

## ✓ Zomato Bangalore Restaurant Trends Analysis

This notebook walks through the process of cleaning, preprocessing, and analyzing Zomato restaurant data for Bangalore. It includes:

- Data cleaning and preprocessing.
- Exploratory data analysis (EDA).
- Merging location data for mapping.
- Extracting actionable insights.

### ✓ Step 1: Data Cleaning and Preprocessing

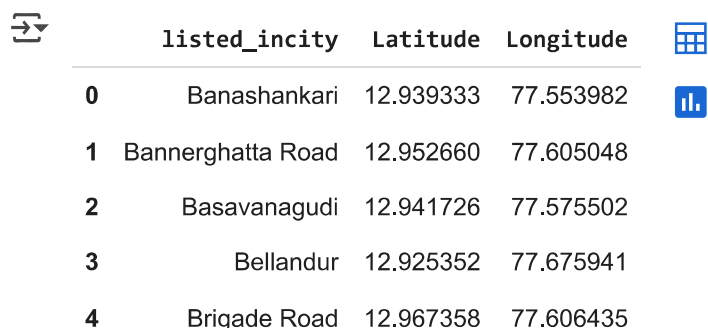
In this step, we handle missing values in the dataset, especially in the categorical columns such as `dish_liked`, `cuisines`, and `rest_type`. We replace missing values with suitable replacements like 'Other' or 'Unknown'.

We will also convert necessary columns to appropriate data types.

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

# Load the datasets
data1 = pd.read_csv('/content/zomato_data.csv')
data2 = pd.read_csv('/content/Geographical Coordinates.csv')

# Display the first few rows of the dataset
data1.head()
data2.head()
```



	listed_incity	Latitude	Longitude
0	Banashankari	12.939333	77.553982
1	Bannerghatta Road	12.952660	77.605048
2	Basavanagudi	12.941726	77.575502
3	Bellandur	12.925352	77.675941
4	Brigade Road	12.967358	77.606435

Next steps: [Generate code with data2](#) [View recommended plots](#) [New interactive sheet](#)

```
data2.head()
```



	listed_incity	Latitude	Longitude
0	Banashankari	12.939333	77.553982
1	Bannerghatta Road	12.952660	77.605048
2	Basavanagudi	12.941726	77.575502
3	Bellandur	12.925352	77.675941
4	Brigade Road	12.967358	77.606435



Next steps:

[Generate code with data2](#)[View recommended plots](#)[New interactive sheet](#)

```
# Step 1: Rating Column (rate)
# Replace '-' values with NaN, remove '/5' and convert to float
data1['rate'] = data1['rate'].str.replace('/5', '', regex=False)
data1['rate'] = pd.to_numeric(data1['rate'], errors='coerce')

# Fill missing ratings with the median
data1['rate'].fillna(data1['rate'].median(), inplace=True)

# Verify the changes using .info(), .isnull().sum(), and .describe()
data1['rate'].info(), data1['rate'].isnull().sum()
```



```
<class 'pandas.core.series.Series'>
RangeIndex: 51717 entries, 0 to 51716
Series name: rate
Non-Null Count  Dtype
-----
51717 non-null  float64
dtypes: float64(1)
memory usage: 404.2 KB
<ipython-input-20-b520bfacd2e1>:7: FutureWarning: A value is trying to be set on a copy of a DataFrame
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate
```

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, in

```
data1['rate'].fillna(data1['rate'].median(), inplace=True)
(None, np.int64(0))
```

```
# Step 2: Cost Column (approx_costfor_two_people)
# Remove commas and convert to numeric
data1['approx_costfor_two_people'] = data1['approx_costfor_two_people'].replace({' ',''}, regex=True)
data1['approx_costfor_two_people'] = pd.to_numeric(data1['approx_costfor_two_people'], errors='coerce')

# Fill missing values with the median
data1['approx_costfor_two_people'].fillna(data1['approx_costfor_two_people'].median(), inplace=True)

# Verify the changes
data1['approx_costfor_two_people'].info(), data1['approx_costfor_two_people'].isnull().sum()
```

```

➦ <class 'pandas.core.series.Series'>
RangeIndex: 51717 entries, 0 to 51716
Series name: approx_costfor_two_people
Non-Null Count  Dtype
-----
51717 non-null  float64
dtypes: float64(1)
memory usage: 404.2 KB
<ipython-input-21-b6436db1d1a0>:7: FutureWarning: A value is trying to be set on a copy of a DataFrame
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, in

data1['approx_costfor_two_people'].fillna(data1['approx_costfor_two_people'].median(), inplace=True)
(None, np.int64(0))

```

```

# Step 3: Categorical Columns
# Replace NaN with "Not Available" or "Other" for categorical columns
data1['dish_liked'].fillna('Not Available', inplace=True)
data1['cuisines'].fillna('Other', inplace=True)
data1['rest_type'].fillna('Unknown', inplace=True)

# Verify the changes
data1[['dish_liked', 'cuisines', 'rest_type']].isnull().sum()

```

```

➦ <ipython-input-22-a0a7a0018dbc>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, in

data1['dish_liked'].fillna('Not Available', inplace=True)
<ipython-input-22-a0a7a0018dbc>:4: FutureWarning: A value is trying to be set on a copy of a DataFrame
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, in

data1['cuisines'].fillna('Other', inplace=True)
<ipython-input-22-a0a7a0018dbc>:5: FutureWarning: A value is trying to be set on a copy of a DataFrame
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, in

data1['rest_type'].fillna('Unknown', inplace=True)


0
dish_liked  0
cuisines    0
rest_type   0

dtype: int64

```

```
# Step 4: Votes Column
# Fill missing values in the votes column with the median
data1['votes'].fillna(data1['votes'].median(), inplace=True)

# Verify the changes
data1['votes'].isnull().sum(), data1['votes'].describe()
```


 <ipython-input-23-43e1b7d70e99>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, in

```
data1['votes'].fillna(data1['votes'].median(), inplace=True)
(np.int64(0),
count      51717.000000
mean        283.697527
std         803.838853
min           0.000000
25%           7.000000
50%          41.000000
75%         198.000000
max        16832.000000
Name: votes, dtype: float64)
```

```
# Step 5: Binary Encoding
# Convert 'online_order' and 'book_table' columns to binary values
data1['online_order'] = data1['online_order'].map({'Yes': 1, 'No': 0})
data1['book_table'] = data1['book_table'].map({'Yes': 1, 'No': 0})

# Verify the changes
data1[['online_order', 'book_table']].head()
```



	online_order	book_table
0	1	1
1	1	0
2	1	0
3	0	0
4	0	0

```
# Step 6: Data Type Conversion
# Convert 'rate' to float, 'votes' to integer, and 'approx_costfor_two_people' to integer
data1['rate'] = data1['rate'].astype(float)
data1['votes'] = data1['votes'].astype(int)
data1['approx_costfor_two_people'] = data1['approx_costfor_two_people'].astype(int)

# Verify the changes
```

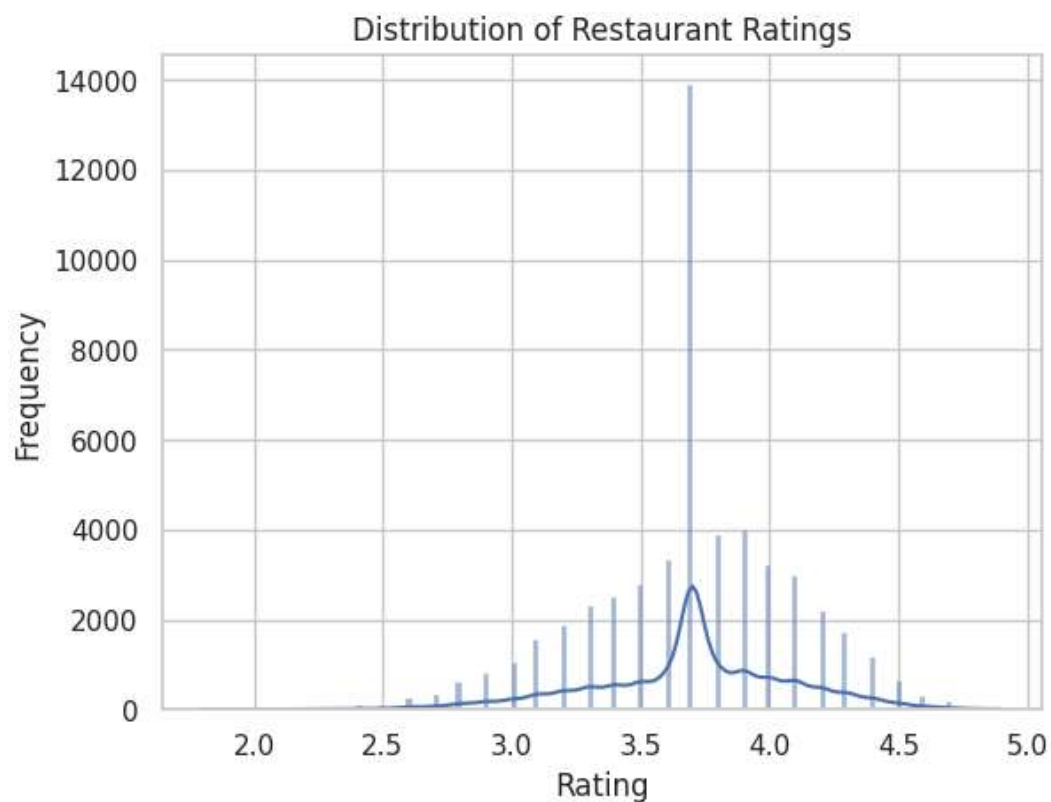
```
data1.info()
```

```
↗ <class 'pandas.core.frame.DataFrame'>
RangeIndex: 51717 entries, 0 to 51716
Data columns (total 10 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   online_order                          51717 non-null  int64
1   book_table                            51717 non-null  int64
2   rate                                  51717 non-null  float64
3   votes                                51717 non-null  int64
4   rest_type                             51717 non-null  object
5   dish_liked                            51717 non-null  object
6   cuisines                              51717 non-null  object
7   approx_costfor_two_people             51717 non-null  int64
8   listed_intype                         51717 non-null  object
9   listed_incity                         51717 non-null  object
dtypes: float64(1), int64(4), object(5)
memory usage: 3.9+ MB
```

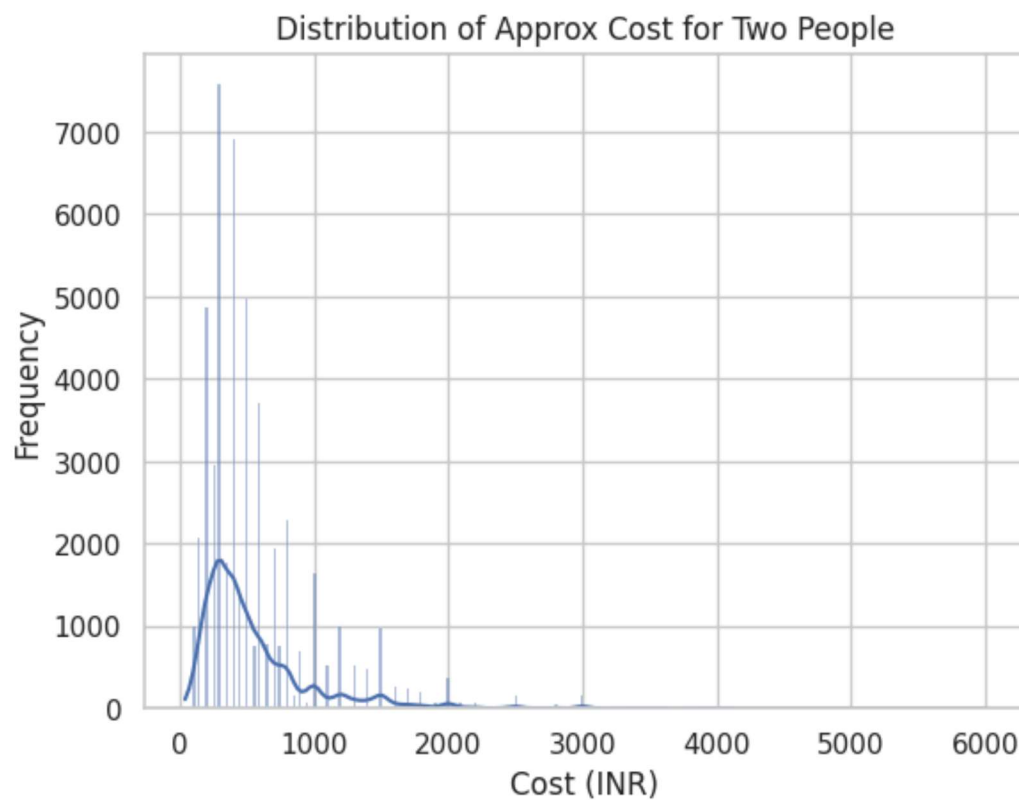
## ✓ Step 2: Exploratory Data Analysis (EDA)

We perform univariate and bivariate analysis to understand the distribution and relationships of key features such as rate, approx\_costfor\_two\_people, votes, and other categorical variables.

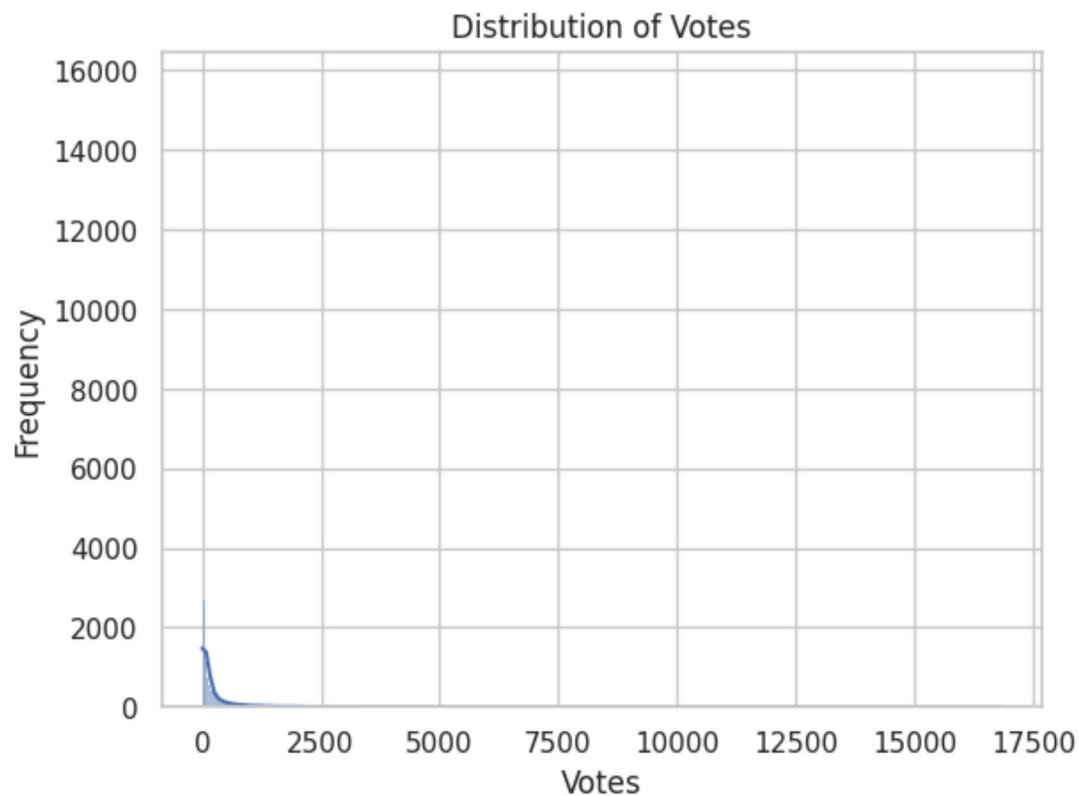
```
# Univariate Analysis
# Distribution of 'rate', 'approx_costfor_two_people', and 'votes'
sns.histplot(data1['rate'], kde=True)
plt.title('Distribution of Restaurant Ratings')
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.show()
```



```
sns.histplot(data1['approx_costfor_two_people'], kde=True)
plt.title('Distribution of Approx Cost for Two People')
plt.xlabel('Cost (INR)')
plt.ylabel('Frequency')
plt.show()
```



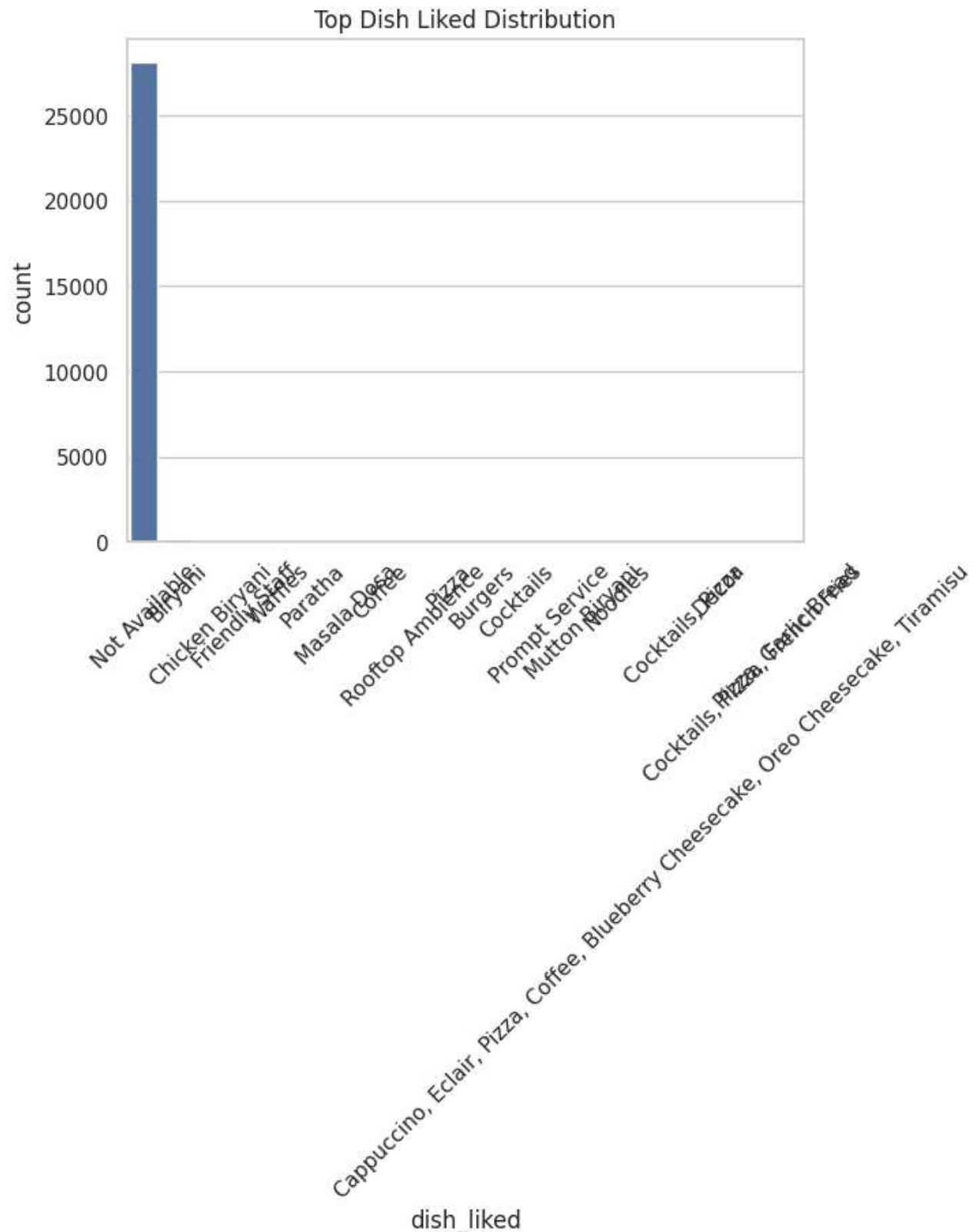
```
sns.histplot(data1['votes'], kde=True)
plt.title('Distribution of Votes')
plt.xlabel('Votes')
plt.ylabel('Frequency')
plt.show()
```



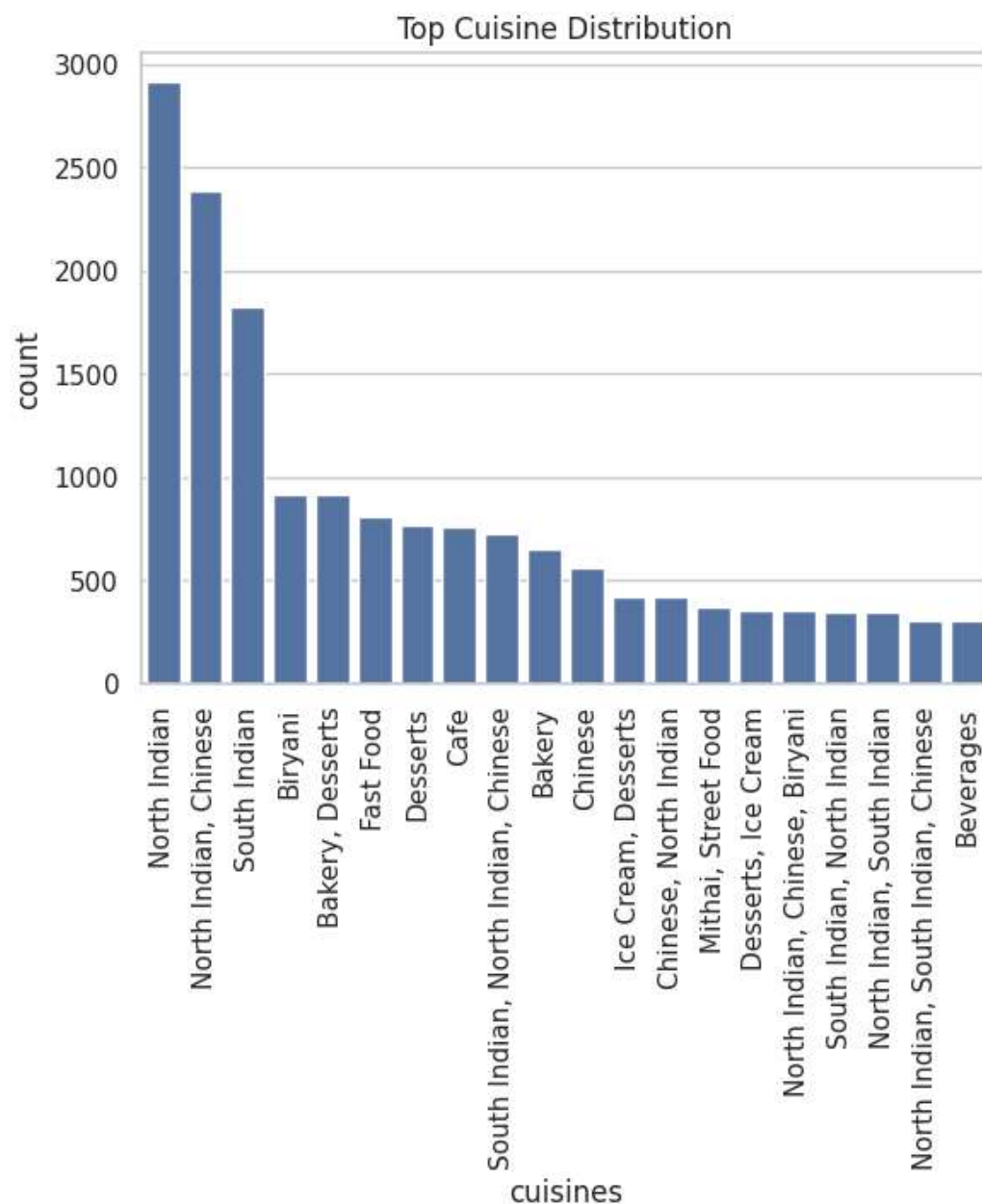
```
# Limit the number of top N categories to visualize
top_n = 20

# Categorical variables analysis with top N categories for better performance
# Plotting 'dish_liked'
top_dish_liked = data1['dish_liked'].value_counts().head(top_n)
sns.countplot(x='dish_liked', data=data1, order=top_dish_liked.index)
plt.title('Top Dish Liked Distribution')
plt.xticks(rotation=45)
plt.show()
```

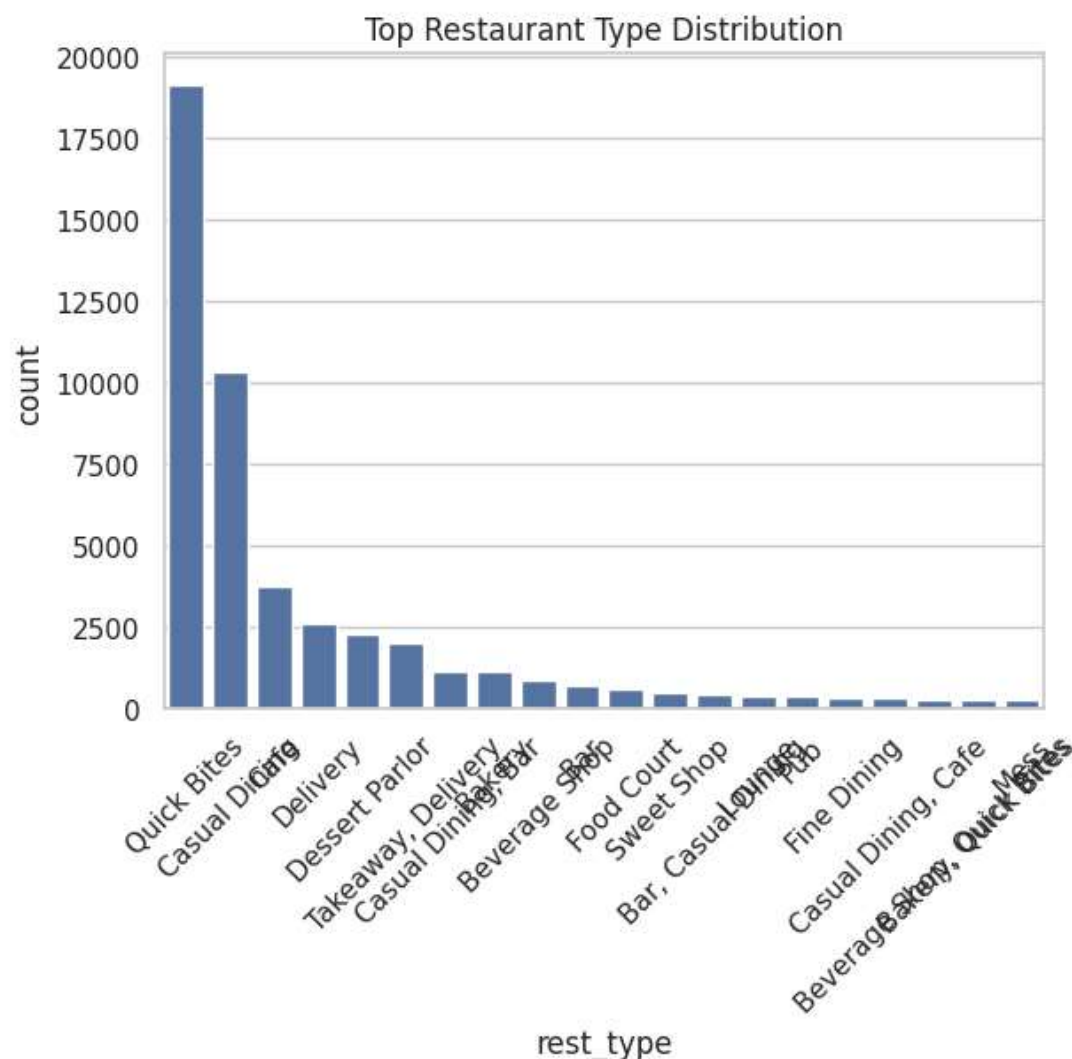




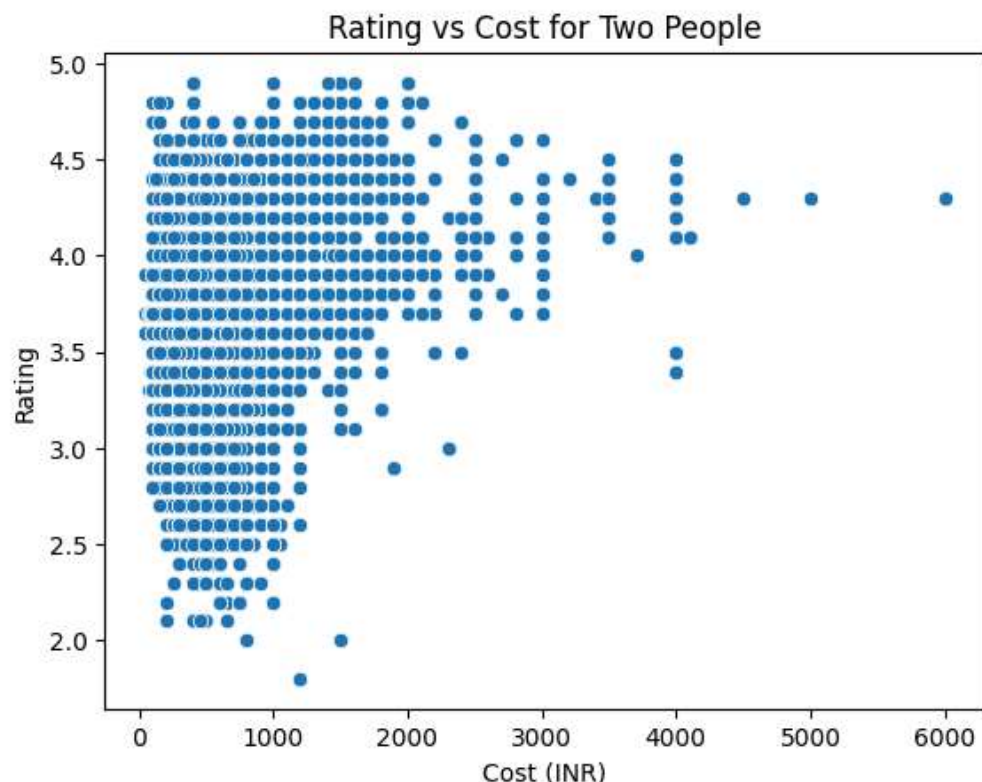
```
# Plotting 'cuisines'
top_cuisines = data1['cuisines'].value_counts().head(top_n)
sns.countplot(x='cuisines', data=data1, order=top_cuisines.index)
plt.title('Top Cuisine Distribution')
plt.xticks(rotation=90)
plt.show()
```



```
# Plotting 'rest_type'
top_rest_type = data1['rest_type'].value_counts().head(top_n)
sns.countplot(x='rest_type', data=data1, order=top_rest_type.index)
plt.title('Top Restaurant Type Distribution')
plt.xticks(rotation=45)
plt.show()
```



```
# Bivariate Analysis: Correlation between 'rate' and 'approx_costfor_two_people'
sns.scatterplot(x='approx_costfor_two_people', y='rate', data=data1)
plt.title('Rating vs Cost for Two People')
plt.xlabel('Cost (INR)')
plt.ylabel('Rating')
plt.show()
```



### ✓ Step 3: Merge Location Data for Mapping

In this step, we will merge the restaurant data ( data1 ) with the geographical coordinates ( data2 ) based on the column `listed_incity`. This will provide us with the geographical coordinates for plotting restaurant locations on a map.

```
# Merge restaurant data (data1) with geographical data (data2) on 'listed_incity'
merged_df = pd.merge(data1, data2, on='listed_incity', how='left')
```

```
# Verify the merged dataset
merged_df.head()
```

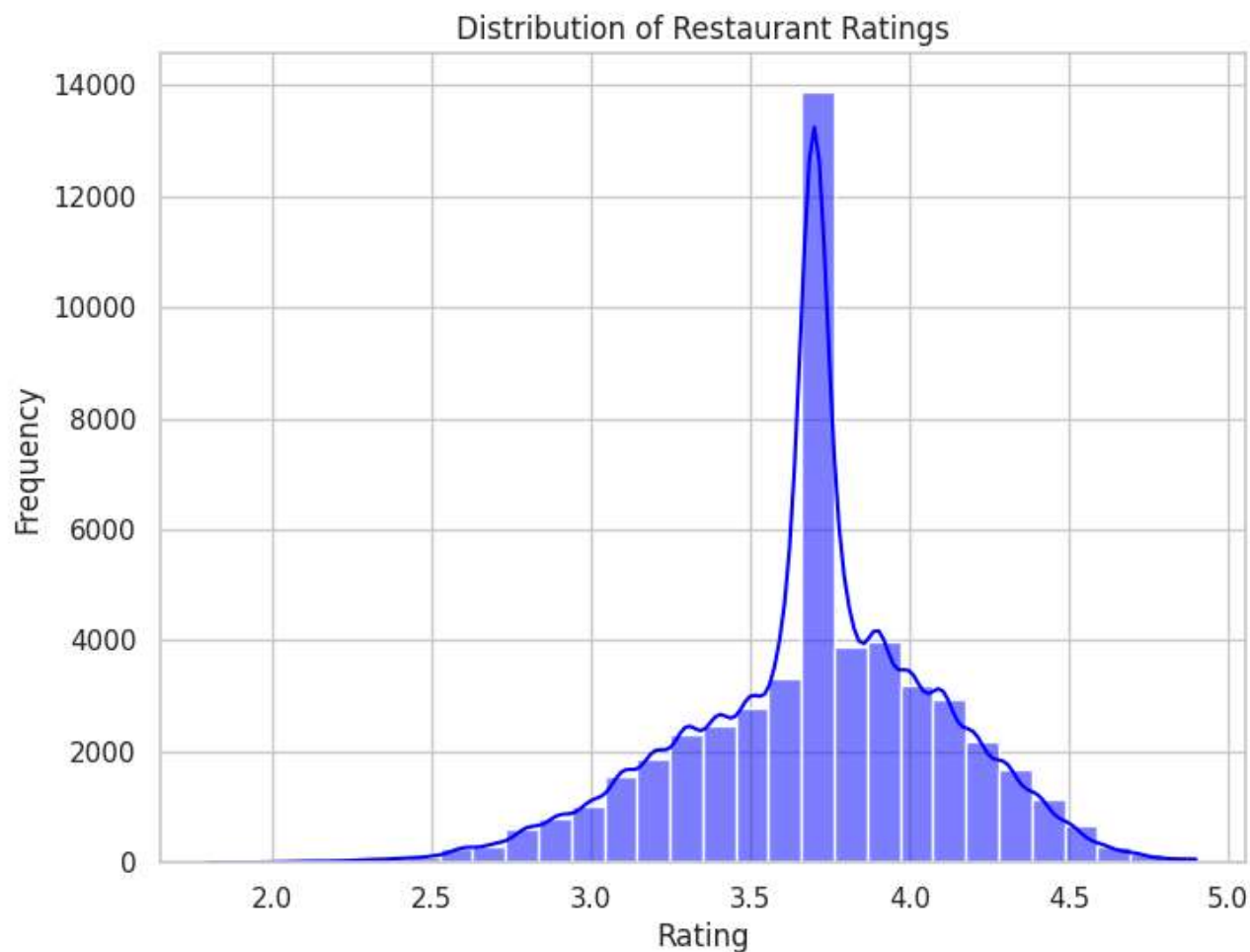
```
# Save the merged dataframe as a CSV file to download
merged_df.to_csv('/content/merged_dataset.csv', index=False)
```

### ✓ Step 4: Extract Actionable Insights

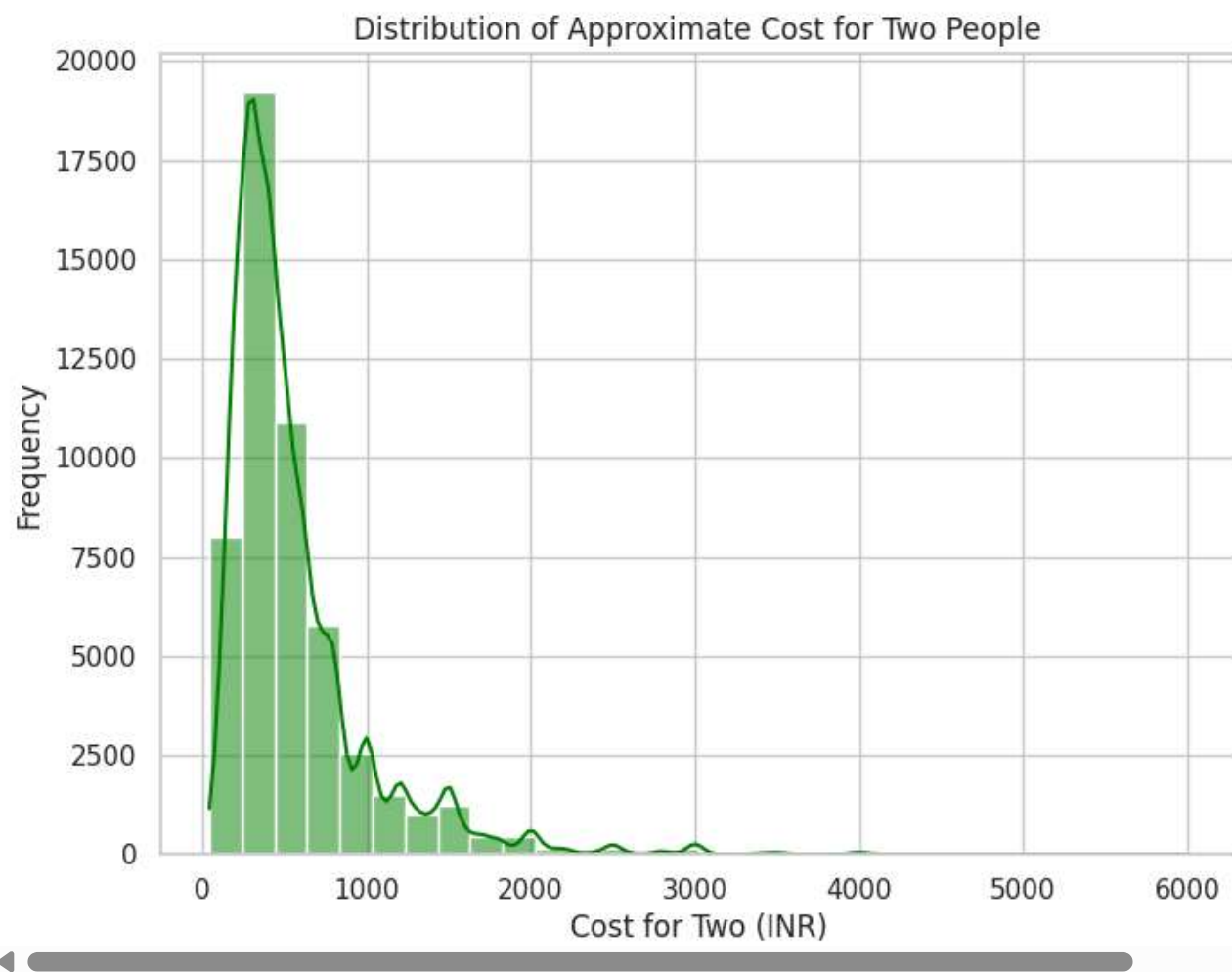
```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Set up plotting style
sns.set(style="whitegrid")
```

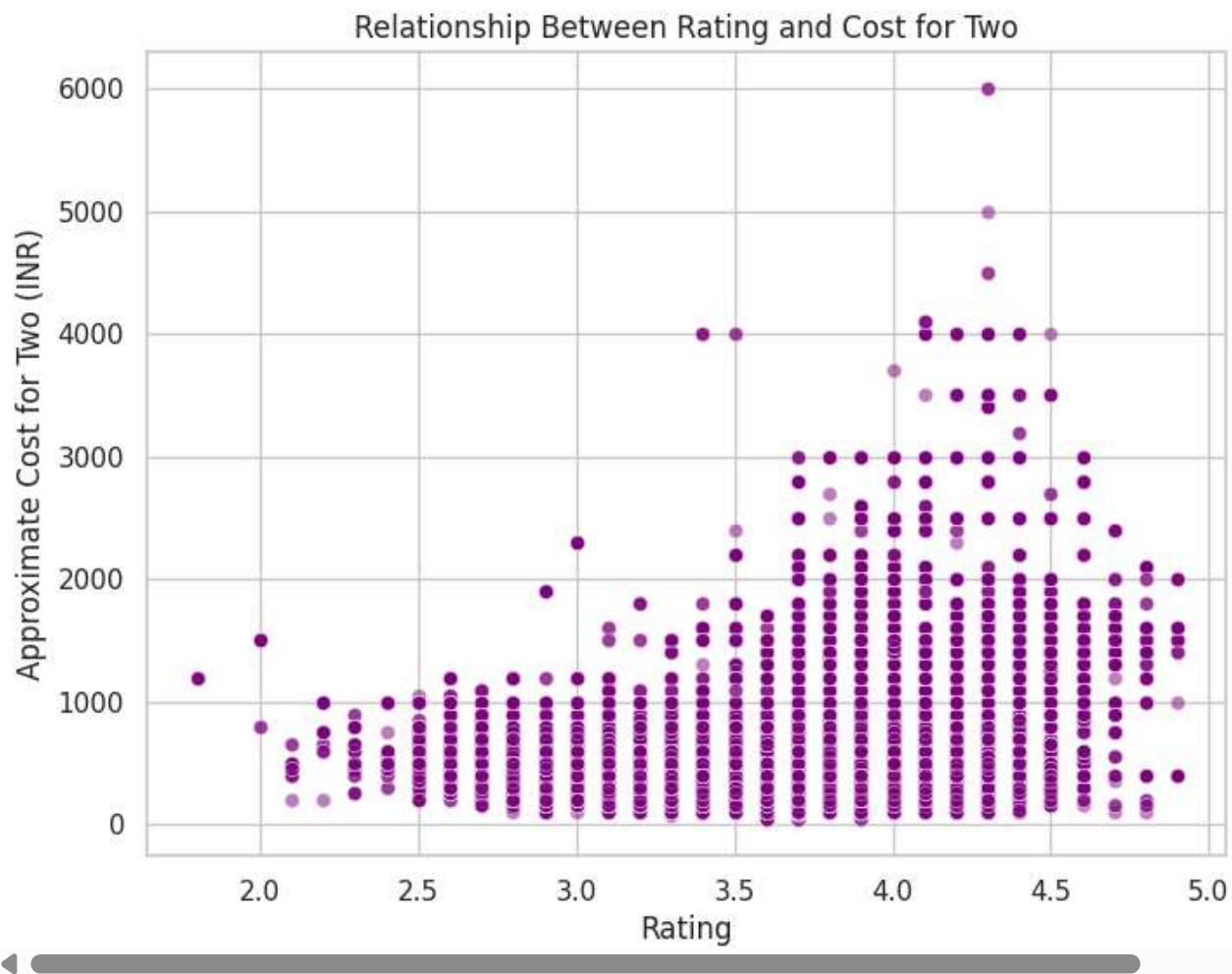
```
# Visualizing the distribution of restaurant ratings
plt.figure(figsize=(8, 6))
sns.histplot(merged_df['rate'], bins=30, kde=True, color='blue')
plt.title('Distribution of Restaurant Ratings')
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.show()
```




```
# Visualizing the distribution of approximate cost for two people
plt.figure(figsize=(8, 6))
sns.histplot(merged_df['approx_costfor_two_people'], bins=30, kde=True, color='green')
plt.title('Distribution of Approximate Cost for Two People')
plt.xlabel('Cost for Two (INR)')
plt.ylabel('Frequency')
plt.show()
```



```
# Visualizing the relationship between rating and cost
plt.figure(figsize=(8, 6))
sns.scatterplot(x='rate', y='approx_costfor_two_people', data=merged_df, color='purple', alpha=0.5)
plt.title('Relationship Between Rating and Cost for Two')
plt.xlabel('Rating')
plt.ylabel('Approximate Cost for Two (INR)')
plt.show()
```



```
# Visualizing the number of restaurants by 'rest_type'
plt.figure(figsize=(10, 6))
rest_type_counts = merged_df['rest_type'].value_counts()
sns.barplot(x=rest_type_counts.index, y=rest_type_counts.values, palette='viridis')
plt.title('Number of Restaurants by Type')
plt.xlabel('Restaurant Type')
plt.ylabel('Number of Restaurants')
plt.xticks(rotation=45, ha='right')
plt.show()
```

 <ipython-input-33-f629ba8a9a53>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `

```
sns.barplot(x=rest_type_counts.index, y=rest_type_counts.values, palette='viridis')
```

