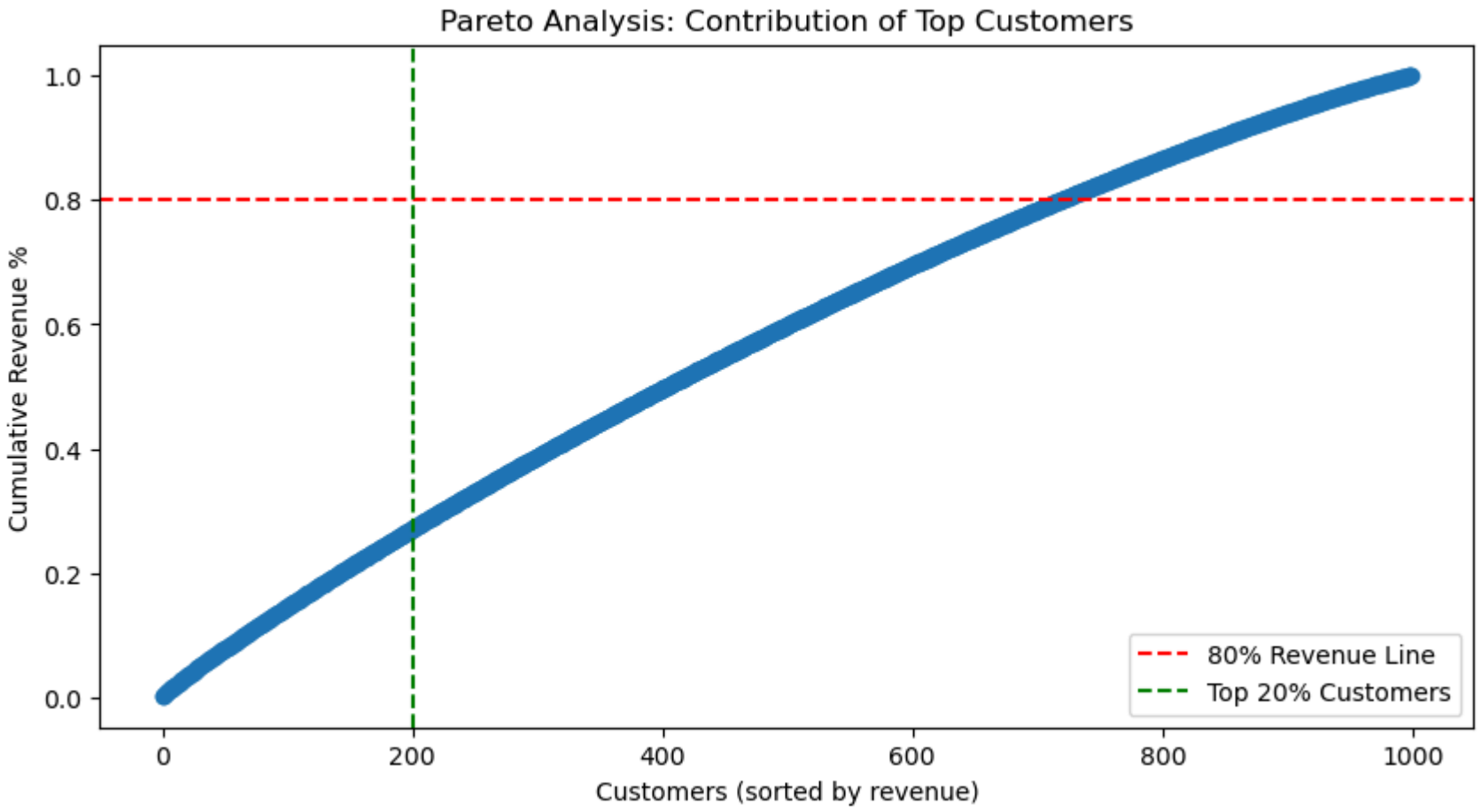
```
    data=rfm,
    x="Segment",
    y="Monetary",
    estimator=np.sum,
    order=rfm.groupby("Segment")["Monetary"].sum().sort_values(ascending=False).index,
    palette="Accent",
    hue="Segment",
    dodge=False)
plt.legend([], [], frameon=False)

plt.title("Revenue Contribution by Segment")
plt.xlabel("Segment")
plt.ylabel("Total Revenue")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [30]:
```
plt.figure(figsize=(8,6))
sns.scatterplot(
    data=rfm,x="Recency",y="Monetary",hue="Segment",palette="tab10",s=70)
plt.title("Recency vs Monetary (Colored by Segment)")
plt.xlabel("Recency (Days Since Last Purchase)")
plt.ylabel("Monetary (Total Spend)")
plt.legend(title="Segment")
plt.show()
```



In [31]:
```python
# Sort customers by revenue (descending)
pareto = rfm.sort_values(by="Monetary", ascending=False).reset_index(drop=True)

# Calculate cumulative revenue %
pareto["CumRevenue"] = pareto["Monetary"].cumsum()
pareto["CumRevenuePct"] = pareto["CumRevenue"] / pareto["Monetary"].sum()

