Objective: To utilize exploratory data analysis (EDA) skills to understand customer preferences, dining trends, and competitive landscape in various regions of India, and to design an effective marketing campaign for a restaurant chain.

In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```python
df=pd.read_csv('zomato_restaurants_in_India.csv')
df.head(5)
```

Out[2]:

	res_id	name	establishment	url	address	city	city_id	locality
0	3400299	Bikanervala	['Quick Bites']	https://www.zomato.com/agra/bikanervala-khanda...	Kalyani Point, Near Tulsi Cinema, Bypass Road,...	Agra	34	Khandari
1	3400005	Mama Chicken Mama Franky House	['Quick Bites']	https://www.zomato.com/agra/mama-chicken-mama-...	Main Market, Sadar Bazaar, Agra Cantt, Agra	Agra	34	Agra Cantt
2	3401013	Bhagat Halwai	['Quick Bites']	https://www.zomato.com/agra/bhagat-halwai-2-sh...	62/1, Near Easy Day, West Shivaji Nagar, Goalp...	Agra	34	Shahganj
3	3400290	Bhagat Halwai	['Quick Bites']	https://www.zomato.com/agra/bhagat-halwai-civi...	Near Anjana Cinema, Nehru Nagar, Civil Lines, ...	Agra	34	Civil Lines
4	3401744	The Salt Cafe Kitchen & Bar	['Casual Dining']	https://www.zomato.com/agra/the-salt-cafe-kitc...	1C,3rd Floor, Fatehabad Road, Tajganj, Agra	Agra	34	Tajganj

5 rows × 26 columns

In [3]:
```python
df.isna().sum()
```

Out[3]:
```
res_id                   0
name                     0
establishment            0
url                      0
address                134
```

```
city                        0
city_id                     0
locality                    0
latitude                    0
longitude                   0
zipcode                163187
country_id                  0
locality_verbose            0
cuisines                 1391
timings                  3874
average_cost_for_two        0
price_range                 0
currency                    0
highlights                  0
aggregate_rating            0
rating_text                 0
votes                       0
photo_count                 0
opentable_support          48
delivery                    0
takeaway                    0
dtype: int64
```

In [4]:

```
missing_values = df.isnull().sum()
```

In [5]:

```
df.dtypes
```

Out[5]:

```
res_id                    int64
name                     object
establishment            object
url                      object
address                  object
city                     object
city_id                   int64
locality                 object
latitude                float64
longitude               float64
zipcode                  object
country_id                int64
locality_verbose         object
cuisines                 object
timings                  object
average_cost_for_two      int64
price_range               int64
currency                 object
highlights               object
aggregate_rating        float64
rating_text              object
votes                     int64
photo_count               int64
opentable_support       float64
delivery                  int64
takeaway                  int64
dtype: object
```

In [6]:

```
df.columns
```

Out[6]:

```
Index(['res_id', 'name', 'establishment', 'url', 'address', 'city', 'city_id',
       'locality', 'latitude', 'longitude', 'zipcode', 'country_id',
       'locality_verbose', 'cuisines', 'timings', 'average_cost_for_two',
       'price_range', 'currency', 'highlights', 'aggregate_rating',
       'rating_text', 'votes', 'photo_count', 'opentable_support', 'delivery',
       'takeaway'],
      dtype='object')
```

Data Cleaning and Preparation:

In [7]:

```
# Getting missing addresses using longitude and latitude
from geopy.geocoders import Nominatim

geolocator = Nominatim(user_agent="my_geocoder")


def get_missing_addresses(row):
    if pd.isnull(row['address']):
        location = geolocator.reverse((row['latitude'], row['longitude']), exactly_one=T
        if location:
            return location.address
        else:
            return None
    else:
        return row['address']

df['address'] = df.apply(get_missing_addresses, axis=1)
```

In [8]:

```
df.isnull().sum()
```

Out[8]:

```
res_id                        0
name                          0
establishment                 0
url                           0
address                       0
city                          0
city_id                       0
locality                      0
latitude                      0
longitude                     0
zipcode                  163187
country_id                    0
locality_verbose              0
cuisines                   1391
timings                    3874
average_cost_for_two          0
price_range                   0
currency                      0
highlights                    0
aggregate_rating              0
rating_text                   0
votes                         0
photo_count                   0
opentable_support            48
delivery                      0
takeaway                      0
dtype: int64
```

In [9]:

```
# Missing cuisines are filled by 'other'
df['cuisines'].fillna("Other", inplace=True)
```

In [10]:

```
# Missing timings are filled by most common timings
most_common_timing = df['timings'].mode()[0]
df['timings'].fillna(most_common_timing, inplace=True)
```

In [11]:

```
# dropped the zipcodes as most are missing and there is no need for zipcodes in our anal
df.drop(columns=['zipcode'], inplace=True)
```

In [12]:

```
# Missing values in opentable support are filled with zero as whole column has zero in i
df['opentable_support'].fillna(0, inplace=True)
```

In [13]:

```
df.isnull().sum()
```

Out[13]:

```
res_id                  0
name                    0
establishment           0
url                     0
address                 0
city                    0
city_id                 0
locality                0
latitude                0
longitude               0
country_id              0
locality_verbose        0
cuisines                0
timings                 0
average_cost_for_two    0
price_range             0
currency                0
highlights              0
aggregate_rating        0
rating_text             0
votes                   0
photo_count             0
opentable_support       0
delivery                0
takeaway                0
dtype: int64
```

In [14]:

```
df['timings'] = df['timings'].str.replace('â€“', 'to')
```

Descriptive Statistics:

Summary of the central tendency, dispersion and shape of the dataset distribution

In [15]:

```
numeric_columns = ['average_cost_for_two', 'aggregate_rating', 'votes', 'photo_count']
df[numeric_columns].describe()
```

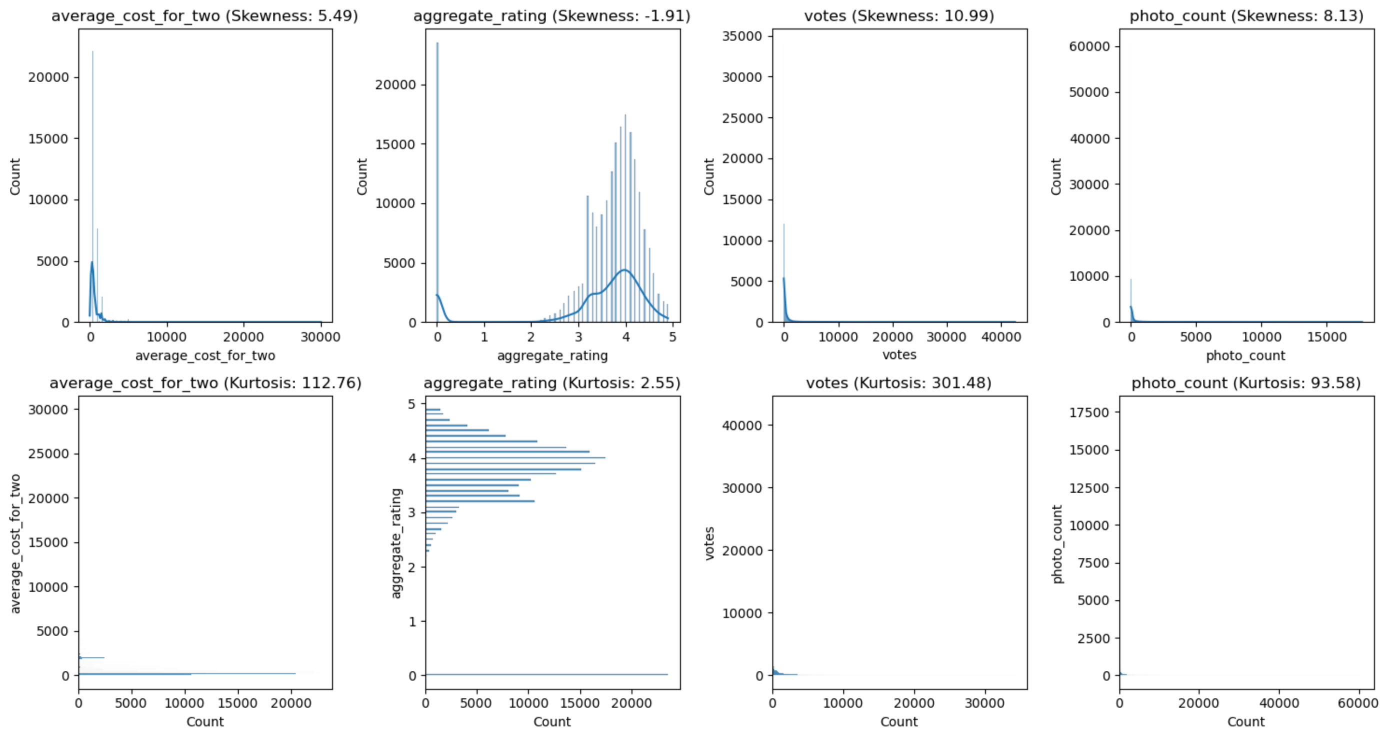
Out[15]:

	average_cost_for_two	aggregate_rating	votes	photo_count
count	211944.000000	211944.000000	211944.000000	211944.000000
mean	595.812229	3.395937	378.001864	256.971224
std	606.239363	1.283642	925.333370	867.668940
min	0.000000	0.000000	-18.000000	0.000000
25%	250.000000	3.300000	16.000000	3.000000
50%	400.000000	3.800000	100.000000	18.000000
75%	700.000000	4.100000	362.000000	128.000000
max	30000.000000	4.900000	42539.000000	17702.000000

In [16]:

