Project Overview: Event Planning and Optimisation for Housing Society

Description

This project focuses on organizing a community event by leveraging **demographic data analysis**, **Event Planning Insights**, **Catering Service Analysis** and **Recommendations** . It involves a comprehensive approach to understanding the community's composition, preferences, and requirements through data manipulation, cleaning, and visualization techniques. The primary objective is to ensure the event caters effectively to the residents, optimizes participation, and efficiently allocates the budget by analyzing residents' data and catering service options.

Goals

1. **Data Understanding and Preparation**: The initial phase involves loading, exploring, and cleaning the residents' demographic data and caterer information to ensure quality and readiness for analysis.
2. **Demographic Analysis**: By analyzing the demographic data, we aim to uncover insights related to the residents' origins, the distribution of flat areas, and ownership status, which are crucial for tailored event planning.
3. **Event Planning Insights**: This part of the project focuses on deriving actionable insights for event planning, such as estimating participation rates, understanding the budget constraints based on donations, and optimizing event resources to ensure maximum engagement.
4. **Catering Service Analysis**: An integral part of the event is the food. This section evaluates catering services based on their ratings and cost to select the best options that balance quality and affordability.
5. **Recommendation System**: Based on the analyses, the project aims to provide a set of recommendations for budget allocation, resident engagement strategies, and caterer selection to ensure the success of the community event.

Methodology

- **Data Loading and Cleaning**: Import demographic and caterer data, followed by cleaning operations such as handling missing values, correcting data types, and removing unnecessary columns to prepare the datasets for analysis.
- **Data Analysis and Visualization**: Use statistical and visualization techniques to analyze the data. Techniques include plotting distributions, calculating participation percentages, and analyzing caterer ratings and costs.
- **Insight Generation**: Generate insights from the analyses to guide the event planning process. Insights include understanding demographic distributions, predicting event participation, and evaluating caterer options.
- **Recommendation Generation**: Based on the generated insights, provide actionable recommendations for various aspects of event planning, such as budgeting, engagement strategies, and caterer selection.

Tools and Libraries Used

- **Pandas**: For data manipulation and analysis.
- **NumPy**: For numerical operations.
- **Matplotlib** and **Seaborn**: For data visualization.
- **Python**: The primary programming language used for scripting and analysis.

Conclusion

This project encapsulates the end-to-end process of planning a community event by analyzing demographic data and catering options. It demonstrates the power of data analysis in making informed decisions that cater to the community's needs and preferences, ensuring the event's success. Through meticulous data preparation, insightful analysis, and strategic recommendations, this project serves as a blueprint for effective community event planning.

==========

Task 1: Data Loading and Overview

In [1]:
```
# Import necessary libraries

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

In [2]:
```
# Ignore any warnings

warnings.filterwarnings('ignore')
```

In [3]:
```
# Load the dataset

final_df = pd.read_csv("Final_Data - Final_Data.csv.csv")
```

In [4]:
```
# Display the first few rows of the dataframe

print(f" Apartment Ownership Details Dataset \n")

final_df.head()
```

```
 Apartment Ownership Details Dataset
```

Out[4]:

	Unnamed: 0	Flat no	Wing	Owner Name	Owner's Spouse Name	No of Resident	Confirmed Members	Origin of Owner	Flat Area (sq.mt)	No of Room	Tenant or owner	Maintenance Amt	Availlability of owner	Flat Vacancy
0	0	101	A	Omkar	-	4	4	Maharashtra	500	2	Owner	6000	Yes	Owned
1	1	102	A	Bhavana	-	5	6	Bangalore	500	2	Owner	6000	Yes	Owned
2	2	103	A	Govind	-	5	3	Rajasthan	500	2	Owner	6000	Yes	Owned
3	3	201	A	Reena	-	4	2	Madhya Pradesh	550	2	Tenant	6000	No	Owned
4	4	202	A	Karishma	-	7	15	Gujurat	600	3	Owner	7000	No	Owned

In [5]:
```
# Display a concise summary of the dataframe

final_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60 entries, 0 to 59
Data columns (total 15 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Unnamed: 0              60 non-null     int64  
 1   Flat no                 60 non-null     int64  
 2   Wing                    60 non-null     object 
 3   Owner Name              60 non-null     object 
 4   Owner's Spouse Name     18 non-null     object 
 5   No of Resident          60 non-null     object 
 6   Confirmed Members       60 non-null     object 
 7   Origin of Owner         60 non-null     object 
 8   Flat Area (sq.mt)       60 non-null     int64  
 9   No of Room              60 non-null     int64  
 10  Tenant or owner         60 non-null     object 
 11  Maintenance Amt         32 non-null     object 
 12  Availlability of owner  60 non-null     object 
 13  Flat Vacancy            43 non-null     object 
 14  Donation                55 non-null     float64
dtypes: float64(1), int64(4), object(10)
memory usage: 7.2+ KB
```

In [6]:
```
# Display basic statistical details like percentile, mean, std etc. of a data frame

final_df.describe()
```

Out[6]:

	Unnamed: 0	Flat no	Flat Area (sq.mt)	No of Room	Donation
count	60.000000	60.000000	60.000000	60.000000	55.000000
mean	29.500000	552.000000	554.166667	2.416667	4136.363636
std	17.464249	289.653212	43.464057	0.497167	3228.138073
min	0.000000	101.000000	500.000000	2.000000	1000.000000
25%	14.750000	302.000000	500.000000	2.000000	1500.000000
50%	29.500000	552.000000	550.000000	2.000000	2500.000000
75%	44.250000	802.000000	600.000000	3.000000	6000.000000
max	59.000000	1003.000000	600.000000	3.000000	10000.000000

In [7]:
```
# Display the number of unique values in each column

print(final_df.nunique())
```

```
Unnamed: 0                60
Flat no                   30
Wing                       2
Owner Name                51
Owner's Spouse Name       12
No of Resident             9
Confirmed Members         14
Origin of Owner           17
Flat Area (sq.mt)          3
No of Room                 2
Tenant or owner            4
Maintenance Amt            4
Availability of owner      3
Flat Vacancy               3
Donation                   6
dtype: int64
```

In [8]:
```
# Display the columns of the dataframe

final_df.columns
```

Out[8]:
```
Index(['Unnamed: 0', 'Flat no', 'Wing', 'Owner Name', 'Owner's Spouse Name',
       'No of Resident', 'Confirmed Members', 'Origin of Owner',
       'Flat Area (sq.mt)', 'No of Room', 'Tenant or owner', 'Maintenance Amt',
       'Availability of owner', 'Flat Vacancy', 'Donation'],
      dtype='object')
```

In [9]:
```
# Display unique values in each column

for i in final_df.columns:
    print(f"\n Unique values in {i}: {final_df[i].unique()}")
```

```
Unique values in Unnamed: 0: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55 56 57 58 59]

Unique values in Flat no: [ 101  102  103  201  202  203  301  302  303  401  402  403  501  502
 503  601  602  603  701  702  703  801  802  803  901  902  903 1001
1002 1003]

Unique values in Wing: ['A' 'B']

Unique values in Owner Name: ['Omkar' 'Bhavana' 'Govind' 'Reena' 'Karishma' 'Ragesh' 'Harshad' 'Abdul'
'Lohit' 'Vijay' 'Praveen' 'Arjun' 'Priya' 'Ravi' 'Deepak' 'Nandini'
'Nisha' 'Sunil' 'Priyanka' 'Rahul' 'Anjali' 'Aman' 'alok' 'Pooja' 'Anil'
'Simran' 'mayuri' 'Naveen' 'Anju' 'Vivek' 'tanmay' 'Rohit' 'Ankita'
'Suresh' 'omi' 'Karan' 'Neha' 'Sameer' 'shilpa' 'Romil' 'Sachin' 'Amit'
'abdul' 'Gaurav' 'Ashish' 'Lokesh' 'Mahesh' 'yash' 'rasika' 'Shivam'
'Vikas']

Unique values in Owner's Spouse Name: ['-' 'Parul' nan 'Shashi' 'Sneha' 'Nisha' 'Raghav' 'Vimesh' 'Chanda'
'Deeksha' 'Ekta' 'Anjali' 'Suman']

Unique values in No of Resident: ['4' '5' '7' '8' '-' '3' '6' '2' '1']

Unique values in Confirmed Members: ['4' '6' '3' '2' '15' '8' '20' '-' '1' '11' '9' '0' '7' '5']

Unique values in Origin of Owner: ['Maharashtra' 'Bangalore' 'Rajasthan' 'Madhya Pradesh' 'Gujurat' 'Kerala'
'Gujarat' 'Tamil Nadu' 'Karnataka' 'Uttar Pradesh' 'Delhi' 'Haryana'
'Punjab' 'Andhra Pradesh' '-' 'Kashmir' 'Assam']

Unique values in Flat Area (sq.mt): [500 550 600]

Unique values in No of Room: [2 3]

 Unique values in Tenant or owner: ['Owner' 'Tenant' 'tenant' 'owner']

 Unique values in Maintenance Amt: ['6000' '7000' '5500' '-' nan]

 Unique values in Availability of owner: ['Yes' 'No' '-']

 Unique values in Flat Vacancy: ['Owned' 'Vacant' '-' nan]

 Unique values in Donation: [ 2500. 10000.  1500.    nan  5000.  1000.  6000.]
```

In [10]:
```
# Check for missing values

print(final_df.isnull().sum())
```

```
Unnamed: 0                0
Flat no                   0
Wing                      0
Owner Name                0
Owner's Spouse Name      42
No of Resident            0
Confirmed Members         0
Origin of Owner           0
Flat Area (sq.mt)         0
No of Room                0
Tenant or owner           0
Maintenance Amt          28
Availability of owner     0
Flat Vacancy             17
Donation                  5
dtype: int64
```

In [11]:
```
# Check for duplicate rows

print(final_df.duplicated().sum())
```

```
0
```

Task 2: Data Cleaning

In [12]:
```
# Replace "-" with NaN

final_df.replace({"-": np.nan}, inplace=True)
```

In [13]:
```
# Drop unnecessary columns

final_df.drop(["Unnamed: 0", "Owner's Spouse Name"], axis=1, inplace=True)
```

In [14]:
```
# Fill missing values with median for numerical columns and mode for categorical columns

numerical_columns = ["Donation", "No of Resident", "Confirmed Members", "Maintenance Amt"]
for column in numerical_columns:
    final_df[column] = final_df[column].fillna(final_df[column].median())

final_df["Availability of owner"] = final_df["Availability of owner"].fillna(final_df["Availability of owner"].mode(

final_df["Origin of Owner"] = final_df["Origin of Owner"].fillna(final_df["Origin of Owner"].mode()[0])
```

In [15]:
```
# Logic for filling "Flat Vacancy" based on "Availability of owner"

final_df.loc[final_df["Availability of owner"] == "Yes", "Flat Vacancy"] = final_df["Flat Vacancy"].fillna("Owned")

final_df.loc[final_df["Availability of owner"] == "No", "Flat Vacancy"] = final_df["Flat Vacancy"].fillna("Vacant")
```

In [16]:
```
# Convert data types

final_df["No of Resident"] = final_df["No of Resident"].astype("int64")

final_df["Confirmed Members"] = final_df["Confirmed Members"].astype("int64")

final_df["Maintenance Amt"] = final_df["Maintenance Amt"].astype("float64")
```

In [17]:
```
# Fill missing "Maintenance Amt" with the mean

final_df["Maintenance Amt"] = final_df["Maintenance Amt"].fillna(final_df["Maintenance Amt"].mean())
```

In [18]:
```
# Display the info to confirm changes

final_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60 entries, 0 to 59
Data columns (total 13 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Flat no                 60 non-null     int64
 1   Wing                    60 non-null     object
 2   Owner Name              60 non-null     object
 3   No of Resident          60 non-null     int64
 4   Confirmed Members       60 non-null     int64
 5   Origin of Owner         60 non-null     object
 6   Flat Area (sq.mt)       60 non-null     int64
 7   No of Room              60 non-null     int64
 8   Tenant or owner         60 non-null     object
 9   Maintenance Amt         60 non-null     float64
 10  Availability of owner   60 non-null     object
 11  Flat Vacancy            60 non-null     object
 12  Donation                60 non-null     float64
dtypes: float64(2), int64(5), object(6)
memory usage: 6.2+ KB
```

Task 3: Data Preparation and Preprocessing

In [19]:
```
# Calculate the number of outsiders

final_df["Outsiders"] = final_df["Confirmed Members"] - final_df["No of Resident"]
final_df["Outsiders"] = final_df["Outsiders"].apply(lambda x: x if x > 0 else 0)
```

In [20]:
```
# Convert string columns to lowercase for consistency

string_columns = ["Origin of Owner", "Tenant or owner", "Availability of owner", "Flat Vacancy"]
for column in string_columns:
    final_df[column] = final_df[column].str.lower()
```

In [21]:
```
# Format the "Owner Name" column to title case

final_df['Owner Name'] = final_df['Owner Name'].str.title()
```

In [22]:
```
# Display the modified dataframe

final_df
```

Out[22]:

	Flat no	Wing	Owner Name	No of Resident	Confirmed Members	Origin of Owner	Flat Area (sq.mt)	No of Room	Tenant or owner	Maintenance Amt	Availlability of owner	Flat Vacancy	Donation	Outside
0	101	A	Omkar	4	4	maharashtra	500	2	owner	6000.0	yes	owned	2500.0	
1	102	A	Bhavana	5	6	bangalore	500	2	owner	6000.0	yes	owned	10000.0	
2	103	A	Govind	5	3	rajasthan	500	2	owner	6000.0	yes	owned	1500.0	
3	201	A	Reena	4	2	madhya pradesh	550	2	tenant	6000.0	no	owned	2500.0	
4	202	A	Karishma	7	15	gujurat	600	3	owner	7000.0	no	owned	2500.0	
5	203	A	Ragesh	4	4	kerala	500	2	tenant	5500.0	no	owned	5000.0	
6	301	A	Harshad	8	8	gujurat	550	2	owner	6000.0	yes	owned	5000.0	
7	302	A	Abdul	4	4	maharashtra	600	3	owner	7000.0	yes	vacant	2500.0	
8	303	A	Lohit	5	2	bangalore	500	2	owner	7000.0	no	vacant	1000.0	
9	401	A	Vijay	5	3	rajasthan	550	2	tenant	7000.0	no	vacant	10000.0	
10	402	A	Bhavana	4	2	madhya pradesh	600	3	tenant	7000.0	no	vacant	2500.0	
11	403	A	Govind	7	20	gujarat	500	2	owner	7000.0	no	vacant	6000.0	
12	501	A	Reena	4	4	kerala	550	2	owner	7000.0	yes	owned	2500.0	
13	502	A	Karishma	8	8	gujarat	600	3	tenant	7000.0	yes	owned	2500.0	
14	503	A	Ragesh	4	3	maharashtra	600	3	owner	7000.0	yes	vacant	10000.0	
15	601	A	Harshad	3	2	tamil nadu	600	3	owner	7000.0	no	vacant	1500.0	
16	602	A	Praveen	6	4	karnataka	600	3	owner	7000.0	no	vacant	2500.0	
17	603	A	Arjun	4	2	maharashtra	600	3	tenant	7000.0	no	vacant	1500.0	
18	701	A	Priya	5	4	uttar pradesh	550	2	owner	7000.0	no	vacant	1000.0	
19	702	A	Ravi	3	2	karnataka	550	2	owner	7000.0	yes	owned	5000.0	
20	703	A	Deepak	4	3	maharashtra	500	2	owner	7000.0	no	vacant	2500.0	
21	801	A	Nandini	4	3	delhi	500	2	tenant	7000.0	yes	vacant	1500.0	
22	802	A	Nisha	5	4	haryana	500	2	owner	7000.0	no	owned	1500.0	
23	803	A	Sunil	4	3	maharashtra	500	2	owner	7000.0	no	owned	2500.0	
24	901	A	Priyanka	3	2	rajasthan	550	2	tenant	7000.0	no	vacant	6000.0	
25	902	A	Rahul	4	3	maharashtra	600	3	owner	7000.0	no	owned	10000.0	
26	903	A	Anjali	5	3	uttar pradesh	600	3	tenant	7000.0	yes	owned	10000.0	
27	1001	A	Aman	4	2	madhya pradesh	600	3	owner	7000.0	yes	owned	5000.0	
28	1002	A	Alok	3	1	maharashtra	600	3	tenant	7000.0	yes	owned	1000.0	
29	1003	A	Pooja	2	2	punjab	600	3	tenant	7000.0	no	vacant	1500.0	
30	101	B	Anil	2	2	punjab	550	2	owner	7000.0	no	owned	6000.0	
31	102	B	Simran	5	11	andhra pradesh	600	3	owner	7000.0	yes	vacant	5000.0	
32	103	B	Mayuri	4	3	kerala	500	2	owner	7000.0	yes	owned	2500.0	
33	201	B	Naveen	3	2	tamil nadu	500	2	owner	7000.0	yes	owned	1500.0	
34	202	B	Anju	2	1	delhi	500	2	tenant	7000.0	yes	owned	2500.0	
35	203	B	Vivek	3	3	maharashtra	500	2	tenant	7000.0	yes	owned	1000.0	
36	301	B	Priyanka	4	3	haryana	550	2	owner	7000.0	no	vacant	1500.0	
37	302	B	Tanmay	5	4	maharashtra	600	3	owner	7000.0	yes	owned	1000.0	
38	303	B	Rohit	3	2	gujarat	500	2	owner	7000.0	no	vacant	2500.0	
39	401	B	Ankita	2	2	rajasthan	550	2	tenant	7000.0	no	vacant	1000.0	
40	402	B	Suresh	5	9	maharashtra	550	2	owner	7000.0	yes	vacant	6000.0	
41	403	B	Pooja	5	4	karnataka	550	2	tenant	7000.0	yes	owned	1500.0	
42	501	B	Omi	4	3	madhya pradesh	550	2	owner	7000.0	yes	vacant	1500.0	
43	502	B	Karan	3	2	tamil nadu	600	3	owner	7000.0	yes	owned	2500.0	
44	503	B	Neha	2	1	kerala	500	2	tenant	7000.0	no	vacant	5000.0	
45	601	B	Sameer	4	3	maharashtra	500	2	owner	7000.0	no	vacant	2500.0	
46	602	B	Shilpa	4	3	haryana	600	3	tenant	7000.0	no	owned	10000.0	
47	603	B	Romil	2	0	maharashtra	500	2	owner	7000.0	yes	owned	6000.0	
48	701	B	Rohit	1	0	kashmir	550	2	tenant	7000.0	no	vacant	2500.0	

	Flat no	Wing	Owner Name	No of Resident	Confirmed Members	Origin of Owner	Flat Area (sq.mt)	No of Room	Tenant or owner	Maintenance Amt	Availlability of owner	Flat Vacancy	Donation	Outside
49	702	B	Sachin	6	4	assam	600	3	tenant	7000.0	no	vacant	1000.0	
50	703	B	Amit	7	7	punjab	500	2	tenant	7000.0	no	vacant	10000.0	
51	801	B	Abdul	5	4	kerala	600	3	owner	7000.0	yes	owned	2500.0	
52	802	B	Gaurav	4	4	tamil nadu	600	3	tenant	7000.0	yes	owned	10000.0	
53	803	B	Ashish	3	5	delhi	600	3	tenant	7000.0	yes	vacant	10000.0	
54	901	B	Lokesh	2	3	maharashtra	550	2	owner	7000.0	no	vacant	1000.0	
55	902	B	Mahesh	3	1	haryana	600	3	owner	7000.0	no	vacant	5000.0	
56	903	B	Yash	4	4	maharashtra	600	3	tenant	7000.0	no	vacant	10000.0	
57	1001	B	Rasika	5	2	kerala	500	2	owner	7000.0	yes	owned	1000.0	
58	1002	B	Shivam	3	2	rajasthan	600	3	tenant	7000.0	yes	owned	6000.0	
59	1003	B	Vikas	3	6	maharashtra	600	3	owner	7000.0	no	owned	2500.0	

Task 4: Importing Caterer Data and Overview

In [23]:
```
# Load the caterer dataset

cat_df = pd.read_csv("Caterer_info - Caterer_info.csv.csv")
```

In [24]:
```
# Display the first few rows of the dataframe

cat_df.head()
```

Out[24]:

	Unnamed: 0	Plate Cost	Decoration Price	Rating	Caterer Name
0	0	280	5000	2.4	a
1	1	420	4300	3.8	b
2	2	390	5900	3.1	c
3	3	420	6900	4.6	d
4	4	280	6500	2.3	e

In [25]:
```
# Display a concise summary of the dataframe

cat_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 5 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Unnamed: 0        20 non-null     int64  
 1   Plate Cost        20 non-null     int64  
 2   Decoration Price  20 non-null     int64  
 3   Rating            20 non-null     float64
 4   Caterer Name      20 non-null     object 
dtypes: float64(1), int64(3), object(1)
memory usage: 932.0+ bytes
```

In [26]:
```
# Check for missing values

cat_df.isnull().sum()
```

Out[26]:
```
Unnamed: 0          0
Plate Cost          0
Decoration Price    0
Rating              0
Caterer Name        0
dtype: int64
```

In [27]:
```
# Display basic statistical details

cat_df.describe(include="all")
```

Out[27]:

	Unnamed: 0	Plate Cost	Decoration Price	Rating	Caterer Name
count	20.00000	20.000000	20.000000	20.000000	20
unique	NaN	NaN	NaN	NaN	20
top	NaN	NaN	NaN	NaN	a
freq	NaN	NaN	NaN	NaN	1
mean	9.50000	374.000000	5430.000000	3.475000	NaN
std	5.91608	74.861275	918.866579	0.741176	NaN
min	0.00000	250.000000	4200.000000	2.300000	NaN
25%	4.75000	317.500000	4775.000000	3.100000	NaN
50%	9.50000	375.000000	5200.000000	3.550000	NaN
75%	14.25000	420.000000	6050.000000	4.000000	NaN
max	19.00000	500.000000	7000.000000	4.800000	NaN

In [28]:
```
# Display the number of unique values in each column

cat_df.nunique()
```

Out[28]:
```
Unnamed: 0          20
Plate Cost          14
Decoration Price    13
Rating              12
Caterer Name        20
dtype: int64
```

In [52]:
```
# Display unique values in each column

for i in cat_df.columns:

    print(f"\n unique values in {i} is: {cat_df[i].unique()}")
```

```
 unique values in Unnamed: 0 is: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]

 unique values in Plate Cost is: [280 420 390 360 250 370 400 380 440 490 320 480 500 310]

 unique values in Decoration Price is: [5000 4300 5900 6900 6500 7000 4200 4800 5800 5600 4700 4900 5400]

 unique values in Rating is: [2.4 3.8 3.1 4.6 2.3 3.3 3.4 4.  4.3 3.2 4.8 3.7]

 unique values in Caterer Name is: ['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'k' 'l' 'm' 'n' 'o' 'p' 'q' 'r'
's' 't']
```

In [30]:
```
# Check for duplicate rows
cat_df.duplicated().sum()
```

Out[30]:
```
0
```

Task 5: Demographic Analysis

The following sections perform various analyses on demographic data, including visualizations to understand the distribution of residents by origin, the distribution of flat areas, and the distribution of flat ownership.

Distribution of Residents by Origin

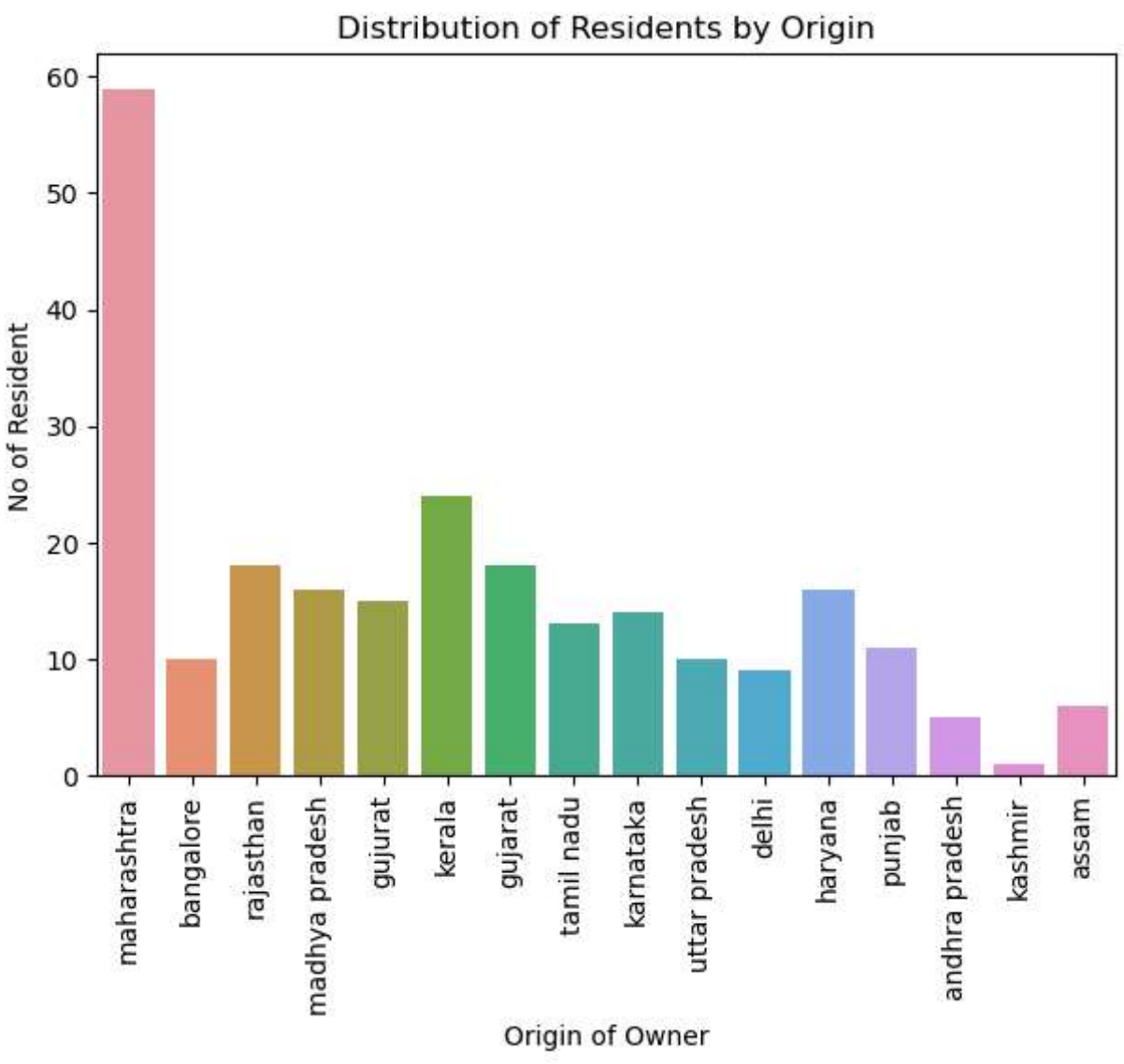
In [31]:
```
# Calculate the average number of residents in all flats.

average_residents = final_df["No of Resident"].mean()
print(f"Average number of residents per flat: {average_residents:.2f}")
```

```
Average number of residents per flat: 4.08
```

In [32]:
```
# Sum the number of residents in flats, grouped by the origin of the owner.
# Display the sum of residents for each owner origin group.

residents_by_owner_origin = final_df.groupby("Origin of Owner")["No of Resident"].sum()

print("Number of residents by owner origin:")
print(residents_by_owner_origin)
```

```
Number of residents by owner origin:
Origin of Owner
andhra pradesh     5
assam              6
bangalore         10
delhi              9
gujarat           18
gujurat           15
haryana           16
karnataka         14
kashmir            1
kerala            24
madhya pradesh    16
maharashtra       59
punjab            11
rajasthan         18
tamil nadu        13
uttar pradesh     10
Name: No of Resident, dtype: int64
```

In [33]:
```python
# Plotting the distribution of residents by origin

plt.figure(figsize=(7,5))
sns.barplot(x=final_df["Origin of Owner"], y=final_df["No of Resident"], ci=None, estimator=np.sum)
plt.xticks(rotation=90)
plt.title("Distribution of Residents by Origin")
plt.show()
```



Distribution of Flat Areas

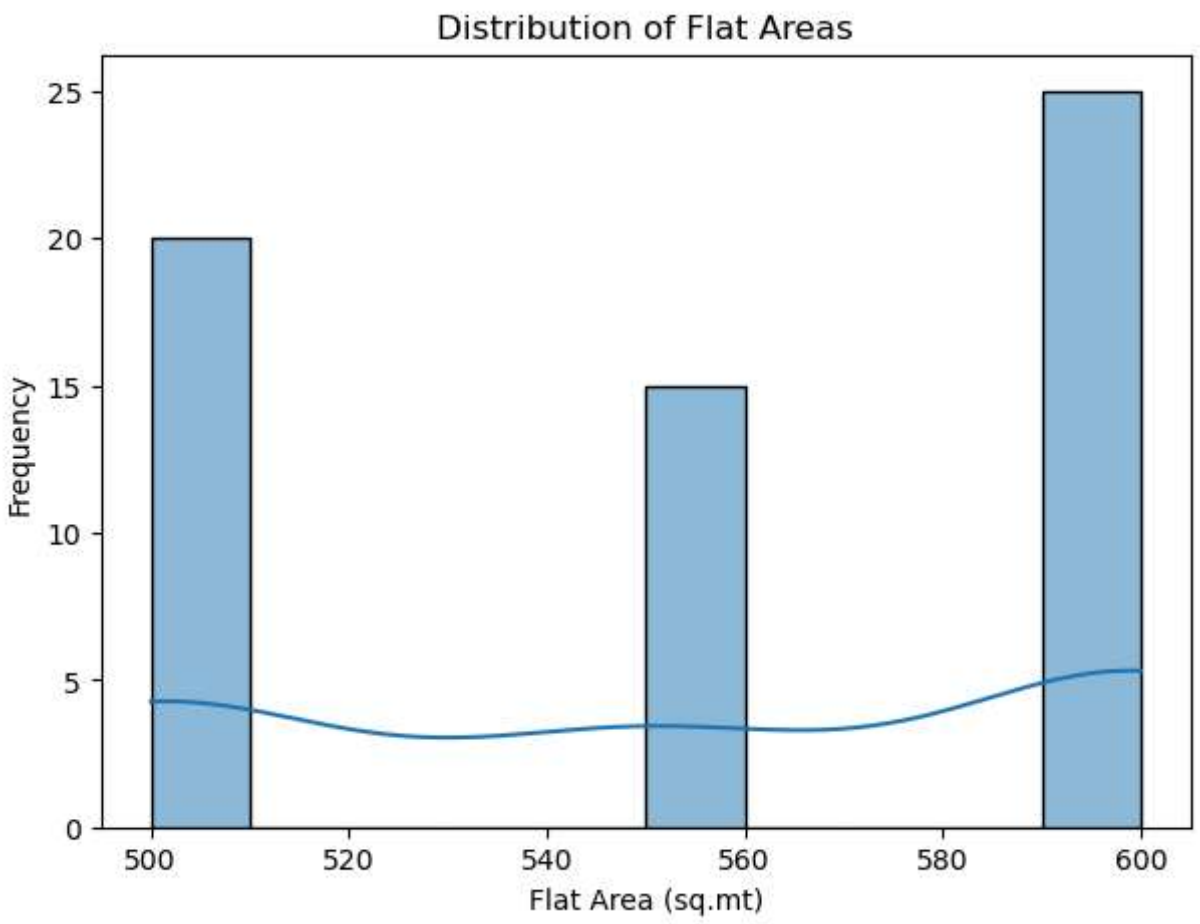
In [34]:
```python
# Calculatethe average area of flats in square meters.

mean_flat_area = final_df["Flat Area (sq.mt)"].mean()
print(f"Average flat area: {mean_flat_area:.2f} sq.mt")
```

```
Average flat area: 554.17 sq.mt
```

In [35]:
```python
# Calculate the most common number of rooms per flat.

most_common_rooms = final_df["No of Room"].mode()[0]
print(f"Most common number of rooms per flat: {most_common_rooms}")
```

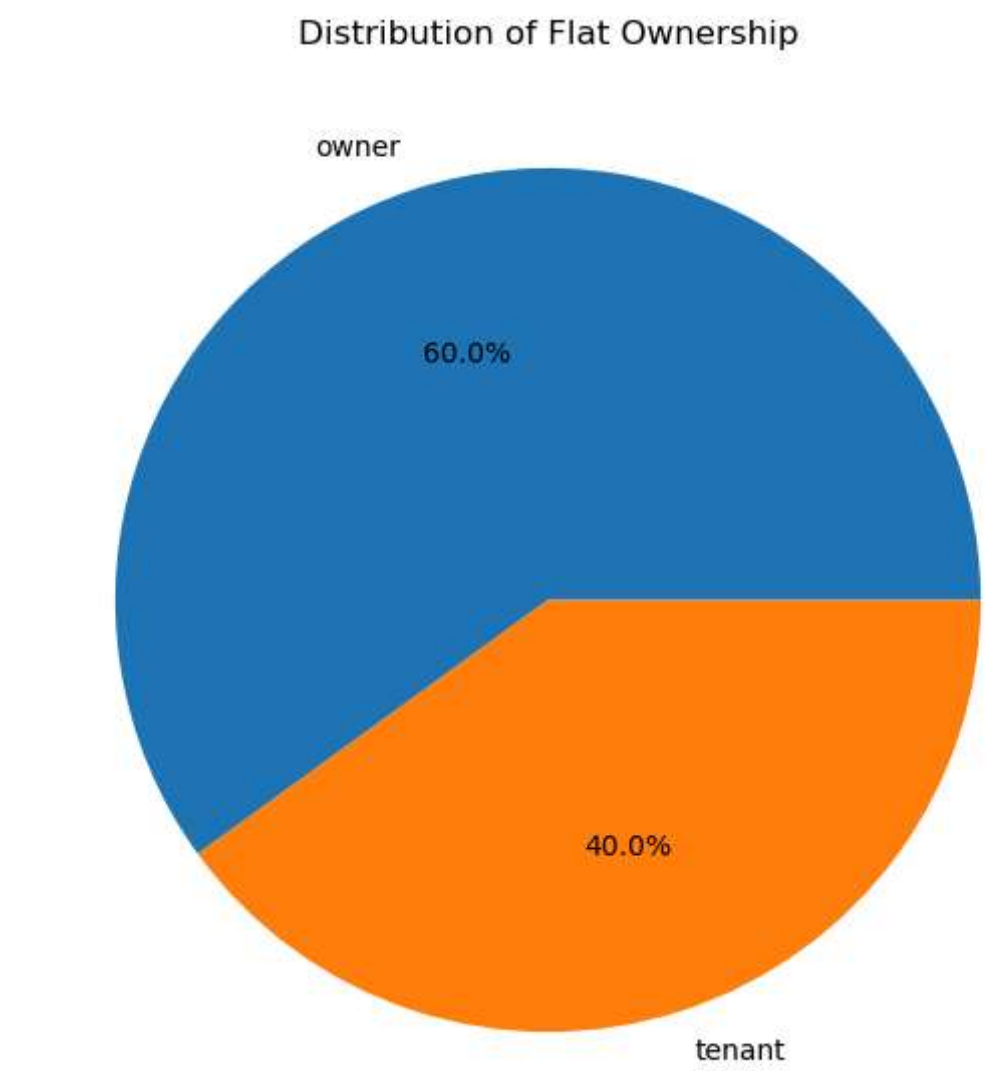
```
Most common number of rooms per flat: 2
```

In [36]:
```python
# Plotting the distribution of flat areas

plt.figure(figsize=(7,5))
sns.histplot(x=final_df['Flat Area (sq.mt)'], bins=10, kde=True)
plt.ylabel("Frequency")
plt.title("Distribution of Flat Areas")
plt.show()
```



Distribution of Flat Ownership

In [37]:
```
# Plotting the distribution of flat ownership

plt.figure(figsize=(7,7))
plt.pie(final_df["Tenant or owner"].value_counts(), autopct="%1.1f%%", labels=final_df["Tenant or owner"].value_counts(
plt.title("Distribution of Flat Ownership")
plt.show()
```



In [38]:
```
# Calculate the percentage of flats owned by their occupants.

num_flats = final_df.shape[0]  # Total number of flats
owners_count = final_df["Tenant or owner"].value_counts()["owner"]
percentage_owners = (owners_count / num_flats) * 100
print(f"Percentage of flat owners: {percentage_owners:.2f}%")
```

```
Percentage of flat owners: 60.00%
```

In [39]:
```
# Calculate the percentage of flats rented by tenants.

tenants_count = final_df["Tenant or owner"].value_counts()["tenant"]
percentage_tenants = (tenants_count / num_flats) * 100
print(f"Percentage of tenants: {percentage_tenants:.2f}%")
```

```
Percentage of tenants: 40.00%
```

In [40]:
```
# Count the number of flats occupied by tenants versus owners.
# Display the counts for tenants and owners.
```

```
tenancy_counts = final_df["Tenant or owner"].value_counts()

print("Counts of flats based on tenancy:")
print(tenancy_counts)
```

```
Counts of flats based on tenancy:
Tenant or owner
owner     36
tenant    24
Name: count, dtype: int64
```

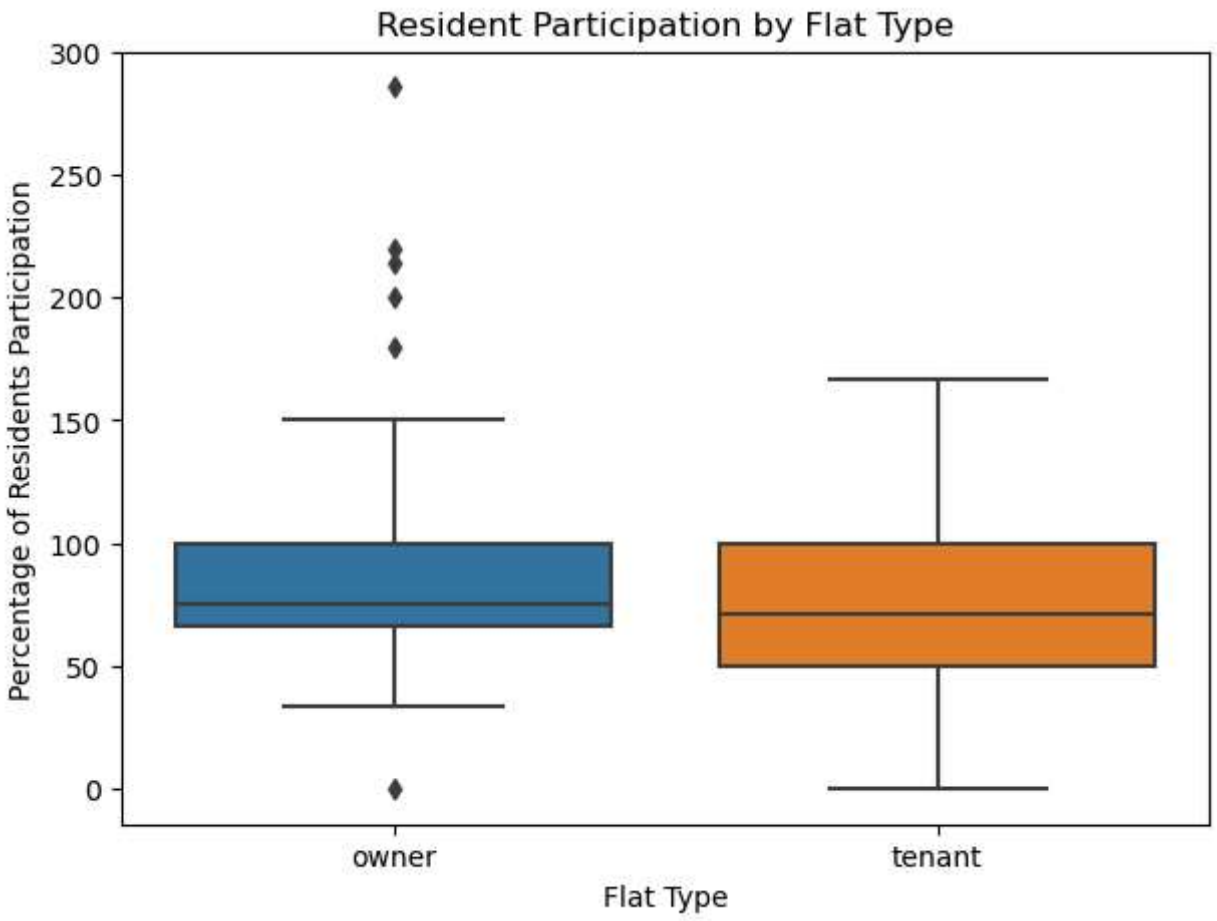
Task 6: Optimizing Event Planning and Resident Engagement

The following code calculates and visualizes the percentage of resident participation based on flat type and gives recommendations based on donation amounts and participation.

In [41]:
```
# Calculate the percentage of resident participation
residents_participation_percentage = (final_df["Confirmed Members"] / final_df["No of Resident"]) * 100

# Formatting and printing each value with two decimals and a percentage sign
formatted_percentage_series = residents_participation_percentage.apply(lambda x: f"{x:.2f}%")

print(formatted_percentage_series)
```

```
0     100.00%
1     120.00%
2      60.00%
3      50.00%
4     214.29%
5     100.00%
6     100.00%
7     100.00%
8      40.00%
9      60.00%
10     50.00%
11    285.71%
12    100.00%
13    100.00%
14     75.00%
15     66.67%
16     66.67%
17     50.00%
18     80.00%
19     66.67%
20     75.00%
21     75.00%
22     80.00%
23     75.00%
24     66.67%
25     75.00%
26     60.00%
27     50.00%
28     33.33%
29    100.00%
30    100.00%
31    220.00%
32     75.00%
33     66.67%
34     50.00%
35    100.00%
36     75.00%
37     80.00%
38     66.67%
39    100.00%
40    180.00%
41     80.00%
42     75.00%
43     66.67%
44     50.00%
45     75.00%
46     75.00%
47      0.00%
48      0.00%
49     66.67%
50    100.00%
51     80.00%
52    100.00%
53    166.67%
54    150.00%
55     33.33%
56    100.00%
57     40.00%
58     66.67%
59    200.00%
dtype: object
```

In [42]:
```
# Plotting the boxplot for percentage of resident participation based on flat type

plt.figure(figsize=(7,5))
sns.boxplot(y=residents_participation_percentage, x=final_df["Tenant or owner"])
plt.ylabel("Percentage of Residents Participation")
plt.xlabel("Flat Type")
plt.title("Resident Participation by Flat Type")
plt.show()
```



In [43]:
```python
# Calculate the average donation amount.

average_donation = final_df["Donation"].mean()
print(f"Average donation amount: {average_donation:.2f}")
```

```
Average donation amount: 4000.00
```

In [44]:
```python
# Calculate the total sum of donations.

total_donation = final_df["Donation"].sum()
print(f"Total donation amount: {total_donation:.2f}")
```

```
Total donation amount: 240000.00
```

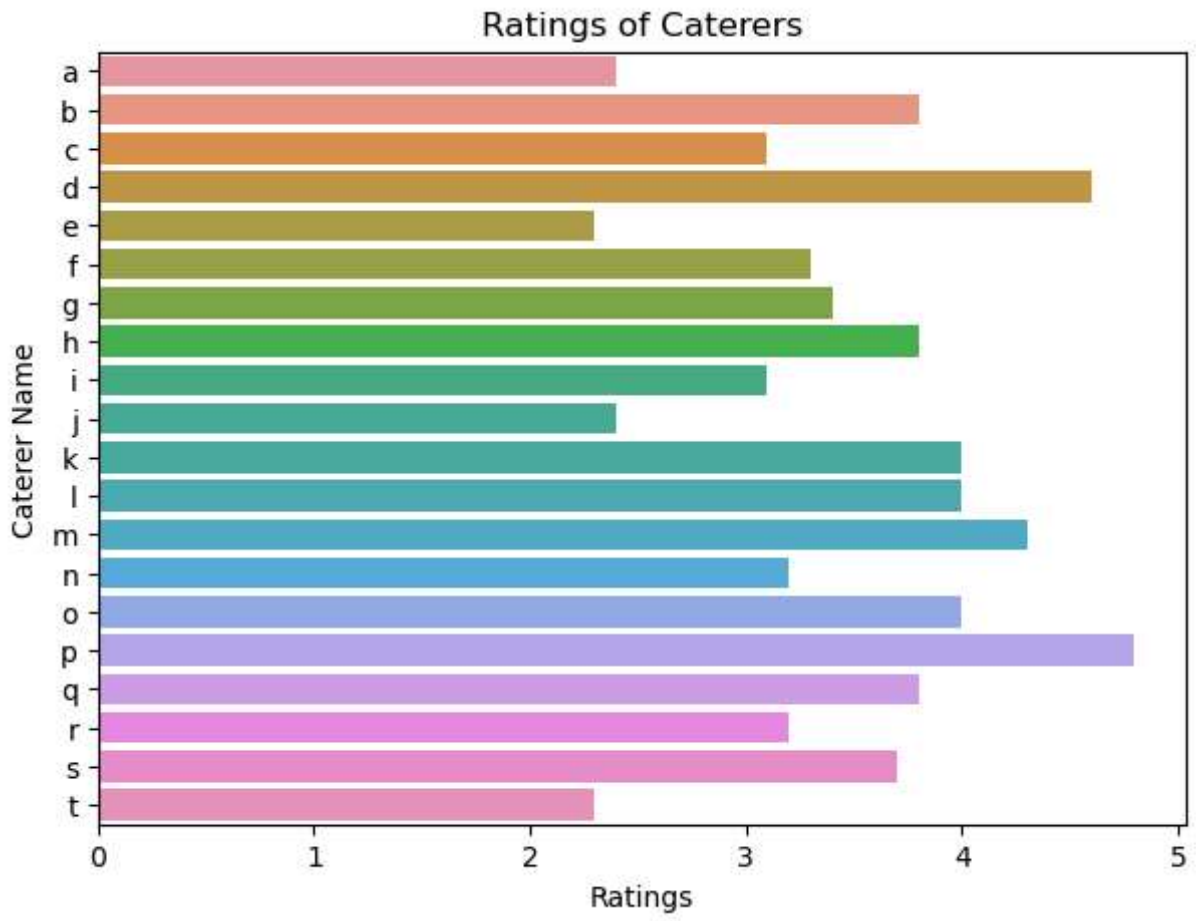
In [45]:
```python
# Recommendations based on donation and participation

print(f"Recommendations based on donation and participation:")
print("-----------------------------------------------------")

# Initialize an empty list to store recommendations.

recommendations = []



# Recommendation based on average donation amount.

if average_donation >= 5000:
    recommendations.append("1. Consider a higher budget for the event")
else:
    recommendations.append("1. Optimize the budget allocation based on donation")



# Assuming 'residents_participation_percentage' is a previously calculated value representing the percentage of resident
# who are expected to participate in the event.
# Recommendation based on residents participation percentage.

if residents_participation_percentage.mean() > 70:
    recommendations.append("2. A good turnout is expected")
else:
    recommendations.append("2. Boost the participation by engaging more")



# Recommendation based on total donation amount.

if total_donation >= 50000:
    recommendations.append("3. Sufficient funds are there")
else:
    recommendations.append("3. Need to collect more funds")



# Display the recommendations.

for recommendation in recommendations:
    print(recommendation)
```

```
Recommendations based on donation and participation:
-----------------------------------------------------
1. Optimize the budget allocation based on donation
2. A good turnout is expected
3. Sufficient funds are there
```

Task 7: Analysis for Catering Services

This section analyzes the catering services' ratings and affordability to assist in selecting an optimal caterer for the event.

In [46]:
```python
# Plotting the ratings of caterers

plt.figure(figsize=(7,5))
sns.barplot(x=cat_df["Rating"], y=cat_df["Caterer Name"], ci=None)
plt.title("Ratings of Caterers")
plt.xlabel("Ratings")
plt.ylabel("Caterer Name")
plt.show()
```



In [47]:
```python
# Find the highest rated decorator(s)

highest_rated_decorator = cat_df[cat_df["Rating"] == cat_df["Rating"].max()]

# If there are multiple highest rated decorators, this will print information for each.

for index, decorator in highest_rated_decorator.iterrows():
    print(f"The highest rated decorator with a rating of {decorator['Rating']} is {decorator['Caterer Name']}.")
```

```
The highest rated decorator with a rating of 4.8 is p.
```

In [48]:
```python
# Find the most cost-effective decorator based on decoration price

cost_effective_decorator = cat_df[cat_df["Decoration Price"] == cat_df["Decoration Price"].min()]

# If there are multiple most cost-effective decorators, this will print information for each.

for index, decorator in cost_effective_decorator.iterrows():
    print(f"The most cost-effective decorator with a decoration price of {decorator['Decoration Price']} Rs is {decorat
```

```
The most cost-effective decorator with a decoration price of 4200 Rs is g.
```

In [49]:
```python
# Find the highest rated caterer (assuming caterers are also in 'cat_df')

highest_rated_caterer = cat_df[cat_df["Rating"] == cat_df["Rating"].max()]

# If there are multiple highest rated caterers, this will print information for each.

for index, caterer in highest_rated_caterer.iterrows():
    print(f"The highest rated caterer with a rating of {caterer['Rating']} is {caterer['Caterer Name']}.")
```

```
The highest rated caterer with a rating of 4.8 is p.
```

In [50]:
```python
# Find the most cost-effective caterer based on plate cost
cost_effective_caterer = cat_df[cat_df["Plate Cost"] == cat_df["Plate Cost"].min()]

# If there are multiple most cost-effective caterers, this will print information for each.
for index, caterer in cost_effective_caterer.iterrows():
    print(f"The most cost-effective caterer with their plate cost of {caterer['Plate Cost']} Rs is {caterer['Caterer Na
```

```
The most cost-effective caterer with their plate cost of 250 Rs is g.
The most cost-effective caterer with their plate cost of 250 Rs is m.
```