# Plot Pareto curve
plt.figure(figsize=(10,5))
plt.plot(pareto.index, pareto["CumRevenuePct"], marker="o")
plt.axhline(0.80, color="red", linestyle="--", label="80% Revenue Line")
plt.axvline(int(len(pareto)*0.2), color="green", linestyle="--", label="Top 20% Customers")
plt.title("Pareto Analysis: Contribution of Top Customers")
plt.xlabel("Customers (sorted by revenue)")
plt.ylabel("Cumulative Revenue %")
plt.legend()
plt.show()
```



In [32]:
```python
rfm.to_csv("rfm_results.csv", index=False)
revenue_segment.to_csv("segment_revenue.csv", index=False)
```

📊 Business Insights

In [34]:
```
print("\n================= BUSINESS INSIGHTS =================\n")

# ==================== 1. Overall Performance ====================
print("📌 OVERALL BUSINESS PERFORMANCE\n")

total_customers = rfm["CustomerID"].nunique()
total_revenue = rfm["Monetary"].sum()
avg_revenue = rfm["Monetary"].mean()

print(f"• Total Customers: {total_customers}")
print(f"• Total Revenue: Rs. {total_revenue:,.2f}")
print(f"• Avg Revenue Per Customer: Rs. {avg_revenue:,.2f}\n")


# ==================== 2. Segment Insights ====================
print("📌 RFM SEGMENT INSIGHTS\n")

segment_counts = rfm["Segment"].value_counts()
segment_revenue = rfm.groupby("Segment")["Monetary"].sum()

print("• Customer Count by Segment:\n")
print(segment_counts)
print("\n• Revenue Contribution by Segment:\n")
print(segment_revenue)

# Identify strongest segments
top_segment = segment_revenue.idxmax()
top_revenue_value = segment_revenue.max()

print(f"\n🔥 Top Performing Segment: {top_segment} (Revenue: Rs. {top_revenue_value:,.2f})\n")


# ==================== 3. Recency Insights ====================
print("📌 RECENCY INSIGHTS\n")

recent_buyers = rfm[rfm["Recency"] <= rfm["Recency"].median()]
inactive_buyers = rfm[rfm["Recency"] > rfm["Recency"].median()]

print(f"• Active / Recent Buyers: {len(recent_buyers)} customers")
print(f"• Inactive / At-Risk Buyers: {len(inactive_buyers)} customers\n")


# ==================== 4. Pareto Analysis (80/20 Rule) ====================
print("📌 PARETO (80/20) INSIGHTS\n")

pareto = rfm.sort_values("Monetary", ascending=False).reset_index(drop=True)
pareto["cum_rev_pct"] = pareto["Monetary"].cumsum() / pareto["Monetary"].sum() * 100

top_20pct_count = int(0.2 * len(pareto))
top_20pct_revenue = pareto.loc[:top_20pct_count, "Monetary"].sum()

print(f"• Top 20% Customers Count: {top_20pct_count}")
print(f"• Revenue by Top 20%: Rs. {top_20pct_revenue:,.2f}")
print(f"• Percentage Contribution: {pareto.loc[top_20pct_count, 'cum_rev_pct']:.2f}% of total revenue\n")


# ==================== 5. Demographic Insights ====================
print("📌 DEMOGRAPHIC INSIGHTS (If present)\n")

if "Age" in rfm.columns:
    print(f"• Average Age: {rfm['Age'].mean():.1f}")
    print("• Age Distribution:")
    print(rfm["Age"].describe(), "\n")

if "Gender" in rfm.columns:
    print("• Gender Distribution:")
    print(rfm["Gender"].value_counts(), "\n")

if "City" in rfm.columns:
    print("• Top Cities by Customer Count:")
    print(rfm["City"].value_counts().head(), "\n")

print("=============== END OF INSIGHTS ===============")
```

```
================= BUSINESS INSIGHTS =================

📌 OVERALL BUSINESS PERFORMANCE

• Total Customers: 1000
• Total Revenue: Rs. 23,053,199.66
• Avg Revenue Per Customer: Rs. 23,053.20

📌 RFM SEGMENT INSIGHTS

• Customer Count by Segment:

Segment
Loyal                 542
Potential Loyalist    248
Champions              84
Others                 77
At Risk                47
Lost                    2
Name: count, dtype: int64

• Revenue Contribution by Segment:

Segment
At Risk                938539.38
Champions             2791699.03
Lost                    29221.39
Loyal                13381926.89
Others                1416376.98
Potential Loyalist    4495435.99
Name: Monetary, dtype: float64

🔥 Top Performing Segment: Loyal (Revenue: Rs. 13,381,926.89)

📌 RECENCY INSIGHTS

• Active / Recent Buyers: 503 customers
• Inactive / At-Risk Buyers: 497 customers

📌 PARETO (80/20) INSIGHTS

• Top 20% Customers Count: 200
• Revenue by Top 20%: Rs. 6,258,992.64
• Percentage Contribution: 27.15% of total revenue

📌 DEMOGRAPHIC INSIGHTS (If present)

=============== END OF INSIGHTS ===============
```