```
numeric_columns = ['average_cost_for_two', 'aggregate_rating', 'votes', 'photo_count']

fig, axes = plt.subplots(nrows=2, ncols=len(numeric_columns), figsize=(15, 8))

for i, col in enumerate(numeric_columns):
    sns.histplot(df[col], ax=axes[0, i], kde=True)
    axes[0, i].set_title(col + " (Skewness: {:.2f})".format(df[col].skew()))
```

```
for i, col in enumerate(numeric_columns):
    sns.histplot(y=df[col], ax=axes[1, i])
    axes[1, i].set_title(col + " (Kurtosis: {:.2f})".format(df[col].kurtosis()))


plt.tight_layout()
plt.show()
```



In [17]:
```
categorical_columns = ['establishment', 'city', 'locality']
df[categorical_columns].apply(lambda x: x.value_counts())
```

Out[17]:

	establishment	city	locality
32nd Avenue, NH8, Gurgaon	NaN	NaN	10.0
800 Jubilee, Jubilee Hills	NaN	NaN	4.0
Hotel Somdeep Palace, Vijay Nagar	NaN	NaN	35.0
ILD Trade Centre Mall, Sohna Road	NaN	NaN	2.0
InterContinental Chennai Mahabalipuram Resort, East Coast Road (ECR)	NaN	NaN	14.0
...	...	...	...
['Quick Bites']	64390.0	NaN	NaN
['Shack']	44.0	NaN	NaN
['Sweet Shop']	6103.0	NaN	NaN
[]	4827.0	NaN	NaN
lebua Lucknow	NaN	NaN	8.0

3851 rows × 3 columns

In [18]:
```
boolean_columns = ['opentable_support', 'delivery', 'takeaway']
df[boolean_columns].apply(pd.Series.value_counts)
```

Out[18]:

	opentable_support	delivery	takeaway

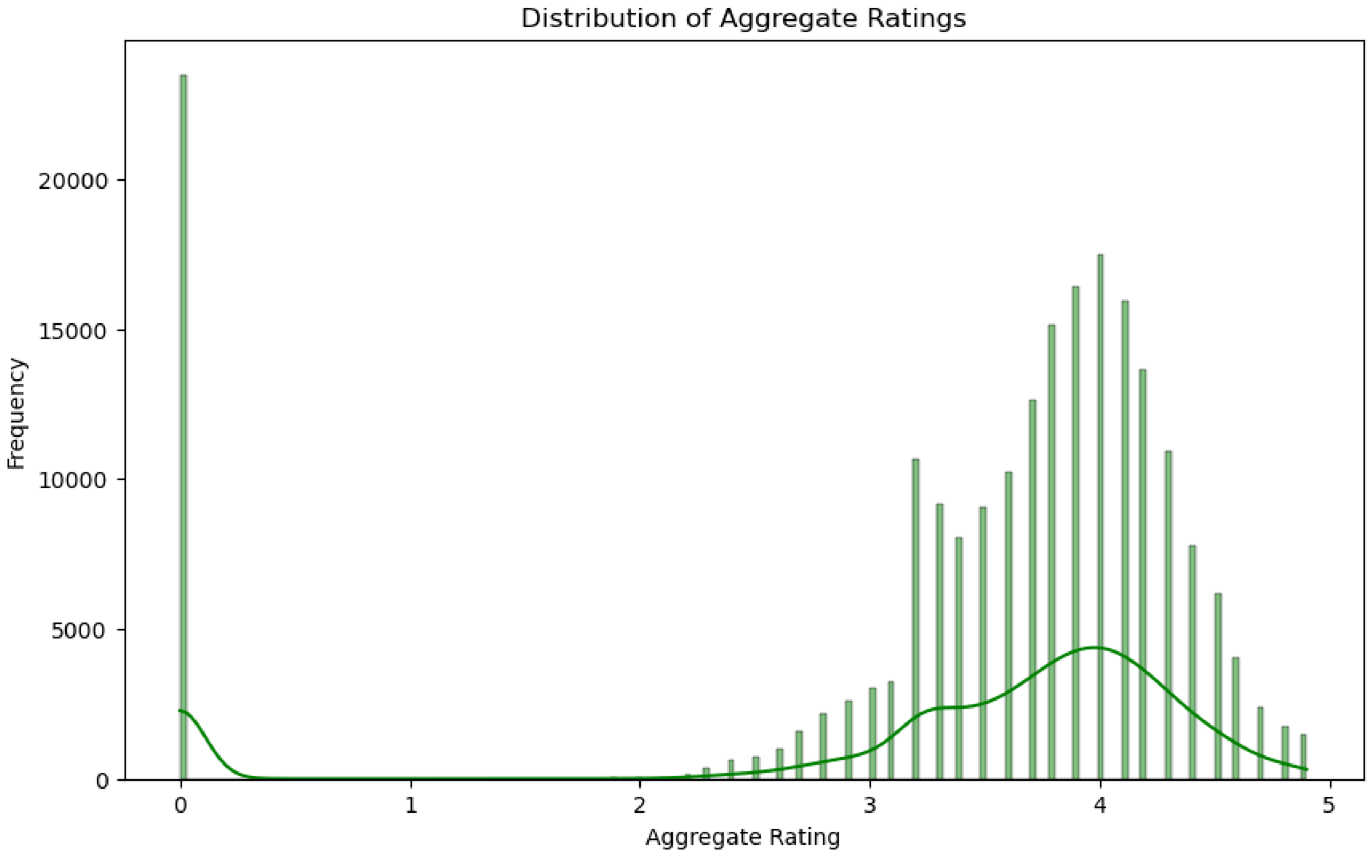
-1.0	NaN	132573	211944.0
0.0	211944.0	1036	NaN
1.0	NaN	78335	NaN

Distribution Analysis:

Analysis of the distribution of key variables (e.g., ratings, price range, cuisines)

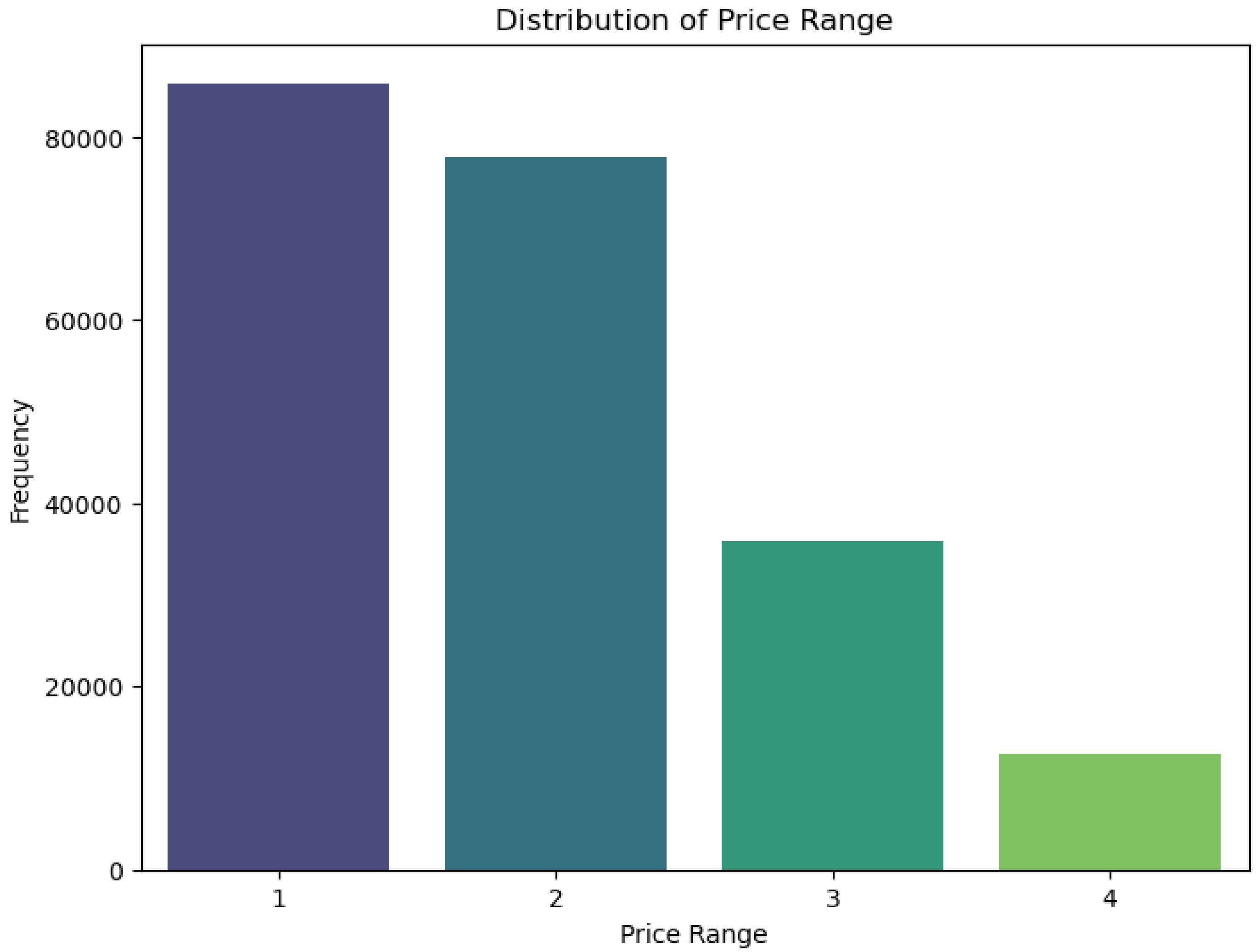
In [19]:

```
# Ratings distribution
plt.figure(figsize=(10, 6))
sns.histplot(df['aggregate_rating'], kde=True, color='green')
plt.title('Distribution of Aggregate Ratings')
plt.xlabel('Aggregate Rating')
plt.ylabel('Frequency')
plt.show()
```

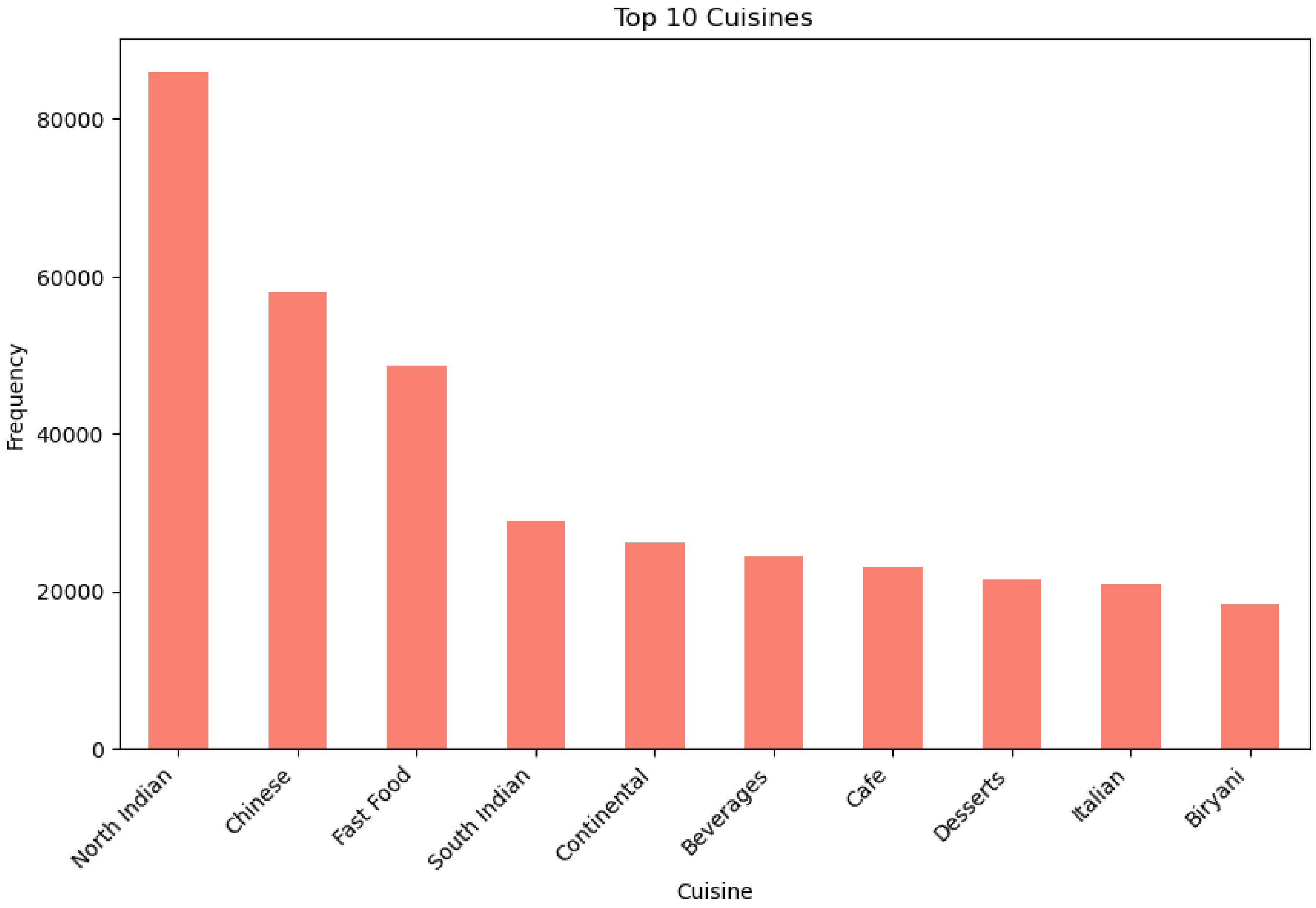


In [20]:

```
# Price range distribution
plt.figure(figsize=(8, 6))
sns.countplot(x='price_range', data=df, palette='viridis')
plt.title('Distribution of Price Range')
plt.xlabel('Price Range')
plt.ylabel('Frequency')
plt.show()
```



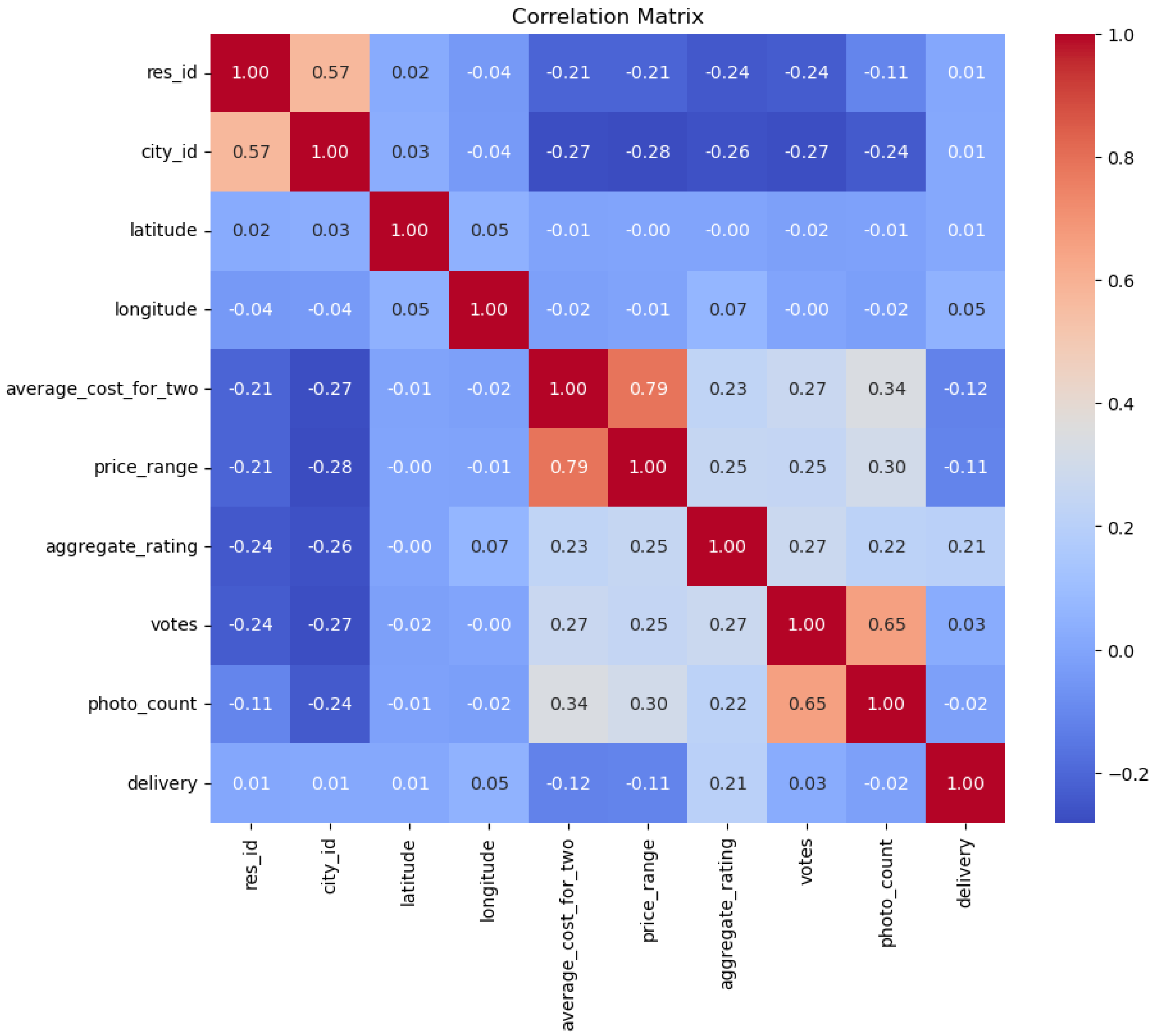
```
In [21]: # Top 10 cuisines distribution
         top_cuisines = df['cuisines'].str.split(', ', expand=True).stack().value_counts().head(1
         plt.figure(figsize=(10, 6))
         top_cuisines.plot(kind='bar', color='salmon')
         plt.title('Top 10 Cuisines')
         plt.xlabel('Cuisine')
         plt.ylabel('Frequency')
         plt.xticks(rotation=45, ha='right')
         plt.show()
```



Correlation Analysis:

In [22]:

```
correlation_matrix = df.corr()
columns_to_hide = ['country_id', 'opentable_support','takeaway']
correlation_matrix_subset = correlation_matrix.drop(columns=columns_to_hide, index=colum

# Visualize the correlation matrix subset using a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix_subset, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```

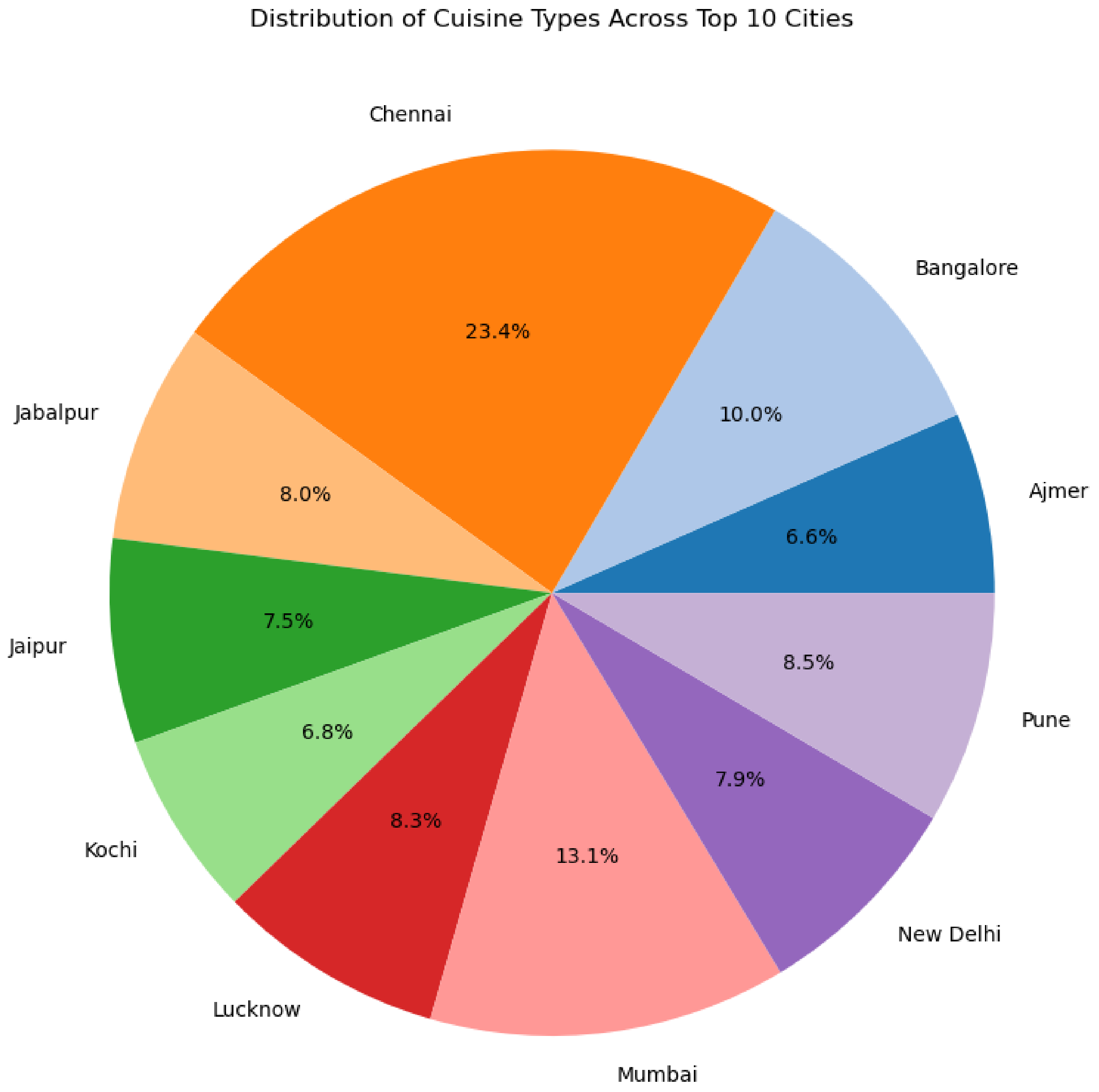


Regional Analysis:

In [24]:

```
# Group the data by city
region_groups = df.groupby('city')

# Calculate the count of each city
city_counts = region_groups.size().sort_values(ascending=False)

# Select the top 10 cities
top_10_cities = city_counts.head(10).index

# Filter the DataFrame to include only data for the top 10 cities
df_top_10 = df[df['city'].isin(top_10_cities)]

# Group the filtered data by city again
region_groups_top_10 = df_top_10.groupby('city')

# Aggregate the data to get the count of each cuisine type in each of the top 10 cities
cuisine_counts_top_10 = region_groups_top_10['cuisines'].value_counts().unstack().fillna

# Plotting the data as a pie chart
plt.figure(figsize=(12, 8))
colors = plt.cm.tab20.colors  # Choose a color map for better distinction
```

```
cuisine_counts_top_10.sum(axis=1).plot(kind='pie', autopct='%1.1f%%', colors=colors)
plt.title('Distribution of Cuisine Types Across Top 10 Cities')
plt.ylabel('')
plt.tight_layout()
plt.show()
```

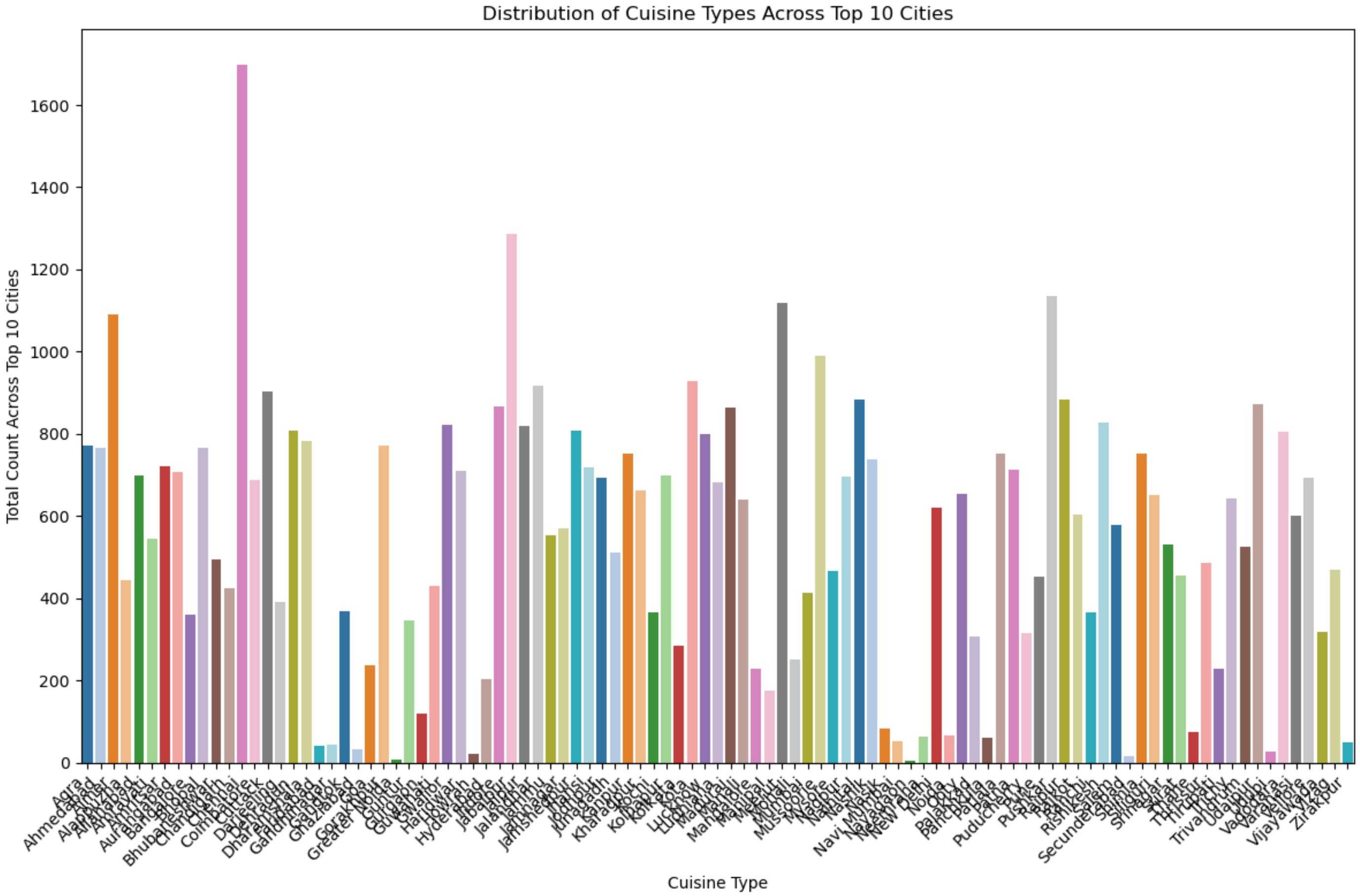


Customer Preference Analysis:

In [27]:

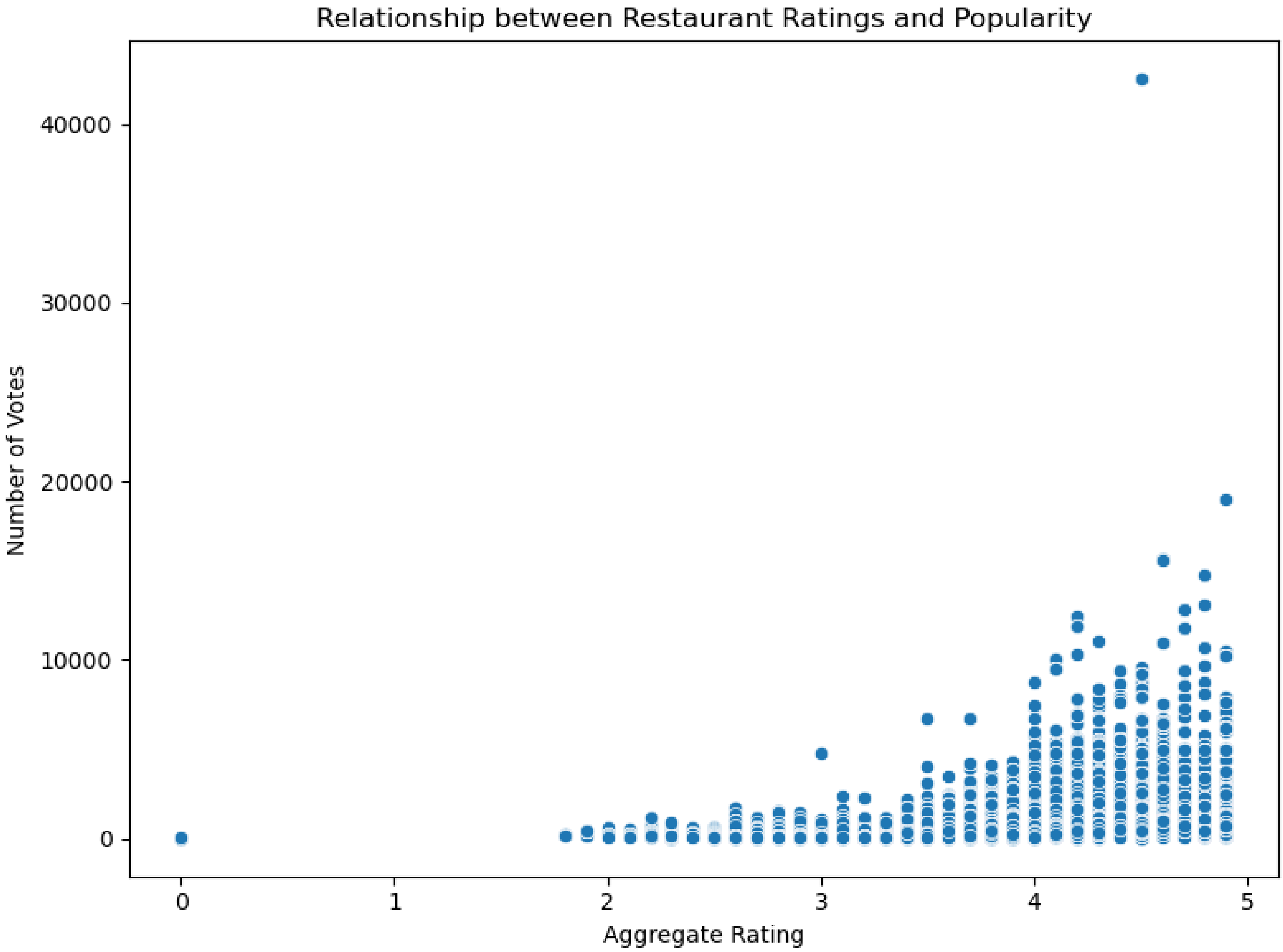
```
import matplotlib.pyplot as plt
import seaborn as sns

# Group the data by city
region_groups = df.groupby('city')

# Aggregate the data to get the count of each cuisine type in each city
cuisine_counts = region_groups['cuisines'].value_counts().unstack().fillna(0)

# Calculate the sum of cuisine counts across cities
```

```
cuisine_counts_sum = cuisine_counts.sum(axis=0)

# Select the top 10 cities
top_10_cities = cuisine_counts_sum.nlargest(10).index

# Filter cuisine counts to include only data for the top 10 cities
cuisine_counts_top_10 = cuisine_counts[top_10_cities]

# Sum the cuisine counts across the top 10 cities
cuisine_counts_sum_top_10 = cuisine_counts_top_10.sum(axis=1)

# Plotting the data as a bar plot
plt.figure(figsize=(12, 8))
sns.barplot(x=cuisine_counts_sum_top_10.index, y=cuisine_counts_sum_top_10.values, palet
plt.title('Distribution of Cuisine Types Across Top 10 Cities')
plt.xlabel('Cuisine Type')
plt.ylabel('Total Count Across Top 10 Cities')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



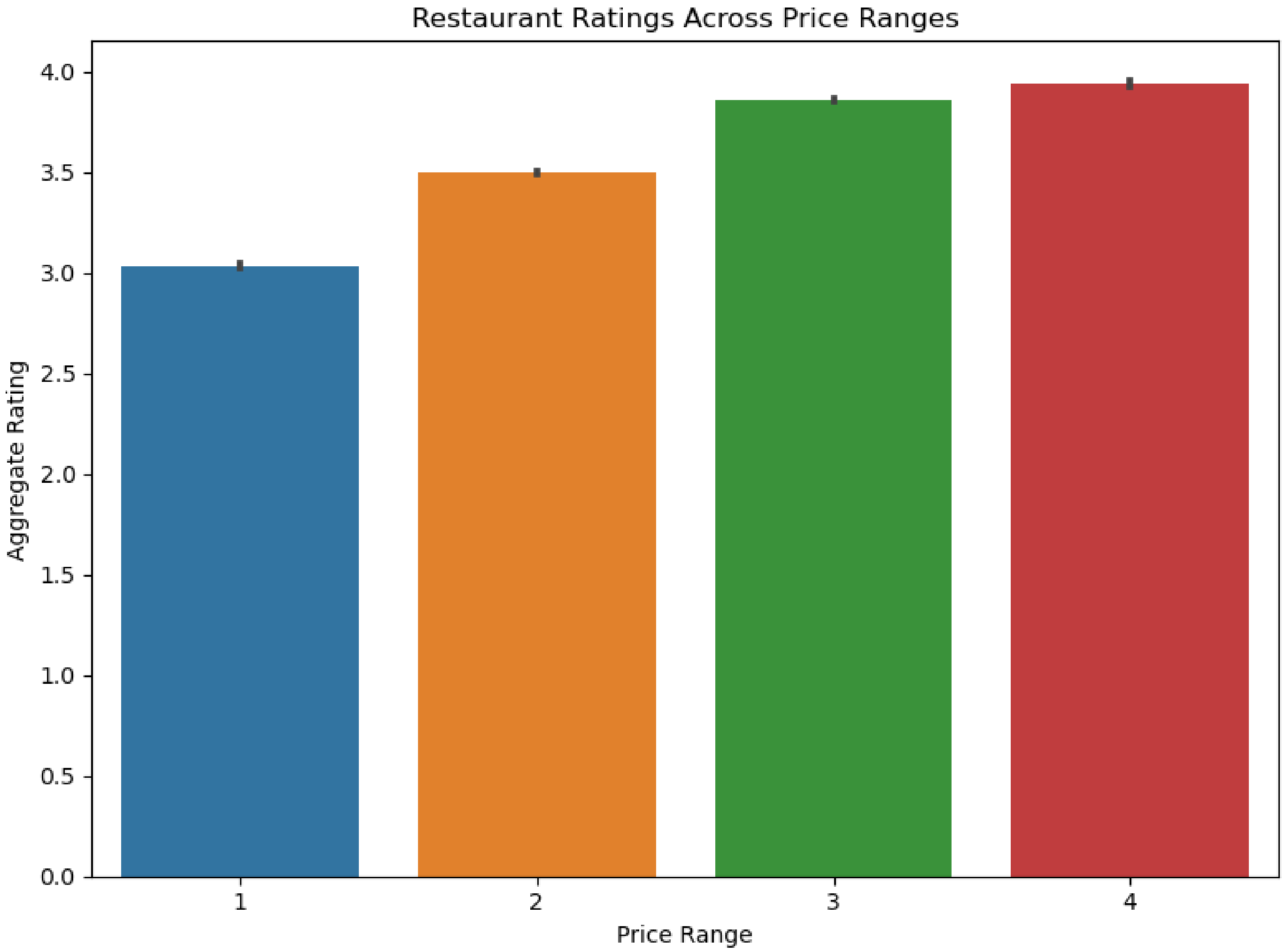
In [28]:

```
# Scatter plot between restaurant ratings and popularity (votes)
plt.figure(figsize=(8, 6))
sns.scatterplot(x='aggregate_rating', y='votes', data=df)
plt.title('Relationship between Restaurant Ratings and Popularity')
plt.xlabel('Aggregate Rating')
plt.ylabel('Number of Votes')
plt.tight_layout()
plt.show()
```



In [29]:

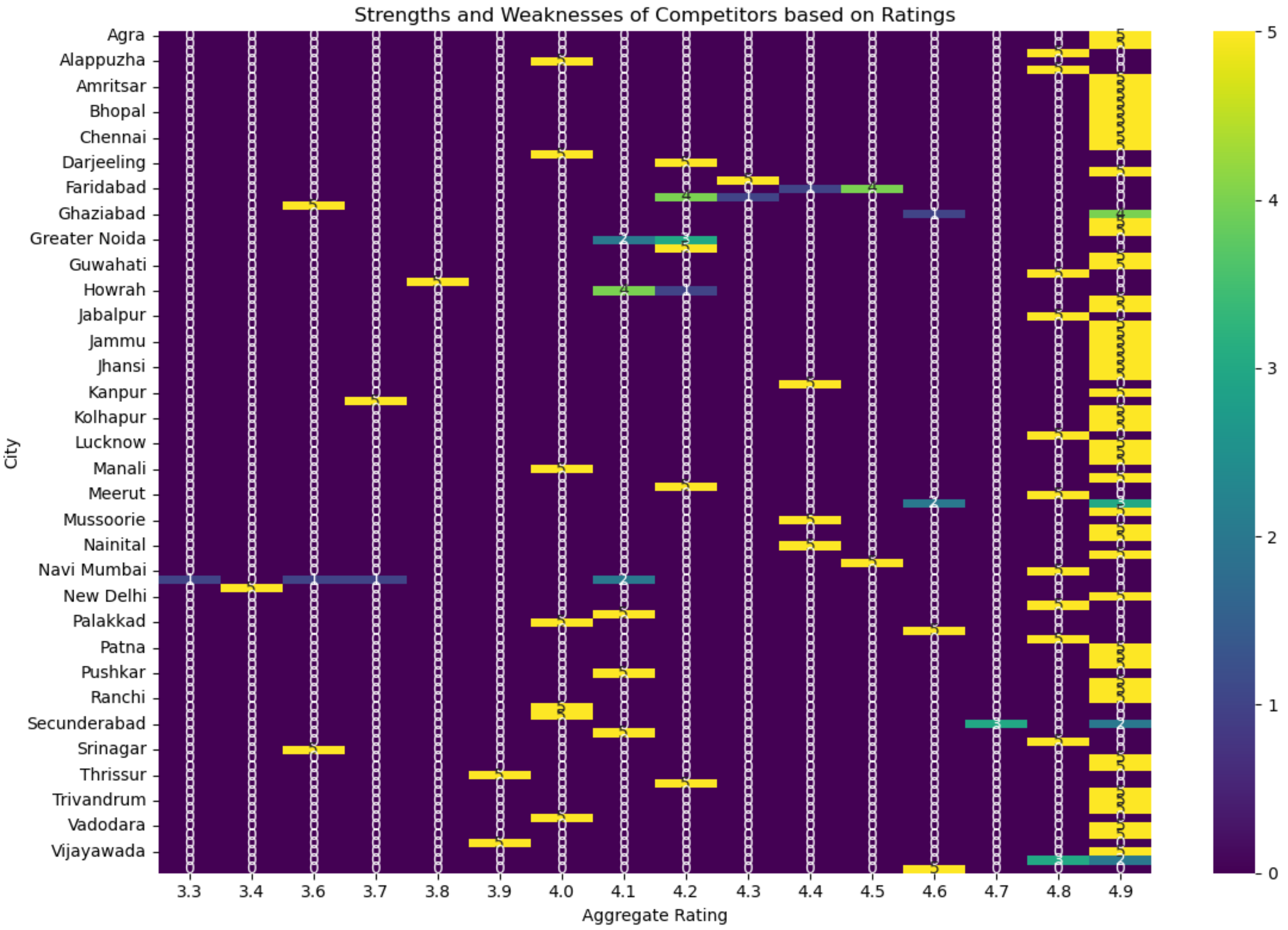
```
# Bar plot to compare restaurant ratings across different price ranges
plt.figure(figsize=(8, 6))
sns.barplot(x='price_range', y='aggregate_rating', data=df)
plt.title('Restaurant Ratings Across Price Ranges')
plt.xlabel('Price Range')
plt.ylabel('Aggregate Rating')
plt.tight_layout()
plt.show()
```



Competitive Analysis:

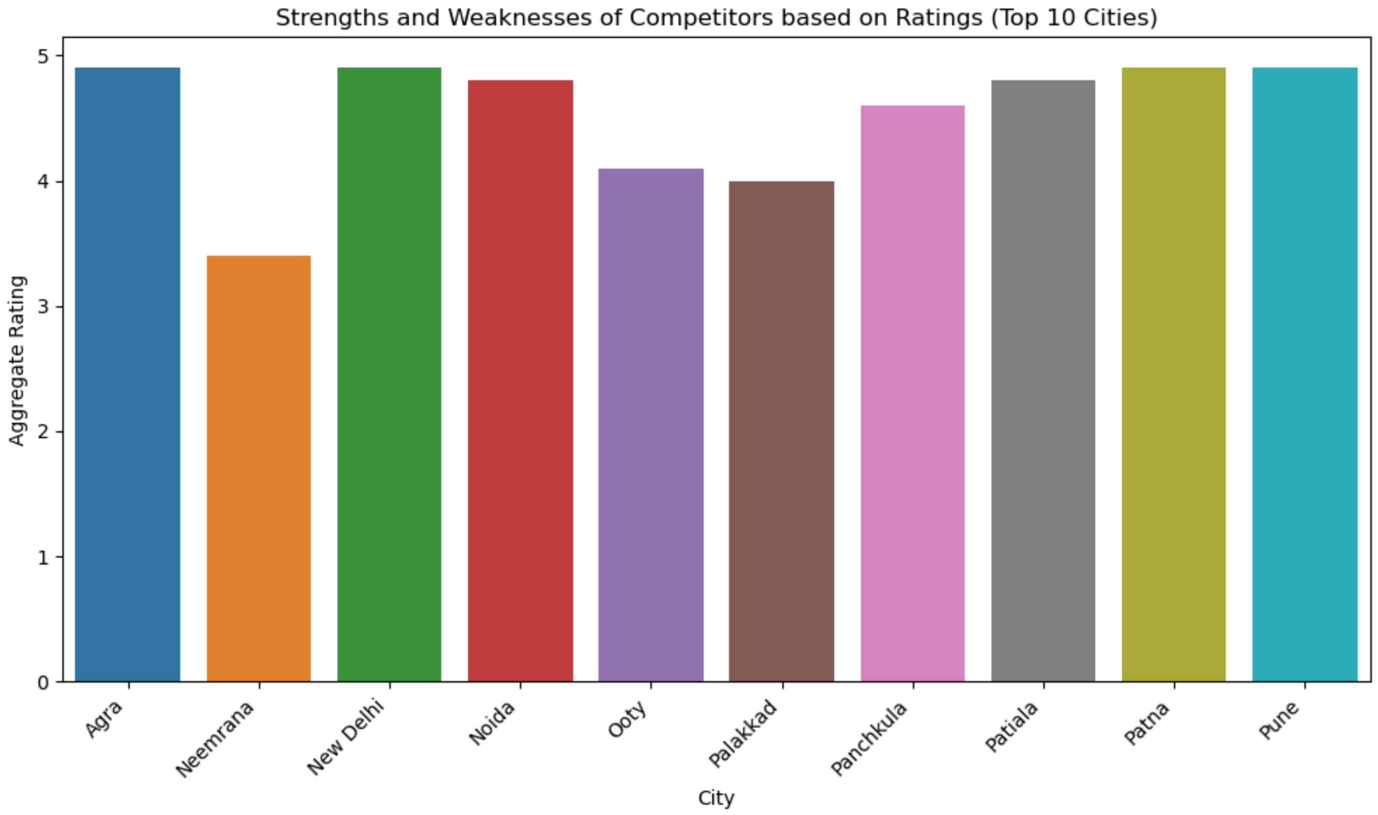
In [30]:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Group the data by region (city)
region_groups = df.groupby('city')

# Identify major competitors in each region based on cuisine, pricing, and ratings
competitors = region_groups.apply(lambda x: x.nlargest(5, 'aggregate_rating'))

# Analyze strengths and weaknesses of competitors
# Strengths: Higher ratings, Positive reviews
# Weaknesses: Lower ratings, Negative reviews
competitor_strengths = competitors[competitors['aggregate_rating'] >= 4.0]
competitor_weaknesses = competitors[competitors['aggregate_rating'] < 4.0]

# Remove the 'city' column before pivoting the DataFrame
competitors_reset = competitors.reset_index(drop=True)

# Create a pivot table to visualize strengths and weaknesses of competitors based on rat
plt.figure(figsize=(12, 8))
heatmap_data = competitors_reset.pivot_table(index='city', columns='aggregate_rating', a
sns.heatmap(heatmap_data, cmap='viridis', annot=True, fmt='d')
plt.title('Strengths and Weaknesses of Competitors based on Ratings')
plt.xlabel('Aggregate Rating')
plt.ylabel('City')
plt.tight_layout()
plt.show()
```



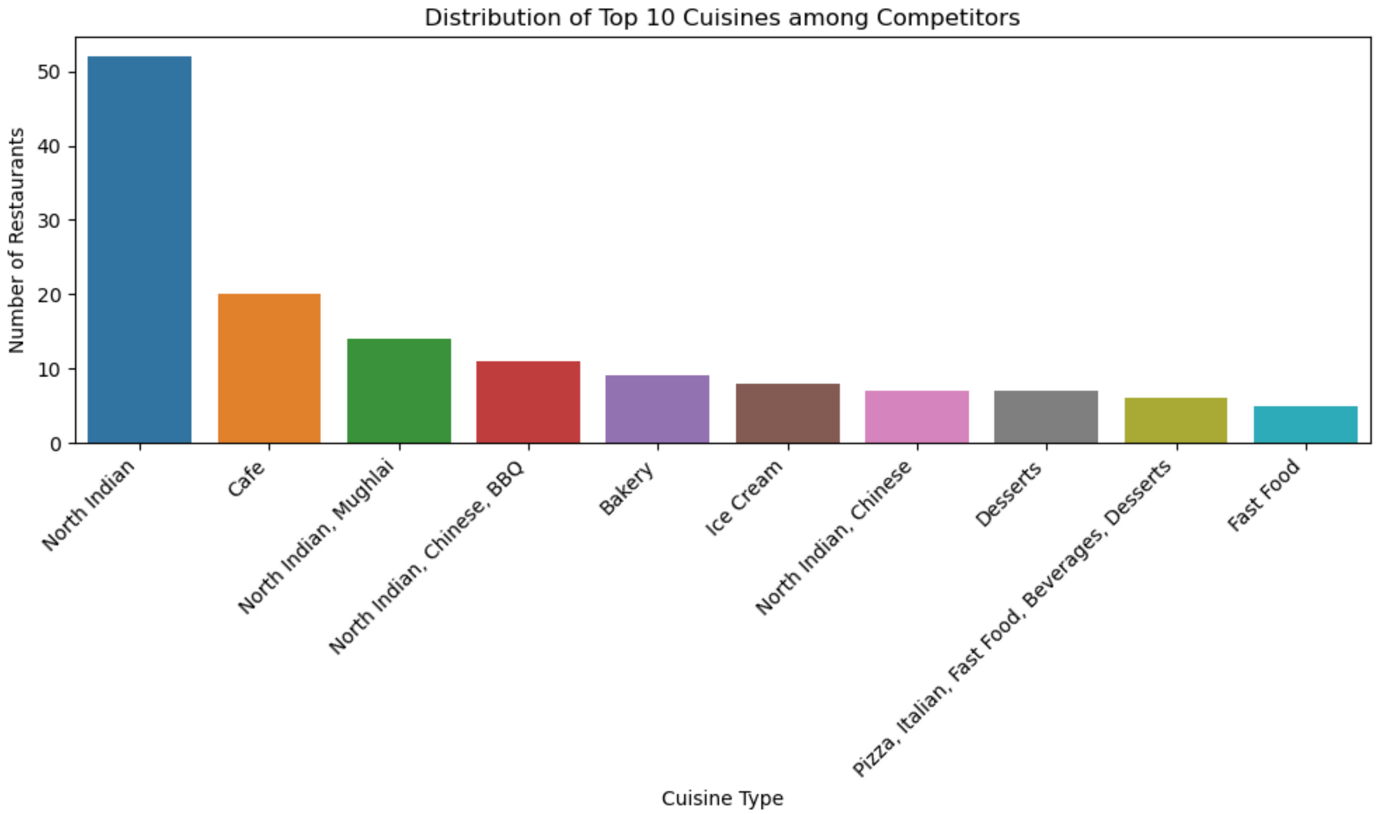
In [31]:

```python
# Group the data by region (city)
region_groups = df.groupby('city')

# Identify major competitors in each region based on cuisine, pricing, and ratings
competitors = region_groups.apply(lambda x: x.nlargest(5, 'aggregate_rating'))

# Select the top 10 cities based on the count of competitors
top_10_cities = competitors['city'].value_counts().head(10).index

# Filter competitors DataFrame to include only data for the top 10 cities
competitors_top_10 = competitors[competitors['city'].isin(top_10_cities)]

# Strengths and weaknesses based on ratings for the top 10 cities
plt.figure(figsize=(10, 6))
sns.barplot(x='city', y='aggregate_rating', data=competitors_top_10)
plt.title('Strengths and Weaknesses of Competitors based on Ratings (Top 10 Cities)')
plt.xlabel('City')
plt.ylabel('Aggregate Rating')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



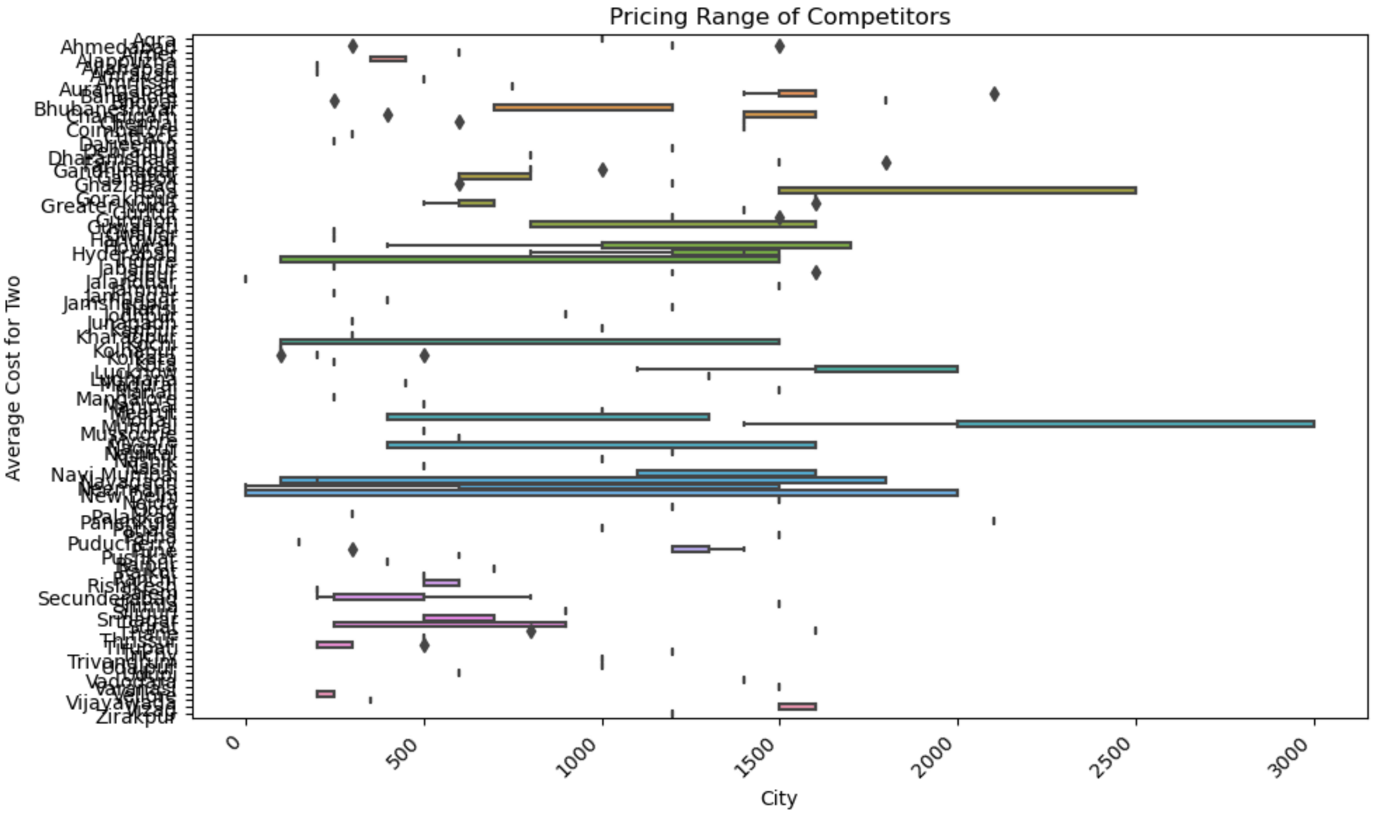
In [32]:

```python
# Group the data by region (city)
region_groups = df.groupby('city')

# Identify major competitors in each region based on cuisine, pricing, and ratings
competitors = region_groups.apply(lambda x: x.nlargest(5, 'aggregate_rating'))

# Count the occurrences of each cuisine type
cuisine_counts = competitors['cuisines'].value_counts()

# Select the top 10 cuisines
top_10_cuisines = cuisine_counts.head(10).index

# Filter competitors DataFrame to include only data for the top 10 cuisines
competitors_top_10_cuisines = competitors[competitors['cuisines'].isin(top_10_cuisines)]

# Distribution of cuisines among competitors for the top 10 cuisines
plt.figure(figsize=(10, 6))
sns.countplot(x='cuisines', data=competitors_top_10_cuisines, order=top_10_cuisines)
plt.title('Distribution of Top 10 Cuisines among Competitors')
plt.xlabel('Cuisine Type')
plt.ylabel('Number of Restaurants')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



In [40]:

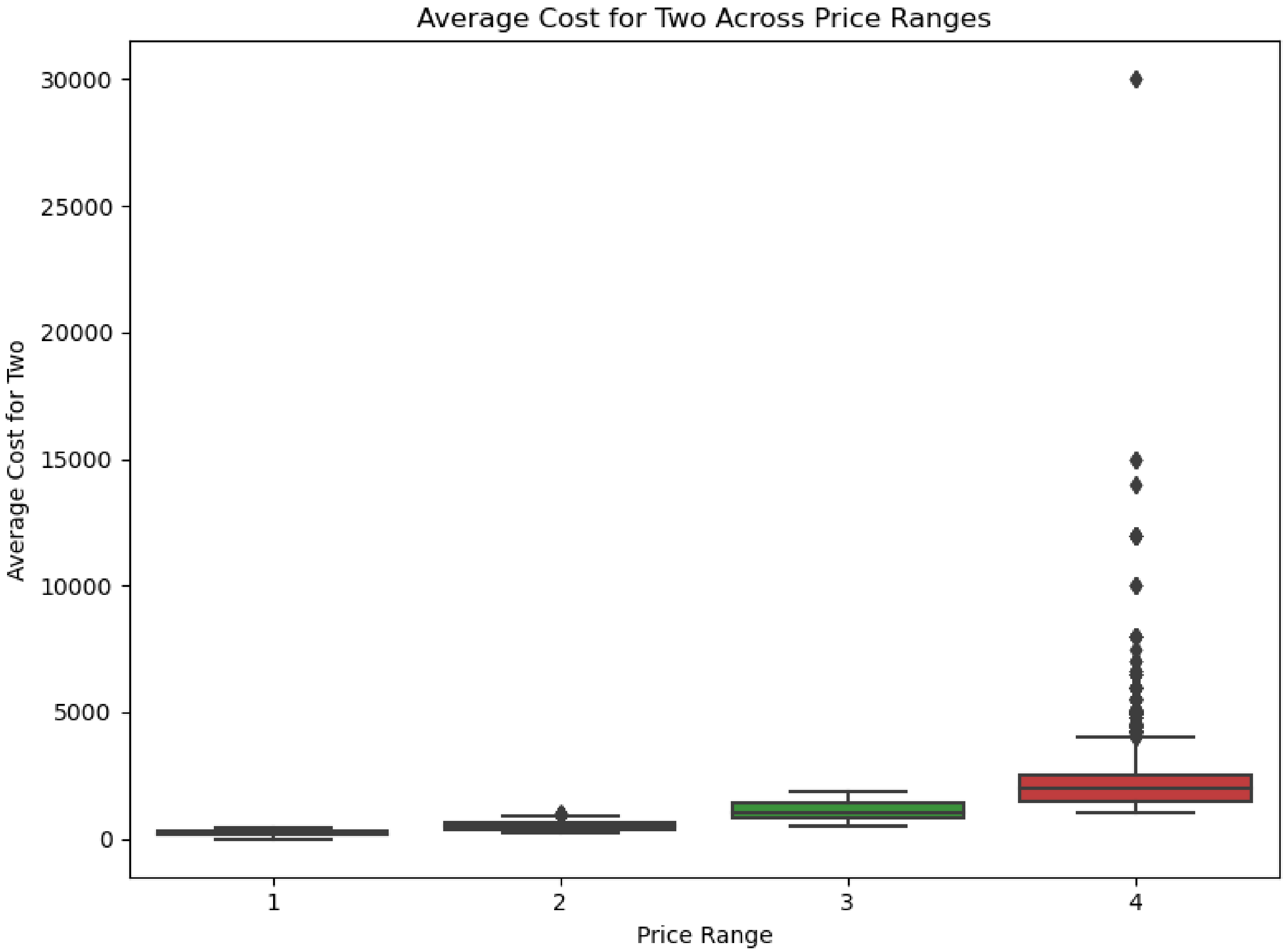
```
#Pricing Range of Competitors
plt.figure(figsize=(10, 6))
sns.boxplot(x='average_cost_for_two', y='city', data=competitors)
plt.title('Pricing Range of Competitors')
plt.xlabel('City')
plt.ylabel('Average Cost for Two')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



Market Gap Analysis:

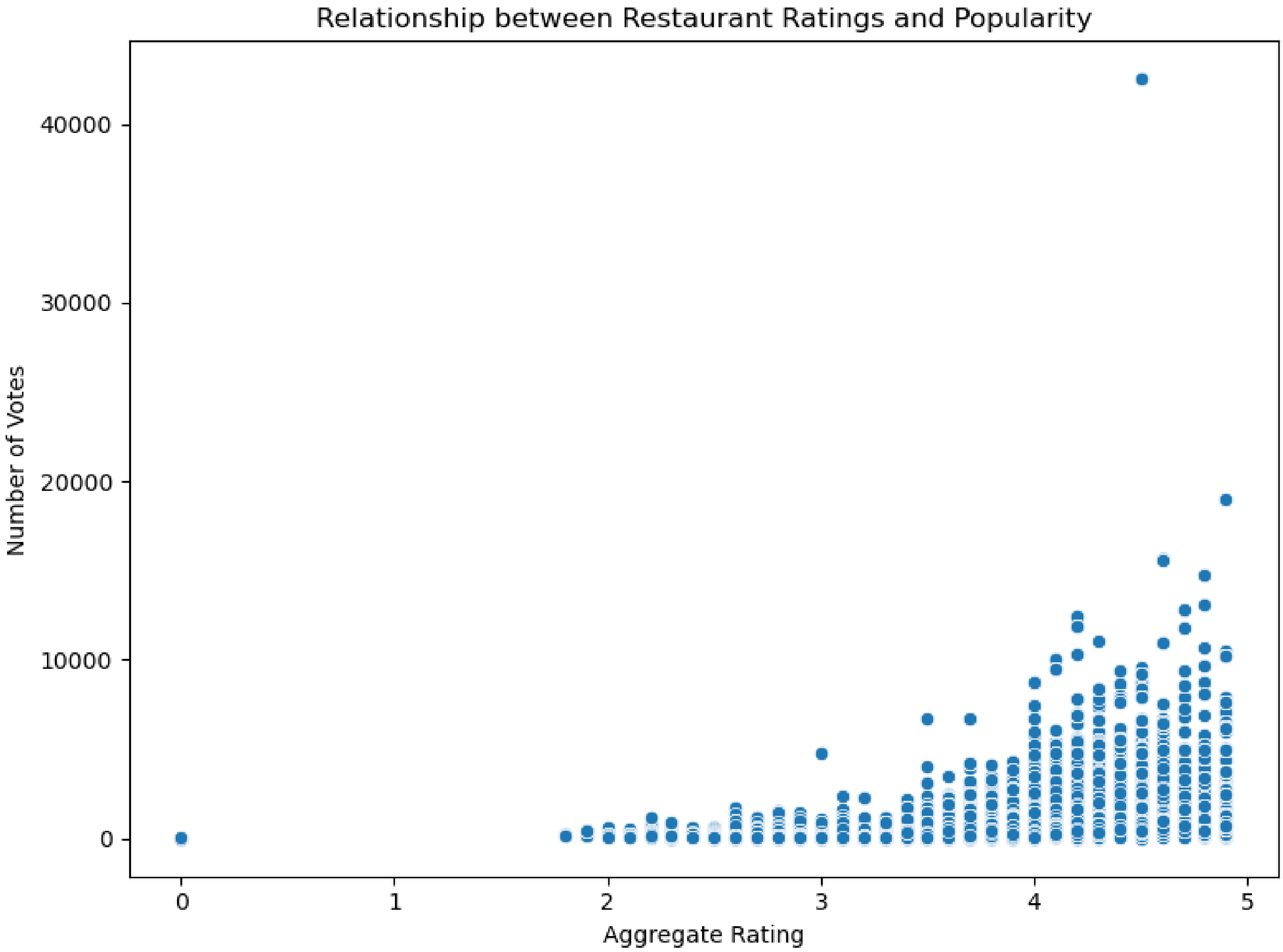
In [35]:

```
# Underrepresented Price Ranges
plt.figure(figsize=(8, 6))
sns.boxplot(x='price_range', y='average_cost_for_two', data=df)
plt.title('Average Cost for Two Across Price Ranges')
plt.xlabel('Price Range')
plt.ylabel('Average Cost for Two')
plt.tight_layout()
plt.show()
```



In [36]:

```
# Analyze Customer Reviews and Ratings
plt.figure(figsize=(8, 6))
sns.scatterplot(x='aggregate_rating', y='votes', data=df)
plt.title('Relationship between Restaurant Ratings and Popularity')
plt.xlabel('Aggregate Rating')
plt.ylabel('Number of Votes')
plt.tight_layout()
plt.show()
```



In [2]:

```
import subprocess

# Define the notebook filename as a string
notebook_filename = "Marketing Campaign for a Restaurant Chain.ipynb"

# Define the nbconvert command with --allow-chromium-download option
command = f"jupyter nbconvert --to webpdf \"{notebook_filename}\" --allow-chromium-downl

# Execute the command
subprocess.run(command, shell=True)
```

Out[2]:

```
CompletedProcess(args='jupyter nbconvert --to webpdf "Marketing Campaign for a Restauran
t Chain.ipynb" --allow-chromium-download', returncode=0)
```

In [ ]: