In [2]:
```
# Import libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from datetime import timedelta

# Load the data

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

In [3]:
```
df_cust.head(20)
```

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
10	CUST10010	Kashvi Goda	nirvi89@hotmail.com	Not Disclosed	73	Ahmedabad	Single	2	2022-04-18
11	CUST10011	Vardaniya Varty	aaryahibala@virk-gopal.org	Not Disclosed	63	Jaipur	Widowed	1	2020-08-06
12	CUST10012	Dishani Jhaveri	zeeshan50@hotmail.com	Not Disclosed	30	Ahmedabad	Divorced	1	2022-12-07
13	CUST10013	Amira Rama	myrachakrabarti@master.com	Female	72	Jaipur	Widowed	1	2022-11-15
14	CUST10014	Arhaan Trivedi	vigadvik@amble-shroff.com	Male	48	Hyderabad	Widowed	1	2023-12-21
15	CUST10015	Faiyaz Sur	riaan12@datta.com	Male	41	Jaipur	Divorced	0	2021-08-20
16	CUST10016	Neelofar Sawhney	usarna@yahoo.com	Not Disclosed	50	Delhi	Married	1	2023-05-15

	CustomerID	Name	Email	Gender	Age	City	MaritalStatus	NumChildren	JoinDate
17	CUST10017	Anay Dugal	aaina34@gmail.com	Female	41	Pune	Single	1	2022-08-03
18	CUST10018	Mahika Kalita	advikissac@badami.com	Female	63	Lucknow	Widowed	2	2021-08-26
19	CUST10019	Inaaya Thaman	hchoudhury@aurora.org	Male	50	Hyderabad	Single	2	2022-06-21

In [4]:
```
df_txn.head(20)
```

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
10	CUST10897	7/16/23	726.91
11	CUST10300	5/9/23	385.41
12	CUST10710	4/4/25	305.66
13	CUST10480	2/4/25	1812.17
14	CUST10480	1/16/24	1560.58
15	CUST10377	10/24/23	465.32
16	CUST10184	2/19/25	672.10
17	CUST10214	6/17/25	1224.70
18	CUST10488	7/12/23	463.81
19	CUST10892	8/13/24	398.16

In [5]:
```
missing_values_customers = df_cust.isnull().sum()

missing_values_txn = df_txn.isnull().sum()

print("Missing Values in Customer dataset:")

print(missing_values_customers)

print("Missing Values in Transaction dataset:")

print(missing_values_txn)
```

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

In [6]:
```
# To validate the give data type are in required data type .
# We need to ensure that Join Date column should be in datetime format

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 [7]:
```
# Converting all the date columns in datetime format

df_cust["JoinDate"] = pd.to_datetime(df_cust["JoinDate"])
df_txn["TransactionDate"]= pd.to_datetime(df_txn["TransactionDate"])
```

```
/var/folders/xg/rx6_m00j4mv43d00pbdvg04w0000gn/T/ipykernel_13261/567917924.py:4: UserWarning: Could not in
fer format, so each element will be parsed 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 [8]:
```
print(f" Total Customer records in the master data:", df_cust.shape[0])
print(f" Total Transaction data:", df_txn.shape[0])
```

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

In [9]:
```
# Keep only valid transactions

# We keep transactions whose CustomerID exists in the master customer data


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

In [10]:
```
# 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 [11]:
```
# for each customer :

# - LastTxnDate = most recent purchase date
# - Frequency = number of rows (purchases)
# - Monetary = sum of TransactionAmount

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

In [12]:
```
rfm.head(25)
```

Out[12]:

	CustomerID	LastTxnDate	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
15	CUST10015	2025-07-24	20	26315.71
16	CUST10016	2025-07-26	24	24607.24
17	CUST10017	2025-06-18	25	21241.48
18	CUST10018	2025-05-19	19	20383.92
19	CUST10019	2025-02-25	25	24529.64
20	CUST10020	2025-07-26	22	19111.42
21	CUST10021	2025-07-11	20	18806.90
22	CUST10022	2025-05-31	15	21368.65
23	CUST10023	2025-05-12	27	30622.22
24	CUST10024	2025-07-22	24	25362.43

In [13]:
```
# Recency (in days) = difference from ref_date

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

rfm.head(25)
```

Out[13]:

	CustomerID	LastTxnDate	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
15	CUST10015	2025-07-24	20	26315.71	6
16	CUST10016	2025-07-26	24	24607.24	4
17	CUST10017	2025-06-18	25	21241.48	42
18	CUST10018	2025-05-19	19	20383.92	72
19	CUST10019	2025-02-25	25	24529.64	155
20	CUST10020	2025-07-26	22	19111.42	4
21	CUST10021	2025-07-11	20	18806.90	19
22	CUST10022	2025-05-31	15	21368.65	60
23	CUST10023	2025-05-12	27	30622.22	79
24	CUST10024	2025-07-22	24	25362.43	8

In [14]:
```
# Put the columns in desired sequence

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

rfm.head(25)
```

Out[14]:

	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
15	CUST10015	6	20	26315.71
16	CUST10016	4	24	24607.24
17	CUST10017	42	25	21241.48
18	CUST10018	72	19	20383.92
19	CUST10019	155	25	24529.64
20	CUST10020	4	22	19111.42
21	CUST10021	19	20	18806.90
22	CUST10022	60	15	21368.65
23	CUST10023	79	27	30622.22
24	CUST10024	8	24	25362.43

In [15]:
```
# 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]  # revered because lower recency is better

rfm["R_Score"]= pd.cut(rfm["Recency"],bins=r_bins,labels=r_labels, include_lowest=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).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).astype(int)
```

```
rfm[ 'M_Score' ] = pd.cut(rfm[ 'Monetary' ],bins=m_bins,labels=m_labels,include_lowest=True).astype(int)
```

In [16]:
```
rfm.head(25)
```

Out[16]:

	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
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 [17]:
```
# RFM score string


rfm["RFM_Score"]= (rfm["R_Score"].astype(str) + rfm["F_Score"].astype(str) + rfm["M_Score"].astype(str))

rfm.head(25)
```

Out[17]:

	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
20	CUST10020	4	22	19111.42	5	4	2	542
21	CUST10021	19	20	18806.90	5	3	2	532
22	CUST10022	60	15	21368.65	4	3	3	433
23	CUST10023	79	27	30622.22	3	4	4	344
24	CUST10024	8	24	25362.43	5	4	3	543

In [18]:
```
# Naming the segments

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 (m<=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 [22]:
```
rfm.head(30)
```

Out[22]:

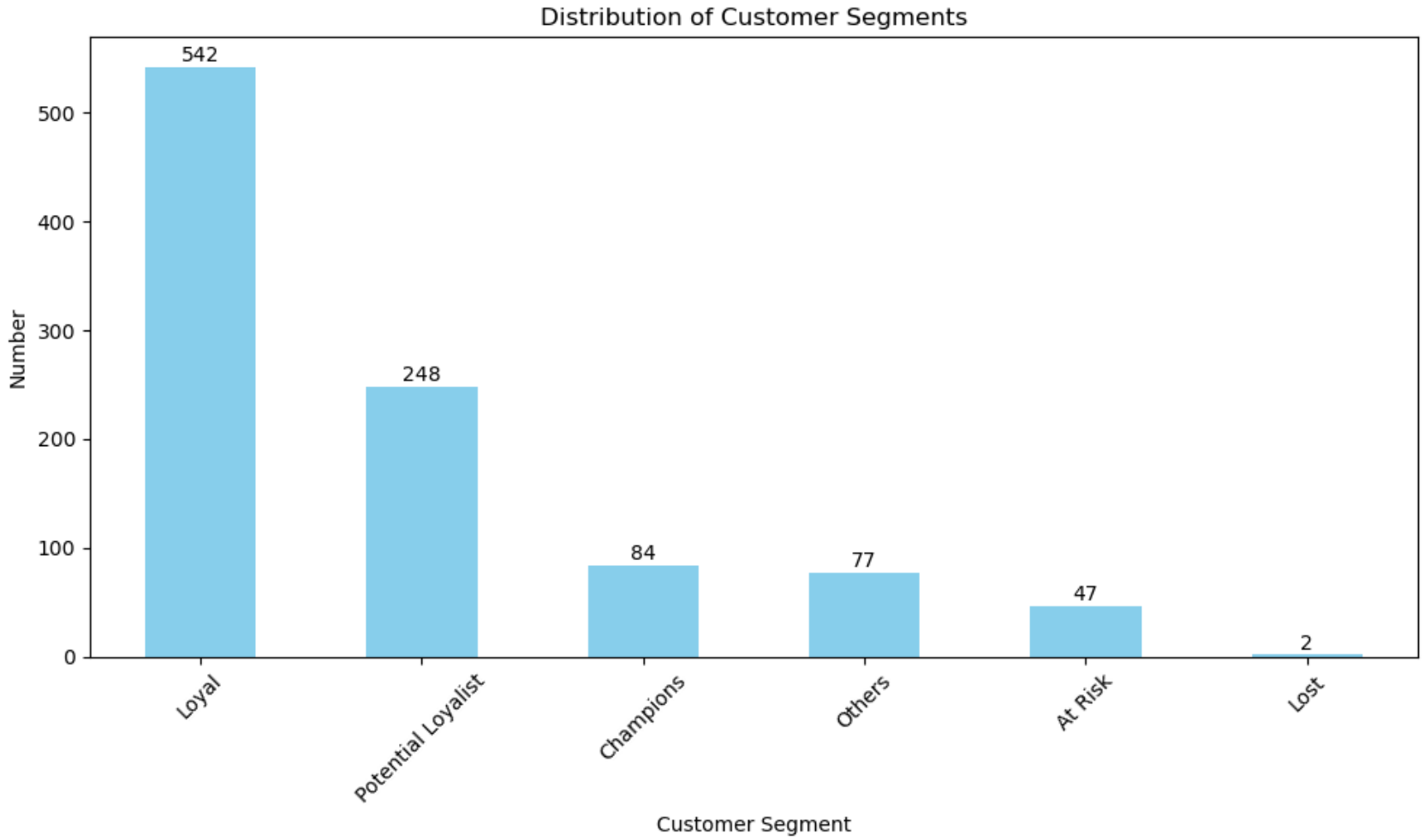
	CustomerID	Recency	Frequency	Monetary	R_Score	F_Score	M_Score	RFM_Score	Segment
0	CUST10000	13	23	21265.49	5	4	3	543	Loyal
1	CUST10001	35	30	28654.31	4	5	3	453	Loyal
2	CUST10002	18	24	23884.03	5	4	3	543	Loyal
3	CUST10003	81	25	24206.03	3	4	3	343	Loyal
4	CUST10004	8	19	25565.30	5	3	3	533	Potential Loyalist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

	CustomerID	Recency	Frequency	Monetary	R_Score	F_Score	M_Score	RFM_Score	Segment
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 [35]:
```
# Simple Business Analysis

print("\n ============= SIMPLE BUSINESS ANALYSIS ==============")

print("\n ==========================")
print("\n Overall Business Summary")
print("\n ==========================")

total_customers = rfm["CustomerID"].shape[0]

total_revenue = rfm["Monetary"].sum()

avg_revenue_per_customer = rfm["Monetary"].mean()


print(f" Total Customers : {total_customers}")
print(f" Total Revenue : Rs. {total_revenue }")
print(f" Average Revenue per Customer: Rs. {avg_revenue_per_customer}")

print("\n ==========================")
print("\n Segment Wise Summary")
print("\n ==========================")

# Customers in each Segment

segment_customers = rfm["Segment"].value_counts()

print(f" Number of Customers in each Segment:")
print(f"\n {segment_customers}")


# Revenue from each Segment

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

print(f"\n Revenue Contribution by each segment:\n")
print(f"\n {segment_revenue}")
```

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

 ============================

 Overall Business Summary

 ============================
 Total Customers : 1000
 Total Revenue : Rs. 23053199.66
 Average Revenue per Customer: Rs. 23053.19966

 ============================

 Segment Wise Summary

 ============================
 Number of Categories 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
0             At Risk    938539.38
1           Champions   2791699.03
2                Lost     29221.39
3               Loyal  13381926.89
4              Others   1416376.98
5  Potential Loyalist   4495435.99
```

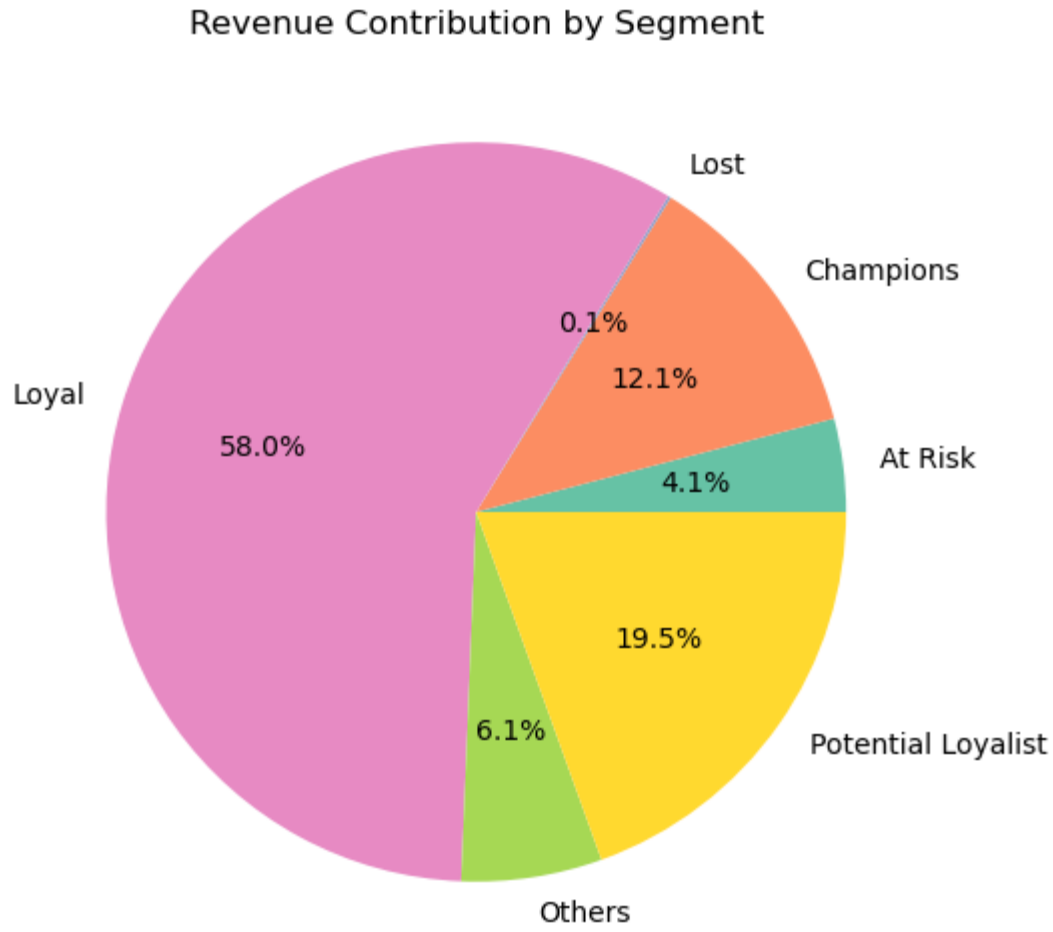
In [57]:
```
segment_customers = rfm["Segment"].value_counts().reset_index()

segment_customers.columns=["Segment","CustomerCount"]

#Bar Chart

segment_counts = rfm["Segment"].value_counts()

# Create a bar plot
plt.figure(figsize=(10, 6))
segment_counts.plot(kind='bar', color='skyblue')

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

# Add value labels on top of each bar
for i, count in enumerate(segment_counts):
    plt.text(i, count + 5, str(count), ha='center')

# Improve layout
plt.xticks(rotation=45)
plt.tight_layout()

# Show the plot
plt.show()


# Customers in each Segment

segment_customers = rfm["Segment"].value_counts()

print(f" Number of Customers in each Segment:")
print(f"\n {segment_customers}")
```




```
 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
```

In [61]:
```
segment_revenue = rfm.groupby("Segment")["Monetary"].sum().reset_index()

print(f"\n Revenue Contribution by each segment:\n")
print(f"\n {segment_revenue}")

# Pie Chart
plt.figure(figsize=(6,6))
plt.pie(segment_revenue["Monetary"],labels=segment_revenue["Segment"],autopct="%1.1f%%", colors=sns.color_
plt.title("Revenue Contribution by Segment")
plt.show()
```

```
 Revenue Contribution by each segment:


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


Revenue Contribution by Segment
Lost
Champions
0.1%
12.1%
Loyal
58.0%
At Risk
4.1%
19.5%
6.1%
Potential Loyalist
Others