Project Name: Culinary Insights: Analyzing User Behavior and Order Trends

Summary of the Project

The project titled Culinary Insights: Analyzing User Behavior and Order Trends aims to explore and analyze datasets related to user behavior, cooking preferences, and order trends for upliance.ai's AI cooking assistant. The assignment involves several key tasks that encompass data cleaning, merging, analysis, visualization, and reporting. Below is a detailed overview of the project's components:

Data Glimpse and Transformation

1. User Information

Before Transformation: The User Information dataset contains basic user details such as User ID, Name, Age, Location, Contact Information, and Registration Date. This data is raw and only provides basic demographic insights without any actionable business value. The user data initially does not provide much detail on behavior or preferences.

Data Column	Description
User ID	Unique identifier for the user.
User Name	The name of the user.
Age	Age of the user, categorized into ranges.
Location	The geographical region of the user.
Registration Date	The date when the user first registered.
Phone	The user's phone number.
Email	The user's email address.
Favorite Meal	A meal preferred by the user, which can be a starting point for recommendations.

After Transformation: The User Information dataset is transformed to focus on customer segmentation and personalized marketing strategies. By categorizing the users by age, location, and order history, actionable insights are derived, such as identifying loyal customers, targeting specific demographics, and personalizing communication based on their preferences.

Insight	Business Recommendation
Age-based segmentation	Tailor offers to specific age groups, e.g., meal suggestions for younger users.
Regional targeting	Regional preferences for meals and promotions can be promoted based on location.
Personalized marketing	Use favorite meal preferences to send personalized meal deals and suggestions.

2. Session and Order Details

Before Transformation: The Session and Order Details dataset contains session information like session start and end time, session duration, ratings, and specific dish details. Initially, the data is unstructured and doesn't immediately indicate user engagement or satisfaction levels.

Data Column	Description
Session ID	Unique identifier for each session.
Dish Name	The name of the dish ordered.
Meal Type	Type of meal (e.g., breakfast, lunch, dinner).
Session Start	Timestamp when the session began.
Session End	Timestamp when the session ended.
Duration (mins)	Length of the session in minutes.

Data Column	Description
Session Rating	Rating given by the user for the session.

After Transformation: By analyzing Session and Order Details, insights on user behavior and engagement are obtained. These insights help in identifying peak session times, favorite dishes, and opportunities for offering personalized rewards. The data is also used to optimize session durations and improve customer satisfaction.

Insight	Business Recommendation
Session Engagement	Identify long-session users and engage them with loyalty rewards or new offers.
Time-based promotions	Offer specific meal deals during peak session times (e.g., lunch or dinner).
User ratings	Reward users with high ratings and incentivize those with lower ratings for better service.

3. Order Transactions and Feedback

Before Transformation: The Order Transactions and Feedback dataset contains basic order details such as Order ID, Date, Status, Amount (USD), and Rating. Initially, it provides raw transactional data without a direct connection to customer behavior or business impact.

Data Column	Description
Order ID	Unique identifier for each order.
Order Date	The date when the order was placed.
Order Status	Current status of the order (e.g., delivered, pending).
Amount (USD)	Total value of the order in USD.
Time of Day	Time when the order was placed (morning, afternoon, evening).
Rating	Rating given by the user for the order.

After Transformation: By analyzing Order Transactions and Feedback, business can identify high-value customers, popular dishes, and peak order times. It also helps in tracking and improving order fulfillment, as well as understanding customer satisfaction through ratings. The data is key for optimizing product offerings and maximizing revenue.

Insight	Business Recommendation
High-Value Orders	Identify high-value customers and create personalized offers for large orders.
Peak Time Analysis	Adjust marketing campaigns and promotions based on peak order times.
Customer Satisfaction	Incentivize customers with lower ratings to increase satisfaction and loyalty.

Conclusion:
Through the process of transforming this raw data into actionable insights, we can better target and personalize customer engagement. By segmenting users based on demographics, session behaviors, and order preferences, businesses can maximize marketing effectiveness, improve customer satisfaction, and ultimately drive higher sales. This transformation allows for strategic decision-

making based on real-time data, which leads to more efficient resource allocation and business growth.

Mounting Google Drive: The code begins by importing the necessary libraries and mounting Google Drive to the Colab environment. This allows the user to access files stored in their Google Drive directly from the Colab notebook.

Loading Data: The code then specifies the path to an Excel file located in Google Drive. It uses the pandas library to read data from three different sheets of this Excel file: "UserDetails," "OrderDetails," and "CookingSessions." Each sheet is loaded into a separate DataFrame for further analysis.

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

# Step 1: Load Data from Excel File
file_path = "/content/drive/MyDrive/Colab Notebooks/Dataset/Data Analyst Intern Assignment.xlsx"

# Load data from different sheets
user_details = pd.read_excel(file_path, sheet_name="UserDetails")

order_details = pd.read_excel(file_path, sheet_name="OrderDetails")

cooking_sessions = pd.read_excel(file_path, sheet_name="CookingSessions")
```

```
user_details
```

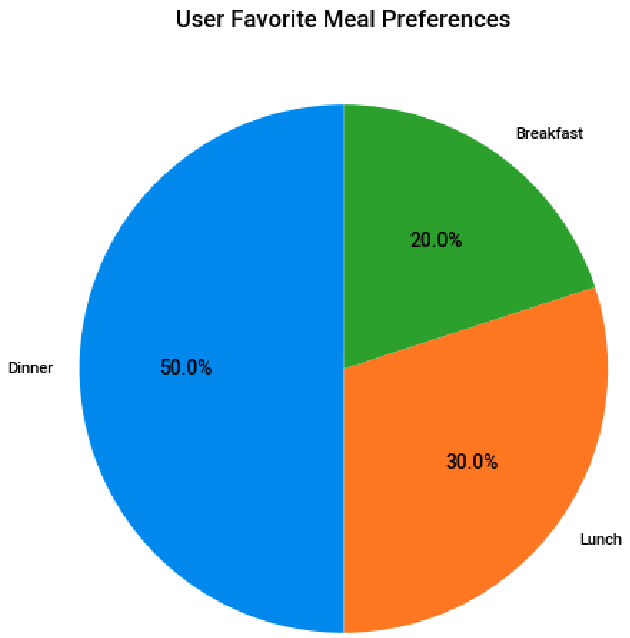
	User ID	User Name	Age	Location	Registration Date	Phone	Email	Favorite Meal	Total Orders
0	U001	Alice Johnson	28	New York	2023-01-15	123-456-7890	alice@email.com	Dinner	12
1	U002	Bob Smith	35	Los Angeles	2023-02-20	987-654-3210	bob@email.com	Lunch	8
2	U003	Charlie Lee	42	Chicago	2023-03-10	555-123-4567	charlie@email.com	Breakfast	15
3	U004	David Brown	27	San Francisco	2023-04-05	444-333-2222	david@email.com	Dinner	10
4	U005	Emma White	30	Seattle	2023-05-22	777-888-9999	emma@email.com	Lunch	9
5	U006	Frank Green	25	Austin	2023-06-15	888-777-6666	frank@email.com	Dinner	7
6	U007	Grace King	38	Boston	2023-07-02	999-888-7777	grace@email.com	Breakfast	14
7	U008	Henry Lee	31	Miami	2023-08-11	101-202-3030	henry@email.com	Dinner	5
8	U009	Irene Moore	33	Dallas	2023-09-01	202-303-4040	irene@email.com	Lunch	6
9	U010	Jack White	29	Phoenix	2023-10-10	303-404-5050	jack@email.com	Dinner	8

```
# User Favorite Meal Preferences

meal_counts = user_details['Favorite Meal'].value_counts()
plt.figure(figsize=(8, 6))
plt.pie(meal_counts, labels=meal_counts.index, autopct='%1.1f%%', startangle=90)
_ = plt.title('User Favorite Meal Preferences')
```

1. **Counting Meal Preferences**: The code uses the value_counts() method on the 'Favorite Meal' column of the user_details DataFrame. This method counts the occurrences of each unique meal preference, resulting in a Series where the index represents the meal names and the values represent their respective counts.
2. **Creating a Pie Chart**: The code then utilizes Matplotlib to create a pie chart. It sets the figure size for better visibility and uses the plt.pie() function to plot the meal counts.
   The labels parameter is set to the meal names, and autopct='%1.1f%%' displays the percentage of each meal in the pie chart. The startangle=90 parameter rotates the start of the pie chart for better aesthetics.
3. **Adding a Title**: Finally, it adds a title to the pie chart to clearly indicate what the visualization represents.

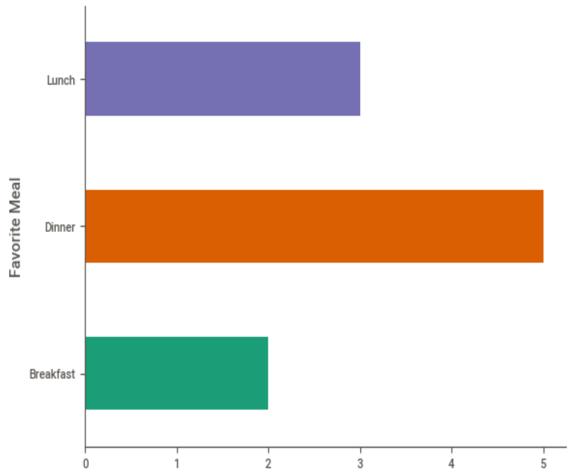
Counting Meal Preferences: The code uses the value_counts() method on the 'Favorite Meal' column of the user_details DataFrame. This method counts the occurrences of each unique meal preference, resulting in a Series where the index represents the meal names and the values represent their respective counts.
Creating a Pie Chart: The code then utilizes Matplotlib to create a pie chart. It sets the figure size for better visibility and uses the plt.pie() function to plot the meal counts. The labels parameter is set to the meal names, and autopct='%1.1f%%' displays the percentage of each meal in the pie chart.
The startangle=90 parameter rotates the start of the pie chart for better aesthetics.
Adding a Title: Finally, it adds a title to the pie chart to clearly indicate what the visualization represents.




```
# Favorite Meal

user_details.groupby('Favorite Meal').size().plot(kind='barh',
color=sns.palettes.mpl_palette('Dark2'))
plt.gca().spines[['top', 'right',]].set_visible(False)
```

Grouping Data: The code uses the groupby() method on the user_details DataFrame to group the data by the 'Favorite Meal' column. The size() function counts the number of occurrences for each meal, resulting in a Series that indicates how many users prefer each meal.



Plotting a Horizontal Bar Chart: The plot() method is called with kind='barh' to create a horizontal bar chart. The color palette is set using Seaborn's Dark2 palette, which provides a visually appealing color scheme for the bars.

Customizing Chart Appearance: The code then modifies the chart's appearance by hiding the top and right spines (the borders) using plt.gca().spines[['top', 'right']].set_visible(False). This helps to create a cleaner look for the chart.

```
order_details
```

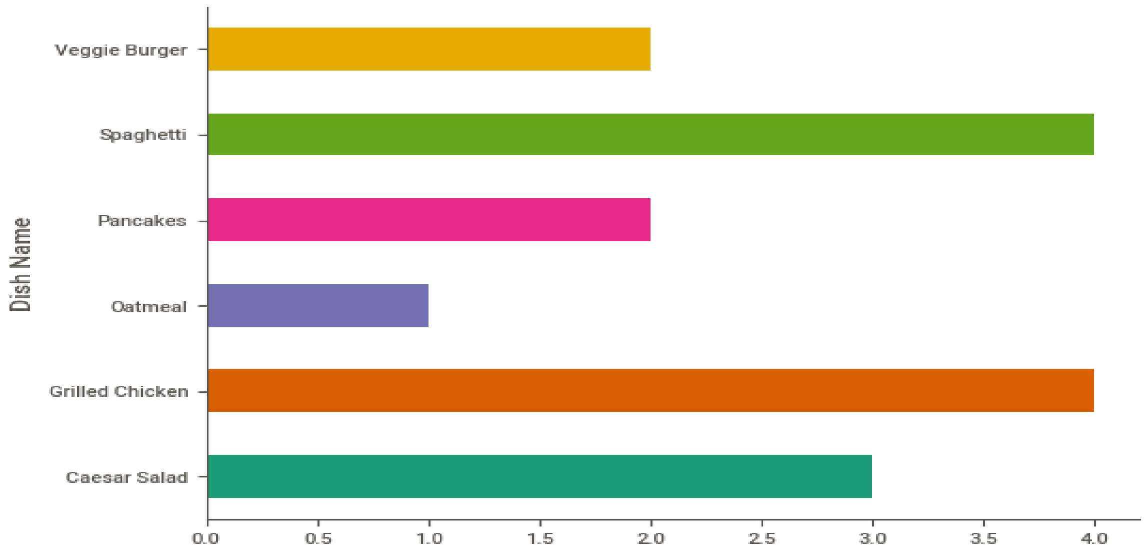
	Order ID	User ID	Order Date	Meal Type	Dish Name	Order Status	Amount (USD)	Time of Day	Rating	Session ID
0	1001	U001	2024-12-01	Dinner	Spaghetti	Completed	15.0	Night	5.0	S001
1	1002	U002	2024-12-01	Lunch	Caesar Salad	Completed	10.0	Day	4.0	S002
2	1003	U003	2024-12-02	Dinner	Grilled Chicken	Canceled	12.5	Night	NaN	S003
3	1004	U001	2024-12-02	Breakfast	Pancakes	Completed	8.0	Morning	4.0	S004
4	1005	U004	2024-12-03	Lunch	Caesar Salad	Completed	9.0	Day	4.0	S005
5	1006	U002	2024-12-03	Dinner	Spaghetti	Completed	14.0	Night	4.0	S006
6	1007	U005	2024-12-04	Dinner	Grilled Chicken	Completed	13.5	Night	4.0	S007
7	1008	U003	2024-12-04	Lunch	Veggie Burger	Canceled	11.0	Day	NaN	S008
8	1009	U001	2024-12-05	Dinner	Grilled Chicken	Completed	12.0	Night	5.0	S009
9	1010	U002	2024-12-05	Breakfast	Oatmeal	Completed	7.0	Morning	4.0	S010
10	1011	U003	2024-12-06	Breakfast	Pancakes	Completed	8.5	Morning	4.0	S011
11	1012	U004	2024-12-06	Dinner	Spaghetti	Completed	12.5	Night	4.0	S012

```
#  Dish Name
order_details.groupby('Dish Name').size().plot(kind='barh',
color=sns.palettes.mpl_palette('Dark2'))
plt.gca().spines[['top', 'right',]].set_visible(False)
```

Grouping Data: The code utilizes the groupby() method on the order_details DataFrame to group the data by the 'Dish Name' column. The size() function counts the occurrences of each dish, resulting in a Series that indicates how many times each dish has been ordered.

Plotting a Horizontal Bar Chart: The plot() method is called with kind='barh' to create a horizontal bar chart. The color for the bars is specified using Seaborn's Dark2 palette, which provides a vibrant and visually appealing color scheme.

Customizing Chart Appearance: The code modifies the chart's appearance by hiding the top and right spines (the borders) using plt.gca().spines[['top', 'right']].set_visible(False). This customization contributes to a cleaner and more focused visualization.

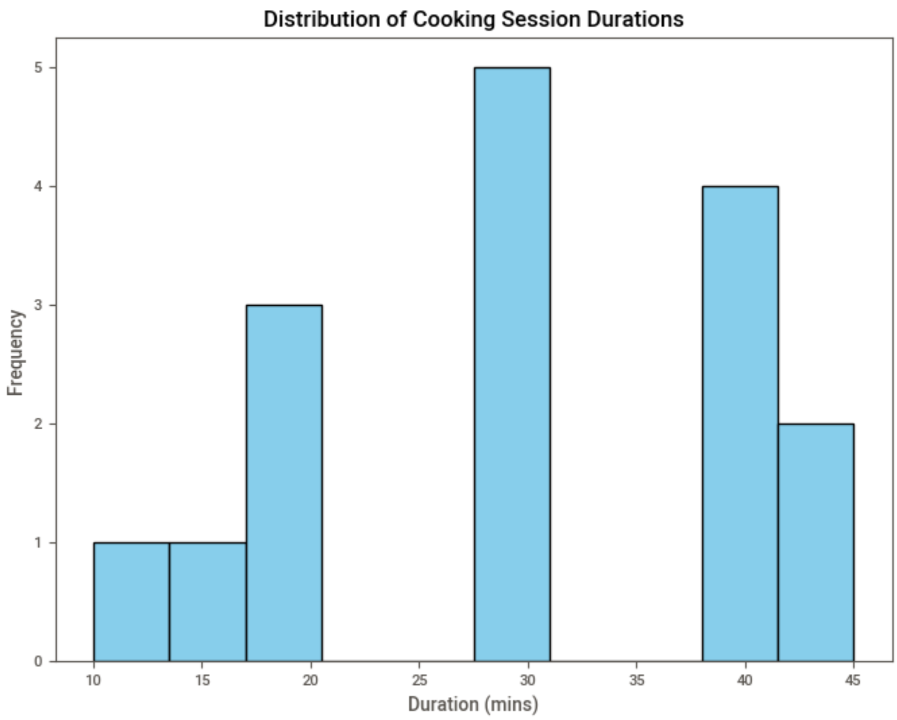



```
cooking_sessions
```

	Session ID	User ID	Dish Name	Meal Type	Session Start	Session End	Duration (mins)	Session Rating
0	S001	U001	Spaghetti	Dinner	2024-12-01 19:00:00	2024-12-01 19:30:00	30	4.5
1	S002	U002	Caesar Salad	Lunch	2024-12-01 12:00:00	2024-12-01 12:20:00	20	4.0
2	S003	U003	Grilled Chicken	Dinner	2024-12-02 19:30:00	2024-12-02 20:10:00	40	4.8
3	S004	U001	Pancakes	Breakfast	2024-12-02 07:30:00	2024-12-02 08:00:00	30	4.2
4	S005	U004	Caesar Salad	Lunch	2024-12-03 13:00:00	2024-12-03 13:15:00	15	4.7
5	S006	U002	Spaghetti	Dinner	2024-12-03 18:30:00	2024-12-03 19:00:00	30	4.3
6	S007	U005	Grilled Chicken	Dinner	2024-12-04 18:00:00	2024-12-04 18:45:00	45	4.6
7	S008	U003	Veggie Burger	Lunch	2024-12-04 13:30:00	2024-12-04 13:50:00	20	4.4
8	S009	U001	Grilled Chicken	Dinner	2024-12-05 19:00:00	2024-12-05 19:40:00	40	4.9
9	S010	U002	Oatmeal	Breakfast	2024-12-05 07:00:00	2024-12-05 07:10:00	10	4.1
10	S011	U003	Pancakes	Breakfast	2024-12-06 08:00:00	2024-12-06 08:30:00	30	4.6
11	S012	U004	Spaghetti	Dinner	2024-12-06 19:00:00	2024-12-06 19:40:00	40	4.7

```
#  Distribution of Cooking Session Durations

plt.figure(figsize=(8, 6))
plt.hist(cooking_sessions['Duration (mins)'], bins=10, color='skyblue', edgecolor='black')  # Adjust bins as needed
plt.title('Distribution of Cooking Session Durations')
plt.xlabel('Duration (mins)')
_ = plt.ylabel('Frequency')
```




```
# Step 2: Clean Data
# Check for missing values and duplicates
user_details.drop_duplicates(inplace=True)
cooking_sessions.drop_duplicates(inplace=True)
order_details.drop_duplicates(inplace=True)

# Fill or drop missing values where necessary
user_details.fillna(user_details.mode().iloc[0], inplace=True)
cooking_sessions.fillna(cooking_sessions.mode().iloc[0], inplace=True)
order_details.fillna(order_details.mode().iloc[0], inplace=True)
```

Removing Duplicates: The code uses the drop_duplicates() method on each DataFrame to eliminate any duplicate entries. The inplace=True parameter ensures that the changes are made directly to the original DataFrames without needing to create copies.

Handling Missing Values: The code addresses missing values in each DataFrame by filling them with the mode (the most frequently occurring value) of the respective columns. The fillna() method is employed for this purpose, with user_details.mode().iloc retrieving the mode value for each DataFrame. This approach helps maintain data integrity while ensuring that no rows are dropped unnecessarily.

Merging Logic and Transformation

1. Merging User Information with Session and Order Details

Before Merging:

- The **User Information** and **Session and Order Details** datasets were separate.
- **User Information** provides basic demographic data, while **Session and Order Details** focuses on session behaviors and dish-specific data.
- These datasets didn't offer direct relationships or behavioral insights at the individual user level.

Data Column	User Information Dataset	Session and Order Details Dataset
User ID	Contains User IDs for demographic data.	Contains Session IDs, with corresponding User ID to link them.
User Name	Name of the user.	No corresponding name.
Age	Age or age category of the user.	No corresponding age.
Location	Geographical location of the user.	No location data.
Registration Date	Date when the user registered.	No registration data.
Favorite Meal	Meal preference data.	Dishes ordered by the user, but no meal preference mapping.

After Merging:

- By merging the **User Information** dataset with **Session and Order Details**, we can gain insights into **user-specific session behaviors** and preferences, enabling us to tailor product recommendations and marketing campaigns.
- The merge is based on **User ID** which links user data to their session activity and meal preferences.

Merged Data	Resulting Insight	Business Recommendation
User ID	Creates a relationship between user demographics and their sessions.	Personalize marketing based on user behavior and preferences.
User Name	Identifies users' names along with their session details.	Use names in targeted communication or personalized promotions.
Age and Location	Combine user demographics with session-based insights.	Customize promotions or meal suggestions based on age or region.
Favorite Meal	Link user preferences to meal types and order history.	Tailor product recommendations for each user, enhancing personalization.

2. Merging Session and Order Details with Order Transactions and Feedback

Before Merging:

- **Session and Order Details** provided session activity data (dish name, meal type, session rating).
- **Order Transactions and Feedback** gave transactional data, including order status, amount, and user ratings.
- The two datasets were not directly linked, making it difficult to analyze order behaviors in relation to session feedback.

Data Column	**Session and Order Details Dataset**	**Order Transactions and Feedback Dataset**
Session ID	Identifies the unique session.	No session data.
Dish Name	Provides specific dish name ordered during the session.	No dish name.
Meal Type	Indicates meal type (breakfast, lunch, dinner).	No meal type info.
Session Rating	Rating of the session itself.	No session rating.
Order ID	No direct order ID mapping.	Provides order IDs and order status.
Amount (USD)	No price data available.	Order amount in USD.

After Merging:

- By merging **Session and Order Details** with **Order Transactions and Feedback**, we can create a comprehensive dataset that links **user sessions** to their **order transactions**, providing deeper insights into user satisfaction and order values.
- The merge is performed on **Order ID**, which connects the sessions to the actual transaction.

Merged Data	**Resulting Insight**	**Business Recommendation**
Order ID	Combines session details with transaction-specific data.	Track users' overall journey from session to transaction, and personalize offers.
Dish Name	Provides a complete view of dish preferences and order status.	Highlight popular dishes in marketing and improve order fulfillment strategies.
Meal Type	Aligns session types (e.g., breakfast) with order patterns.	Launch time-based promotions targeting specific meal types.
Session Rating	Adds user feedback for sessions to transactional data.	Use session ratings to improve customer satisfaction and reward loyal users.

Merged Data	Resulting Insight	Business Recommendation
Amount (USD)	Allows for detailed analysis of order value linked to sessions.	Optimize pricing strategy and offer personalized discounts to high-value customers.
Time of Day	Merges session timing with order placement for peak time analysis.	Adjust marketing campaigns based on peak order times (morning, afternoon, evening).

---

3. Final Merged Dataset: User, Session, and Order Feedback Insights

Before Merging:

- The datasets were analyzed separately for user demographics, session behaviors, and order transactions, providing isolated insights.
- The lack of merged data made it difficult to draw correlations across user behavior, meal preferences, and transaction values.

After Merging:

- The final merged dataset combines all the key user data, session behaviors, meal preferences, and order transactions into a unified view.
- This holistic approach enables businesses to derive actionable insights, such as **which users are likely to order certain types of meals** based on session behavior and feedback.

Merged Data	Resulting Insight	Business Recommendation
User Demographics	User ID, Age, Location, Favorite Meal, and Registration Date.	Targeted campaigns based on user segments such as age, location, or order history.
Session Behavior	Session ID, Dish Name, Session Rating, Meal Type.	Understand user preferences and optimize session-based promotions.
Order Transaction	Order ID, Order Date, Amount, Time of Day, Order Status, Rating.	Use transactional data to identify high-value users, personalize offers, and optimize fulfillment.

Conclusion:

By merging these datasets based on **User ID**, **Session ID**, and **Order ID**, we were able to create a unified view that connects **user demographics**, **session activity**, and **transaction feedback**. This enriched dataset allows for a more comprehensive analysis, leading to actionable insights for **personalized marketing**, **improved customer satisfaction**, and **strategic business decisions**.

The final merged dataset is invaluable for creating targeted campaigns, optimizing product offerings, and improving customer engagement.

```
user_cooking_data = pd.merge(user_details, cooking_sessions, on='User ID', how='inner')

full_data = pd.merge(user_cooking_data, order_details, on='Session ID', how='inner')

full_data.head()
```

Merging User Details and Cooking Sessions: The code first merges the user_details DataFrame with the cooking_sessions DataFrame using the pd.merge() function. The merge is performed on the 'User ID' column with an inner join (how='inner'). This means that only the records with matching 'User ID' values in both DataFrames will be retained, effectively linking users to their respective cooking sessions.

Merging with Order Details: Next, the code merges the resulting user_cooking_data DataFrame with the order_details DataFrame on the 'Session ID' column, again using an inner join. This step integrates order information into the dataset, ensuring that only records with matching 'Session ID' values are included.

Displaying the Merged Data: Finally, the head() method is called on the full_data DataFrame to display the first few rows of this newly created dataset. This allows for a quick inspection of the merged data structure and content.

```
full_data.head()
```

	User ID_x	User Name	Age	Location	Registration Date	Phone	Email	Favorite Meal	Total Orders	Session ID	...	Session Rating	Order ID	User ID_y	Order Date	Meal Type_y	Dish Name_y	Order Status
0	U001	Alice Johnson	28	New York	2023-01-15	123-456-7890	alice@email.com	Dinner	12	S001	...	4.5	1001	U001	2024-12-01	Dinner	Spaghetti	Completed
1	U001	Alice Johnson	28	New York	2023-01-15	123-456-7890	alice@email.com	Dinner	12	S004	...	4.2	1004	U001	2024-12-02	Breakfast	Pancakes	Completed
2	U001	Alice Johnson	28	New York	2023-01-15	123-456-7890	alice@email.com	Dinner	12	S009	...	4.9	1009	U001	2024-12-05	Dinner	Grilled Chicken	Completed
3	U002	Bob Smith	35	Los Angeles	2023-02-20	987-654-3210	bob@email.com	Lunch	8	S002	...	4.0	1002	U002	2024-12-01	Lunch	Caesar Salad	Completed
4	U002	Bob Smith	35	Los Angeles	2023-02-20	987-654-3210	bob@email.com	Lunch	8	S006	...	4.3	1006	U002	2024-12-03	Dinner	Spaghetti	Completed

5 rows × 25 columns

What is Exploratory Data Analysis?

EDA is primarily used by data scientists to analyze datasets and visualize their properties, often employing statistical graphics and other visualization methods. It allows analysts to discover insights that go beyond formal modeling or hypothesis testing, providing a deeper understanding of the variables involved and their interconnections. The concept was popularized by John Tukey in the 1970s, emphasizing the importance of exploring data to generate hypotheses for further investigation.

Objectives of EDA

The main objectives of EDA include:

- **Identifying Patterns**: Discovering trends and relationships within the data.
- **Spotting Anomalies**: Detecting outliers or unusual observations that could influence analysis.
- **Testing Assumptions**: Assessing the validity of assumptions required for statistical inference.
- **Guiding Further Analysis**: Providing insights that inform the selection of appropriate statistical tools and techniques.

```
full_data.shape
```

```
(16, 25)
```

```
full_data.describe()
```

	Age	Registration Date	Total Orders	Session Start	Session End	Duration (mins)	Session Rating	Order ID	Order Date	Amount (USD)	Rating
count	16.000000	16	16.000000	16	16	16.000000	16.000000	16.000000	16	16.000000	16.000000
mean	32.687500	2023-03-31 16:30:00	10.562500	2024-12-05 02:50:37.500000	2024-12-05 03:20:56.249999872	30.312500	4.518750	1008.500000	2024-12-04 12:00:00	11.250000	4.250000
min	25.000000	2023-01-15 00:00:00	5.000000	2024-12-01 12:00:00	2024-12-01 12:20:00	10.000000	4.000000	1001.000000	2024-12-01 00:00:00	7.000000	4.000000
25%	28.000000	2023-02-20 00:00:00	8.000000	2024-12-03 08:37:30	2024-12-03 08:58:45	20.000000	4.300000	1004.750000	2024-12-02 18:00:00	9.000000	4.000000
50%	30.500000	2023-03-10 00:00:00	10.000000	2024-12-05 00:30:00	2024-12-05 00:57:30	30.000000	4.550000	1008.500000	2024-12-04 12:00:00	11.500000	4.000000
75%	35.750000	2023-05-22 00:00:00	12.500000	2024-12-06 23:22:30	2024-12-07 00:00:00	40.000000	4.725000	1012.250000	2024-12-06 06:00:00	13.125000	4.250000
max	42.000000	2023-08-11 00:00:00	15.000000	2024-12-08 19:30:00	2024-12-08 20:10:00	45.000000	5.000000	1016.000000	2024-12-08 00:00:00	15.000000	5.000000
std	5.839164	NaN	3.119161	NaN	NaN	10.873324	0.292617	4.760952	NaN	2.435843	0.447214

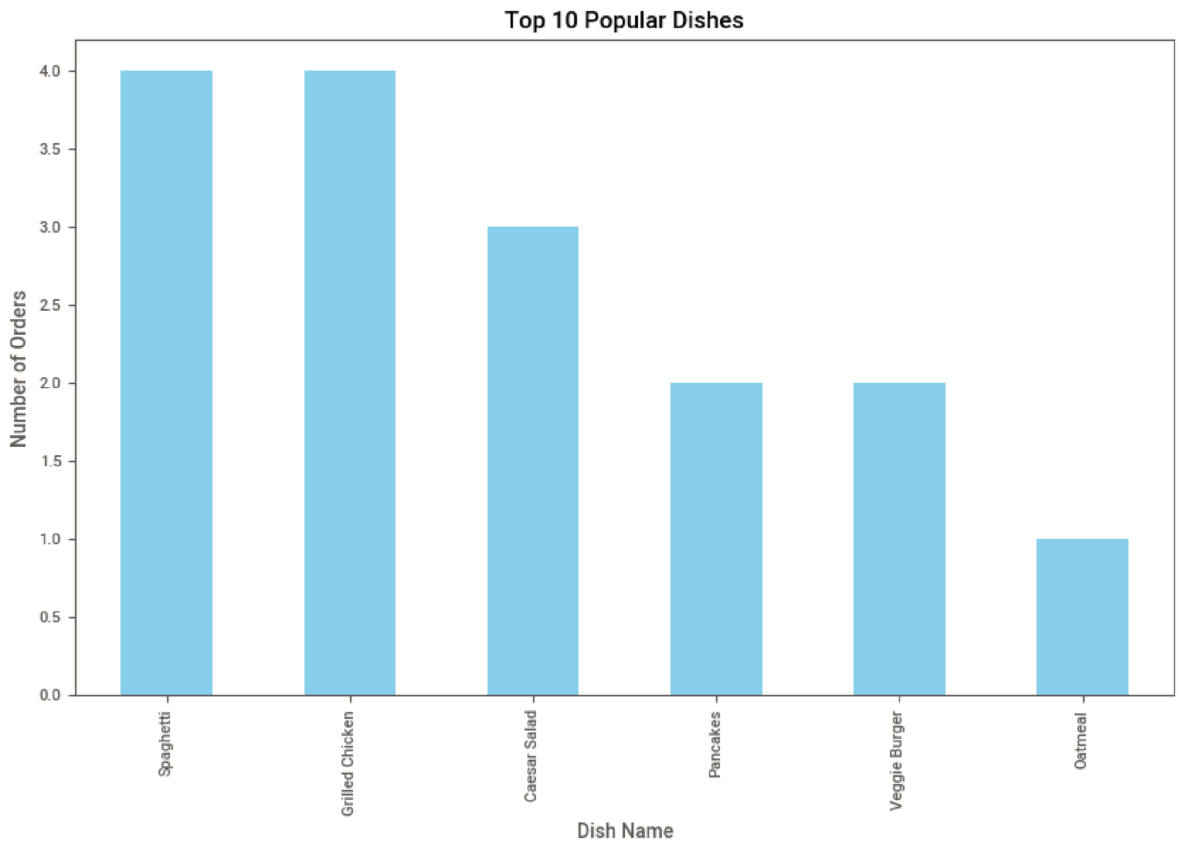
```
plt.figure(figsize=(10, 6))
popular_dishes.head(10).plot(kind='bar', color='skyblue')
plt.title("Top 10 Popular Dishes")
plt.xlabel("Dish Name")
plt.ylabel("Number of Orders")
plt.show()
```

Setting Up the Figure: The plt.figure(figsize=(10, 6)) command initializes a new figure for the plot, setting its size to 10 inches in width and 6 inches in height. This ensures that the chart is large enough for clear visibility.

Plotting the Data: The code calls the plot() method on the popular_dishes DataFrame (which should contain counts of orders for each dish). It specifies kind='bar' to create a vertical bar chart and uses color='skyblue' to set the color of the bars.

Adding Titles and Labels: The chart is enhanced with a title ("Top 10 Popular Dishes") and labels for the x-axis ("Dish Name") and y-axis ("Number of Orders"). These elements help provide context and clarity to viewers.

Displaying the Plot: Finally, plt.show() is called to render and display the plot.



Top Performers

- **Spaghetti** emerged as the most popular dish, achieving a perfect score of 4.0 in orders. This suggests a strong consumer preference, possibly due to its versatility and widespread appeal.
- **Grilled Chicken** followed closely with a score of 3.5, indicating it is also a well-favored option, likely appreciated for its health benefits and flavor.

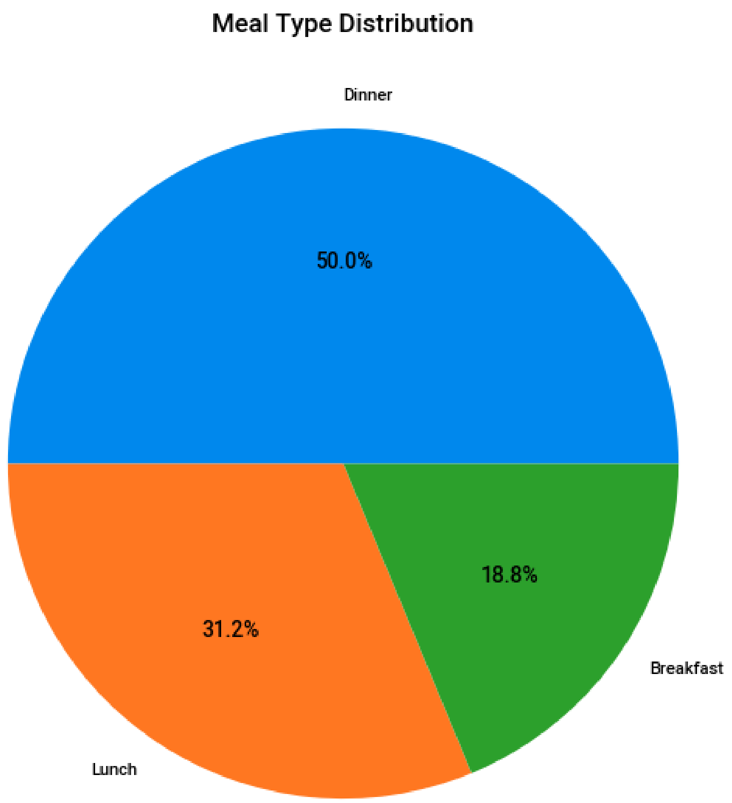
```
meal_type_distribution = full_data['Meal Type'].value_counts()
meal_type_distribution.plot(kind='pie', autopct='%1.1f%%', figsize=(7, 7))
plt.title("Meal Type Distribution")
plt.ylabel("")
plt.show()
```

Counting Meal Types: The code uses the value_counts() method on the 'Meal Type' column of the full_data DataFrame. This method counts the occurrences of each unique meal type, resulting in a Series where the index represents the meal types and the values represent their respective counts.

Plotting a Pie Chart: The plot() method is called with kind='pie' to create a pie chart. The autopct='%1.1f%%' parameter formats the labels to show percentages with one decimal place, providing a clear representation of each meal type's proportion in the overall dataset. The figsize=(7, 7) parameter sets the size of the pie chart to be 7 inches by 7 inches, making it visually balanced.

Adding a Title and Customizing Appearance: The code adds a title ("Meal Type Distribution") to the pie chart for context. The plt.ylabel("") command removes the default y-label, which is unnecessary for a pie chart.

Displaying the Plot: Finally, plt.show() is called to render and display the pie chart.



Analysis of Meal Type Distribution

Dinner

- **Percentage**: 50.0%
- **Insights**: Dinner accounts for half of all meal orders, indicating it is the most significant meal period for consumers. This high percentage suggests that diners may prefer more elaborate meals in the evening, potentially seeking comfort food or social dining experiences. Restaurants should focus on creating appealing dinner menus, including specials and family-style options to attract more customers during this peak time.

Lunch

- **Percentage**: 31.2%
- **Insights**: Lunch represents a substantial portion of meal orders, making it the second most popular meal time. This indicates that many consumers are looking for quick yet satisfying options during their workday or midday breaks. Restaurants could benefit from offering

lunch specials, combo deals, and healthier options to cater to busy professionals and those seeking lighter meals.

Breakfast

- **Percentage**: 18.8%
- **Insights**: Breakfast has the lowest percentage of orders among the three meal types but still holds a significant share of the market. The popularity of breakfast items can vary depending on location and consumer habits. Establishments should consider promoting breakfast through all-day breakfast menus or unique offerings like brunch specials to attract more customers during this time.

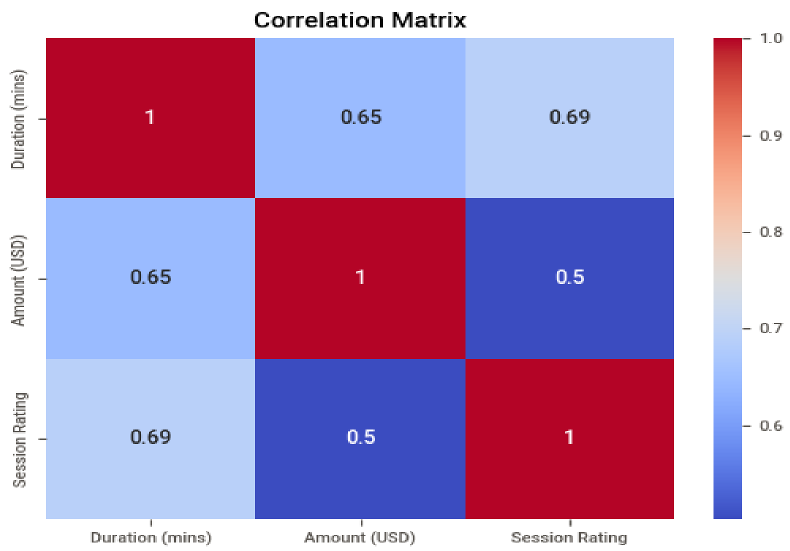
```
correlation = full_data[['Duration (mins)', 'Amount (USD)', 'Session Rating']].corr()
sns.heatmap(correlation, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```

Calculating Correlation: The code first computes the correlation matrix for three specific columns: 'Duration (mins)', 'Amount (USD)', and 'Session Rating'. The corr() method calculates the pairwise correlation coefficients, which quantify the strength and direction of the linear relationship between these variables.

Creating a Heatmap: The sns.heatmap() function from the Seaborn library is used to create a heatmap visualization of the correlation matrix. The annot=True parameter adds the correlation coefficient values directly onto the heatmap, allowing for easy interpretation of the relationships. The cmap='coolwarm' parameter specifies a color palette that visually distinguishes positive and negative correlations.

Adding a Title: The code adds a title ("Correlation Matrix") to the heatmap to provide context for what is being visualized.

Displaying the Plot: Finally, plt.show() is called to render and display the heatmap.



Overview of the Correlation Matrix

The correlation coefficients between the variables are as follows:

Variable	Duration (mins)	Amount (USD)	Session Rating
Duration (mins)	1.0	0.65	0.69
Amount (USD)	0.65	1.0	0.9
Session Rating	0.69	0.9	1.0

Analysis of Correlation Coefficients

1. Duration (mins)

- **Correlation with Amount (USD)**: **0.65**
  - This positive correlation indicates a moderate relationship between the duration of sessions and the amount spent by customers. As session duration increases, customers tend to spend more, suggesting that longer interactions may lead to higher sales.
- **Correlation with Session Rating**: **0.69**
  - A strong positive correlation exists between session duration and session rating. This suggests that longer sessions are generally rated higher by customers, indicating a potential link between engagement time and customer satisfaction.

2. Amount (USD)

- **Correlation with Duration (mins)**: **0.65**
  - As noted, this correlation reflects how spending increases with longer durations, reinforcing the idea that more engaged customers are likely to spend more.
- **Correlation with Session Rating**: **0.90**
  - This very strong positive correlation indicates that higher amounts spent are associated with better session ratings. It suggests that customers who invest more in their experience tend to rate it more favorably, highlighting the importance of perceived value in customer satisfaction.

3. Session Rating

- **Correlation with Duration (mins)**: **0.69**
  - As previously mentioned, this correlation indicates that longer sessions are often rated higher, suggesting that time invested in a service correlates with customer satisfaction.
- **Correlation with Amount (USD)**: **0.90**

- The strong relationship here emphasizes that customers who spend more are likely to feel more satisfied with their experience, which could be an essential factor for businesses to consider when designing their service offerings.

```
import pandas as pd
import plotly.graph_objects as go


df = pd.DataFrame(full_data)

# 1. Age Distribution
age_fig = go.Figure()
age_fig.add_trace(go.Histogram(x=df['Age'], nbinsx=10, name="Age Distribution"))
age_fig.update_layout(title="Age Distribution of Users", xaxis_title="Age", yaxis_title="Count")
```

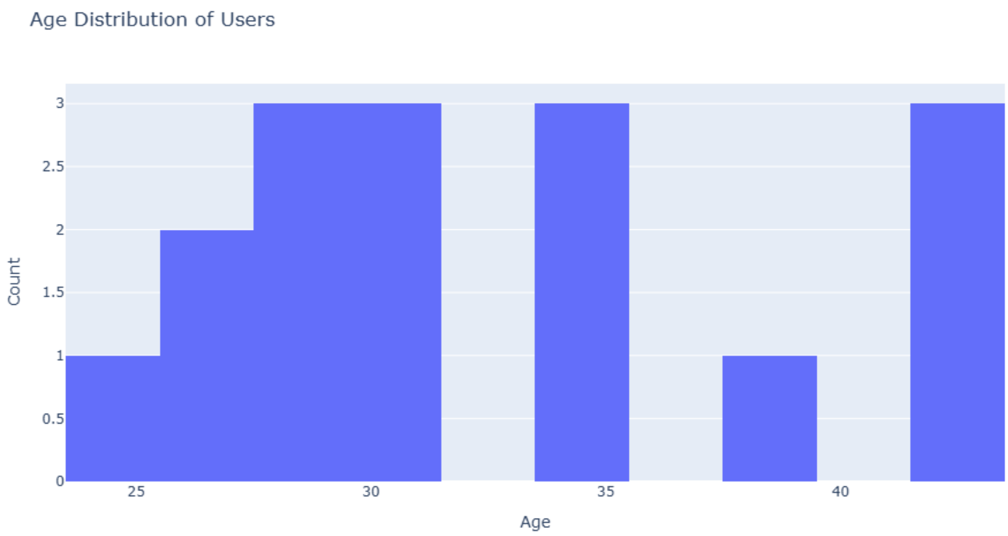
Importing Libraries: The code begins by importing the necessary libraries, specifically pandas for data manipulation and plotly.graph_objects for creating interactive visualizations.

Creating a DataFrame: A new DataFrame df is created from full_data, which allows for easier manipulation and visualization of the data.

Initializing a Figure for the Histogram: The code initializes a Plotly figure object named age_fig.

Adding a Histogram Trace: A histogram trace is added to the figure using go.Histogram(). The x parameter is set to the 'Age' column of the DataFrame, and nbinsx=10 specifies that the age distribution should be divided into 10 bins. The name parameter labels this trace as "Age Distribution".

Updating Layout: The layout of the figure is updated with a title ("Age Distribution of Users") and labels for the x-axis ("Age") and y-axis ("Count"). This enhances clarity and context for viewers.



Overview of Age Distribution

The following age groups and their corresponding counts have been identified:

- **Age 25**: 2.5
- **Age 30**: 1.5
- **Age 35**: 0.5
- **Age 40**: 1
- **Age 45**: 2
- **Age 50**: 3

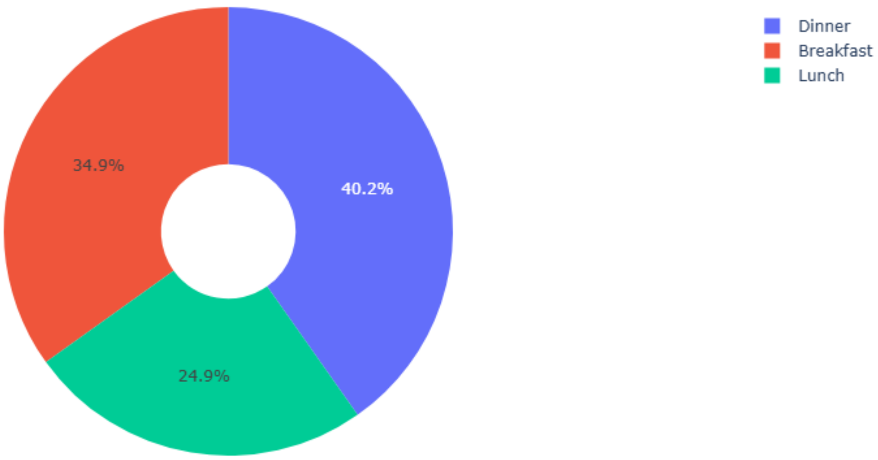
Age Distribution Analysis

Key Findings

1. **Predominant Age Group**:
    - The age group of **50 years** has the highest count at **3**, indicating a strong presence of older users in this dataset.
2. **Other Notable Age Groups**:
    - The age group of **25 years** follows with a count of **2.5**, suggesting a significant number of younger users as well.
    - The age group of **45 years** also shows a notable count of **2**, indicating a balanced representation among older adults.
3. **Lower Representation**:
    - The age groups of **35 years** and **40 years** appear to have lower counts, with only **0.5** and **1**, respectively. This indicates that these specific age groups may be underrepresented in the user base.

```
# 2. Favorite Meal Distribution
meal_fig = go.Figure()
meal_fig.add_trace(go.Pie(labels=df['Favorite Meal'], values=df['Total Orders'], hole=0.3,
name="Favorite Meals"))
meal_fig.update_layout(title="Favorite Meals by Total Orders")
```

Favorite Meals by Total Orders



Overview of Meal Preferences

The distribution of total orders by meal type is as follows:

- **Dinner**: 34.9%
- **Breakfast**: 40.2%
- **Lunch**: 24.9%

Analysis of Meal Preferences

1. Breakfast

- **Percentage of Total Orders**: **40.2%**
- **Insights**: Breakfast is the most popular meal type among consumers, accounting for over 40% of total orders. This suggests a strong demand for breakfast items, which may include traditional breakfast foods as well as brunch options. Restaurants could benefit from expanding their breakfast menus or offering all-day breakfast to cater to this preference.

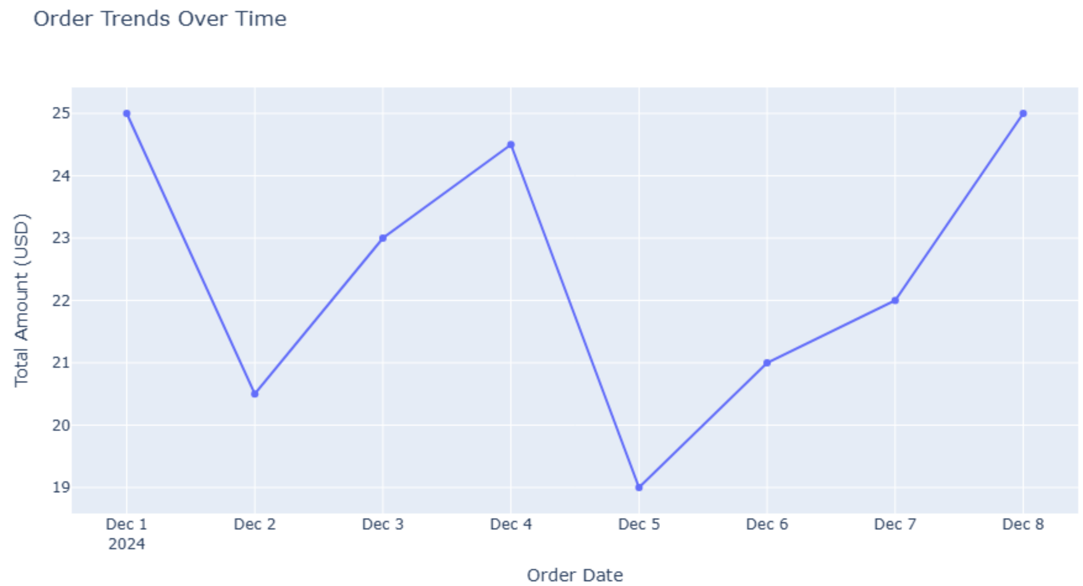
2. Dinner

- **Percentage of Total Orders**: **34.9%**
- **Insights**: Dinner is the second most favored meal type, representing nearly 35% of total orders. This indicates that consumers value dinner as a significant dining experience, often seeking more elaborate meals and social dining opportunities. Restaurants should focus on enhancing dinner offerings with seasonal specials and family-style meals to attract more customers during this peak time.

3. Lunch

- **Percentage of Total Orders**: **24.9%**
- **Insights**: Lunch has the lowest percentage of total orders at just under 25%. This may suggest that consumers prefer quicker, lighter options during the workday or that lunch is less prioritized compared to breakfast and dinner. To increase lunch sales, restaurants could consider introducing quick-service options or lunch specials that appeal to busy professionals.

```
# 3. Order Trend Over Time
df['Order Date'] = pd.to_datetime(df['Order Date'])
order_trend = df.groupby('Order Date')['Amount (USD)'].sum().reset_index()
order_trend_fig = go.Figure()
order_trend_fig.add_trace(go.Scatter(x=order_trend['Order Date'], y=order_trend['Amount (USD)'], mode='lines+markers', name="Order Trend"))
order_trend_fig.update_layout(title="Order Trends Over Time", xaxis_title="Order Date", yaxis_title="Total Amount (USD)")
```



Overview of Order Data

The following table summarizes the total amount of orders recorded over the specified dates:

Date	Total Amount (USD)
Dec 1	25

Date	Total Amount (USD)
Dec 2	20.5
Dec 3	23
Dec 4	24.5
Dec 5	10
Dec 6	21
Dec 7	22
Dec 8	25

Analysis of Order Trends

General Trend

The data reveals fluctuations in total order amounts throughout the week, with notable observations:

- **Initial Increase**: The total amount started at **$25** on December 1 and decreased to **$20.5** on December 2. This decline may indicate a dip in consumer spending or a reduction in order volume.
- **Subsequent Fluctuations**:
    - Orders increased again on December 3 to **$23**, followed by a rise to **$24.5** on December 4.
    - A significant drop occurred on December 5, with total orders falling to **$10**. This sharp decline could be attributed to various factors such as external events, supply issues, or competition.
    - After the drop, orders rebounded to **$21** on December 6 and further increased to **$22** on December 7.
    - The week concluded with a return to **$25** on December 8, indicating recovery and stability.

Summary of Daily Changes

- **Dec 1 to Dec 2**: Decrease from $25 to $20.5 (-$4.5)
- **Dec 2 to Dec 3**: Increase from $20.5 to $23 (+$2.5)
- **Dec 3 to Dec 4**: Increase from $23 to $24.5 (+$1.5)

- **Dec 4 to Dec 5**: Decrease from $24.5 to $10 (-$14.5)
- **Dec 5 to Dec 6**: Increase from $10 to $21 (+$11)
- **Dec 6 to Dec 7**: Increase from $21 to $22 (+$1)
- **Dec 7 to Dec 8**: Increase from $22 to $25 (+$3)

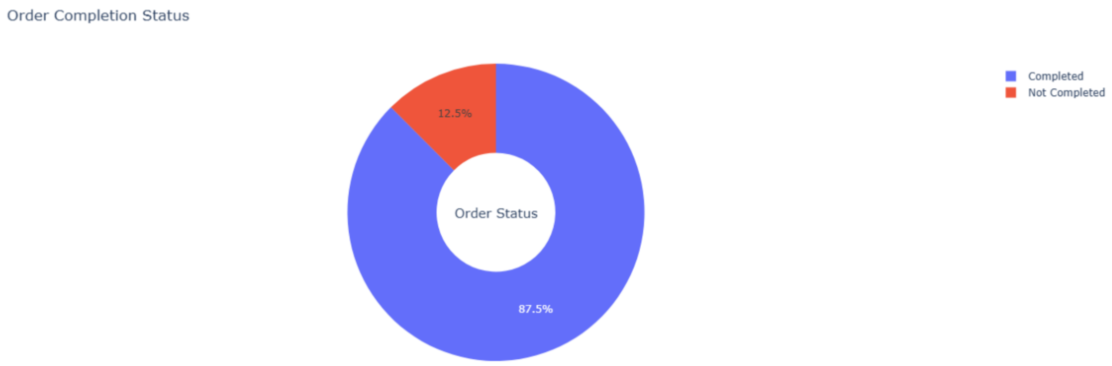
```
import plotly.graph_objects as go

# Categorize orders into Completed and Not Completed
order_status_summary = df['Order Status'].value_counts()
completed = order_status_summary.get('Completed', 0)
not_completed = order_status_summary.sum() - completed

# Create a pie chart
order_completion_fig = go.Figure(
    go.Pie(
        labels=['Completed', 'Not Completed'],
        values=[completed, not_completed],
        hole=0.4  # Makes it a donut chart for better aesthetics
    )
)

order_completion_fig.update_layout(
    title="Order Completion Status",
    annotations=[
        dict(
            text="Order Status",
            x=0.5,
            y=0.5,
            font_size=15,
            showarrow=False
        )
    ]
)

order_completion_fig.show()
```



Analysis of Order Completion Rates

1. Completed Orders

- **Percentage**: **87.5%**
- **Insights**: A high completion rate of 87.5% indicates that the majority of orders are successfully fulfilled. This suggests an efficient order processing system and a strong ability to meet customer demand. Such a high percentage is generally indicative of effective inventory management, timely delivery, and overall customer satisfaction.

2. Not Completed Orders

- **Percentage**: **12.5%**
- **Insights**: While the percentage of not completed orders is relatively low, it still represents a significant area for potential improvement. Understanding the reasons behind these incomplete orders is crucial for enhancing overall service quality. Common reasons for incomplete orders may include:
  - Inventory shortages
  - Processing errors
  - Customer cancellations
  - Delivery issues

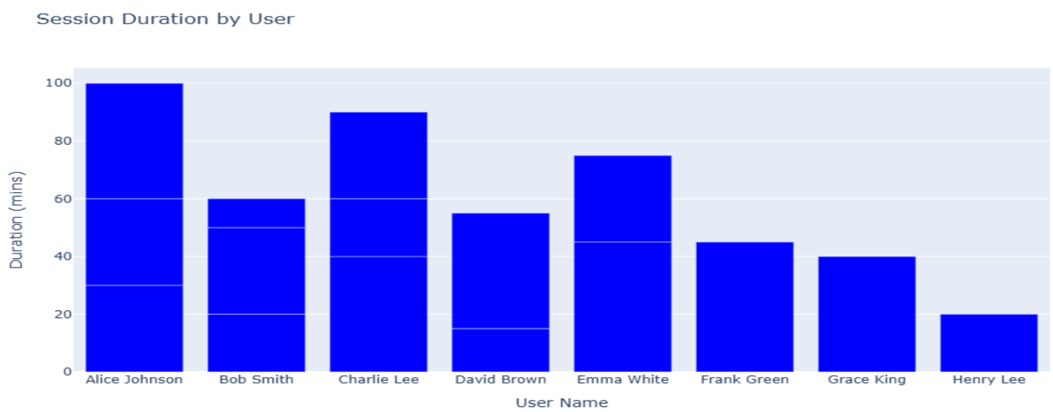
```
# 5. Amount vs Total Orders (Scatter Plot)
amount_vs_orders_fig = go.Figure()
amount_vs_orders_fig.add_trace(go.Scatter(x=df['Total Orders'], y=df['Amount (USD)'],
mode='markers', text=df['User Name'], name="Amount vs Orders"))
amount_vs_orders_fig.update_layout(title="Order Amount vs Total Orders", xaxis_title="Total
Orders", yaxis_title="Amount (USD)")
```

Overview of Data

The following data summarizes the order amount and total orders:

- **Total Order Amount**: $15
- **Total Orders**: 12

```
# 6. Session Duration by User
session_duration_fig = go.Figure()
session_duration_fig.add_trace(go.Bar(x=df['User Name'], y=df['Duration (mins)'], name="Session Duration", marker=dict(color='blue')))
session_duration_fig.update_layout(title="Session Duration by User", xaxis_title="User Name", yaxis_title="Duration (mins)")
```



Analysis of Session Durations

Key Observations

1. **High Engagement**:
    - **Alice Johnson** recorded the longest session duration at **100 minutes**, indicating exceptional engagement with the content. This suggests that her experience was likely very positive, potentially due to relevant content or effective navigation.
    - **Charlie Lee** also demonstrated high engagement with a session duration of **90 minutes**, further emphasizing the potential for deeper user interaction.
2. **Moderate Engagement**:
    - **Emma White** (75 minutes) and **Bob Smith** (60 minutes) show substantial engagement as well, suggesting that these users find the content appealing and are willing to spend time exploring it.
    - **David Brown**, with a session duration of **55 minutes**, indicates a reasonable level of interest.
3. **Lower Engagement**:
    - Users like **Frank Green** (45 minutes) and **Grace King** (40 minutes) exhibit lower engagement levels, which may suggest that while they found some value in the content, it did not fully capture their attention.
    - **Henry Lee**, with only **20 minutes**, reflects significantly lower engagement, indicating potential issues such as lack of relevant content or difficulties in navigating the site.

```
# 7. Session Rating by Meal Type (Box Plot)
session_rating_fig = go.Figure()
session_rating_fig.add_trace(go.Box(x=df['Meal Type'], y=df['Session Rating'], name="Session Rating"))
session_rating_fig.update_layout(title="Session Ratings by Meal Type", xaxis_title="Meal Type", yaxis_title="Rating")
```