In [49]:
```
#First, We have to import libraries.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import timedelta

#Now, we have to load CSV file for reading the given csv file with the help of pandas library.

df_Cust=pd.read_csv("Customer_Master_Data.csv")
df_txn=pd.read_csv("Customer_Transactions.csv")
```

In [50]:
```
# For preview, we have used head function.
df_Cust.head(10)
```

Out[50]:

	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 [51]:
```
df_txn.head(10)
```

Out[51]:

	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 [52]:

```
# For shape, we have used shape function.
print(f"Total Customer record in the Customer Master Data",df_Cust.shape[0])
print(f"Total Transaction record in the Customer_Transactions Data",df_txn.shape[0])
```

```
Total Customer record in the Customer Master Data 1000
Total Transaction record in the Customer_Transactions Data 23050
```

In [53]:

```
# For structure of data set, we have used info function.
df_Cust.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
```

In [54]:
```
df_txn.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
```

In [55]:
```
# then we have tried to find missing values by using "isnull().sum()" function
#so that I could clean the data set for visualization.
missing_values_cust=df_Cust.isnull().sum()

print("Total missing vlaues in the Customer Master Data:\n", missing_values_cust)

missing_values_txn=df_txn.isnull().sum()

print("Total missing vlaues in the Customer_Transactions Data:\n",missing_values_txn)
```

```
Total missing vlaues in the Customer Master Data:
 CustomerID       0
Name            0
Email           0
Gender          0
Age             0
City            0
MaritalStatus   0
NumChildren     0
JoinDate        0
dtype: int64
Total missing vlaues in the Customer_Transactions Data:
 CustomerID           0
TransactionDate     0
TransactionAmount   0
dtype: int64
```

In [56]:
```
# we have checked the data set by using info function to validate
# that the dataset are in correct formate.
df_Cust.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
```

In [57]:
```
df_txn.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
```

In [58]:
```
# We have convert "Join date column" in Customer_master_dataset and
#" Transaction Date in Transaction dataset because both are in object type.
df_Cust["JoinDate"]=pd.to_datetime(df_Cust["JoinDate"])
df_txn["TransactionDate"]=pd.to_datetime(df_txn["TransactionDate"])
```

```
C:\Users\A\AppData\Local\Temp\ipykernel_11828\4124459641.py:3: UserWarning: Could not infer format, so each element will be par
sed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expected, please specify a format.
  df_txn["TransactionDate"]=pd.to_datetime(df_txn["TransactionDate"])
```

In [59]:
```
# Keep only Valid Transaction
# We will keep only those transaction whose customer_id is
# in the Customer_master_dataset.
df_txn=df_txn[df_txn["CustomerID"].isin(df_Cust["CustomerID"])].copy()
```

In [60]:
```
# Now, we will do RFM calculation
#Reference_Date=Max_Transaction_date +1 day
ref_date=df_txn["TransactionDate"].max()+timedelta(days=1)
print(ref_date)
```

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

In [61]:
```
# For each Customer
# Last_Transaction_date = Most recent purchase date
# Frequency = Number of rows(purchases)
# Monetary = sum of Transaction amount
rfm=(df_txn.groupby("CustomerID").agg(Last_Transaction_date=("TransactionDate","max"),Frequency=("TransactionDate","count"),
                                     Monetary=("TransactionAmount","sum")).reset_index())
```

In [62]:
```
rfm.head(15)
```

Out[62]:

	CustomerID	Last_Transaction_date	Frequency	Monetary
0	CUST10000	2025-07-17	23	21265.49
1	CUST10001	2025-06-25	30	28654.31
2	CUST10002	2025-07-12	24	23884.03
3	CUST10003	2025-05-10	25	24206.03
4	CUST10004	2025-07-22	19	25565.30
5	CUST10005	2025-07-06	29	29459.82
6	CUST10006	2025-07-19	28	27922.36
7	CUST10007	2025-05-05	15	14957.06
8	CUST10008	2025-07-27	19	19479.25
9	CUST10009	2025-07-23	25	22832.83
10	CUST10010	2025-07-16	20	20932.93
11	CUST10011	2025-07-13	22	23159.47
12	CUST10012	2025-06-01	28	26626.07
13	CUST10013	2025-07-17	19	14536.66
14	CUST10014	2025-06-28	21	23325.00

In [63]:

```
#Recency (in days) = Difference from ref_date
rfm["Recency"]=(ref_date-rfm["Last_Transaction_date"]).dt.days
rfm.head(15)
```

Out[63]:

	CustomerID	Last_Transaction_date	Frequency	Monetary	Recency
0	CUST10000	2025-07-17	23	21265.49	13
1	CUST10001	2025-06-25	30	28654.31	35
2	CUST10002	2025-07-12	24	23884.03	18
3	CUST10003	2025-05-10	25	24206.03	81
4	CUST10004	2025-07-22	19	25565.30	8
5	CUST10005	2025-07-06	29	29459.82	24
6	CUST10006	2025-07-19	28	27922.36	11
7	CUST10007	2025-05-05	15	14957.06	86
8	CUST10008	2025-07-27	19	19479.25	3
9	CUST10009	2025-07-23	25	22832.83	7
10	CUST10010	2025-07-16	20	20932.93	14
11	CUST10011	2025-07-13	22	23159.47	17
12	CUST10012	2025-06-01	28	26626.07	59
13	CUST10013	2025-07-17	19	14536.66	13
14	CUST10014	2025-06-28	21	23325.00	32

In [64]:

```
# put the column in desired sequence
rfm=rfm[["CustomerID","Recency","Frequency","Monetary"]]
rfm.head(15)
```

Out[64]:

	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
10	CUST10010	14	20	20932.93
11	CUST10011	17	22	23159.47
12	CUST10012	59	28	26626.07
13	CUST10013	13	19	14536.66
14	CUST10014	32	21	23325.00

In [65]:

```
# Now,We will define the score for R/F/M
#Recency(Lower is better)
#<=30 days   :5
#<=60 days   :4
#<=120 days  :3
#<=240 days  :2
#>240 days   :1
r_bins=[0,30,60,120,240,float("inf")]
r_labels=[5,4,3,2,1]
rfm["R_Score"]=pd.cut(rfm["Recency"],bins=r_bins,labels=r_labels,include_lowest=True,right=True).astype(int)
```

```
#Frequency(Higher is better)
#<=7  :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)
```

In [66]:

```
rfm.head(25)
```

Out[66]:

	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
5	CUST10005	24	29	29459.82	5	5	3
6	CUST10006	11	28	27922.36	5	4	3
7	CUST10007	86	15	14957.06	3	3	2
8	CUST10008	3	19	19479.25	5	3	2
9	CUST10009	7	25	22832.83	5	4	3
10	CUST10010	14	20	20932.93	5	3	3
11	CUST10011	17	22	23159.47	5	4	3
12	CUST10012	59	28	26626.07	4	4	3
13	CUST10013	13	19	14536.66	5	3	2
14	CUST10014	32	21	23325.00	4	3	3
15	CUST10015	6	20	26315.71	5	3	3
16	CUST10016	4	24	24607.24	5	4	3
17	CUST10017	42	25	21241.48	4	4	3
18	CUST10018	72	19	20383.92	3	3	3
19	CUST10019	155	25	24529.64	2	4	3
20	CUST10020	4	22	19111.42	5	4	2
21	CUST10021	19	20	18806.90	5	3	2

	CustomerID	Recency	Frequency	Monetary	R_Score	F_Score	M_Score
22	CUST10022	60	15	21368.65	4	3	3
23	CUST10023	79	27	30622.22	3	4	4
24	CUST10024	8	24	25362.43	5	4	3

In [67]:
```
#Now we will add new cloumn as "RFM_Score"
rfm["RFM_Score"]=(rfm["R_Score"].astype(str)+rfm["F_Score"].astype(str)+rfm["M_Score"].astype(str))
rfm.head(20)
```

Out[67]:

	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
5	CUST10005	24	29	29459.82	5	5	3	553
6	CUST10006	11	28	27922.36	5	4	3	543
7	CUST10007	86	15	14957.06	3	3	2	332
8	CUST10008	3	19	19479.25	5	3	2	532
9	CUST10009	7	25	22832.83	5	4	3	543
10	CUST10010	14	20	20932.93	5	3	3	533
11	CUST10011	17	22	23159.47	5	4	3	543
12	CUST10012	59	28	26626.07	4	4	3	443
13	CUST10013	13	19	14536.66	5	3	2	532
14	CUST10014	32	21	23325.00	4	3	3	433
15	CUST10015	6	20	26315.71	5	3	3	533
16	CUST10016	4	24	24607.24	5	4	3	543
17	CUST10017	42	25	21241.48	4	4	3	443
18	CUST10018	72	19	20383.92	3	3	3	333
19	CUST10019	155	25	24529.64	2	4	3	243

In [68]:

```
#Now we will add new cloumn as "Segement"
def segment_row(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 (f<=2):
        return "Lost"
    else:
        return "Others"
rfm["Segment"]=[segment_row(r,f,m) for r,f,m, in zip(rfm["R_Score"],rfm["F_Score"],rfm["M_Score"])]
```

In [69]:

```
rfm.head(30)
```

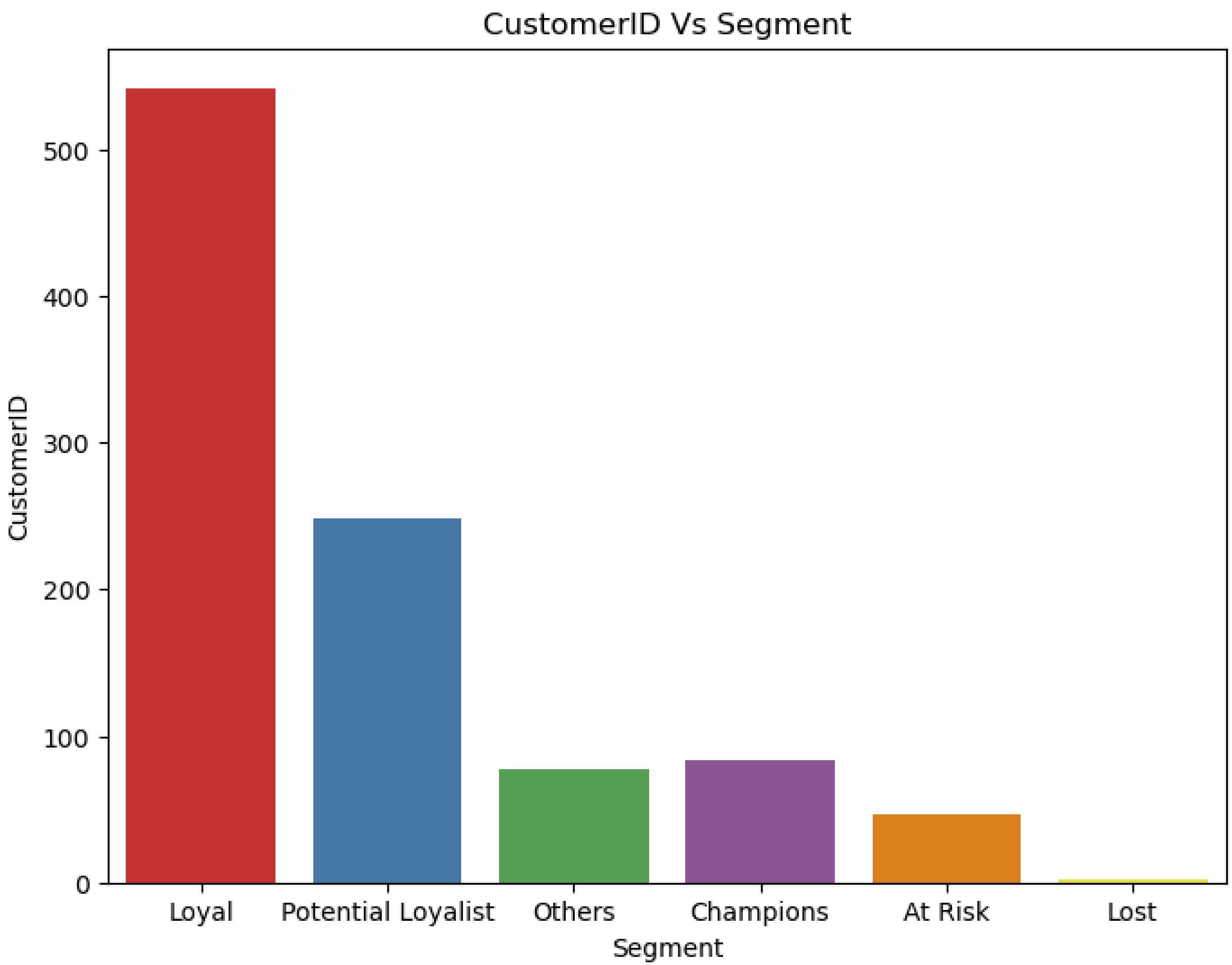
Out[69]:

	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
10	CUST10010	14	20	20932.93	5	3	3	533	Potential Loyalist
11	CUST10011	17	22	23159.47	5	4	3	543	Loyal
12	CUST10012	59	28	26626.07	4	4	3	443	Loyal
13	CUST10013	13	19	14536.66	5	3	2	532	Potential Loyalist
14	CUST10014	32	21	23325.00	4	3	3	433	Potential Loyalist
15	CUST10015	6	20	26315.71	5	3	3	533	Potential Loyalist
16	CUST10016	4	24	24607.24	5	4	3	543	Loyal
17	CUST10017	42	25	21241.48	4	4	3	443	Loyal
18	CUST10018	72	19	20383.92	3	3	3	333	Others
19	CUST10019	155	25	24529.64	2	4	3	243	Loyal
20	CUST10020	4	22	19111.42	5	4	2	542	Loyal
21	CUST10021	19	20	18806.90	5	3	2	532	Potential Loyalist

	CustomerID	Recency	Frequency	Monetary	R_Score	F_Score	M_Score	RFM_Score	Segment
22	CUST10022	60	15	21368.65	4	3	3	433	Potential Loyalist
23	CUST10023	79	27	30622.22	3	4	4	344	Loyal
24	CUST10024	8	24	25362.43	5	4	3	543	Loyal
25	CUST10025	4	19	14074.55	5	3	2	532	Potential Loyalist
26	CUST10026	69	25	23621.41	3	4	3	343	Loyal
27	CUST10027	30	20	22327.31	5	3	3	533	Potential Loyalist
28	CUST10028	11	18	18921.46	5	3	2	532	Potential Loyalist
29	CUST10029	41	24	27318.86	4	4	3	443	Loyal

In [70]:

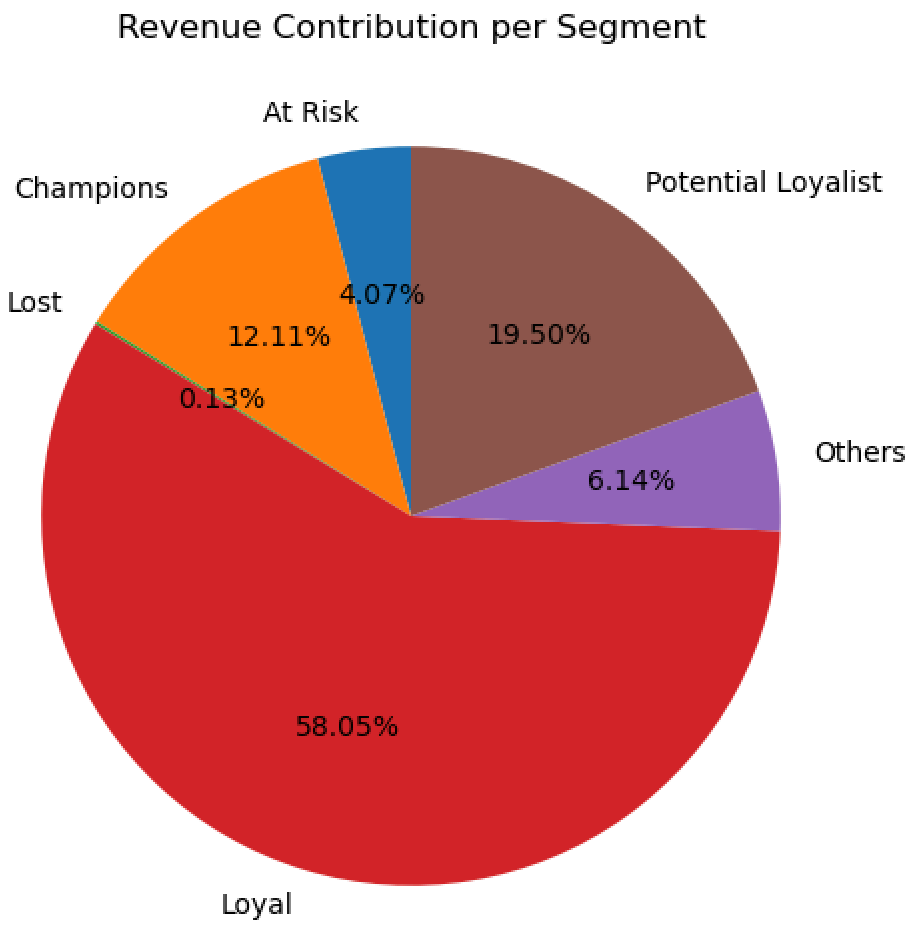
```
#Visualization
#Count of customers in each segment
plt.figure(figsize=(8,6))
sns.countplot(x="Segment",data=rfm,hue="Segment",palette="Set1")
plt.title("CustomerID Vs Segment")
plt.xlabel("Segment")
plt.ylabel("CustomerID")
plt.show()
```



In [71]:
```
rfm["Segment"].value_counts()
```

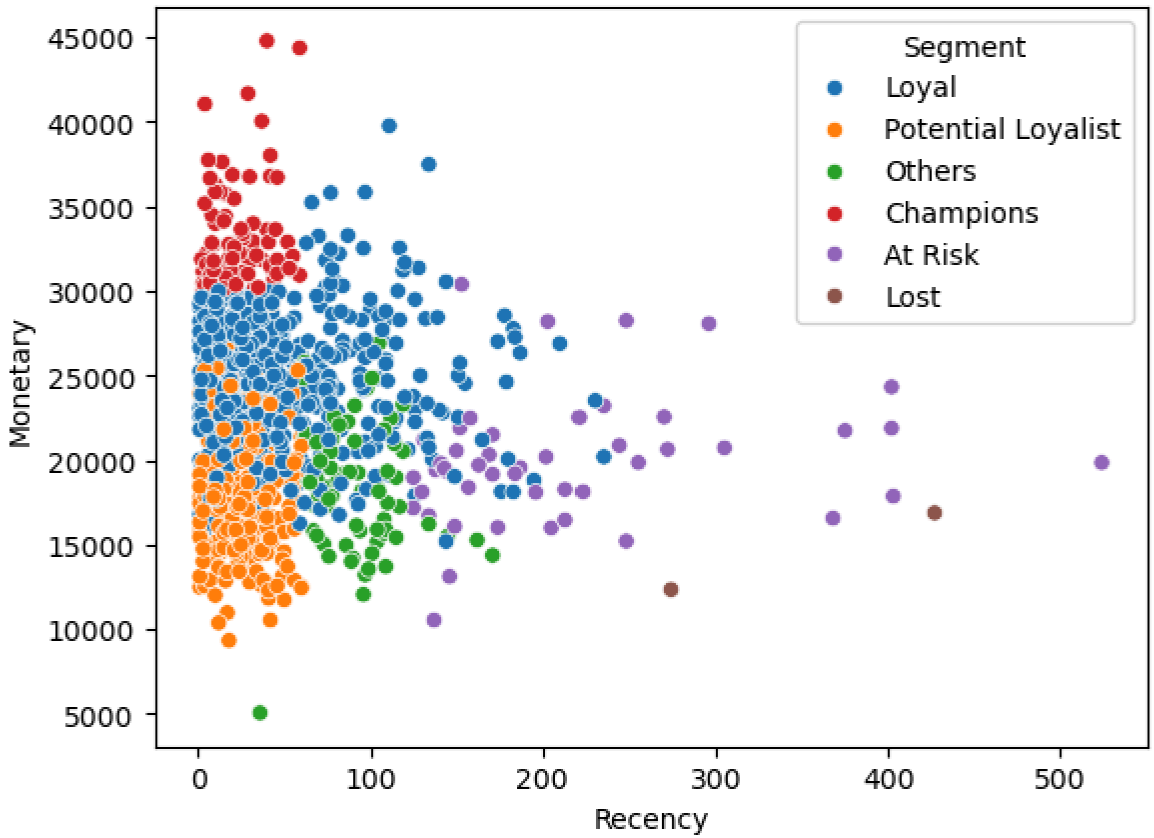
Out[71]:
```
Segment
Loyal                 542
Potential Loyalist    248
Champions              84
Others                 77
At Risk                47
Lost                    2
Name: count, dtype: int64
```

In [72]:
```
#Visualization
#Revenue contribution per segment
Rev_per_Seg = rfm.groupby('Segment')['Monetary'].sum()
plt.figure(figsize=(6,6))
plt.pie(Rev_per_Seg , labels=Rev_per_Seg .index, autopct='%1.2f%%', startangle=90)
plt.title("Revenue Contribution per Segment")
plt.show()
```



In [73]:
```
#Recency vs Monetary scatter plot colored by segment
sns.scatterplot(x=rfm["Recency"],y=rfm["Monetary"],hue=rfm["Segment"])
```

Out[73]: `<Axes: xlabel='Recency', ylabel='Monetary'>`



In [74]:
```
#Pareto Analysis
print(rfm["Monetary"].describe())
```

```
count     1000.00000
mean     23053.19966
std       5622.44101
min       5052.69000
25%      18965.46250
50%      22969.82000
75%      26827.39250
max      44784.99000
Name: Monetary, dtype: float64
```

In [76]:
```
#First,we will sort " Monetary" in descending order
rfm_sorted=rfm.sort_values("Monetary",ascending=False)

# We will find "Cumulative Revenue" and then we will sort accordingly.
rfm["CumuRevenue"]=rfm_sorted["Monetary"].cumsum()/rfm_sorted["Monetary"].sum()*100

#We will find how many customers contribute to the first 80% of total revenue.
X80=rfm_sorted.loc[rfm_sorted["CumuRevenue"]>=80,"CustomerID"].index[0]+1

#then we will find percentage of customers needed to contribute 80% of revenue.
pct_customers=X80/len(rfm_sorted)*100

print(f"\n Top {pct_customers:.2f}% of customers contribute~80% of total Revenue.")
```

```
Top 61.90% of customers contribute~80% of total Revenue.
```

In [ ]: