- **Importing Libraries**

Lines 1-13: Essential libraries are imported

- ❖ **Requests, BeaurifulSoup, pandas, numpy and re : These libraries are for data extraction and cleaning**
- ❖ **Selenium and webdriver : Used to automate a web browser for javascript-render pages(e.g for scrolling)**
- ❖ **Warnings: supress unnessary warnimgs**

Initialize Selenium WebDriver

Line 17: Initializes a Chrome WebDriver instance to control a browser.

Line 18: uses the driver.get() method to open makemytrip hotels listing page goa

Extract Page Source with BeautifulSoup

Line 19: Extracts the current page's HTML source.

Line 20: Parses the HTML using BeautifulSoup to make it easier to search through.

Scroll to Load More Content

1. **Lines 22-33: Code to scroll the page, which is necessary for pages where content loads as you scroll.**
    - **Line 25: Retrieves the initial scroll height.**
    - **Lines 27-32: Scrolls down in a loop, waits to load new content, and checks if there's more content by comparing scroll heights.**
    - **Line 29: Waits 6 seconds after scrolling to allow content to load fully.**
    - **Line 33: Updates the scroll height, breaks out of the loop when no new content appears.**

Extract Data into Lists

1. **Line 34: Extracts the updated page source after scrolling.**
2. **Line 36: Re-parses the full page source using BeautifulSoup.**

Find Hotel Names

1. **Line 38: Initializes an empty Names list for storing hotel names.**
2. **Lines 40-46: Loops over div elements containing hotel details.**
    - **Line 41: Uses regex to check for valid hotel names and adds them to Names. Adds NaN if no match is found.**

Debugging Code for Verification

1. **Lines 47-51: Debugging code to check the content extracted (commented out).**
2. **Lines 52-55: Cleans up the hotel names by removing extraneous text (also commented out).**

Initialize Lists for Hotel Details

1. **Lines 57-59: Creates empty lists for hotel details like Hotel, Location, Ratings, and walk.**

Extracting Hotel Details

1. **Lines 61-78: Loops through hotel entries on the page to extract and store specific details.**
    - **Line 63: Finds hotel name from span with class wordBreak appendRight10.**
    - **Line 65: Extracts the hotel location from span with class blueText.**
    - **Line 67: Retrieves rating value from span with class ratingValue.**
    - **Line 69: Extracts nearby location description (such as walking distance) from span with class latoRegular.**

Unused Example Code

1. **Lines 80-84: Another attempt at extracting information from different div elements (commented out).**

Ratings, Reviews, and Counts Lists

1. **Lines 86-88: Initializes lists for ratings, number of ratings, and reviews to store details for each hotel.**

This script automates scrolling through a webpage, collects details from each listed hotel, and stores them in respective lists.

Certainly! Here's a breakdown of each section and step in your code.

Import Libraries and Set Up

```python

import requests

from bs4 import BeautifulSoup

import pandas as pd

import numpy as np

import time

import re

from selenium import webdriver

import warnings

warnings.filterwarnings('ignore')

```

1. **Requests: To handle HTTP requests.**

2. **BeautifulSoup: For parsing HTML and XML documents.**

3. **Pandas: Used to work with data in DataFrame format.**

4. **NumPy: Provides support for numerical operations.**

5. **Time: For adding pauses, particularly in web scraping.**

6. **Regular Expressions (re): Used for searching patterns in text.**

7. **Selenium WebDriver: Automates browsing actions, e.g., scrolling.**

8. **Warnings: Ignores any warnings during code execution.**

Web Driver Setup

```python

driver = webdriver.Chrome()

driver.get('https://www.makemytrip.com/hotels/hotel-listing/?checkin=10142024&checkout=10162024&city=CTGOI&country=IN&locusId=CTGOI&locusType=city&regionNearByExp=3&roomStayQualifier=2e0e&rsc=1e2eundefinede&searchText=Goa')

soup = BeautifulSoup(driver.page_source)

```

1. **Initialize WebDriver: Opens the Chrome browser.**

2. **Navigate to URL: Opens the MakeMyTrip hotel listings page.**

3. **Create BeautifulSoup Object: Parses the page's source code, enabling easy data extraction.**

Scroll Automation

```python

last_height = driver.execute_script("return document.body.scrollHeight")

for _ in range(500):

    driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")

    time.sleep(6)

    new_height = driver.execute_script("return document.body.scrollHeight")

    if new_height == last_height:

        pass

    last_height = new_height

```

1. **Initial Scroll Height**: Captures the starting height of the page.

2. **Loop Through Scrolls**: Scrolls down 500 times, waiting for the content to load (`time.sleep(6)`) after each scroll.

3. **Update Scroll Height**: Checks the new height after each scroll and breaks if no new content loads (page end reached).

### Extracting Hotel Information

#### Names

```python

Names = []

for i in soup.find_all('div', class_='flexOne appendLeft20'):

    if re.findall("^[A-Za-z0-9 &',\-.a]+", i.text):

        Names.append(re.findall("^[A-Za-z0-9 &',\-.a]+", i.text)[0])

**else:**

**Names.append(np.nan)**

**```**

**1. **Initialize `Names` List**: Holds the names of hotels.**

**2. **Loop Through Elements**: Finds and extracts hotel names from the specific div class.**

**3. **Use Regex**: Filters the extracted text based on the pattern to include letters, numbers, and special characters in hotel names.**

**#### Ratings, Number of Ratings, and Reviews**

**```python**

**Ratings = []**

**No_of_Ratings = []**

**No_of_Reviews = []**

**for i in soup.find_all('div', class_="textRight priceDetailsNewPers padding16"):**

**if re.findall('(?:Very Good|Excellent|Good|Fair|Poor)(\d\.\d)', i.text):**

**Ratings.append(re.findall('(?:Very Good|Excellent|Good|Fair|Poor)(\d\.\d)', i.text)[0])**

**else:**

**Ratings.append(np.nan)**

**if re.findall('(\d+)\sRatings', i.text):**

**No_of_Ratings.append(re.findall('(\d+)\sRatings', i.text)[0])**

**else:**

**No_of_Ratings.append(np.nan)**

**if re.findall('(\d+)\sreviews', i.text):**

**No_of_Reviews.append(re.findall('(\d+)\sreviews', i.text)[0])**

**else:**

**No_of_Reviews.append(np.nan)**

**```**

**1. **Initialize Lists**: `Ratings`, `No_of_Ratings`, and `No_of_Reviews` store respective data.**

**2. **Extract Ratings**: Regex finds ratings in formats like "Very Good4.5" and stores the numerical rating.**

**3. **Extract Number of Ratings**: Finds the count of ratings based on the pattern.**

**4. **Extract Reviews**: Searches for the number of reviews. Missing values are handled by appending `np.nan`.**

**#### Pricing and Taxes**

**```python**

**Original_price = []**

**Discount_price = []**

**Taxes = []**

**for i in soup.find_all('div', class_="textRight priceDetailsNewPers padding16"):**

**if re.findall("₹(\s\d.{4})", i.text):**

**Original_price.append(re.findall("₹(\s\d.{4})", i.text)[0])**

```
    else:
        Original_price.append(np.nan)
    if re.findall("₹\s\d.{4}₹(\s\d.{4})", i.text):
        Discount_price.append(re.findall("₹\s\d.{4}₹(\s\d.{4})", i.text)[0])
    else:
        Discount_price.append(np.nan)
    if re.findall("₹\s(\d\,?\d+)\staxes", i.text):
        Taxes.append(re.findall("₹\s(\d\,?\d+)\staxes", i.text)[0])
    else:
        Taxes.append(np.nan)
```

1. **Original Price**: Extracts and stores the original price for each hotel.
2. **Discount Price**: Looks for the discounted price pattern.
3. **Taxes**: Extracts tax amounts, handling cases with commas.

Creating DataFrame

```python
df_1 = pd.DataFrame({
    "Hotel": Hotel,
    "Location": Location,
    "Ratings": Ratings,
    "No_of_Ratings": No_of_Ratings,
    "No_of_Reviews": No_of_Reviews,
```

"Original_price": Original_price,

"Discount_price": Discount_price,

"Taxes": Taxes

})

df_1.info()

df_1.isnull().sum()

df_1.tail()

```

1. **Create DataFrame: Organizes data into a structured format.**

2. **Check DataFrame Info: Shows the columns, data types, and non-null counts.**

3. **Null Count: Checks for any missing values.**

4. **View Tail: Displays the last few rows of the DataFrame for inspection.**

Saving DataFrame to Excel

```python

df_1.to_excel(r'C:\Users\DELL\Final Project\files\makemytripdf_1.xlsx')

```

- **Save Data: Exports the DataFrame to an Excel file at the specified path.**

DATA Cleaning

Loading and Cleaning Excel Data

```python

df_1 = pd.read_excel(r'C:\Users\DELL\Final Project\files\makemytripdf_1.xlsx')

df_1.drop(columns='Unnamed: 0', inplace=True)

df_1.info()

```

1. **Load Excel Data: Reads the saved Excel file into a DataFrame.**

2. **Drop Extra Column: Removes the automatically generated 'Unnamed: 0' column.**

3. **Inspect Data: Displays DataFrame information, showing it's ready for analysis.**

Certainly! Let's walk through each step of your code to understand how it works in merging building type and house rule data from different Excel files into a single DataFrame.

Step-by-Step Explanation

Step 1: Reading and Preparing the Data from `buildingtype_1df.xlsx`

1. **Read the Excel file:**

```python

df_2 = pd.read_excel(r'C:\Users\DELL\Final Project/files/buildingtype_1df.xlsx')

```

This reads the first file containing building types for various hotels into `df_2`.

2. **Drop Unwanted Columns:**

```python

df_2.drop(columns='Unnamed: 0', inplace=True)

```

Here, `Unnamed: 0` column is dropped, as it's usually an auto-generated index column that is unnecessary.

3. **Merge `df_1` with `df_2`:**

```python

df = pd.merge(df_1, df_2, on='Hotel', how='left')

```

This merges the original `df_1` DataFrame with `df_2` on the common column `'Hotel'`, keeping all rows from `df_1` (left join) and bringing in matching data from `df_2`.

Step 2: Reading and Merging Additional Building Type Data

1. **Load Data from `buildingtype_2df.xlsx`:**

```python

df_3 = pd.read_excel(r'C:\Users\DELL\Final Project/files/buildingtype_2df.xlsx')

df_3.drop(columns='Unnamed: 0', inplace=True)

df = pd.merge(df, df_3, on='Hotel', how='left')

```

This repeats the previous process for the second file, `buildingtype_2df.xlsx`. The data is read, the unnecessary column is dropped, and it is merged with the existing `df`.

2. **Combine `Building_Type` Columns:**

```python

df['Building_Type'] = df['Building_Type_x'].combine_first(df['Building_Type_y'])

df.drop(columns=['Building_Type_x', 'Building_Type_y'], inplace=True)

```

Since merging added `Building_Type_x` and `Building_Type_y` columns, `combine_first()` fills `Building_Type` with non-null values from `Building_Type_x` and falls back to `Building_Type_y` where necessary. The original columns are then dropped.

Step 3: Checking the Data Summary After Each Merge

After each merge, you check the structure of the data with:

```python

df.info()

```

This command provides an overview of the DataFrame, including the data types and number of non-null values in each column, helping you monitor the number of missing values in `Building_Type`.

Step 4: Repeating the Process for Other Building Type Files

The process is repeated with additional files (`buildingtype_3df.xlsx` to `buildingtype_6df.xlsx`). Each file is:

- Loaded into a temporary DataFrame (`df_4`, `df_5`, `df_6`, etc.).
- Cleaned by removing the `Unnamed: 0` column.
- Merged with the main DataFrame `df` on `'Hotel'` using `how='left'`.
- Combined with the `Building_Type` column using `combine_first()` to fill missing values, and the temporary columns (`Building_Type_x` and `Building_Type_y`) are dropped.

Step 5: Summarizing the `Building_Type` Column

You check the frequency of each building type and the number of null values:

```python
df.Building_Type.value_counts()
df.Building_Type.isnull().sum()
```

These commands help identify the number of occurrences for each building type and how many rows still have missing `Building_Type` values, helping you assess data completeness.

Summary

This approach efficiently combines building type data from multiple files into a single, comprehensive DataFrame by repeatedly loading, cleaning, merging, and updating the `Building_Type` column. After these steps, you can proceed to merge `House_Rule` or other columns following a similar process.

It looks like you've successfully merged multiple datasets into a single DataFrame, adding columns such as `House_Rule` and `hotel_type` for hotels in Goa. You've combined `House_Rule` and `hotel_type` columns from multiple files, handling missing values with `.combine_first()` and subsequently removing redundant columns like `House_Rule_x` and `House_Rule_y`.

To clean and prepare the data for further analysis, consider addressing the remaining missing values in columns such as `Ratings`, `No_of_Ratings`, `Discount_price`, `Taxes`, `Building_Type`, and `House_Rule`. Here are a few steps to proceed:

1. **Handling Missing Values:**

- For numeric columns like `Ratings`, `No_of_Ratings`, `No_of_Reviews`, and `Discount_price`, you might consider filling missing values with the median or mean, or dropping rows with a high percentage of missing data if applicable.

- For categorical columns like `House_Rule` and `Building_Type`, use the mode or create a new category like "Unknown."

2. **Data Type Conversion:**

- Convert price columns (e.g., `Original_price` and `Discount_price`) to numeric values after removing currency symbols and commas.

- Ensure all columns are in the correct data types for easy analysis.

3. **Save the Cleaned Data:**

- After making these changes, save the cleaned DataFrame to an Excel file or any other format you prefer for future analysis.

Let me know if you'd like help with code for any specific cleaning step, or if there's another step you're looking to address!

Certainly! Here's a breakdown of each step in your code and its purpose.

1. Importing the Necessary Packages

To begin with, you would need to import the relevant packages, such as `pandas` for data manipulation, which is assumed here to be already imported as `pd`.

2. Merging Star Category Data (4-Star Hotels)

1. **Reading the Star Category Data:**

```python

df_13 = pd.read_excel(r'C:\Users\DELL\Final Project/files/star_category_4df.xlsx')

```

- You read in an Excel file containing 4-star hotel information. This file has at least two columns: "Hotel" and "hotel_type," where `hotel_type` indicates that these hotels have been categorized as "4 star."

2. **Cleaning the Data:**

```python

df_13.drop(columns='Unnamed: 0', inplace=True)

```

- Here, you drop an unnecessary "Unnamed: 0" column, which may be a result of exporting or indexing during the data preparation phase.

3. **Merging the DataFrames:**

```python

df = pd.merge(df, df_13, on='Hotel', how='left')

```

- You perform a left merge between `df` (your primary dataset) and `df_13` (the 4-star category data). The merge is done based on the "Hotel" column to add the star category information from `df_13` into `df` wherever there is a match.

4. **Combining hotel_type Columns:**

```python

df['hotel_type'] = df['hotel_type_x'].combine_first(df['hotel_type_y'])

```

- Here, you use `combine_first()` to combine the values from `hotel_type_x` (existing data) and `hotel_type_y` (newly merged data). If `hotel_type_x` is `NaN`, it takes the value from `hotel_type_y`; otherwise, it keeps the existing value.

5. **Dropping Redundant Columns:**

```python

df.drop(columns=['hotel_type_x', 'hotel_type_y'], inplace=True)

```

- After merging and combining, you drop the intermediary columns `hotel_type_x` and `hotel_type_y` to clean up the dataset.

6. **Checking the Last Few Rows:**

```python

df.tail(4)

```

- This command displays the last four rows of the dataset, allowing you to verify the successful merging of 4-star hotel information.

7. **Counting Null Values:**

```python

df.isnull().sum()

```

- This line checks for missing values in each column, which helps to understand where further data cleaning might be necessary.

3. Merging Star Category Data (3-Star Hotels)

1. **Reading the 3-Star Category Data:**

```python

df_14 = pd.read_excel(r'C:\Users\DELL\Final Project/files/star_category_3df.xlsx')

```

- Here, you load an Excel file containing information on 3-star hotels.

2. **Dropping Unnecessary Columns:**

```python

df_14.drop(columns='Unnamed: 0', inplace=True)

```

- Similar to the previous step, you drop the "Unnamed: 0" column, which is unnecessary for the analysis.

3. **Merging 3-Star Data into Main Dataset:**

```python

df = pd.merge(df, df_14, on='Hotel', how='left')

```

- You merge the 3-star data (`df_14`) with the main dataset (`df`) based on the "Hotel" column, performing a left join to incorporate 3-star information wherever there is a match.

4. **Combining Columns for hotel_type**:

```python
df['hotel_type'] = df['hotel_type_x'].combine_first(df['hotel_type_y'])
```

- The `combine_first()` method is used again to merge hotel type information from the existing and new data, ensuring no information is lost.

5. **Dropping Temporary Columns**:

```python
df.drop(columns=['hotel_type_x', 'hotel_type_y'], inplace=True)
```

- As before, you remove the temporary columns used during the merge to keep the data tidy.

6. **Displaying the Last Few Rows**:

```python
df.tail(4)
```

- This command lets you view the last four rows to ensure the data from the 3-star file has been successfully merged.

7. **Checking Null Values After 3-Star Merge:**

```python

df.isnull().sum()

```

- By rechecking missing values, you gain insight into which columns still need cleaning or if further merges may fill some gaps.

4. Checking Distribution of Hotel Types

1. **Counting Unique Values in hotel_type:**

```python

df.hotel_type.value_counts()

```

- This command counts the occurrences of each unique value in the `hotel_type` column. Based on the output, there are 353 hotels labeled as 3-star, 145 as 4-star, and 55 as 5-star.

Here's a concise summary of your code:

1. **Load Data:**

- You start by loading your main hotel dataset, as well as supplementary datasets containing 3-star and 4-star hotel categories.

2. **Clean Data:**

- For each supplementary dataset (3-star and 4-star), you drop any unnecessary columns, like "Unnamed: 0".

3. **Merge Star Category Data:**

- Using a left join, you merge the 3-star and 4-star datasets with the main hotel dataset based on the "Hotel" column.

- After each merge, you handle `hotel_type` by combining columns from the existing and merged data, ensuring all non-null values are retained.

4. **Clean up Merged Columns:**

- After combining the `hotel_type` values, you drop temporary columns (`hotel_type_x` and `hotel_type_y`) created during each merge.

5. **Inspect and Analyze:**

- You check the last few rows of the main dataset after each merge to confirm the integration of new data.

- Finally, you examine missing values across columns and count the occurrences of each hotel type (3-star, 4-star, 5-star) to understand the distribution in the dataset.

This process prepares a unified dataset by combining hotel type categories and allows for further analysis of missing values and star distribution.

Summary of Code Workflow

1. **Dataset Loading & Initial Exploration**

- Imported required libraries such as `pandas` and `numpy`.

- Loaded an Excel dataset containing hotel information, and displayed initial rows, shape, and columns to understand its structure.

2. **Dataset Inspection**

- Checked for unique values in columns like `Building_Type`.

- Used `.info()` and `.dtypes` to get an overview of data types and identify columns with missing values.

- Calculated missing values for each column and displayed their count and percentage.

3. **Data Filtering**

- Filtered out rows with missing values in the `hotel_type` column to form a new DataFrame (`df2`).

4. **Missing Value Analysis**

- Analyzed missing values specifically within `df2` to identify columns with the most null values.

5. **Column Management**

- Attempted to drop unnecessary columns (`Unnamed: 0` and `No_of_Reviews`) from the filtered DataFrame (`df2`), although there was an error for `Unnamed: 0` due to an incorrect attempt after its removal in a previous step.

Observations

- The dataset contains a significant amount of missing values in columns such as `Building_Type`, `Discount_price`, `House_Rule`, etc.

- Further handling of missing data or imputation could be beneficial depending on the analysis needs.

- Column-dropping errors indicate that some columns had already been removed, so a check before attempting additional drops could improve code stability.

Let me know if you need assistance with resolving the KeyError or any specific cleaning strategies!

It looks like you've made significant progress in cleaning up your hotel dataset. Here's a summary and some suggestions for further steps:

Summary of Cleaning Steps Completed

1. **Handling Missing Values:**

- Filled missing values in `Ratings` with `0` and filtered rows where `Ratings` was `NaN` while `No_of_Ratings` was present.

- Filled missing values in `No_of_Ratings` and `Taxes` with `0`.

- Dropped rows with remaining missing values in key columns (`Ratings`, `No_of_Ratings`, and `Taxes`).

2. **Data Type Adjustments:**

- Converted `Original_price` and `Discount_price` by removing special characters (`₹`, commas) and converting them to integers.

- Cleaned `Taxes` column, removing symbols and commas to ensure numeric formatting.

3. **Current Shape and Column Consistency:**

- After cleaning, `df2` has 406 entries, and all relevant columns are non-null with consistent types.

Next Steps and Tips

1. **Convert Columns to Numeric Type:**

- Ensure that columns like `Discount_price` and `Taxes` are in integer format:

```python

df2['Discount_price'] = df2['Discount_price'].astype(int)

df2['Taxes'] = df2['Taxes'].astype(int)

```

2. **Save and Validate:**

- Consider saving this cleaned data to a new CSV for easier re-access and validation.

3. **Outlier Detection:**

- Run basic statistics on `Ratings`, `Original_price`, `Discount_price`, and `Taxes` to detect any outliers or data inconsistencies.

4. **Optional Data Imputation for Ratings:**

- If you plan to analyze ratings, consider imputing them using the average ratings based on similar hotels (e.g., by `Location` and `hotel_type`) rather than using `0`.

This structure should streamline further analysis on value-for-money hotels and facilitate any visualization or statistical modeling.

Data analysis:
Absolutely, I'll go through each line of code from the notebook in detail.

Step 1: Import Libraries
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = 'all'
```

- `import pandas as pd`: Imports the `pandas` library, which is used for data manipulation and analysis. It allows for reading and writing data, handling missing values, and various other data processing tasks. Here, it's imported as `pd` to make referencing it more concise.

- `import seaborn as sns`: Imports the `seaborn` library, a Python library for creating statistical data visualizations. It builds on top of `matplotlib` and integrates well with `pandas` DataFrames. It's imported as `sns` to simplify references.

- `import matplotlib.pyplot as plt`: Imports `matplotlib.pyplot`, a library primarily used for plotting in Python. It provides a range of functions to create static, animated, and interactive visualizations.

- `from IPython.core.interactiveshell import InteractiveShell`: Imports `InteractiveShell` from the IPython core, which is a part of the IPython system used to configure the behavior of IPython notebooks.

- `InteractiveShell.ast_node_interactivity = 'all'`: Sets the notebook's output to display all outputs within a single cell. By default, only the last output of a cell is displayed, but this command changes it so that each statement's result is shown.

Step 2: Load Dataset
```python
df = pd.read_excel(r"D:\\INNOMATICS\\DATA ANALYTICS\\01-Python\\final project\\data_cleaned_df (1).xlsx")
```

- `df = pd.read_excel(...)`: Reads an Excel file into a `pandas` DataFrame.
- `pd.read_excel` is a function from `pandas` that reads Excel files.

- `r"D:\\INNOMATICS\\DATA ANALYTICS\\01-Python\\final project\\data_cleaned_df (1).xlsx"` specifies the path of the Excel file to load. The `r` prefix before the string means "raw string," which tells Python to treat backslashes (`\`) as literal characters.

- `df`: This variable holds the resulting DataFrame, making it accessible for further data processing and analysis.

Step 3: Calculate Discount
```python
df.insert(8,"Discount",df.Original_price-df.Discounted_price)
```

- `df.insert(8, "Discount", ...)`: Inserts a new column named `"Discount"` into the DataFrame `df` at the 8th position (Python is zero-indexed, so this is technically the 9th column).
- The 8 here is the column index position in the DataFrame.
- `"Discount"` is the name of the new column.
- `df.Original_price - df.Discounted_price` calculates the difference between the values in the columns `Original_price` and `Discounted_price` for each row. This difference represents the amount saved (or discount received) when buying at the discounted price instead of the original price.

- The resulting `"Discount"` column now contains the calculated discount for each item in the DataFrame.

Step 4: View Data
```python
df.head()
```

- `df.head()`: Displays the first five rows of the DataFrame `df` by default.
  - This is a useful way to quickly inspect the structure of the data and ensure that columns have been loaded and calculated correctly.
  - It’s especially helpful after making modifications to confirm that they have been applied as expected.

Step 5: View Column Names
```python
df.columns
```

- `df.columns`: Lists all the column names in the DataFrame `df`.
  - This allows you to see all column names and confirm the addition of new columns (such as the `"Discount"` column) or to get an overview of the data’s structure.
  - This command returns an `Index` object that displays the column names as a list.

Alright, let's go through the next steps in the notebook in detail.

Step 6: Plot a Count Plot for `Location`

```python
plt.figure(figsize=(15,8))
sns.countplot(x="Location", data=df)
plt.xticks(rotation=90)
plt.show()
```

- `plt.figure(figsize=(15,8))`: Creates a new figure for plotting with a specified width and height. `figsize=(15,8)` sets the width to 15 units and the height to 8 units, ensuring the plot has enough space to display all the data clearly, especially for locations.

- `sns.countplot(x="Location", data=df)`: Creates a count plot with the `Location` column on the x-axis.
  - `x="Location"` specifies the column to be counted and displayed on the x-axis.
  - `data=df` specifies that the data for the plot comes from the `df` DataFrame.
  - A count plot shows the count of unique values for a particular column, in this case, `Location`.

- `plt.xticks(rotation=90)`: Rotates the x-axis labels by 90 degrees. This is often done when labels are long, as rotating them helps prevent overlap and improves readability.

- `plt.show()`: Displays the plot within the notebook.

Step 7: Plot a Histogram for `Ratings`

```python

plt.figure(figsize=(8,5))

sns.histplot(df["Ratings"], kde=True)

plt.show()

```

- `plt.figure(figsize=(8,5))`: Creates a new figure for the histogram with a specified size (8 units wide and 5 units high).

- `sns.histplot(df["Ratings"], kde=True)`: Creates a histogram to display the distribution of values in the `Ratings` column.

** - `df["Ratings"]` specifies that the data for the histogram should come from the `Ratings` column.**

** - `kde=True` adds a Kernel Density Estimate (KDE) line, which is a smooth curve that represents the data distribution, overlayed on the histogram.**

- `plt.show()`: Displays the histogram within the notebook.

Step 8: Plot a Box Plot for `Discount`

```python
plt.figure(figsize=(8,5))
sns.boxplot(x=df["Discount"])
plt.show()
```

- `plt.figure(figsize=(8,5))`: Creates a new figure for the box plot, setting it to 8 units wide and 5 units high.

- `sns.boxplot(x=df["Discount"])`: Creates a box plot to show the distribution and outliers in the `Discount` column.
  - `x=df["Discount"]` specifies that the `Discount` column data will be plotted on the x-axis.
  - A box plot is useful for visualizing the spread of data, identifying quartiles, and detecting any outliers.

- `plt.show()`: Displays the box plot within the notebook.

Step 9: Calculate Quartiles and IQR for `Discount`

```python
Q1 = df['Discount'].quantile(0.25)
Q3 = df['Discount'].quantile(0.75)
IQR = Q3 - Q1
```

```

- `Q1 = df['Discount'].quantile(0.25)`: Calculates the first quartile (25th percentile) of the `Discount` column and stores it in `Q1`. The first quartile is the value below which 25% of the data falls.

- `Q3 = df['Discount'].quantile(0.75)`: Calculates the third quartile (75th percentile) of the `Discount` column and stores it in `Q3`. The third quartile is the value below which 75% of the data falls.

- `IQR = Q3 - Q1`: Calculates the Interquartile Range (IQR) by subtracting `Q1` from `Q3`. The IQR represents the range of the middle 50% of the data, providing insights into data spread and is often used to identify outliers.

Step 10: Calculate Outlier Bounds

```python

Lower_Bound = Q1 - 1.5 * IQR

Upper_Bound = Q3 + 1.5 * IQR

```

- `Lower_Bound = Q1 - 1.5 * IQR`: Calculates the lower bound for outliers in the `Discount` column. Any value below this bound is considered a potential outlier. The `1.5 * IQR` rule is commonly used to determine outlier thresholds.

- `Upper_Bound = Q3 + 1.5 * IQR`: Calculates the upper bound for outliers in the `Discount` column. Any value above this bound is considered a potential outlier.

These bounds help in identifying and potentially filtering out values that fall outside of the usual range in the data.

Step 11: Filter Outliers from `Discount`

```python

outliers = df[(df["Discount"] < Lower_Bound) | (df["Discount"] > Upper_Bound)]

outliers

```

- `outliers = df[(df["Discount"] < Lower_Bound) | (df["Discount"] > Upper_Bound)]`: Creates a new DataFrame `outliers` that contains only rows where `Discount` values are outside the calculated lower and upper bounds.

- `(df["Discount"] < Lower_Bound)` selects rows where the `Discount` value is below the lower bound.

- `(df["Discount"] > Upper_Bound)` selects rows where the `Discount` value is above the upper bound.

- The `|` operator combines these two conditions, meaning any value outside either bound is considered an outlier.

- `outliers`: Displays the filtered outliers DataFrame, which contains records from `df` where `Discount` values fall outside the calculated bounds.

Sure, let's go through the next steps in detail.

Step 12: Add `Price Range` Column Based on `Discounted_price`

```python

df['price_range'] = pd.cut(df['Discounted_price'], bins=[0, 2000, 4000, 6000, 8000, 10000], labels=['0-2K', '2-4K', '4-6K', '6-8K', '8-10K'])

```

- `df['price_range'] =`: Creates a new column in the `df` DataFrame called `price_range`.

- `pd.cut(df['Discounted_price'], ...)`:

- This function is used to divide data into specified bins or intervals. Here, it divides the values in the `Discounted_price` column into different price ranges based on the specified intervals.

- `bins=[0, 2000, 4000, 6000, 8000, 10000]`: Sets the boundaries for each price range. For example, `0-2000` is one bin, representing prices between 0 and 2000.

- `labels=['0-2K', '2-4K', '4-6K', '6-8K', '8-10K']`: Labels each bin with a specific price range label.

This code adds a `price_range` column, which categorizes each `Discounted_price` value into a price range category.

Step 13: Plot Count Plot of `Price Range`

```python

plt.figure(figsize=(10,5))

sns.countplot(x="price_range", data=df)

plt.show()

```

- `plt.figure(figsize=(10,5))`: Creates a new figure for the plot, setting it to 10 units wide and 5 units high.

- `sns.countplot(x="price_range", data=df)`: Creates a count plot to display the frequency of each price range category in the `price_range` column.

- `x="price_range"` specifies that the `price_range` column values are on the x-axis.

- `data=df` specifies that the data for the plot comes from the `df` DataFrame.

- `plt.show()`: Displays the plot within the notebook, showing the count of records for each price range.

Step 14: Create a Scatter Plot of `Discounted_price` vs. `Ratings`

```python

plt.figure(figsize=(10,6))

sns.scatterplot(x="Discounted_price", y="Ratings", data=df)

plt.show()

```

- `plt.figure(figsize=(10,6))`: Creates a new figure for the scatter plot, setting it to 10 units wide and 6 units high.

- `sns.scatterplot(x="Discounted_price", y="Ratings", data=df)`: Creates a scatter plot to examine the relationship between `Discounted_price` and `Ratings`.

- `x="Discounted_price"` specifies that the `Discounted_price` column values are on the x-axis.

- `y="Ratings"` specifies that the `Ratings` column values are on the y-axis.

- `data=df` specifies that the data for the plot comes from the `df` DataFrame.

- `plt.show()`: Displays the scatter plot within the notebook, allowing us to see if there's any pattern or relationship between the discounted price of hotels and their ratings.

Step 15: Calculate and Display Correlation Matrix

```python

corr = df.corr()

plt.figure(figsize=(12,8))

sns.heatmap(corr, annot=True, cmap='coolwarm')

plt.show()

```

- `corr = df.corr()`: Calculates the correlation matrix for numerical columns in `df`, storing it in `corr`. Correlation values range from -1 to 1, where:

- Values closer to 1 indicate a strong positive correlation.

- Values closer to -1 indicate a strong negative correlation.

- Values close to 0 indicate little or no linear correlation.

- `plt.figure(figsize=(12,8))`: Creates a new figure for the heatmap, setting it to 12 units wide and 8 units high.

- `sns.heatmap(corr, annot=True, cmap='coolwarm')`: Creates a heatmap to visualize the correlation matrix.

  - `corr` is the matrix being visualized.

  - `annot=True` displays the correlation values within each cell.

  - `cmap='coolwarm'` applies a color map to differentiate positive and negative correlations, making it easy to identify relationships visually.

- `plt.show()`: Displays the heatmap within the notebook, showing the correlations among numerical features.

Step 16: Filter Hotels for "Value for Money" Analysis

```python
value_hotels = df[(df['Ratings'] >= 4) & (df['Discounted_price'] <= df['Discounted_price'].mean())]
value_hotels
```

- `value_hotels =`: Creates a new DataFrame called `value_hotels` to store hotels that meet the "value for money" criteria.

- `df[(df['Ratings'] >= 4) & (df['Discounted_price'] <= df['Discounted_price'].mean())]`: Filters rows in the `df` DataFrame based on specific conditions.

- `df['Ratings'] >= 4`: Selects hotels with ratings of 4 or higher, indicating customer satisfaction.

- `df['Discounted_price'] <= df['Discounted_price'].mean()`: Selects hotels with a discounted price less than or equal to the average discounted price in `df`, targeting affordability.

- The `&` operator combines both conditions, meaning only hotels that meet both criteria will be selected.

- `value_hotels`: Displays the filtered `value_hotels` DataFrame, showing hotels that offer high ratings at an affordable discounted price.

This provides a comprehensive walkthrough of each line of code in your notebook up to this point. Let me know if you would like further explanation on any specific step or if there are additional sections to go through!x