In [4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

These libraries are commonly used together in data science and analytics workflows because they provide a powerful combination of data manipulation, numerical computation, and data visualization capabilities. By importing these libraries, you can: - Manipulate and analyze data with Pandas - Perform numerical computations with NumPy - Create visualizations with Matplotlib and Seaborn This sets the stage for a wide range of data science tasks, from data cleaning and preprocessing to machine learning and visualization.

In [5]:
```
dataframe = pd.read_csv("Zomato data .csv")
print(dataframe.head())
```

```
                   name online_order book_table   rate  votes  \
0                 Jalsa          Yes        Yes  4.1/5    775
1        Spice Elephant          Yes         No  4.1/5    787
2       San Churro Cafe          Yes         No  3.8/5    918
3  Addhuri Udupi Bhojana           No         No  3.7/5     88
4          Grand Village           No         No  3.8/5    166

   approx_cost(for two people) listed_in(type)
0                          800          Buffet
1                          800          Buffet
2                          800          Buffet
3                          300          Buffet
4                          600          Buffet
```

* pd.read_csv("Zomato data .csv"): - This line reads the CSV file named "Zomato data .csv" into a Pandas DataFrame. - The read_csv() function is a part of the pandas library, which is why we use the pd alias. * dataframe = ...: - The resulting DataFrame is assigned to a variable named dataframe. * print(dataframe.head()): - This line prints the first few rows of the DataFrame using the head() method. - By default, head() returns the first 5 rows of the DataFrame. You can specify a different number of rows by passing an integer argument, e.g., head(10).

In [6]:
```
def handleRate(value):
        value=str(value).split('/')
        value=value[0];
        return float(value)

dataframe['rate']=dataframe['rate'].apply(handleRate)
print(dataframe.head())
#Before proceeding, let’s convert the data type of the “rate” column to float and remove the denominator.
```

```
                    name online_order book_table  rate  votes  \
0                  Jalsa          Yes        Yes   4.1    775
1         Spice Elephant          Yes         No   4.1    787
2        San Churro Cafe          Yes         No   3.8    918
3  Addhuri Udupi Bhojana           No         No   3.7     88
4          Grand Village           No         No   3.8    166

   approx_cost(for two people) listed_in(type)
0                          800          Buffet
1                          800          Buffet
2                          800          Buffet
3                          300          Buffet
4                          600          Buffet
```

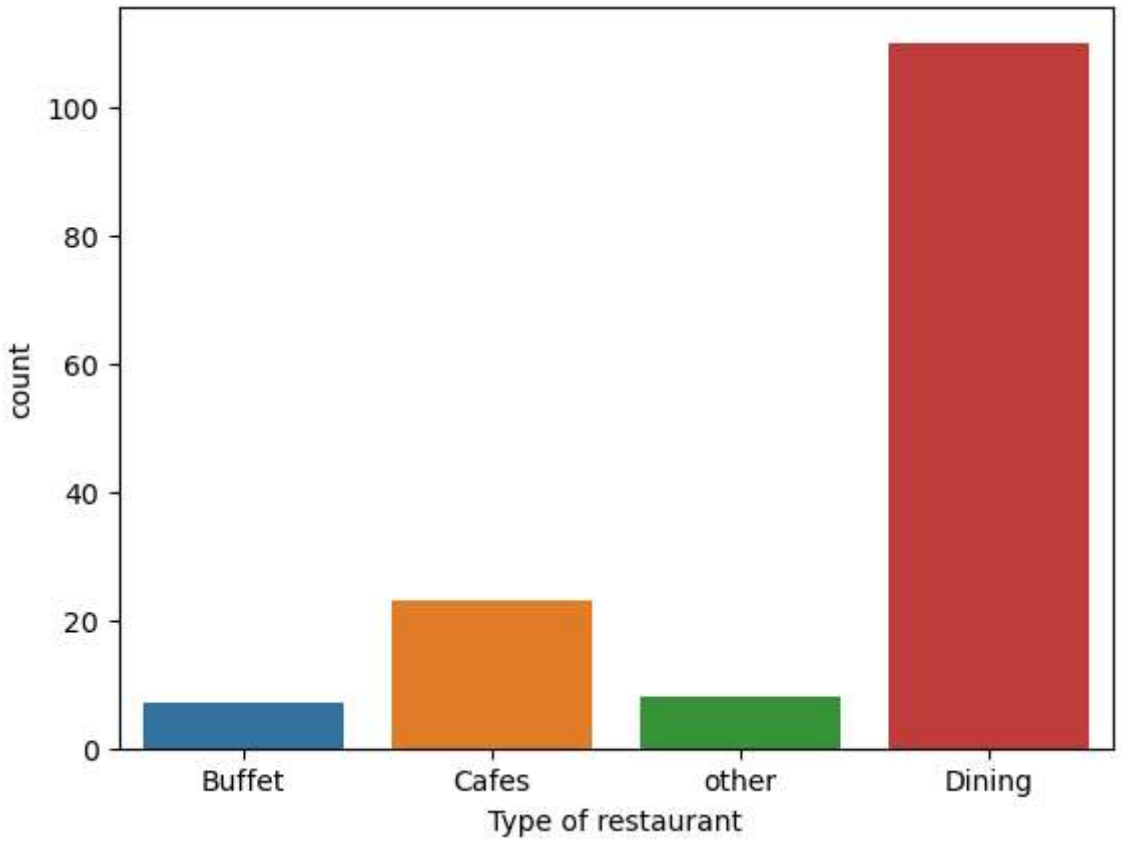
In [8]:
```
#To obtain a summary of the data frame, you can use the following code:-
dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 148 entries, 0 to 147
Data columns (total 7 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   name                         148 non-null    object
 1   online_order                 148 non-null    object
 2   book_table                   148 non-null    object
 3   rate                         148 non-null    float64
 4   votes                        148 non-null    int64
 5   approx_cost(for two people)  148 non-null    int64
 6   listed_in(type)              148 non-null    object
dtypes: float64(1), int64(2), object(4)
memory usage: 8.2+ KB
```

We will now examine the data frame for the presence of any null values. This stage scans each column to see whether there are any missing values or empty cells. This allows us to detect any potential data gaps that must be addressed. There is no NULL value in dataframe.

In [10]:
```
sns.countplot(x=dataframe['listed_in(type)'])
plt.xlabel("Type of restaurant")
#Conclusion: The majority of the restaurants fall into the dining category.
```

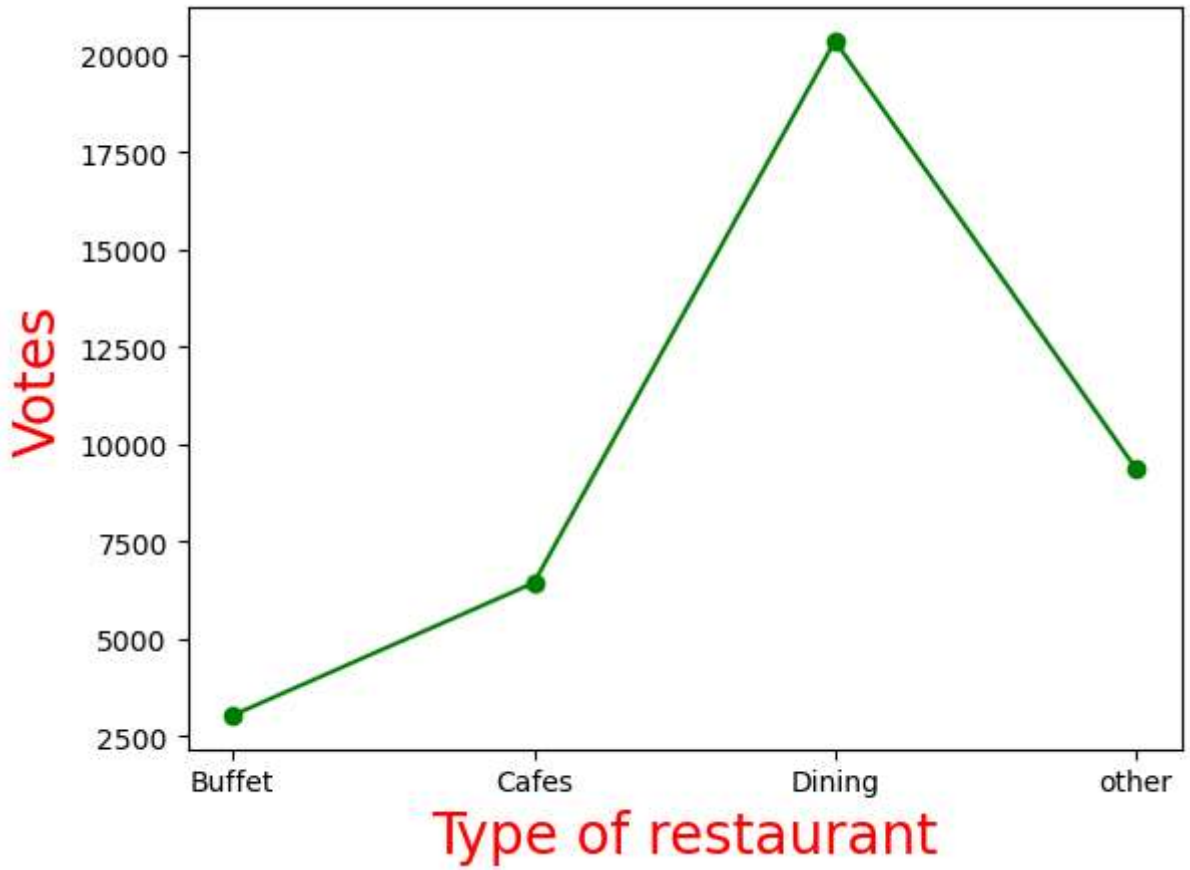
Out[10]:
```
Text(0.5, 0, 'Type of restaurant')
```



* sns.countplot(): - This function creates a count plot, where the x-axis represents the unique values in the specified column, and the y-axis represents the frequency of each value. * x=dataframe['listed_in(type)']: - This specifies the column in your DataFrame that you want to visualize. In this case, it's the listed_in(type) column, which presumably contains the type of restaurant (e.g., Buffet, cafes, dining, other). * plt.xlabel("Type of restaurant"): - This sets the label for the x-axis, making it clear what the plot is showing. When you run this code, you should see a bar chart with the following characteristics: - The x-axis will display the unique values from the listed_in(type) column. - The y-axis will display the frequency of each value, represented by the height of the bars. - The plot will give you a quick visual sense of the distribution of restaurant types in your data.

In [15]:
```
grouped_data = dataframe.groupby('listed_in(type)')['votes'].sum()
result = pd.DataFrame({'votes': grouped_data})
plt.plot(result, c="green", marker="o")
plt.xlabel("Type of restaurant", c="red", size=20)
plt.ylabel("Votes", c="red", size=20)
#This code can be used to visualize the popularity of different types of restaurants in a city, based on user votes.
#Conclusion: Dining restaurants are preferred by a larger number of individuals.
```

Out[15]:
```
Text(0, 0.5, 'Votes')
```



* Grouping data - dataframe.groupby('listed_in(type)'): groups the data by the 'listed_in(type)' column, which contains the type of restaurant. - ['votes']: selects only the 'votes' column from the grouped data. - .sum(): calculates the sum of votes for each group. * Creating a new DataFrame - pd.DataFrame(): creates a new DataFrame. - {'votes': grouped_data}: creates a dictionary with a single key-value pair, where the key is 'votes' and the value is the grouped data. * Plotting - plt.plot(): creates a line plot. - result: passes the new DataFrame as the data to plot. - c="green": sets the line color to green. - marker="o": adds circle markers to the plot. * Adding labels - plt.xlabel(): adds a label to the x-axis. - plt.ylabel(): adds a label to the y-axis. - "Type of restaurant": sets the x-axis label text. - "Votes": sets the y-axis label text. - c="red": sets the label color to red. - size=20: sets the label font size to 20.

In [16]:
```
#Now we will determine the restaurant’s name that received the maximum votes based on a given dataframe.
max_votes = dataframe['votes'].max()
restaurant_with_max_votes = dataframe.loc[dataframe['votes'] == max_votes, 'name']
print("Restaurant(s) with the maximum votes:")
print(restaurant_with_max_votes)
```

```
Restaurant(s) with the maximum votes:
38    Empire Restaurant
Name: name, dtype: object
```

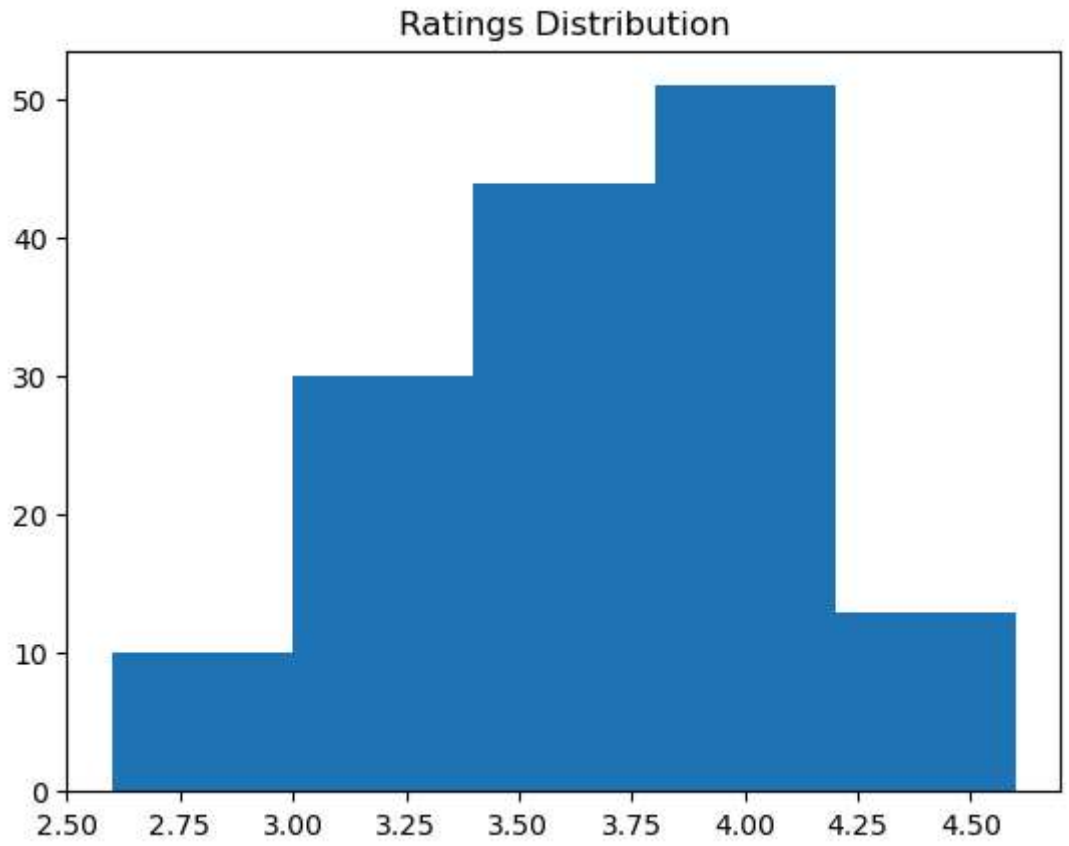
* Find the Maximum Number of Votes - This line of code finds the maximum value in the votes column of the dataframe. - The max() function is used to find the maximum value. * Find the Restaurant(s) with the Maximum Votes - This line of code finds the restaurant(s) with the maximum number of votes. - The loc[] function is used to access a group of rows and columns by label(s). - The condition dataframe['votes'] == max_votes filters the rows where the number of votes is equal to the maximum number of votes. - The 'name' column is selected to get the names of the restaurants. * Print the Result - The first print() statement prints a header message. - The second print() statement prints the names of the restaurant(s) with the maximum number of votes.

In [19]:
```
#Lets explore the online_order column
sns.countplot(x=dataframe['online_order'])
#Conclusion: This suggests that a majority of the restaurants do not accept online orders.
```

Out[19]:
```
<Axes: xlabel='online_order', ylabel='count'>
```

- sns: Seaborn, a Python data visualization library based on matplotlib. - countplot: a Seaborn function that creates a count plot. - x=dataframe['online_order']: the input data, which is a column named online_order from a Pandas DataFrame dataframe. This column is assumed to be categorical (i.e., it contains discrete values). When you run this code, Seaborn will create a bar chart with the unique values of online_order on the x-axis and the count of each value on the y-axis. The height of each bar represents the frequency of each value in the online_order column. For example, if the online_order column contains values like ['yes', 'no', 'yes', 'no', 'yes', ...], the resulting count plot would show two bars: one for yes and one for no, with the height of each bar representing the number of times each value appears in the column.

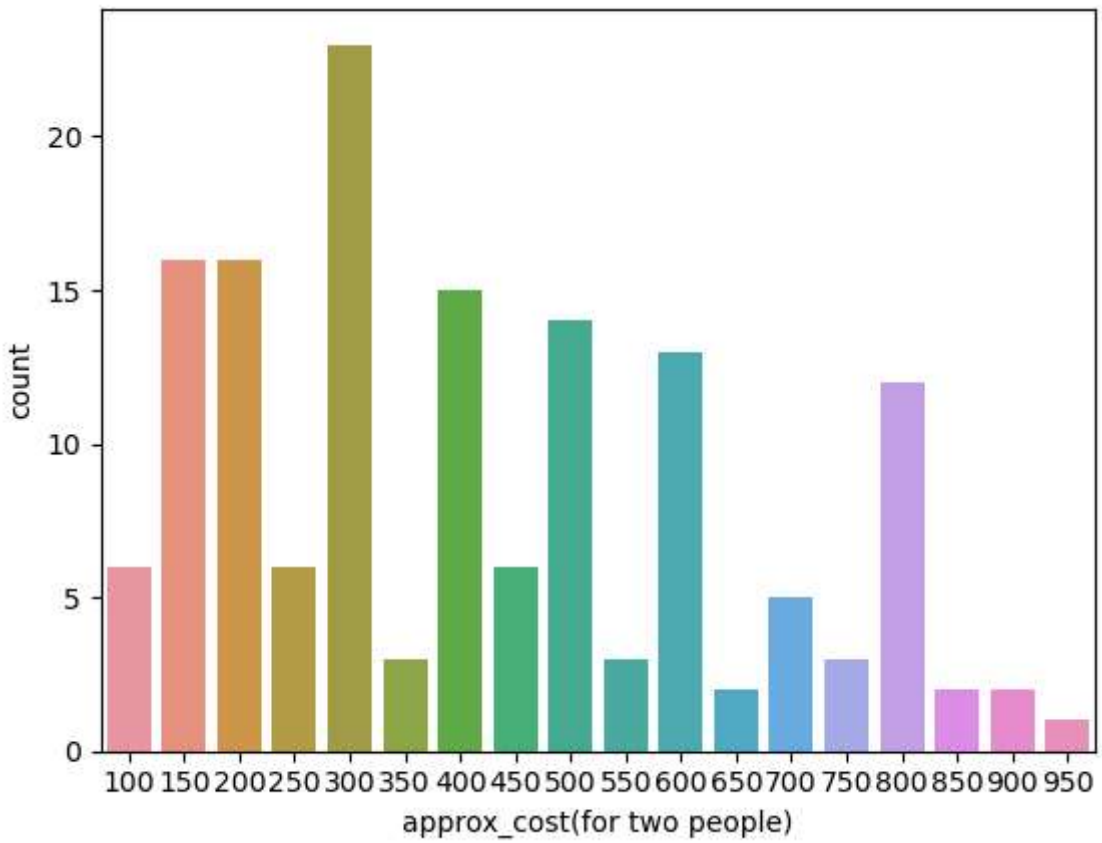
In [22]:
```
#Let's explore the rate column.
plt.hist(dataframe['rate'],bins=5)
plt.title("Ratings Distribution")
plt.show()
#Conclusion: The majority of restaurants received ratings ranging from 3.5 to 4.
```



- plt.hist(dataframe['rate'], bins=5): Creates a histogram of the rate column in the DataFrame, with 5 bins (i.e., 5 bars in the histogram). The hist function calculates the frequency of values in each bin. - plt.title("Ratings Distribution"): Sets the title of the histogram to "Ratings Distribution". - plt.show(): Displays the histogram. The resulting plot will show the distribution of ratings, with the x-axis representing the rating values and the y-axis representing the frequency (or count) of each rating value. The 5 bins will group the ratings into 5 ranges, giving a rough idea of how the ratings are distributed.

In [23]:
```
#Let's explore the approx_cost(for two people) column.
couple_data=dataframe['approx_cost(for two people)']
sns.countplot(x=couple_data)
#Conclusion: The majority of couples prefer restaurants with an approximate cost of 300 rupees.
```

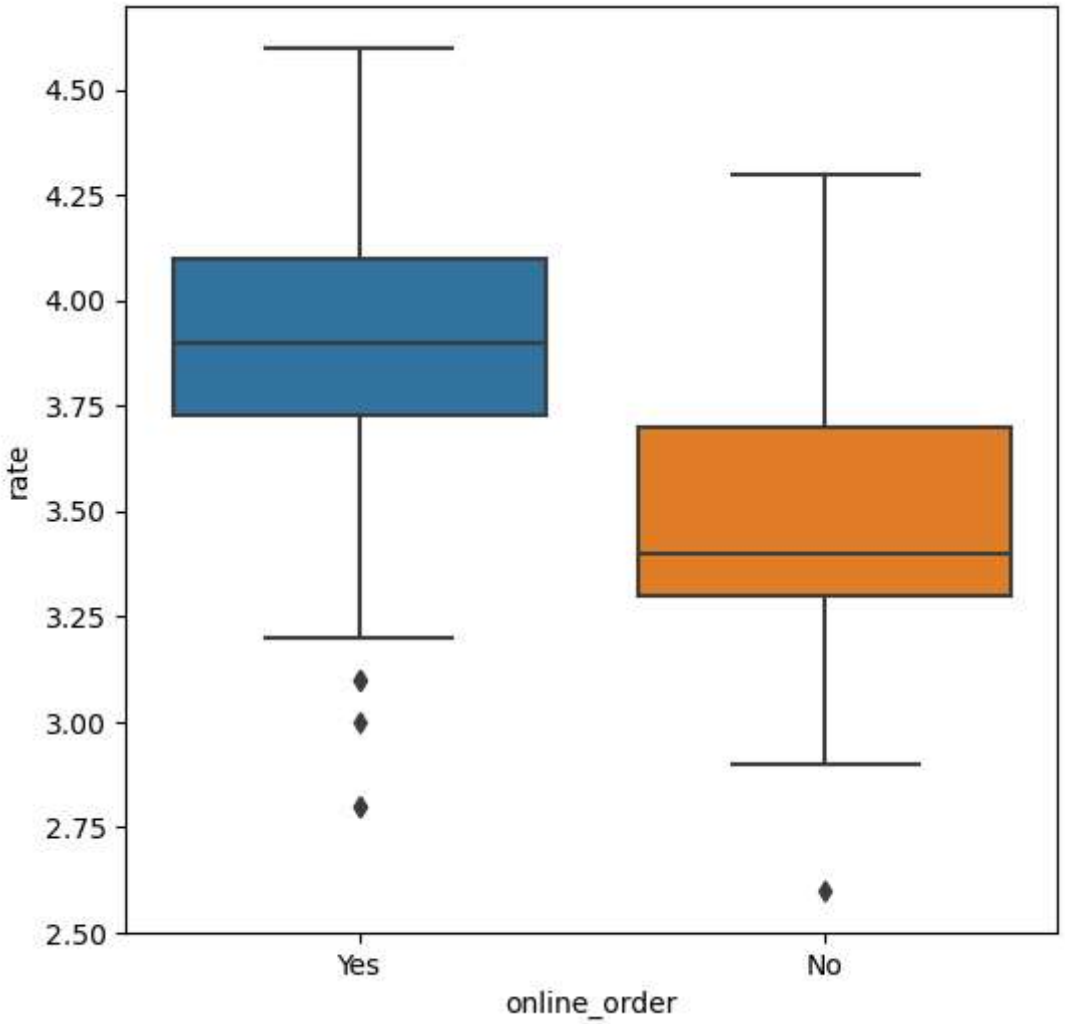
Out[23]: <Axes: xlabel='approx_cost(for two people)', ylabel='count'>



- couple_data = dataframe['approx_cost(for two people)']: This line extracts a column named 'approx_cost(for two people)' from a pandas DataFrame (dataframe) and assigns it to a variable named couple_data. This column likely contains data on the approximate cost for two people. - sns.countplot(x=couple_data): This line creates a count plot using seaborn's countplot function. The x parameter is set to couple_data, which means the plot will display the count of each unique value in the couple_data column. This code is used to visualize the distribution of approximate costs for two people in a dataset. The resulting plot will show the frequency of each cost value, providing insights into the most common costs for two people.

In [24]:
```
#Now we will examine whether online orders receive higher ratings than offline orders.
plt.figure(figsize = (6,6))
sns.boxplot(x = 'online_order', y = 'rate', data = dataframe)
#CONCLUSION: Offline orders received lower ratings in comparison to online orders, which obtained excellent ratings.
```

Out[24]: <Axes: xlabel='online_order', ylabel='rate'>



* plt.figure(figsize = (6,6)): - plt is an alias for the matplotlib.pyplot module, which is a popular data visualization library in Python. - figure() is a function that creates a new figure. - figsize = (6,6) sets the size of the figure in inches. In this case, the figure will be 6 inches wide and 6 inches tall. * sns.boxplot(x = 'online_order', y = 'rate', data = dataframe): - sns is an alias for the seaborn library, which is built on top of matplotlib and provides a high-level interface for drawing attractive and informative statistical graphics. - boxplot() is a function that creates a boxplot, which is a graphical representation of the distribution of a set of data. - x = 'online_order' specifies the column in the dataframe that will be used for the x-axis of the boxplot. - y = 'rate' specifies the column in the dataframe that will be used for the y-axis of the boxplot. - data = dataframe specifies the dataframe that contains the data to be plotted. This code will create a 6x6 inch boxplot where the x-axis represents 'online_order' and the y-axis represents 'rate', using the data from the specified dataframe.

In [28]:
```
pivot_table = dataframe.pivot_table(index='listed_in(type)', columns='online_order', aggfunc='size', fill_value=0)
sns.heatmap(pivot_table, annot=True, cmap="YlGnBu", fmt='d')
plt.title("Heatmap")
```

```
plt.xlabel("Online Order")
plt.ylabel("Listed In (Type)")
plt.show()
#CONCLUSION: Dining restaurants primarily accept offline orders, whereas cafes primarily receive online orders.
#This suggests that clients prefer to place orders in person at restaurants, but prefer online ordering at cafes.
```

* pivot_table = dataframe.pivot_table(index='listed_in(type)', columns='online_order', aggfunc='size', fill_value=0): - Creates a pivot table from the dataframe object. - Sets the index to the values in the listed_in(type) column. - Sets the columns to the values in the online_order column. - Uses the size aggregation function to count the number of rows for each combination of index and column values. - Fills missing values with 0. * sns.heatmap(pivot_table, annot=True, cmap="YlGnBu", fmt='d'): - Creates a heatmap from the pivot table using the heatmap function from Seaborn. - Displays the numerical values in each cell using annot=True. - Uses the "YlGnBu" colormap to color the cells. - Formats the numerical values as integers using fmt='d'. * plt.title("Heatmap"), plt.xlabel("Online Order"), and plt.ylabel("Listed In (Type)"): - Sets the title, x-axis label, and y-axis label of the plot. * plt.show(): - Displays the plot. The resulting heatmap shows the count of rows for each combination of listed_in(type) and online_order values, with the counts displayed as integers in each cell. The color of each cell represents the count, with higher counts corresponding to darker colors.