

# Introduction to Python Project : FoodHub Data Analysis

## Problem Statement

### Context

The number of restaurants in New York is increasing day by day. Lots of students and busy professionals rely on those restaurants due to their hectic lifestyles. Online food delivery service is a great option for them. It provides them with good food from their favorite restaurants. A food aggregator company FoodHub offers access to multiple restaurants through a single smartphone app.

The app allows restaurants to receive a direct online order from a customer. The app assigns a delivery person from the company to pick up the order after it is confirmed by the restaurant. The delivery person then uses the map to reach the restaurant and waits for the food package. Once the food package is handed over to the delivery person, he/she confirms the pick-up in the app and travels to the customer's location to deliver the food. The delivery person confirms the drop-off in the app after delivering the food package to the customer. The customer can rate the order in the app. The food aggregator earns money by collecting a fixed margin of the delivery order from the restaurants.

### Objective

The food aggregator company has stored the data of the different orders made by the registered customers in their online portal. They want to analyze the data to get a fair idea about the demand of different restaurants which will help them in enhancing their customer experience. Suppose you are hired as a Data Scientist in this company and the Data Science team has shared some of the key questions that need to be answered. Perform the data analysis to find answers to these questions that will help the company improve its business.

## Data Dictionary

The data includes various information related to a food order. A detailed data dictionary is provided below.

### Data Dictionary

- **order\_id:** Unique ID of the order
- **customer\_id:** ID of the customer who ordered the food
- **restaurant\_name:** Name of the restaurant
- **cuisine\_type:** Cuisine ordered by the customer
- **cost\_of\_the\_order:** Cost of the order

- day\_of\_the\_week: Indicates whether the order is placed on a weekday or weekend (The weekday is from Monday to Friday and the weekend is Saturday and Sunday)
- rating: Rating given by the customer out of 5
- food\_preparation\_time: Time (in minutes) taken by the restaurant to prepare the food. This is calculated by taking the difference between the timestamps of the restaurant's order confirmation and the delivery person's pick-up confirmation.
- delivery\_time: Time (in minutes) taken by the delivery person to deliver the food package. This is calculated by taking the difference between the timestamps of the delivery person's pick-up confirmation and drop-off information

## Let us start by importing the required libraries

```
In [1]: # import libraries for data manipulation
import pandas as pd

# import libraries for data visualization
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

# misc/other imports
import warnings

# import libraries for python utils
import tabulate
```

```
In [2]: # GLOBAL OPTIONS ---
# To avoid clutter in the output, suppress warnings
warnings.filterwarnings("ignore")

# Set global figure size for all plots
plt.rcParams["figure.figsize"] = (15, 6)
plt.rcParams["font.size"] = 12

# Set the grid style globally (for both major and minor grids)
# plt.rcParams['axes.grid'] = True
plt.rcParams["grid.linestyle"] = "--"
plt.rcParams["grid.alpha"] = 0.6

plt.rcParams["grid.color"] = "#E2E0E0FF"

# Set global font sizes for title, xlabel, and ylabel
plt.rcParams["axes.titlesize"] = 16 # Default font size for plot titles
plt.rcParams["axes.labelsize"] = 12 # Default font size for x and y labels

# Seaborn style ---
# Set global Seaborn style and context
# ref: https://r02b.github.io/seaborn_palettes/
sns.set_theme(style="whitegrid", palette="muted", context="notebook")
```

## ❖ Python Utils

```
In [3]: # UTILS (plot) ---

def set_fig_size(width, height):
```

```

"""
Set figure size (for upcoming plot only)
NOTE: this will apply to next plot only (& not all plots)
"""

plt.figure(figsize=(width, height))

def set_small_fig():
    """
    (5, 5) -> ar: 1
    Square shaped fig (eg: scatter plot)
    """
    set_fig_size(5, 5)

def set_medium_fig():
    """
    (9, 6) -> ar: 1.5
    Normal | Medium
    """
    # alternate = (12, 8)
    set_fig_size(9, 6)

def set_large_fig():
    """
    (12, 6) -> ar: 2
    Large figure
    """
    set_fig_size(12, 6)

def set_vlarge_fig():
    """
    (13, 5) -> ar: 2.5
    Very Large figure (Eg Time Series covering long span)
    """
    set_fig_size(15, 6)

```

In [4]: # UTILS (PANDAS) ---

```

def tableit(series: pd.Series, index_label: str = None, value_label: str = ''):
    """
    Display a pandas Series as a formatted table using the tabulate library.
    Args:
        series (pd.Series): The pandas Series to be displayed.
        index_name (str): The name of the index column. Default to `series.name`.
    Returns:
        None
    """
    i_name = index_label if index_label else series.name
    table = tabulate.tabulate(
        series.items(), headers=[i_name, value_label], tablefmt="grid"
    )
    print(table)

# todo: decide default value for show_index
def tableit_df(df: pd.DataFrame, show_index: bool = True):
    """
    Display a pandas DataFrame as a formatted table using the tabulate library.
    Args:
        df (pd.DataFrame): The pandas DataFrame to be displayed.
    """

```

```

        show_index (bool): Whether to display the DataFrame index. Default :
    Returns:
        None
    .....
    # Convert DataFrame to a tabulated string
    table = tabulate.tabulate(
        df, headers="keys", tablefmt="pretty", showindex=show_index
    )
    print(table)

```

In [5]: # UTILS (Python | Primitive) ---

```

def snake_to_pascal(snake_str, join_with=" "):
    """Convert snake_case to PascalCase (eg my_name -> MyName)
Args:
    snake_str (str): string to convert
    join_with (str): character to join the components, default is space
"""
components = snake_str.split("_")
return join_with.join(x.title() for x in components)

def format_pct(val):
    """Format a val as percentage i.e max 2 decimal value & adding % at the
    return f"{val:.2f}%"
```

In [6]: # UTILS (Python | Stats) ---

```

def calc_iqr(series: pd.Series):
    """
    series: array of numerical values
    """
    Q1 = series.quantile(0.25)
    Q3 = series.quantile(0.75)
    IQR = Q3 - Q1
    return Q1, Q3, IQR

def to_percentage(value):
    """value is expected to be a normalized float value in [0, 1]"""
    return format_pct(value * 100)
```

### ⚡ Note

Please note that some lines in this notebook may appear as commented-out code (e.g., `print()`/`repr()`) in favor of using `tableit()`/`tableit_df()` for more structured outputs. These comments are intentionally left in place so that, in case the notebook is exported to a different format and the display results do not appear as expected, they can be easily reverted to the default behavior (e.g., using `repr()` for outputs in the original notebook).

## Understanding the structure of the data

In [7]: # uncomment and run the following lines for Google Colab  
# from google.colab import drive

```
# drive.mount('/content/drive')
```

In [8]:

```
# Load the dataset
df = pd.read_csv('foodhub_order.csv')
# Make a copy of the original DataFrame (to have a original copy of the data)
original_df = df.copy()
```

In [9]:

```
# View the first 5 rows
df.head(5)
```

Out [9]:

|   | order_id | customer_id | restaurant_name           | cuisine_type | cost_of_the_order | day_of_the_week |
|---|----------|-------------|---------------------------|--------------|-------------------|-----------------|
| 0 | 1477147  | 337525      | Hangawi                   | Korean       | 30.75             | Weekend         |
| 1 | 1477685  | 358141      | Blue Ribbon Sushi Izakaya | Japanese     | 12.08             | Weekend         |
| 2 | 1477070  | 66393       | Cafe Habana               | Mexican      | 12.23             | Weekday         |
| 3 | 1477334  | 106968      | Blue Ribbon Fried Chicken | American     | 29.20             | Weekend         |
| 4 | 1478249  | 76942       | Dirty Bird to Go          | American     | 11.59             | Weekday         |

### 🔍 Observation:

1. The dataset contains information about food orders from various restaurants.
2. There are 9 columns in the dataset: order\_id, customer\_id, restaurant\_name, cuisine\_type, cost\_of\_the\_order, day\_of\_the\_week, rating, food\_preparation\_time, and delivery\_time.
3. The 'rating' column contains both numeric and non-numeric values (e.g., 'Not given').
4. The 'day\_of\_the\_week' column indicates whether the order was placed on a weekday or weekend.
5. The 'food\_preparation\_time' and 'delivery\_time' columns are measured in minutes.

## Question 1: How many rows and columns are present in the data? [0.5 mark]

In [10]:

```
rows, cols = df.shape
rows, cols
```

Out [10]:

```
(1898, 9)
```

### 🔍 Observation:

1. The dataset contains 1898 rows and 9 columns.
2. rows corresponds to orders and columns corresponds to features of the order.

## Question 2: What are the datatypes of the different columns in the dataset? (The info() function can be used) [0.5 mark]

In [11]:

```
tableit(df.dtypes, index_label='column')
```

| column                | value   |
|-----------------------|---------|
| order_id              | int64   |
| customer_id           | int64   |
| restaurant_name       | object  |
| cuisine_type          | object  |
| cost_of_the_order     | float64 |
| day_of_the_week       | object  |
| rating                | object  |
| food_preparation_time | int64   |
| delivery_time         | int64   |

### 🔍 Observation:

1. The 'order\_id', 'customer\_id', 'food\_preparation\_time', and 'delivery\_time' have integer values (ie numeric).
2. The 'cost\_of\_the\_order' holds float values (ie numeric).
3. The 'restaurant\_name', 'cuisine\_type', 'day\_of\_the\_week', and 'rating' columns are of type object, indicating they contain string values.

### 🔧 Actionable Insights

Although the 'rating' column is currently of type `object`, it should ideally be of type `int64`, as ratings are generally *ordinal* in nature. This will be addressed later (during **data cleaning/preprocessing** step).

```
In [12]: # ? As suggested in question, so using the `info()` as well
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1898 entries, 0 to 1897
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
 --- 
 0   order_id         1898 non-null   int64  
 1   customer_id      1898 non-null   int64  
 2   restaurant_name  1898 non-null   object  
 3   cuisine_type     1898 non-null   object  
 4   cost_of_the_order 1898 non-null   float64 
 5   day_of_the_week  1898 non-null   object  
 6   rating           1898 non-null   object  
 7   food_preparation_time 1898 non-null   int64  
 8   delivery_time    1898 non-null   int64  
dtypes: float64(1), int64(4), object(4)
memory usage: 133.6+ KB
```

**Question 3:** Are there any missing values in the data? If yes, treat them using an appropriate method. [1 mark]

```
In [14]: has_missing_values = df.isnull().any().any()
has_missing_values
```

Out[14]: False

**Observation:**

There are *no* explicit missing values in the dataset. However, the **rating** column contains entries labeled as "**Not given**" which indicate the *absence* of a customer rating. These entries should be treated as missing values during analysis to ensure accurate insights.

### Question 4: Check the statistical summary of the data. What is the minimum, average, and maximum time it takes for food to be prepared once an order is placed? [2 marks]

```
In [15]: stat_summary = df.describe(include='all').T
stat_summary
```

|                              | count  | unique | top         | freq | mean          | std           |
|------------------------------|--------|--------|-------------|------|---------------|---------------|
| <b>order_id</b>              | 1898.0 | NaN    | NaN         | NaN  | 1477495.5     | 548.049724    |
| <b>customer_id</b>           | 1898.0 | NaN    | NaN         | NaN  | 171168.478398 | 113698.139743 |
| <b>restaurant_name</b>       | 1898   | 178    | Shake Shack | 219  | NaN           | NaN           |
| <b>cuisine_type</b>          | 1898   | 14     | American    | 584  | NaN           | NaN           |
| <b>cost_of_the_order</b>     | 1898.0 | NaN    | NaN         | NaN  | 16.498851     | 7.483812      |
| <b>day_of_the_week</b>       | 1898   | 2      | Weekend     | 1351 | NaN           | NaN           |
| <b>rating</b>                | 1898   | 4      | Not given   | 736  | NaN           | NaN           |
| <b>food_preparation_time</b> | 1898.0 | NaN    | NaN         | NaN  | 27.37197      | 4.632481      |
| <b>delivery_time</b>         | 1898.0 | NaN    | NaN         | NaN  | 24.161749     | 4.972637      |

**Observation:**

1. The dataset contains 1898 entries for each column.
2. The average cost of the orders is approximately 16.50 dollars.
3. The cost of an order lies between 4 to 36 dollars.
4. The average food preparation time is approximately 27 minutes.
5. Food preparation times range from a minimum of 20 minutes to a maximum of 35 minutes.
6. The average delivery time is around 24 minutes, with most orders being delivered within 30 minutes.
7. The restaurant with the most orders is "Shake Shack," receiving a total of 219 orders.
8. The most common cuisine type is "American" with 584 orders.
9. The majority of orders (ie 1351) are placed on weekends.
10. The 'rating' column has 736 entries labeled as "Not given", indicating missing ratings.

```
In [16]: # statistical summary for the 'food_preparation_time' column
food_preparation_summary = df['food_preparation_time'].describe()
food_preparation_summary
```

```
Out[16]: count    1898.000000
mean      27.371970
std       4.632481
min      20.000000
25%     23.000000
50%     27.000000
75%     31.000000
max     35.000000
Name: food_preparation_time, dtype: float64
```

### ❓ Question

What is the minimum, average, and maximum time it takes for food to be prepared once an order is placed ?

=>

The minimum time for food preparation is 20 minutes, the average time is 27.37 minutes, and the maximum time is 35 minutes

### 📌 Points

1. 25% of the orders have a preparation time of 23 minutes or less.
2. The median food preparation time is 27 minutes, meaning that half of the orders are prepared in 27 minutes or less.
3. 25% of the orders have a preparation time of 31 minutes or more.

### 📝 Gist

On average, restaurants take about 27 minutes to prepare food, with most orders falling between 23 and 31 minutes. While the fastest orders are ready in 20 minutes, some take up to 35 minutes. The slight variability in preparation time (around 4 minutes) suggests a consistent but slightly flexible pace across restaurants.

## Question 5: How many orders are not rated? [1 mark]

```
In [17]: df['rating'].value_counts()
```

```
Out[17]: Not given    736
5            588
4            386
3            188
Name: rating, dtype: int64
```

### ✅ Answer

There are 736 orders that are not rated, as indicated by the "Not given" entries in the rating column.

### ⚡ Note

There are no missing values in the 'rating' column, so these entries clearly represent the absence of a rating & hence `value_counts()` can be reliable to answer the above question

If that's was not the case, then we may need to determine it with different approach

```
In [18]: # GENERIC WAY :- An alternate way to find missing values
# NOTE: this will be helpful especially when there are more than 1 rating va
#       (ie for eg 'Not Given', 'Not rated', 'NA', etc)
#       In present case, there is only 1 value 'Not rated',
#       so it was straight forward to determine just via seeing stats-info :| 

# find non-numeric values in the 'rating' column by converting the column to
non_numeric_ratings_count = pd.to_numeric(df['rating'], errors='coerce').isna().sum()
print(f"Number of non-numeric values in the 'rating' column: {non_numeric_ra
```

Number of non-numeric values in the 'rating' column: 736

### Observation

The 'rating' column contains 736 entries marked as 'Not given,' indicating that these orders were not rated by customers.

---

## Data Cleaning / Data PreProcessing

```
In [19]: # ! NOTE: only attribute which are used further in some sort of computation,
#       # 1.
# NOTE: the modification is done inplace, as original df backup us already
#       # convert the 'rating' column to a numeric type and replace 'Not given' with
df['rating'] = pd.to_numeric(df['rating'], errors='coerce')
```

```
In [20]: # Verify the changes
df['rating'].dtype
```

```
Out[20]: dtype('float64')
```

```
In [21]: distinct_ratings = df['rating'].nunique()
distinct_ratings
```

```
Out[21]: 3
```

```
In [22]: df['rating'].value_counts()
```

```
Out[22]: 5.0    588
4.0    386
3.0    188
Name: rating, dtype: int64
```

```
In [23]: # get non-numeric values count in the new 'rating' column
new_val = df['rating'].isnull().sum()
prev_val = non_numeric_ratings_count
print(f"For verification (non-numeric count): {new_val} (previously {prev_val})")
```

For verification (non-numeric count): 736 (previously 736), is\_same = True

```
In [24]: df['rating'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 1898 entries, 0 to 1897
Series name: rating
Non-Null Count Dtype
-----
1162 non-null    float64
dtypes: float64(1)
memory usage: 15.0 KB
```

```
In [25]: # !! Assumption : Before this computation, ratings column should be converted to float
```

```
# Check:-  
# Top 5 restaurants with the highest average rating  
top_5_restaurant_with_highest_ratings = df.groupby('restaurant_name')['rating'].mean().reset_index()  
tableit(top_5_restaurant_with_highest_ratings, index_label='Restaurant', value_col='Average Rating')
```

| Restaurant         | Average Rating |
|--------------------|----------------|
| 'wichcraft         | 5              |
| 67 Burger          | 5              |
| Anjappar Chettinad | 5              |
| Asuka Sushi        | 5              |
| Balade             | 5              |

## Exploratory Data Analysis (EDA)

### Univariate Analysis

#### ❖ Python Utils

```
In [26]: class UnivariatePlotHelper:  
    def __init__(self, df: pd.DataFrame):  
        self.df = df  
  
    def draw_countplot(  
        self,  
        attribute: str,  
        *,  
        label=None,  
        rot=0,  
        order=None,  
        sort=True,  
        palette=None,  
        showgrid=None,  
    ):  
        ....  
        sort: will be used only if order is None  
        ....  
        # prep (meta) --  
        xlabel = label if label else snake_to_pascal(attribute)  
        priority = None  
        if order is not None:
```

```

        priority = order
    elif sort:
        # sort by count
        # NOTE: value_counts() returns a Series sorted by values (in DESCENDING ORDER)
        priority = df[attribute].value_counts().index

        # plot (crux) --
        ax = sns.countplot(data=df, x=attribute, order=priority, palette=palettes[0])
        # display count above each bar
        ax.bar_label(ax.containers[0])

        # aesthetics --
        plt.title(f"Bar Graph of {xlabel}")
        plt.xlabel(xlabel)
        plt.ylabel("count")
        plt.xticks(rotation=rot)
        if showgrid:
            plt.grid(True)
        plt.show()

    def draw_boxplot(self, attribute: str, *, label=None):
        """
        attribute: name of the column
        """
        # prep
        xlabel = label if label else snake_to_pascal(attribute)
        # plot
        sns.boxplot(data=df, x=attribute)
        # aesthetics
        plt.title(f"Box Plot of {xlabel}")
        plt.xlabel(xlabel)
        plt.show()

    def draw_boxplot_fromseries(self, series: pd.Series, *, label=None):
        """
        series: array of numeric values
        """
        # prep
        xlabel = label if label else snake_to_pascal(series.name)
        # plot
        sns.boxplot(x=series)
        # aesthetics
        plt.title(f"Box Plot of {xlabel}")
        plt.xlabel(xlabel)
        plt.show()

    def draw_histplot(self, attribute: str, *, bins="auto", label=None, color="blue"):
        # prep
        xlabel = label if label else snake_to_pascal(attribute)
        # plot
        sns.histplot(data=df, x=attribute, bins=bins, kde=True, color=color)
        # aesthetics
        plt.title(f"Histogram of {xlabel}")
        plt.xlabel(xlabel)
        plt.ylabel("frequency")
        plt.show()

    def draw_pieplot(self, attribute: str, *, label=None):
        # prep
        label = label if label else snake_to_pascal(attribute)
        # plot
        counts = df[attribute].value_counts()
        # aesthetics
        plt.title(f"Pie Chart of {label}", pad=20)

```

```
plt.pie(counts, labels=counts.index, autopct="%1.1f%%", startangle=90)
plt.axis("equal")
plt.show()
```

**Question 6:** Explore all the variables and provide observations on their distributions. (Generally, histograms, boxplots, countplots, etc. are used for univariate exploration.) [9 marks]

In [27]: `# helpers (init | singleton)`

```
plot_helper = UnivariatePlotHelper(df)
```

In [28]: `df.columns.to_list()`

Out[28]:

```
['order_id',
 'customer_id',
 'restaurant_name',
 'cuisine_type',
 'cost_of_the_order',
 'day_of_the_week',
 'rating',
 'food_preparation_time',
 'delivery_time']
```

In [29]: `# total orders`

```
total_orders = df.shape[0]
total_customers = df['customer_id'].nunique()
print(f'Total Orders: {total_orders} & Total Customers: {total_customers}')
```

Total Orders: 1898 & Total Customers: 1200

 Note: Since order\_id and customer\_id are nominal in nature, used primarily for identification, they are excluded from further analysis.

**Attribute: restaurant\_name**

In [30]: `# Descriptive stats`

```
restaurant_order_counts = df['restaurant_name'].value_counts()
restaurant_order_counts
```

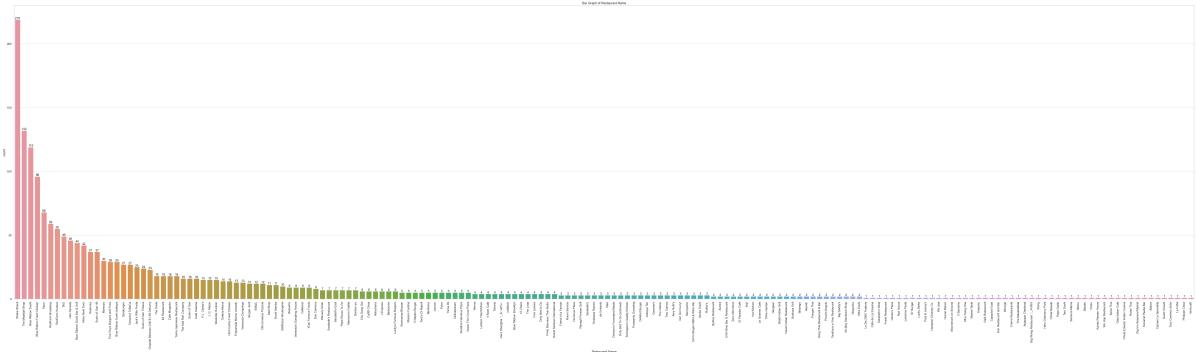
Out[30]:

|                           |     |
|---------------------------|-----|
| Shake Shack               | 219 |
| The Meatball Shop         | 132 |
| Blue Ribbon Sushi         | 119 |
| Blue Ribbon Fried Chicken | 96  |
| Parm                      | 68  |
| ...                       |     |
| Sushi Choshi              | 1   |
| Dos Caminos Soho          | 1   |
| La Follia                 | 1   |
| Philippe Chow             | 1   |
| 'wichcraft                | 1   |

Name: restaurant\_name, Length: 178, dtype: int64

In [31]: `# PLOT`

```
# as there are too many categories, so setting bit unconventional figsize
set_fig_size(80, 20)
# restaurant names -> nominal variable | categorical
plot_helper.draw_countplot('restaurant_name', rot=90)
```



Note:

As the restaurant\_name is nominal in nature, this plot is hard to visualize. Hence let try to pick some priority restaurants and plot the graph again.

```
In [32]: top_n = 10
# Top 10 restaurants with the highest demand (ie maximum orders)
top_restaurants_order_count = df['restaurant_name'].value_counts().nlargest(10)
```

```
In [33]: top_restaurants_names = top_restaurants_order_count.index.to_list()
print(f'Top {top_n} restaurants with the highest demand (ie maximum orders):')
print(top_restaurants_names)
```

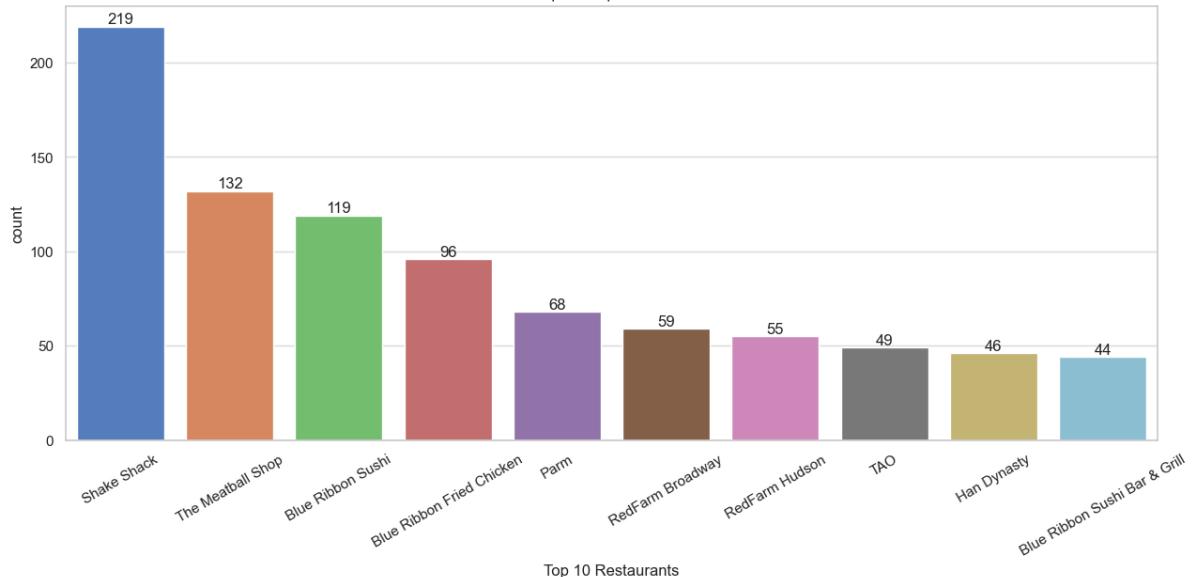
Top 10 restaurants with the highest demand (ie maximum orders):  
['Shake Shack', 'The Meatball Shop', 'Blue Ribbon Sushi', 'Blue Ribbon Fried Chicken', 'Parm', 'RedFarm Broadway', 'RedFarm Hudson', 'TAO', 'Han Dynasty', 'Blue Ribbon Sushi Bar & Grill']

```
In [34]: tableit(top_restaurants_order_count, value_label="#orders")
```

| restaurant_name               | #orders |
|-------------------------------|---------|
| Shake Shack                   | 219     |
| The Meatball Shop             | 132     |
| Blue Ribbon Sushi             | 119     |
| Blue Ribbon Fried Chicken     | 96      |
| Parm                          | 68      |
| RedFarm Broadway              | 59      |
| RedFarm Hudson                | 55      |
| TAO                           | 49      |
| Han Dynasty                   | 46      |
| Blue Ribbon Sushi Bar & Grill | 44      |

```
In [35]: plot_helper.draw_countplot('restaurant_name', label=f'Top {top_n} Restaurant
```

Bar Graph of Top 10 Restaurants

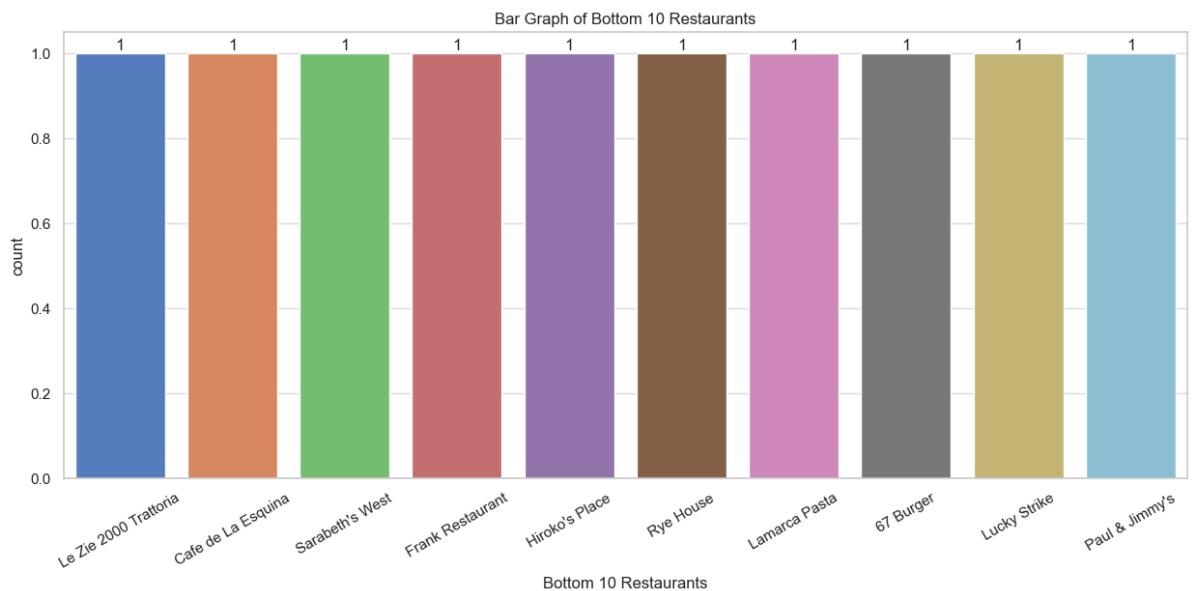


```
In [36]: # less demanding restaurants order counts
least_n = 10
bottom_restaurants_order_count = df['restaurant_name'].value_counts().nlargest(least_n)
```

```
In [37]: bottom_restaurants_names = bottom_restaurants_order_count.index.to_list()
print(f'The {least_n} restaurants with the least demand (ie fewest orders):')
print(bottom_restaurants_names)
```

The 10 restaurants with the least demand (ie fewest orders):  
 ['Le Zie 2000 Trattoria', 'Cafe de La Esquina', "Sarabeth's West", 'Frank Restaurant', "Hiroko's Place", 'Rye House', 'Lamarca Pasta', '67 Burger', 'Lucky Strike', "Paul & Jimmy's"]

```
In [38]: plot_helper.draw_countplot('restaurant_name', label=f'Bottom {least_n} Restaurants')
```



🔍 Insights :

All the bottom restaurants have only 1 order

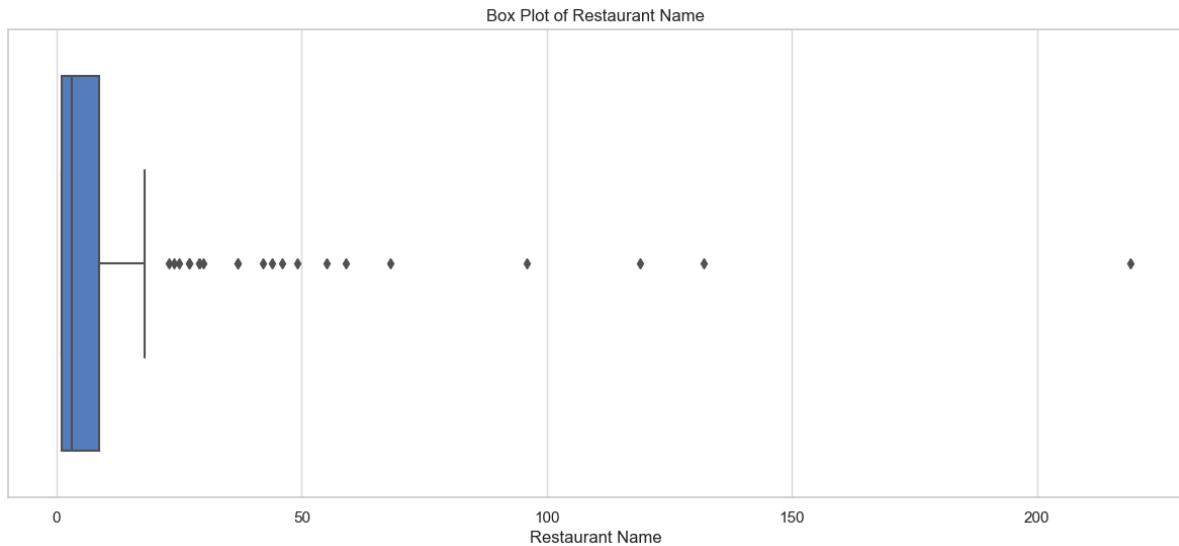
```
In [39]: # Descriptive stats
restaurant_order_counts.describe()
```

```
Out[39]: count    178.000000
          mean     10.662921
          std      24.047008
          min      1.000000
          25%     1.000000
          50%     3.000000
          75%     8.750000
          max     219.000000
          Name: restaurant_name, dtype: float64
```

Idea : Let try to reckon restaurant order demand summary

In [40]: # PLOT

```
plot_helper.draw_boxplot_fromseries(restaurant_order_counts)
```



In [41]: # STATS

```
q1, q3, _ = calc_iqr(restaurant_order_counts)
q1, q3
```

Out[41]: (1.0, 8.75)

In [42]: # Explore | Stats

```
# restaurants with more than 10
restaurant_names_with_more_than_10_orders = restaurant_order_counts[restaurant_order_counts['order_count'] > 10]

# get total orders count from all such restaurants
filtered_df = df[df['restaurant_name'].isin(restaurant_names_with_more_than_10_orders)]

filtered_df.shape
```

Out[42]: (1494, 9)

In [43]: df.shape

Out[43]: (1898, 9)

```
diff = df.shape[0] - filtered_df.shape[0]
diff_pct = diff / df.shape[0] * 100
print(f"Total orders from restaurants with more than 10 orders: {diff} (ie {diff_pct:.2f}%)")
```

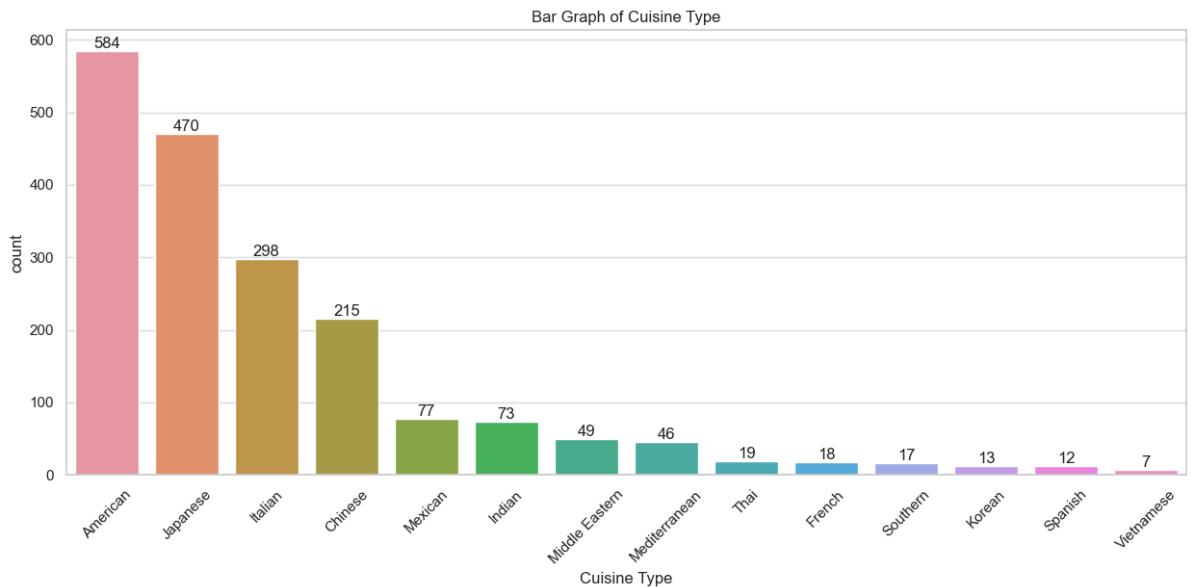
Total orders from restaurants with more than 10 orders: 404 (ie 21.29%)

## 🔍 Insights

Only 400 (ie approx 20%) restaurants have more than 10 orders

### Attribute: cuisine\_type

```
In [45]: # categorical
x = 'cuisine_type'
plot_helper.draw_countplot(x, rot=45)
```



### 📌 Points

- The most common cuisine types are American, Japanese, and Italian.
- There are very fewer orders for cuisines like Korean, Spanish, and Vietnamese.

The distribution indicates a higher preference for certain cuisines among customers, Hence restaurant should focus on such cuisines to attract more customers.

```
In [46]: # helper cell
# : to find next column name
df.columns
```

```
Out[46]: Index(['order_id', 'customer_id', 'restaurant_name', 'cuisine_type',
       'cost_of_the_order', 'day_of_the_week', 'rating',
       'food_preparation_time', 'delivery_time'],
      dtype='object')
```

### Attribute: cost\_of\_the\_order

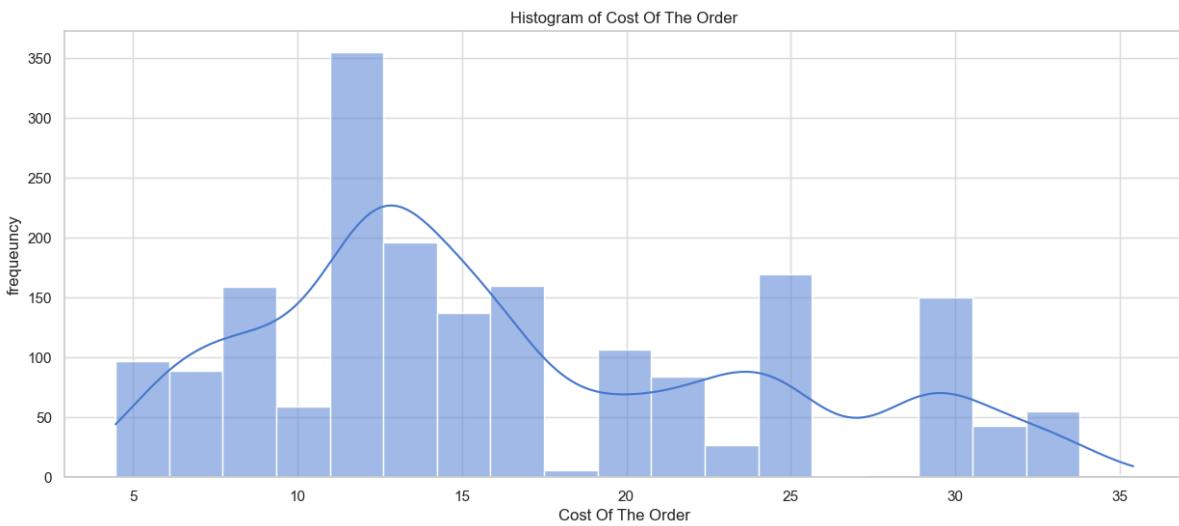
```
In [47]: x = 'cost_of_the_order'
```

```
In [48]: df[x].describe()
```

```
Out[48]: count    1898.000000
          mean     16.498851
          std      7.483812
          min      4.470000
          25%     12.080000
          50%     14.140000
          75%     22.297500
          max     35.410000
          Name: cost_of_the_order, dtype: float64
```

In [49]: # Plot

```
plot_helper.draw_histplot(x)
```



### 📌 Points :

- The histogram for 'Cost of the Order' shows the distribution of order costs in the dataset.
- The distribution appears to be right-skewed, indicating that most orders are relatively inexpensive, with fewer high-cost orders.
- There are some outliers with significantly higher costs, which could be due to aspects.
- The presence of a KDE (Kernel Density Estimate) line helps visualize the probability density of the order costs.

In [50]: # Descriptive stats

```
q1, q3, iqr = calc_iqr(df[x])
q1, q3, iqr
```

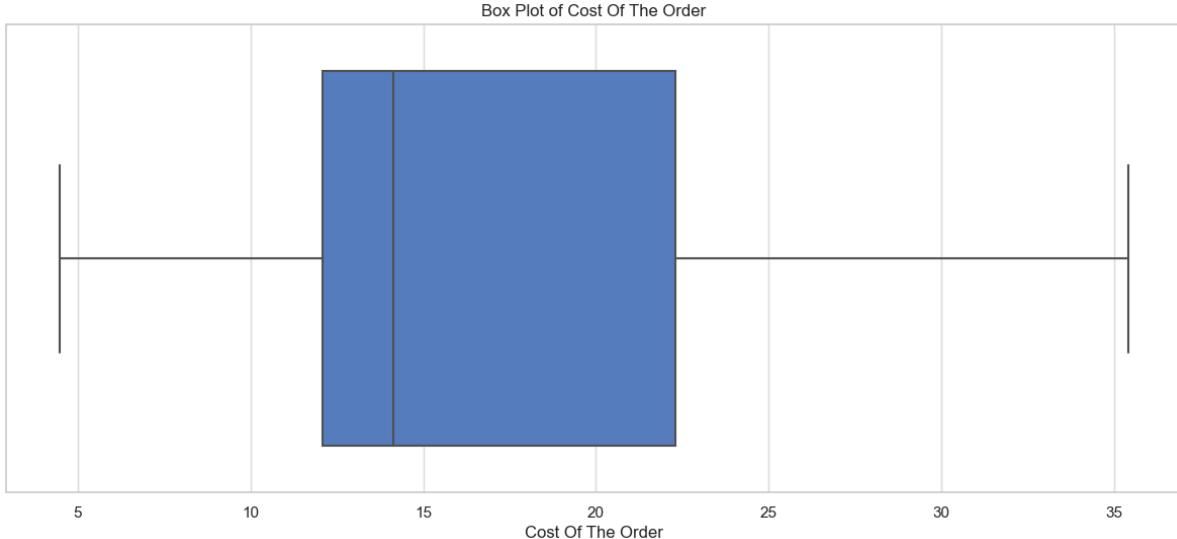
Out[50]: (12.08, 22.2975, 10.2175)

### 🔍 Insights:

The IQR for the cost of the order is 10.21 units, i.e Half of all the orders cost somewhere between 10 units more or less than typical usual price, showing a moderate variation in how much customers are spending.

In [51]: # Plot

```
plot_helper.draw_boxplot(x)
```



### Points:

- The boxplot for 'Cost of the Order' shows the summary of order costs in the dataset.
- The median cost of the orders is around 15 dollar.
- The interquartile range (IQR) is approximately 10 units, indicating that the middle 50% of the orders cost between 12 and 23 dollars.
- There are several outliers with higher costs, which could be due to large or multiple orders. The presence of outliers suggests that while most orders are within a certain range, there are some orders that are significantly more expensive.

In [52]: # Stats | More

```
# deeper insights
print(f"skewness: {df['cost_of_the_order'].skew()}")
print(f"kurtosis: {df['cost_of_the_order'].kurt()}")
```

skewness: 0.6217469995871637  
kurtosis: -0.6078508734318953

### Insights:

The cost of orders has a slight right skew (skewness of 0.6), meaning there are a few higher-cost orders compared to the majority. The negative kurtosis value (-0.6) indicates that the distribution has lighter tails and is flatter than a normal distribution, suggesting fewer extreme values (outliers) in the cost data.

### Observation:

The boxplot for 'Cost of the Order' reveals that the median order costs about 15 units, with most orders falling between 12 to 23 dollars. This means that half of the orders are within this price range. However, there are a few orders that are much more expensive, likely due to larger or multiple items being ordered. This shows that while most people are placing affordable orders, some orders can be significantly higher in cost.

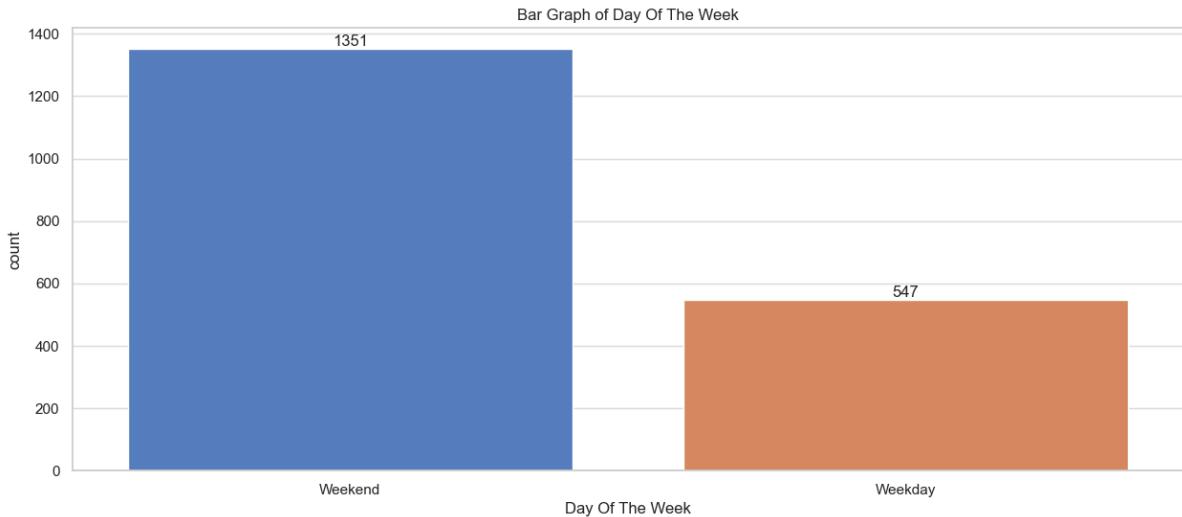
In [53]: # helper cell  
df.columns

```
Out[53]: Index(['order_id', 'customer_id', 'restaurant_name', 'cuisine_type',
   'cost_of_the_order', 'day_of_the_week', 'rating',
   'food_preparation_time', 'delivery_time'],
  dtype='object')
```

### Attribute: day\_of\_the\_week

```
In [54]: # categorical
x = 'day_of_the_week'
```

```
In [55]: plot_helper.draw_countplot(x)
```



```
In [56]: # Descriptive Statistics
order_count_by_day_category = df['day_of_the_week'].value_counts()
order_count_by_day_category_pct = df['day_of_the_week'].value_counts(normalize=True)

print('Order Counts by Day of the Week:', order_count_by_day_category, sep='')
print('\nPercentage of Orders by Day of the Week:', order_count_by_day_category_pct)
```

Order Counts by Day of the Week:

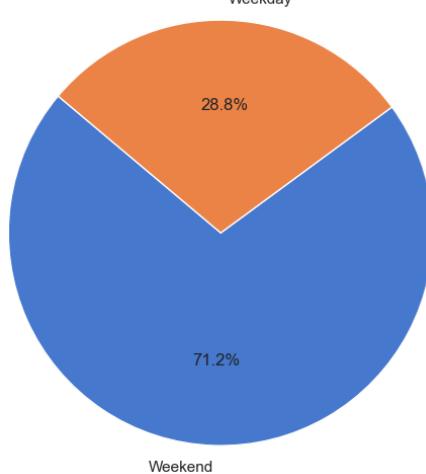
```
Weekend    1351
Weekday     547
Name: day_of_the_week, dtype: int64
```

Percentage of Orders by Day of the Week:

```
Weekend    71.18019
Weekday    28.81981
Name: day_of_the_week, dtype: float64
```

```
In [57]: plot_helper.draw_pieplot(x)
```

Pie Chart of Day Of The Week



### 📌 Points :

- The pie chart for 'Orders by Day of the Week' shows the distribution of orders between weekdays and weekends.
- The majority of orders are placed on weekends, accounting for approximately 71.2% of the total orders.
- Weekday orders make up the remaining 28.8% of the total orders.

### 🔍 Observation:

There is higher demand for food orders during weekends compared to weekdays. Restaurants may consider increasing their staff and resources during weekends to handle the higher volume of orders.

```
In [58]: # helper cell
df.columns
```

```
Out[58]: Index(['order_id', 'customer_id', 'restaurant_name', 'cuisine_type',
       'cost_of_the_order', 'day_of_the_week', 'rating',
       'food_preparation_time', 'delivery_time'],
      dtype='object')
```

### Attribute: rating

```
In [59]: # Categorical | Ordinal
x = 'rating'
df[x].dtype
```

```
Out[59]: dtype('float64')
```

```
In [60]: # Note; here null is same as "Not given" as we have already converted "Not given" to null
ratings_not_given = df[x].isnull().sum()
ratings_not_given
```

```
Out[60]: 736
```

```
In [61]: ratings_not_given_pct = ratings_not_given / total_orders * 100
print(f"Percentage of orders without rating: {format_pct(ratings_not_given_pct)}")
```

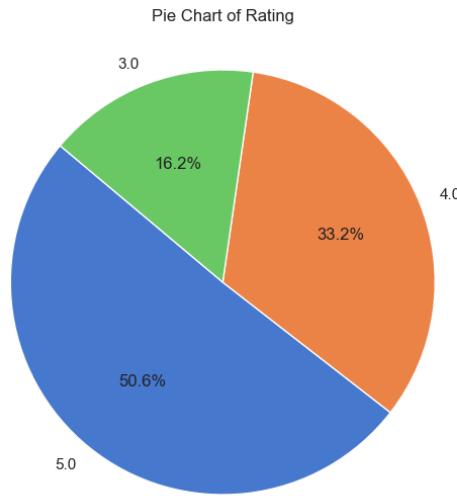
Percentage of orders without rating: 38.78%

### 🔍 Insights:

A significant portion of the ratings are 'Not given', indicating that many customers did not provide a rating.

NOTE: Plots addresses orders with rating

```
In [62]: # Plot  
plot_helper.draw_pieplot(x)
```



📌 **Points:**

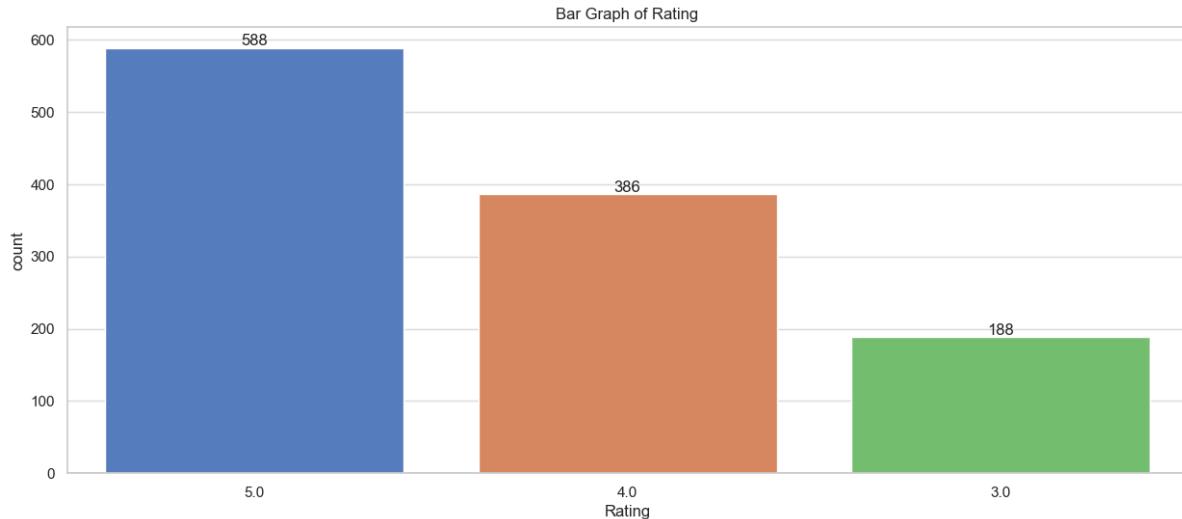
- The pie chart for 'Rating' shows the distribution of ratings given by customers.
- Among the given ratings, the most common rating is 5, followed by 4 and 3.

```
In [63]: # Stats
```

```
tableit(df[x].value_counts(), index_label='Rating', value_label='#Orders')
```

| Rating | #Orders |
|--------|---------|
| 5      | 588     |
| 4      | 386     |
| 3      | 188     |

```
In [64]: plot_helper.draw_countplot(x)
```



### 🔍 Observation:

This suggests that customers who do leave ratings generally tend to give high scores. The large number of 'Not given' ratings highlights the potential opportunity for the company to encourage more customers to share their feedback.

```
In [65]: # helper cells
df.columns
```

```
Out[65]: Index(['order_id', 'customer_id', 'restaurant_name', 'cuisine_type',
       'cost_of_the_order', 'day_of_the_week', 'rating',
       'food_preparation_time', 'delivery_time'],
      dtype='object')
```

### Attribute: food\_preparation\_time

```
In [66]: x = 'food_preparation_time'
```

```
In [67]: # Stats
df[x].describe()
```

```
Out[67]: count    1898.000000
mean      27.371970
std       4.632481
min      20.000000
25%     23.000000
50%     27.000000
75%     31.000000
max     35.000000
Name: food_preparation_time, dtype: float64
```

```
In [68]: df[x].nunique()
```

```
Out[68]: 16
```

### 🔍 Insights:

There are only 16 unique food preparation time (minutes), indicating it to be as discrete in nature

```
In [69]: # explore | stats
```

```
food_preparation_time_counts = df[x].value_counts()
```

```
food_preparation_time_counts_pct = df[x].value_counts(normalize=True) * 100

food_preparation_time_counts

Out[69]:
```

|    |     |
|----|-----|
| 21 | 135 |
| 23 | 123 |
| 27 | 123 |
| 22 | 123 |
| 28 | 121 |
| 24 | 121 |
| 20 | 119 |
| 30 | 119 |
| 33 | 118 |
| 35 | 117 |
| 31 | 116 |
| 26 | 115 |
| 25 | 113 |
| 34 | 113 |
| 32 | 113 |
| 29 | 109 |

Name: food\_preparation\_time, dtype: int64

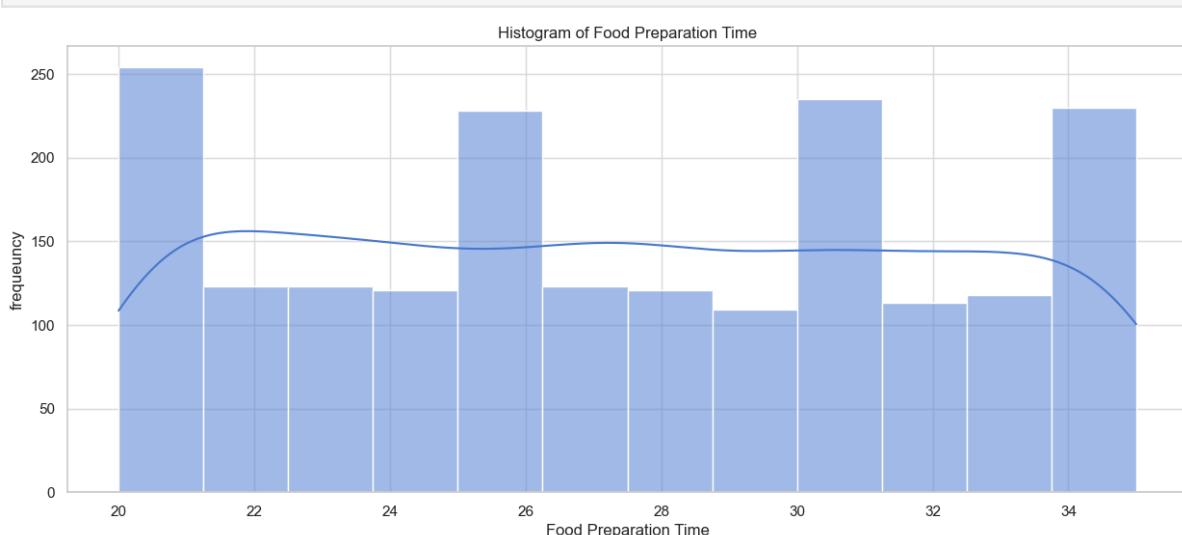
```
In [70]: food_preparation_time_counts_pct
```

```
Out[70]:
```

|    |          |
|----|----------|
| 21 | 7.112750 |
| 23 | 6.480506 |
| 27 | 6.480506 |
| 22 | 6.480506 |
| 28 | 6.375132 |
| 24 | 6.375132 |
| 20 | 6.269758 |
| 30 | 6.269758 |
| 33 | 6.217071 |
| 35 | 6.164384 |
| 31 | 6.111697 |
| 26 | 6.059009 |
| 25 | 5.953635 |
| 34 | 5.953635 |
| 32 | 5.953635 |
| 29 | 5.742887 |

Name: food\_preparation\_time, dtype: float64

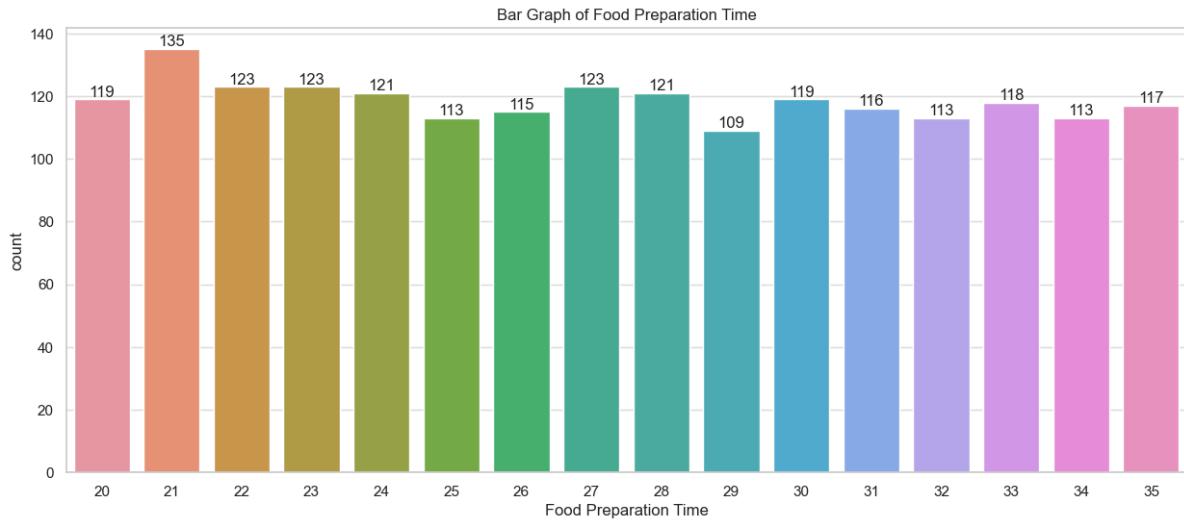
```
In [71]: plot_helper.draw_histplot(x)
```



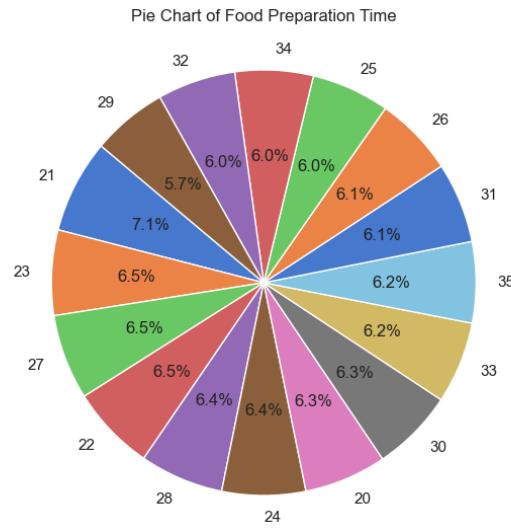
🔍 Insights:

The histogram of food preparation time appears to be plateaued, with a consistent distribution across the range, indicating little variation in preparation times

In [72]: `plot_helper.draw_countplot(x, sort=False)`



In [73]: `plot_helper.draw_pieplot(x)`

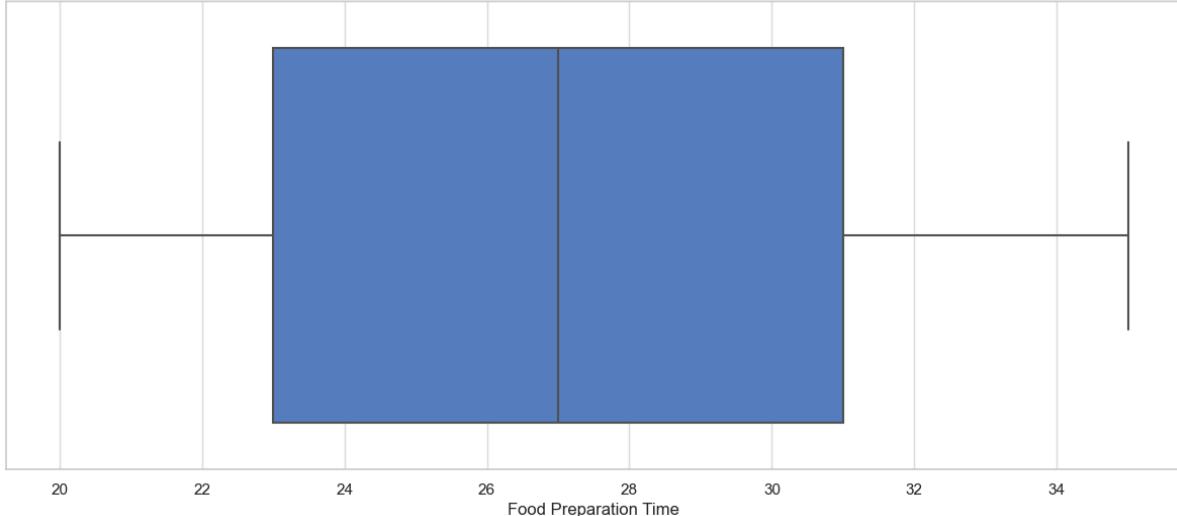


🔍 Insights:

The pie chart shows that the most common food preparation time is 21 minutes, making up 7.1% of all orders.

In [74]: `plot_helper.draw_boxplot(x)`

Box Plot of Food Preparation Time



### 🔍 Observation:

The 'Food Preparation Time' column contains only 16 unique values, meaning that the time it takes for restaurants to prepare orders is fairly consistent. This could suggest that most restaurants have similar preparation times for their orders, or they may follow a standard process for preparing meals. It also indicates that there aren't many variations in how long it takes to prepare food across different orders.

The distribution of 'Food Preparation Time' shows that most orders are prepared within a time frame of 23 to 31 minutes.

```
In [75]: df.columns
```

```
Out[75]: Index(['order_id', 'customer_id', 'restaurant_name', 'cuisine_type',
       'cost_of_the_order', 'day_of_the_week', 'rating',
       'food_preparation_time', 'delivery_time'],
      dtype='object')
```

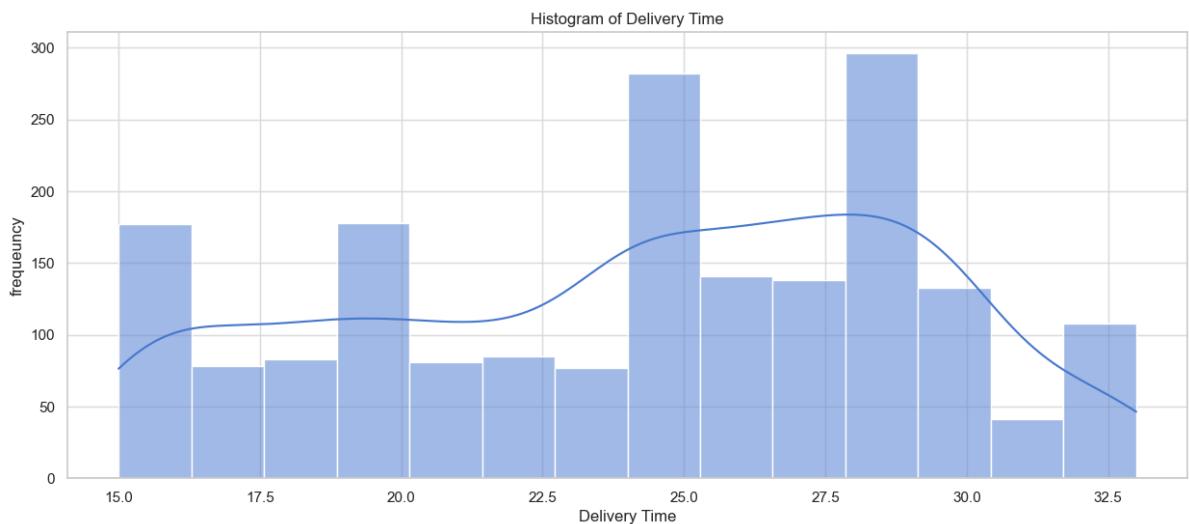
### Attribute: delivery\_time

```
In [76]: # numeric
x = 'delivery_time'
```

```
In [77]: df[x].describe()
```

```
Out[77]: count    1898.000000
mean     24.161749
std      4.972637
min     15.000000
25%    20.000000
50%    25.000000
75%    28.000000
max     33.000000
Name: delivery_time, dtype: float64
```

```
In [78]: plot_helper.draw_histplot(x)
```



🔍 Insights:

The histogram of delivery time is slightly left-skewed, with higher frequencies observed for longer delivery times on the right side

In [79]: # Stats

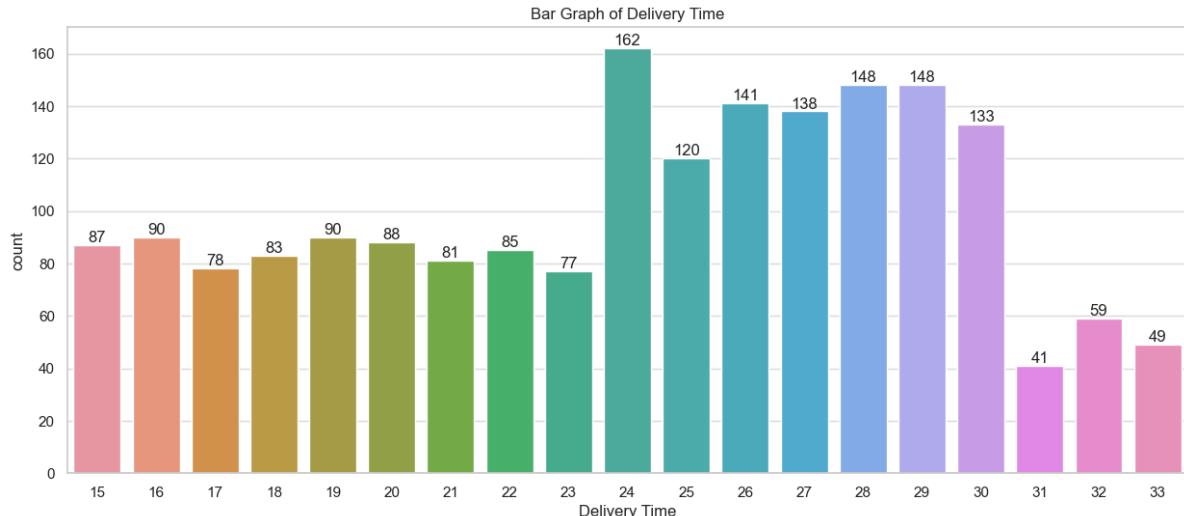
```
# frequency of each delivery time
delivery_time_counts = df[x].value_counts()
delivery_time_counts
```

Out[79]:

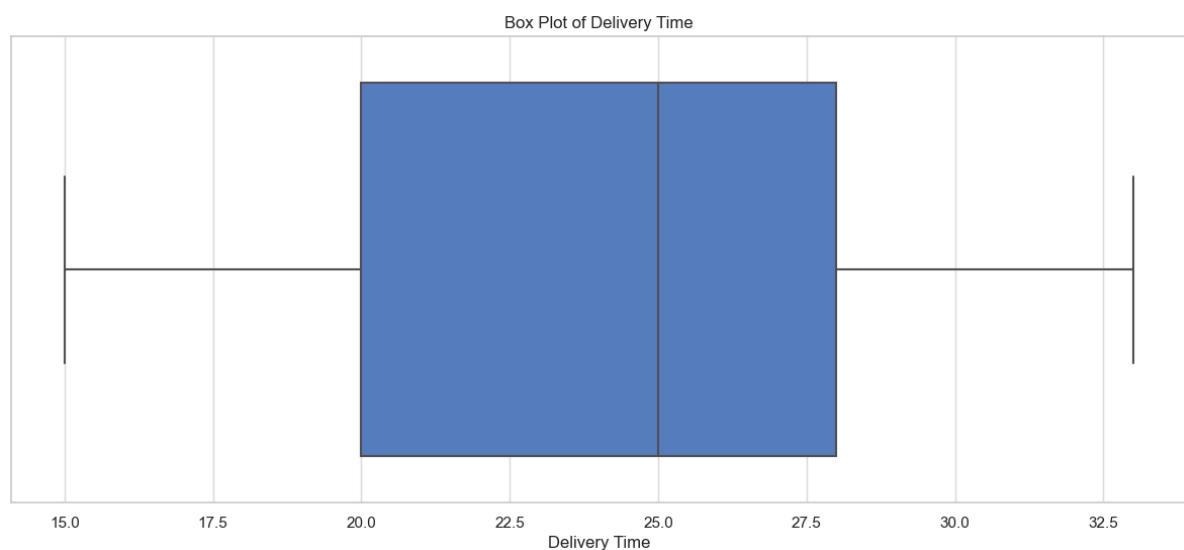
|    |     |
|----|-----|
| 24 | 162 |
| 28 | 148 |
| 29 | 148 |
| 26 | 141 |
| 27 | 138 |
| 30 | 133 |
| 25 | 120 |
| 19 | 90  |
| 16 | 90  |
| 20 | 88  |
| 15 | 87  |
| 22 | 85  |
| 18 | 83  |
| 21 | 81  |
| 17 | 78  |
| 23 | 77  |
| 32 | 59  |
| 33 | 49  |
| 31 | 41  |

Name: delivery\_time, dtype: int64

In [80]: plot\_helper.draw\_countplot(x, sort=False)



```
In [81]: plot_helper.draw_boxplot(x)
```



```
In [82]: df[x].describe()
```

```
Out[82]:
```

|                                     | count       | mean      | std      | min       | 25%       | 50%       | 75%       | max       |
|-------------------------------------|-------------|-----------|----------|-----------|-----------|-----------|-----------|-----------|
| Name: delivery_time, dtype: float64 | 1898.000000 | 24.161749 | 4.972637 | 15.000000 | 20.000000 | 25.000000 | 28.000000 | 33.000000 |

### 📌 Points:

- The average delivery time is approximately 24.16 minutes.
- The standard deviation is around 4.97 minutes, indicating some variability in delivery times.
- The minimum delivery time is 15 minutes.
- The 25th percentile (Q1) is 20 minutes, meaning 25% of the orders have a delivery time of 20 minutes or less.
- The median (50th percentile) delivery time is 25 minutes.
- The 75th percentile (Q3) is 28 minutes, meaning 25% of the orders have a delivery time of 28 minutes or more.

- The maximum delivery time is 33 minutes.

 **Observation:**

1. 25% of the deliveries are very fast, taking only 20 minutes or less
2. 50% of the deliveries take 25 minutes or less

On an average, deliveries take about 24 minutes, with most orders falling between 20 and 28 minutes. While the fastest deliveries are completed in 15 minutes, some take up to 33 minutes. Moreover, Half of all orders are delivered within 25 minutes.

The slight variability in delivery time (around 5 minutes) suggests a consistent but slightly flexible pace across delivery services

---

### Question 7: Which are the top 5 restaurants in terms of the number of orders received? [1 mark]

```
In [83]: # Q7.
k = 5
# NOTE: The resulting object will be in descending order so that the first element is the most frequent.
top_5_restaurant = df['restaurant_name'].value_counts().head(5)
top_5_restaurant
```

```
Out[83]: Shake Shack      219
          The Meatball Shop 132
          Blue Ribbon Sushi  119
          Blue Ribbon Fried Chicken 96
          Parm                68
          Name: restaurant_name, dtype: int64
```

```
In [84]: tableit(top_5_restaurant, index_label='Restaurant', value_label='Order Count')
```

| Restaurant                | Order Count |
|---------------------------|-------------|
| Shake Shack               | 219         |
| The Meatball Shop         | 132         |
| Blue Ribbon Sushi         | 119         |
| Blue Ribbon Fried Chicken | 96          |
| Parm                      | 68          |

 **Observation:**

The top 5 restaurants in terms of the number of orders received are:

1. Shake Shack with **219** orders,
2. The Meatball Shop with **132** orders,
3. Blue Ribbon Sushi with **119** orders,
4. Blue Ribbon Fried Chicken **96** orders, and
5. Parm with **68** orders

## Question 8: Which is the most popular cuisine on weekends? [1 mark]

```
In [85]: weekend_orders = df[df['day_of_the_week'] == 'Weekend']
cuisine_order_count = weekend_orders['cuisine_type'].value_counts()
cuisine_order_count
```

```
Out[85]: American      415
Japanese     335
Italian       207
Chinese       163
Mexican        53
Indian         49
Mediterranean   32
Middle Eastern   32
Thai           15
French          13
Korean          11
Southern         11
Spanish          11
Vietnamese       4
Name: cuisine_type, dtype: int64
```

```
In [86]: # most popular cuisine and its percentage
most_popular_cuisine = cuisine_order_count.idxmax()
percentage = (cuisine_order_count.max() / cuisine_order_count.sum()) * 100
print(f"The most popular cuisine on weekends is {most_popular_cuisine} with
```

The most popular cuisine on weekends is American with 30.72% of weekend orders.

### Observation:

The most popular cuisine on weekends is *American* with a total of 415 orders, constituting around 31% of weekend orders. This indicates a strong preference for American cuisine among customers during weekends.

Restaurants offering American cuisine may experience higher demand on weekends and should be prepared to handle the increased volume of orders.

## Question 9: What percentage of the orders cost more than 20 dollars? [2 marks]

```
In [87]: # mask for orders with cost greater than 20
mask_cost_above_20 = df['cost_of_the_order'] > 20
percent_of_order_above_20 = mask_cost_above_20.mean() * 100
print(f"Percentage of orders with a cost above 20 dollar is {format_pct(percent_of_order_above_20)}")
```

Percentage of orders with a cost above 20 dollar is 29.24%

```
In [88]: df['cost_of_the_order'].describe()
```

```
Out[88]: count    1898.000000
          mean     16.498851
          std      7.483812
          min      4.470000
          25%     12.080000
          50%     14.140000
          75%     22.297500
          max     35.410000
Name: cost_of_the_order, dtype: float64
```

**Observation:**

- This indicates that nearly one-third of the orders are relatively high-cost, suggesting that a significant portion of customers are willing to spend more on their food orders. This could suggest a preference for higher-end items or larger orders.

### Question 10: What is the mean order delivery time? [1 mark]

```
In [89]: avg_delivery_time = df['delivery_time'].mean()
print(f"The average delivery time is {avg_delivery_time:.2f} minutes.")
```

The average delivery time is 24.16 minutes.

**Observations:**

the mean order delivery time is 24.16 minutes

### Question 11: The company has decided to give 20% discount vouchers to the top 3 most frequent customers. Find the IDs of these customers and the number of orders they placed. [1 mark]

```
In [90]: # Count the number of orders for each customer
order_counts = df['customer_id'].value_counts()

# Get the top 3 customers
top_3_customers = order_counts.head(3)

top_3_customers
```

```
Out[90]: 52832    13
        47440    10
        83287     9
Name: customer_id, dtype: int64
```

```
In [91]: tableit(top_3_customers, index_label='Customer ID', value_label='Order Count')
```

| Customer ID | Order Count |
|-------------|-------------|
| 52832       | 13          |
| 47440       | 10          |
| 83287       | 9           |

**Observation:**

The top 3 most frequent customers based on the number of orders placed are:

1. Customer ID 52832 with 13 order,
2. Customer ID 47440 with 10 order,
3. Customer ID 83287 with 9 order. These customers will be eligible for the 20% discount vouchers.

The top 3 customers with the most orders, suggesting a small group of loyal customers place multiple orders. This indicates potential for targeted loyalty programs or special offers to further encourage repeat business from these high-value customers.

 *Explore (New Insight):*

#### Customer Repeat

```
In [92]: # Explore more (Customer Repeat)
order_counts.describe()
```

```
Out[92]: count    1200.000000
mean      1.581667
std       1.090560
min      1.000000
25%      1.000000
50%      1.000000
75%      2.000000
max      13.000000
Name: customer_id, dtype: float64
```

```
In [93]: order_counts[order_counts > 2].count()
```

```
Out[93]: 149
```

```
In [94]: 149 / 1200 * 100
```

```
Out[94]: 12.416666666666666
```

 *Insights:*

- Only around 12.5% of customers placed order more than twice

**There are very few repeating customers** as most of the customers placed order for once only. However, there is a small group of customers who place multiple orders, with the highest order count reaching 13.

Since most customers only place one order, there is an opportunity to boost customer retention through loyalty programs or promotions aimed at encouraging repeat orders. Targeting the small group of high-frequency customers with exclusive offers could further strengthen their loyalty and increase their order frequency.

## Multivariate Analysis

**Question 12:** Perform a multivariate analysis to explore relationships between the important variables in the dataset. (It is a good idea to explore relations between numerical variables

## as well as relations between numerical and categorical variables) [10 marks]

### 1. Cuisine Type and Day of the Week

(This can help identify popularity of cuisines among weekdays and weekends)

```
In [95]: x = 'cuisine_type'
y = 'day_of_the_week'
```

```
In [96]: df[x].nunique(), df[y].nunique()
```

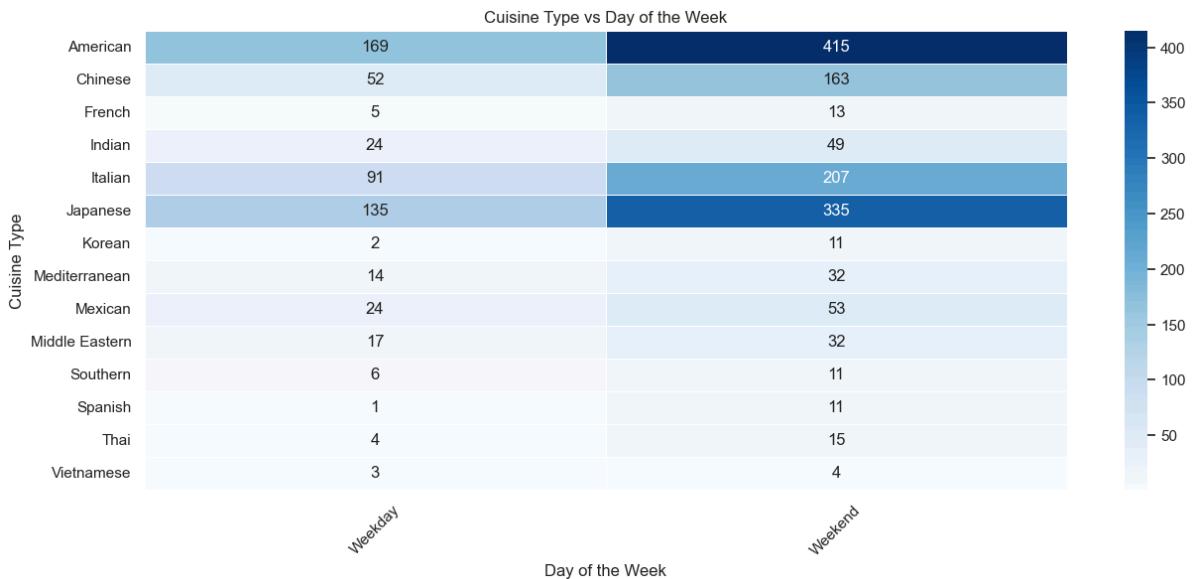
```
Out[96]: (14, 2)
```

```
In [97]: # Plot
heatmap_data = pd.crosstab(df['cuisine_type'], df['day_of_the_week'])

sns.heatmap(heatmap_data, annot=True, fmt="d", cmap="Blues", cbar=True, line
```

# aesthetics

```
plt.title("Cuisine Type vs Day of the Week")
plt.xlabel("Day of the Week")
plt.ylabel("Cuisine Type")
plt.xticks(rotation=45)
plt.show()
```



#### 📌 Points:

The heatmap shows the relationship between cuisine type and the day of the week (Weekday vs Weekend).

- **American cuisine** is the most popular on both weekdays and weekends, with a significantly higher number of orders on weekends.
- **Japanese cuisine** is the second most popular, with a higher number of orders on weekends compared to weekdays.
- **Italian cuisine** also shows a higher number of orders on weekends.
- **Chinese cuisine** has a notable number of orders on weekends, but fewer on weekdays.

- Other cuisines like **Mexican, Indian, and Mediterranean** also have more orders on weekends than weekdays, but the difference is less pronounced compared to American and Japanese cuisines.
- **Korean, Southern, Spanish, and Vietnamese cuisines** have relatively few orders overall, with a slight increase on weekends.

Overall, the heatmap indicates that weekends generally see a higher volume of orders across most cuisine types, with American and Japanese cuisines being the most popular.

In [98]: `# Plot 2 (Stacked Bar Chart)`

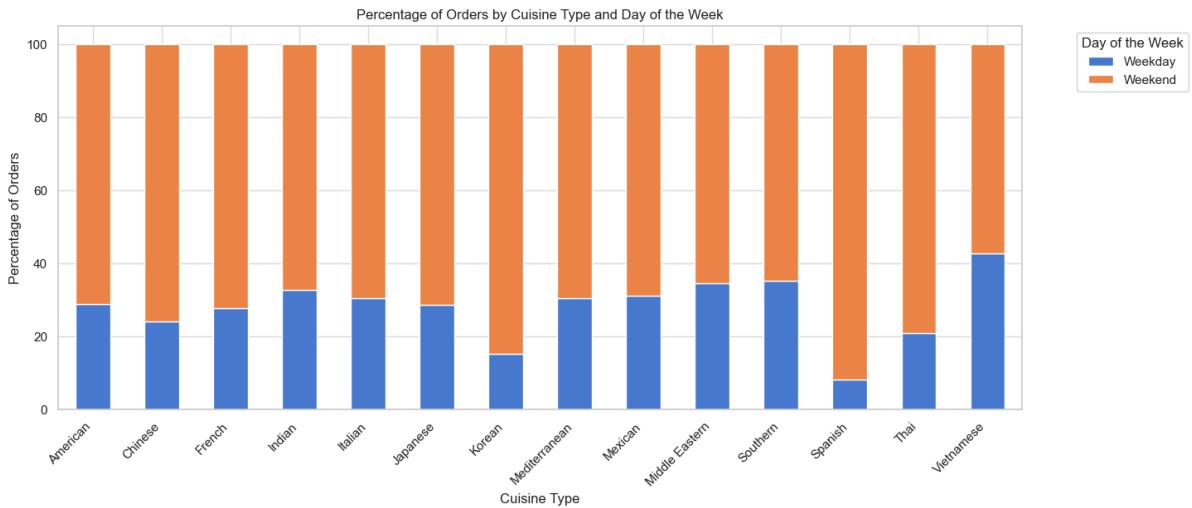
```
# Calculate the count of orders for each cuisine_type and day_of_the_week
order_counts = pd.crosstab(df['cuisine_type'], df['day_of_the_week'])

# Normalize to get percentages
order_percentages = order_counts.div(order_counts.sum(axis=1), axis=0) * 100

# Plot the stacked bar plot
order_percentages.plot(kind='bar', stacked=True)

# Customize the plot
plt.title('Percentage of Orders by Cuisine Type and Day of the Week')
plt.xlabel('Cuisine Type')
plt.ylabel('Percentage of Orders')
plt.xticks(rotation=45, ha='right')
plt.legend(title='Day of the Week', bbox_to_anchor=(1.05, 1), loc='upper left')

plt.show()
```



🔍 **Insights:**

- Only vietnamese cuisine has more than 40% share of orders on weekdays.
- Spanish cuisine are mostly ordered on weekend (with less than 10% contribution during weekdays)

Vietnamese cuisine is the only cuisine that shows a relatively balanced distribution, with 40% of orders placed on weekdays and 60% on weekends. Unlike other cuisines, which have a greater concentration of orders on weekends (below 40% on weekdays), Vietnamese cuisine maintains a more even split across the week.

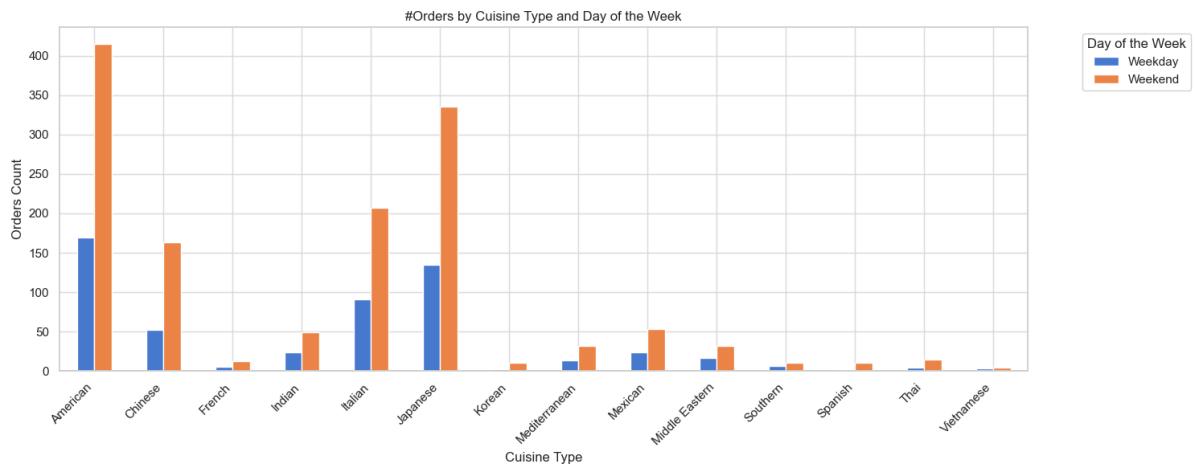
In [99]: # Plot 3

```
# Calculate the count of orders for each cuisine_type and day_of_the_week
order_counts = pd.crosstab(df['cuisine_type'], df['day_of_the_week'])

# Plot the grouped bar plot
order_counts.plot(kind='bar', stacked=False)

# Customize the plot
plt.title('#Orders by Cuisine Type and Day of the Week')
plt.xlabel('Cuisine Type')
plt.ylabel('Orders Count')
plt.xticks(rotation=45, ha='right')
plt.legend(title='Day of the Week', bbox_to_anchor=(1.05, 1), loc='upper left')

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



### 🔍 Insights:

Italian and Chinese cuisines rank just after Japanese and American cuisines in terms of orders, showing a strong customer preference for these options as well.

The business could leverage this insight by offering combo deals or promotions that highlight Italian and Chinese dishes, especially during peak ordering times, to further drive sales. This could also help in balancing the demand across different cuisines and maintaining customer interest.

### 🔍 Observation: (todo:)

1. American and Japanese cuisines are most famous amongst all.
2. Vietnamese & Thai cuisines are least sought by customers.
3. Chinese and Italian cuisines follow, showing a similar trend of increased orders on weekends.
4. Other cuisines like French, Indian, and Mediterranean have relatively fewer orders but still show a weekend preference.
5. Vietnamese cuisine is the only cuisine that shows a relatively balanced distribution, with 40% of orders placed on weekdays and 60% on weekends. Unlike other cuisines, which have a greater concentration of orders on weekends (below 40% on weekdays), Vietnamese cuisine maintains a more even split across the week.

6. Spanish cuisines are mostly ordered on weekend, having roughly less than 10% of orders placed on weekdays. This indicates that customers tend to prefer Spanish cuisine primarily during the weekends.

The data indicates a general trend of higher demand for various cuisines during weekends, suggesting that customers are more likely to order food during weekends.

Restaurants may consider increasing their inventory and staffing on weekends to accommodate the higher volume of orders.

## 2. Customer Ratings & Cost of the Orders

(This can provide insights into whether customers tend to rate more expensive orders differently than cheaper ones)

In [100]:

```
# Plot 1
sns.violinplot(data=df, x='rating', y='cost_of_the_order')
plt.title('Distribution of Order Costs by Customer Ratings')
plt.xlabel('Rating')
plt.ylabel('Cost of the Order')
plt.show()
```



In [101]:

```
avg_cost_by_rating = original_df.groupby('rating')['cost_of_the_order'].mean()
avg_cost_by_rating
```

Out[101]:

|   | rating    | cost_of_the_order |
|---|-----------|-------------------|
| 0 | 3         | 16.223351         |
| 1 | 4         | 16.710337         |
| 2 | 5         | 16.965697         |
| 3 | Not given | 16.085340         |

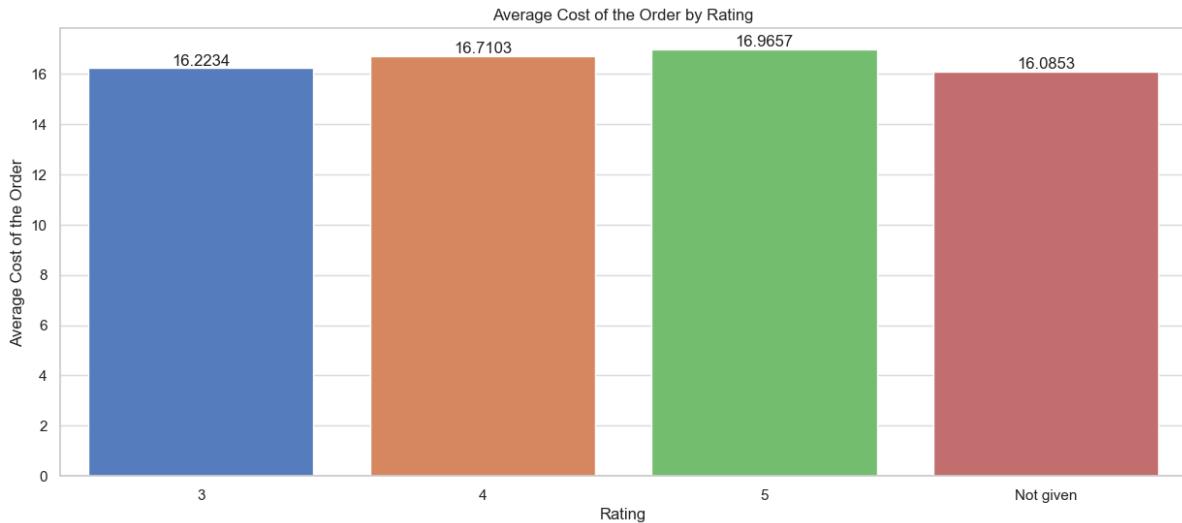
In [102]:

```
# Plot 2 (Count)
# create a bar plot to show the average cost for each rating
ax = sns.barplot(data=avg_cost_by_rating, x='rating', y='cost_of_the_order')

# add count above each bar
ax.bar_label(ax.containers[0])
```

```
# Add title and labels
plt.title('Average Cost of the Order by Rating')
plt.xlabel('Rating')
plt.ylabel('Average Cost of the Order')

# Show the plot
plt.show()
```



### **Observation:**

This suggests that cost of an order does not vary significantly based on the rating given by the customer. In other words, whether the customer gives a high or low rating, the average amount spent on orders remains relatively consistent. This could further imply that pricing of the orders is independent of customer satisfaction, or there may be a broad range of pricing within each rating group

The average cost appears to be almost identical across all ratings. Additionally, even when ratings were not provided, the average cost remains consistent. This indicates cost is independent of customer feedback.

### **Recommendations :**

- Focus on improving non-cost-related aspects like food presentation, freshness, delivery accuracy, and speed.
- Encourage customers to leave feedback on what they enjoyed or disliked about their experience to identify areas for improvement.
- Consider adding value to the customer experience through small gestures like complimentary items or personalized notes, which could help boost ratings without impacting average costs.

## 3. Day of Week & Delivery Time

(This can help identify peak days with longer delivery times and optimize operations for faster service)

```
In [103]: x = 'day_of_the_week'
y = 'delivery_time'
df[x].nunique(), df[y].nunique()
```

Out[103]: (2, 19)

In [104...]: # Plot 1

```
# box plot between day of the week and delivery time
sns.boxplot(data=df, x=x, y=y)

# add title and labels
plt.title('Delivery Time Comparision: Weekdays vs Weekends')
plt.xlabel('Day of the Week')
plt.ylabel('Delivery Time (mins)')

# show the plot
plt.show()
```



### 🔍 Insights:

This is quite interesting, as weekend observes more orders compare to weekday, still orders are delivered in lesser time during weekend. This suggests that there may be some other factors affecting the delivery time during weekday (maybe traffic of New-York)

```
In [105...]: # calculate median time for each day of the week
median_delivery_time = df.groupby('day_of_the_week')['delivery_time'].median()
median_delivery_time
```

Out[105]: day\_of\_the\_week delivery\_time

| 0 | Weekday | 28.0 |
|---|---------|------|
| 1 | Weekend | 22.0 |

In [106...]: # Plot 2

```
# Strip plot for day_of_the_week vs delivery_time
sns.stripplot(x='day_of_the_week', y='delivery_time', data=df, jitter=True)

# Adding labels and title
plt.title('Delivery Time Comparision: Weekdays vs Weekends')
plt.xlabel('Day of the Week')
plt.ylabel('Delivery Time')
```

```
# Show plot
plt.show()
```



### 🔍 Observation:

Delivery times on weekdays tend to be higher and more spread out compared to weekends, where they are generally faster and more consistent. This could suggest that weekend orders may be handled more efficiently or have less traffic, while weekdays could experience more delays or variability, possibly due to increased order volume or delivery congestion.

## 4. Delivery Time & Customer Ratings

(This will help us understand if higher or lower ratings are associated with faster or slower delivery times.)

```
In [107...]: x = 'rating'
y = 'delivery_time'
df[x].nunique(), df[y].nunique()
```

```
Out[107]: (3, 19)
```

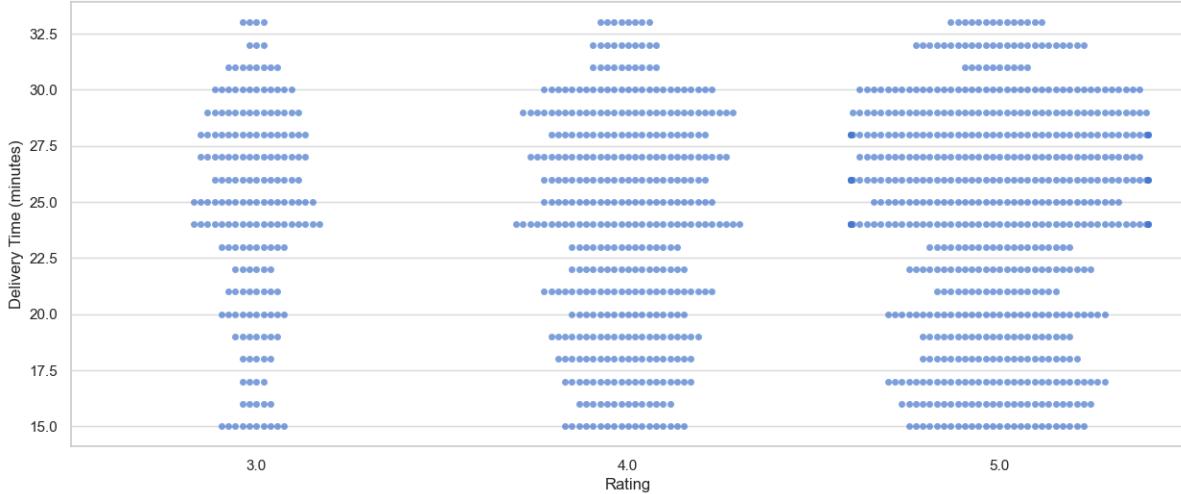
```
In [108...]: # Plot 1

# Drop rows with NaN values in the 'rating' column (for this pair analysis)
df_clean = df.dropna(subset=['rating'])

sns.swarmplot(data=df_clean, x='rating', y='delivery_time', alpha=0.7)

plt.title('Customer Rating vs Delivery Time')
plt.xlabel('Rating')
plt.ylabel('Delivery Time (minutes)')
plt.show()
```

Customer Rating vs Delivery Time



### 🔍 Insights:

Here we get good idea about density of ratings around particular Delivery Time (minutes) value. Because we use swarm plot we will get better idea about width (ie spread) of ratings distribution acrosss each delivery time stamp

### 💡 Quick Observation :

Most of orders are delivered in 20 to 30 mins, regardless of ratings

In [109...]

```
# plot 2
sns.violinplot(data=df, x='rating', y='delivery_time')
plt.title('Customer Rating vs Delivery Time')
plt.xlabel('Rating')
plt.ylabel('Delivery Time (minutes)')
plt.show()
```



### 🔍 Insights:

The distribution of delivery times for rating 5 resembles to a large extent with a violin like shape, comparatively amongst 3, indicating a balanced spread of values.

In [110...]

```
sns.boxplot(data=df, x='rating', y='delivery_time')
plt.title('Customer Rating vs Delivery Time')
plt.xlabel('Rating')
```

```
plt.ylabel('Delivery Time (minutes)')
plt.show()
```



### 📍 Points:

The Interquartile Range (IQR) for all rating categories is nearly the same when considering delivery time

- Consistent Spread**: regardless of the rating given by the customer, the delivery times have a similar range and dispersion
- No Strong Influence of Rating on Delivery Time Variability**: The ratings given by customers don't seem to significantly influence the variability of delivery times
- Uniform Experience Across Ratings**: Customers across all ratings are experiencing similar ranges of delivery times, which could imply that the company's delivery system has a consistent performance regardless of customer satisfaction

```
In [111]: # stats
avg_delivery_time_by_rating = df.groupby('rating')['delivery_time'].mean()
avg_delivery_time_by_rating
```

```
Out[111]:   rating  delivery_time
0      3.0    24.558511
1      4.0    23.867876
2      5.0    24.212585
```

### 🔍 Observation:

- The average delivery time by rating shows that the delivery times are relatively consistent across different ratings.

This finding might indicate that delivery time is not a major driver of the variance in customer ratings, or that other factors (such as food quality, customer service, etc.) might play a more substantial role in influencing ratings & customer satisfaction.

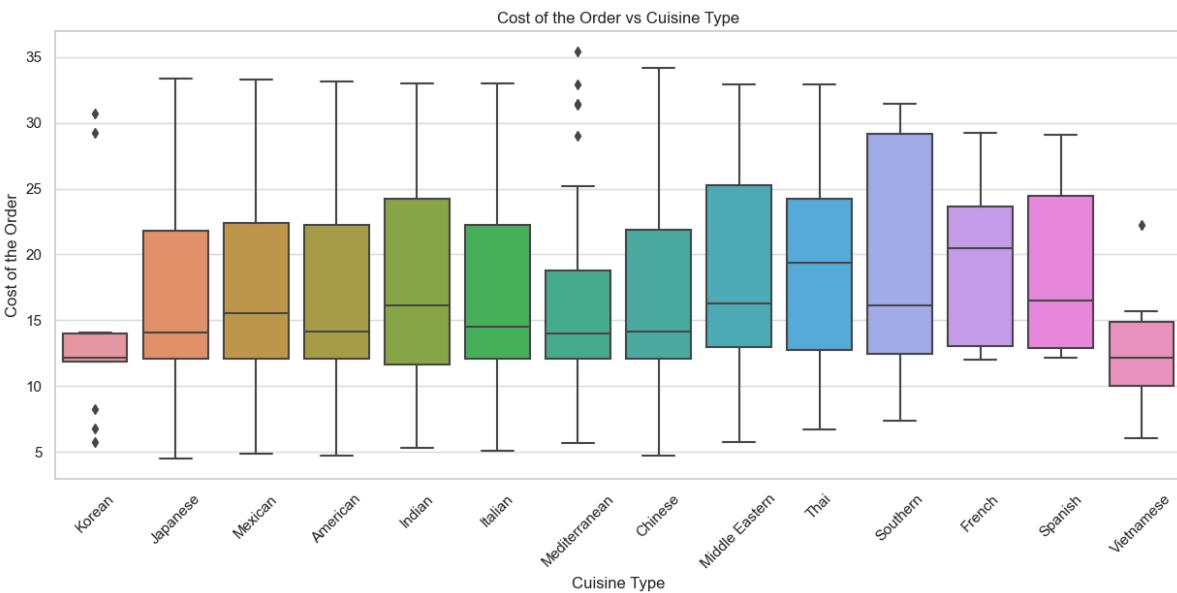
Whether the delivery time is slightly longer or shorter, customers tend to give similar ratings.

## 5. Cost of Order & Cuisine Types

(This can help derive strategic insights for pricing, promotions, and even menu optimization based on customer preferences and spending behavior.)

In [112...]

```
sns.boxplot(data=df, x='cuisine_type', y='cost_of_the_order')
plt.title('Cost of the Order vs Cuisine Type')
plt.xlabel('Cuisine Type')
plt.ylabel('Cost of the Order')
plt.xticks(rotation=45)
plt.show()
```



### 📌 Points:

- Either korean cuisines are too cheaper or too costly (demonstrating unconventional phenomenon)
- Mediterranean cuisines also being charged a lot than normalcy sometimes, which could be due to the specialized ingredients or materials required for these dishes that may not be readily available, resulting in higher costs for certain items.
- Southern dishes has good sort of varieties for lower-spender as well as high-spender.
- Typically Korean & Vietnamese dishes are cheaper compare to others
- Middle eastern and southern dishes are available in high price ranges.

In [113...]

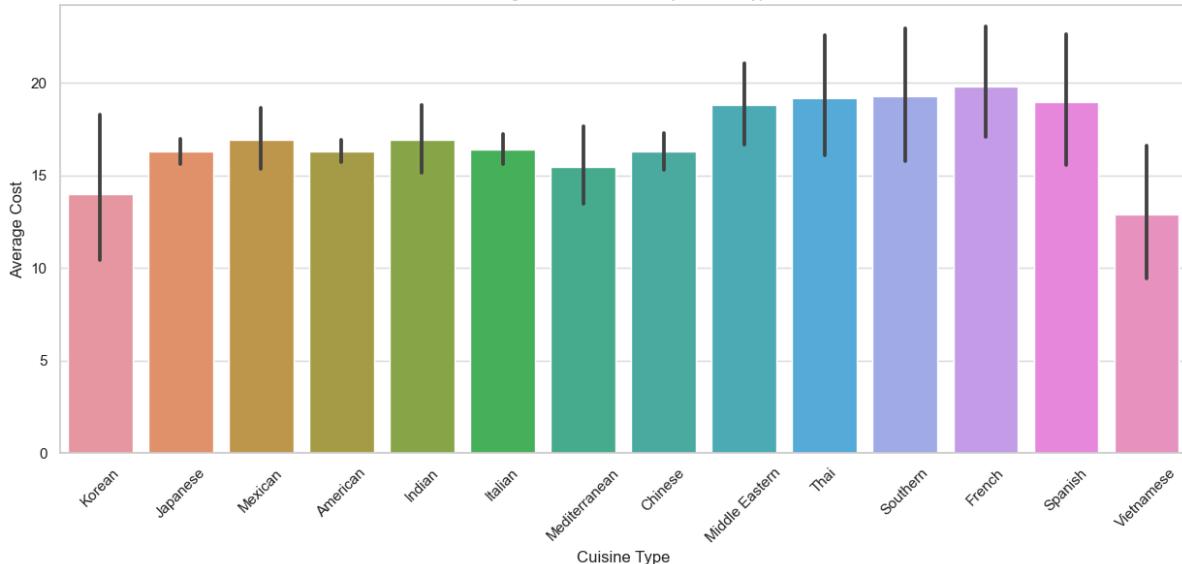
```
# create bar plot
sns.barplot(data=df, x='cuisine_type', y='cost_of_the_order')

# title and labels
plt.title('Average Cost of the Order by Cuisine Type')
plt.xlabel('Cuisine Type')
plt.ylabel('Average Cost')

# Rotate x-axis labels
plt.xticks(rotation=45)

# show the plot
plt.show()
```

Average Cost of the Order by Cuisine Type



In [114...]

## # Stats

```
avg_cost_by_cuisine = df.groupby('cuisine_type')['cost_of_the_order'].mean()
avg_cost_by_cuisine
```

Out [114]:

|    | cuisine_type   | cost_of_the_order |
|----|----------------|-------------------|
| 0  | American       | 16.319829         |
| 1  | Chinese        | 16.305209         |
| 2  | French         | 19.793889         |
| 3  | Indian         | 16.919726         |
| 4  | Italian        | 16.418691         |
| 5  | Japanese       | 16.304532         |
| 6  | Korean         | 14.001538         |
| 7  | Mediterranean  | 15.474783         |
| 8  | Mexican        | 16.933117         |
| 9  | Middle Eastern | 18.820612         |
| 10 | Southern       | 19.300588         |
| 11 | Spanish        | 18.994167         |
| 12 | Thai           | 19.207895         |
| 13 | Vietnamese     | 12.882857         |

## 🔍 Observation:

1. The average cost of orders varies across different cuisine types.
2. French cuisine has the highest average cost at approximately 19.79 dollars, followed by Spanish and Southern cuisines.
3. Vietnamese cuisine has the lowest average cost at approximately 12.88 dollars.
4. Southern cuisines might be having variety of dishes serving each segment of consumer needs.

Most of the cuisines have an average cost between 14 and 19 dollars.

The variation in average costs suggests that certain cuisines may be priced higher due to factors such as ingredient costs, preparation complexity, or restaurant pricing strategies.

## 6. Restaurant Name & Cusine Type

(This may help to discover what are specialities of certain restaurants)

```
In [115]: x = 'restaurant_name'
y = 'cuisine_type'

df[x].nunique(), df[y].nunique()

Out[115]: (178, 14)
```

**NOTE:** As there are too many restaurants, we will pick top performing restaurants (ie which receives more orders comparatively)

```
In [116]: # PREP (Data Preparation)

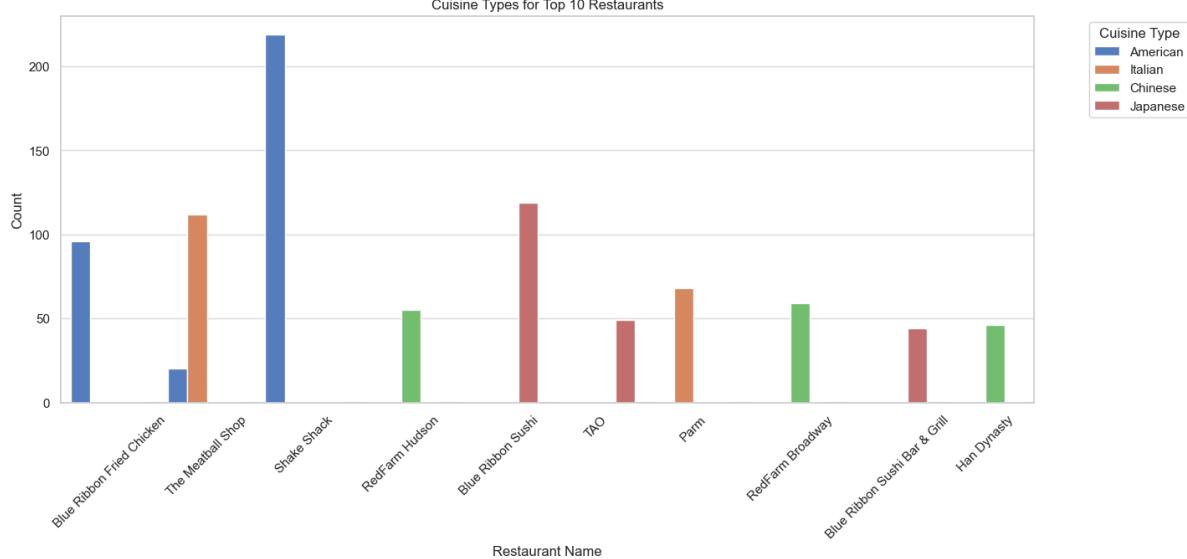
# Filter for top 10 restaurants by order count
famous_restaurants = df['restaurant_name'].value_counts().head(10).index
filtered_df = df[df['restaurant_name'].isin(famous_restaurants)]

# hence we will deal with filtered_df
filtered_df.head()
```

|    | order_id | customer_id | restaurant_name           | cuisine_type | cost_of_the_order | day_of_the_v |
|----|----------|-------------|---------------------------|--------------|-------------------|--------------|
| 3  | 1477334  | 106968      | Blue Ribbon Fried Chicken | American     | 29.20             | Wee          |
| 6  | 1477894  | 157711      | The Meatball Shop         | Italian      | 6.07              | Wee          |
| 12 | 1476966  | 129969      | Blue Ribbon Fried Chicken | American     | 24.30             | Wee          |
| 15 | 1477414  | 66222       | Shake Shack               | American     | 16.20             | Wee          |
| 18 | 1477138  | 47280       | RedFarm Hudson            | Chinese      | 14.12             | Wee          |

```
In [117]: # Plot 1

sns.countplot(data=filtered_df, x='restaurant_name', hue='cuisine_type')
plt.title('Cuisine Types for Top 10 Restaurants')
plt.xlabel('Restaurant Name')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.legend(title='Cuisine Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



### 📍 Points:

It seems that Most of the restaurant possess their own area of speciality or expertise for specific cuisine.

- **Shake Shack** is famous for American cuisines
- **Blue Ribbon Sushi** is famous for Japanese cuisines
- **The Meatball Shop**, though sells American Cuisine, still Customer prefer it for Italian cuisines & it seems to be leader in italic cuisine segment.

Chinese cuisines also seems to hold good market. & There is healthy competition for chinese cuisine between **RedFarm Hudson**, **RedFarm Broadway**, & **Han Dynasty** restaurants.

In [118]:

# Stats

```
# get order count for restaurant by cuisine type
restaurant_cuisine_count = filtered_df.groupby(['restaurant_name', 'cuisine_type']).size().reset_index()
restaurant_cuisine_count
```

Out[118]:

|  | cuisine_type | American | Chinese | Italian | Japanese |
|--|--------------|----------|---------|---------|----------|
| restaurant_name                          |              |          |         |         |          |
| <b>Blue Ribbon Fried Chicken</b>         |              | 96.0     | 0.0     | 0.0     | 0.0      |
| <b>Blue Ribbon Sushi</b>                 |              | 0.0      | 0.0     | 0.0     | 119.0    |
| <b>Blue Ribbon Sushi Bar &amp; Grill</b> |              | 0.0      | 0.0     | 0.0     | 44.0     |
| <b>Han Dynasty</b>                       |              | 0.0      | 46.0    | 0.0     | 0.0      |
| <b>Parm</b>                              |              | 0.0      | 0.0     | 68.0    | 0.0      |
| <b>RedFarm Broadway</b>                  |              | 0.0      | 59.0    | 0.0     | 0.0      |
| <b>RedFarm Hudson</b>                    |              | 0.0      | 55.0    | 0.0     | 0.0      |
| <b>Shake Shack</b>                       |              | 219.0    | 0.0     | 0.0     | 0.0      |
| <b>TAO</b>                               |              | 0.0      | 0.0     | 0.0     | 49.0     |
| <b>The Meatball Shop</b>                 |              | 20.0     | 0.0     | 112.0   | 0.0      |

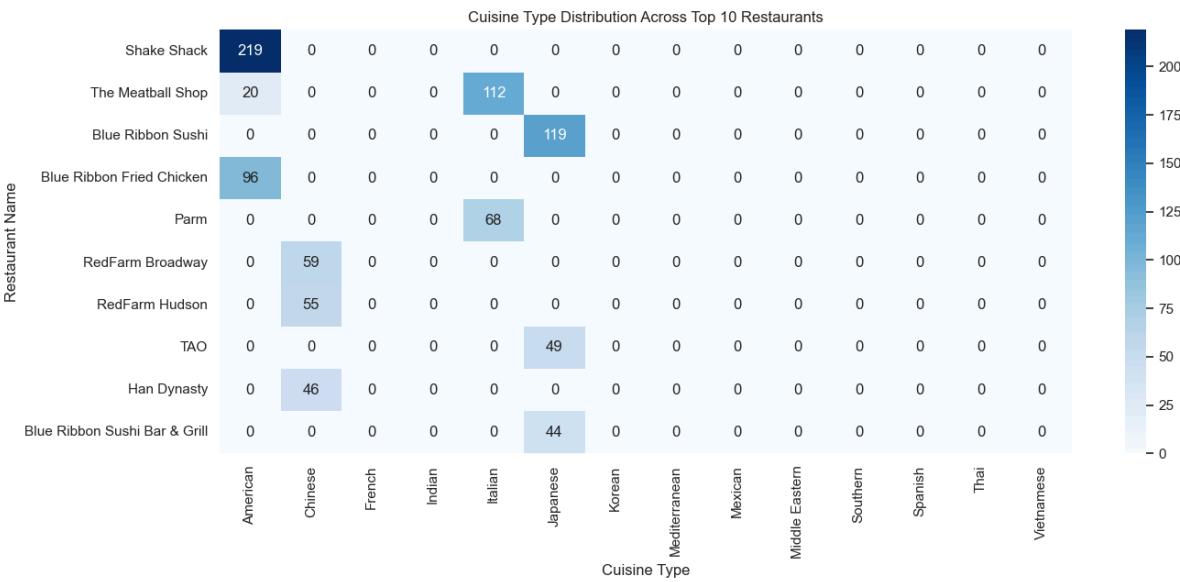
In [119...]

```
# Plot 2 (Pivot Table)

# Create a pivot table
pivot_table = df.pivot_table(index='restaurant_name', columns='cuisine_type'

# Filter for top 10 restaurants
k = 10
top_restaurant_pivot = pivot_table.loc[pivot_table.sum(axis=1).nlargest(k).index]

sns.heatmap(top_restaurant_pivot, annot=True, fmt='d', cmap='Blues', cbar=True)
plt.title(f'Cuisine Type Distribution Across Top {k} Restaurants')
plt.xlabel('Cuisine Type')
plt.ylabel('Restaurant Name')
plt.show()
```



### 🔍 Observation:

1. Shake Shack is exclusively known for American cuisine, with a total of 219 orders.
2. The Meatball Shop offers both American and Italian cuisines, with a higher preference for Italian cuisine (112 orders) compared to American (20 orders).
3. Blue Ribbon Sushi specializes in Japanese cuisine, with 119 orders.
4. Blue Ribbon Fried Chicken is known for American cuisine, with 96 orders.
5. Parm is exclusively known for Italian cuisine, with 68 orders.
6. RedFarm Broadway and RedFarm Hudson are known for Chinese cuisine, with 59 and 55 orders respectively.
7. TAO specializes in Japanese cuisine, with 49 orders.
8. Han Dynasty is known for Chinese cuisine, with 46 orders.
9. Blue Ribbon Sushi Bar & Grill specializes in Japanese cuisine, with 44 orders.

These observations indicate that certain restaurants have a strong specialization in specific cuisines, and they tend to receive a high volume of orders for those cuisines. This could suggest a strong customer preference for these restaurants and their specialized cuisines.

Moreover top restaurants don't have much variety in cuisine type, they are specialized in one or two cuisine types.

## 7. Restaurant Name & Food Preparation Time

Investigate the distribution of food preparation times across restaurants, which can help identify restaurants with consistently higher or lower preparation times & detect if there are any outliers (ie exceptions) in preparation times for specific restaurants.

This can drive potential insights into operational & efficiency issues

```
In [120...]: x = 'restaurant_name'
y = 'food_preparation_time'
df[x].nunique(), df[y].nunique()

Out[120]: (178, 16)
```

**NOTE:** As there are too many restaurants, we will pick top performing restaurants (ie which receives more orders comparatively)

```
In [121...]: # PREP (Data Preparation)

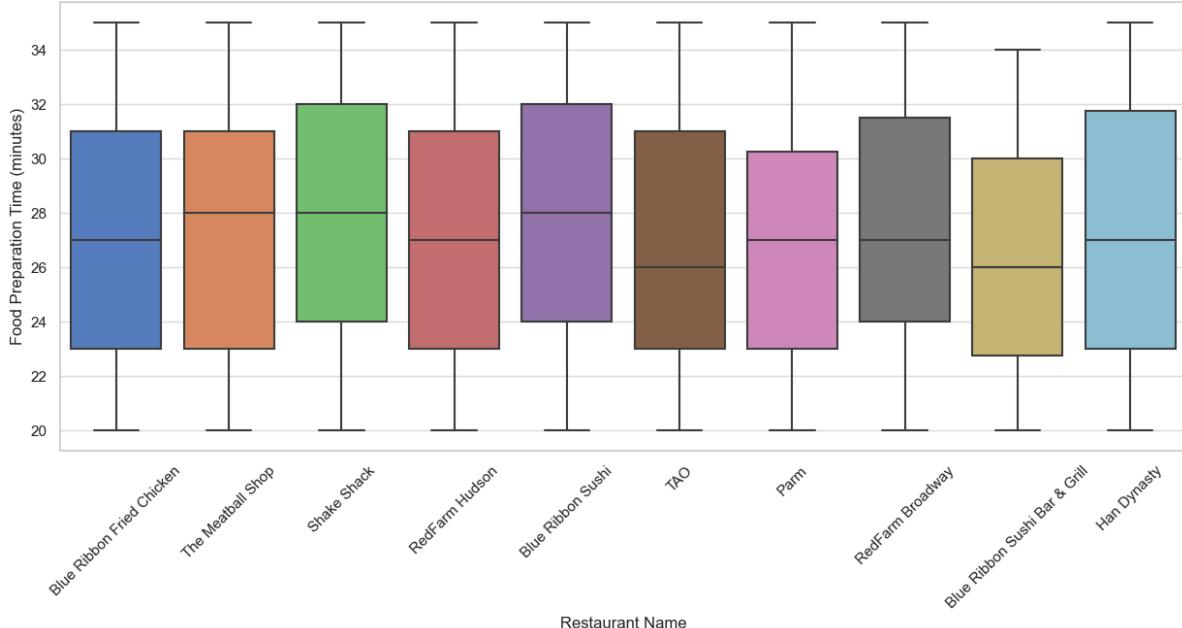
# Filter for top 10 restaurants by order count
famous_restaurants = df['restaurant_name'].value_counts().head(10).index
filtered_df = df[df['restaurant_name'].isin(famous_restaurants)]
```

```
In [122...]: print("Top 10 Restaurants by Order Count :")
print(famous_restaurants)

Top 10 Restaurants by Order Count :
Index(['Shake Shack', 'The Meatball Shop', 'Blue Ribbon Sushi',
       'Blue Ribbon Fried Chicken', 'Parm', 'RedFarm Broadway',
       'RedFarm Hudson', 'TAO', 'Han Dynasty',
       'Blue Ribbon Sushi Bar & Grill'],
      dtype='object')
```

```
In [123...]: # Plot
sns.boxplot(data=filtered_df, x='restaurant_name', y='food_preparation_time')
plt.title('Food Preparation Time Distribution for Top 10 Restaurants')
plt.xlabel('Restaurant Name')
plt.ylabel('Food Preparation Time (minutes)')
plt.xticks(rotation=45)
plt.show()
```

Food Preparation Time Distribution for Top 10 Restaurants



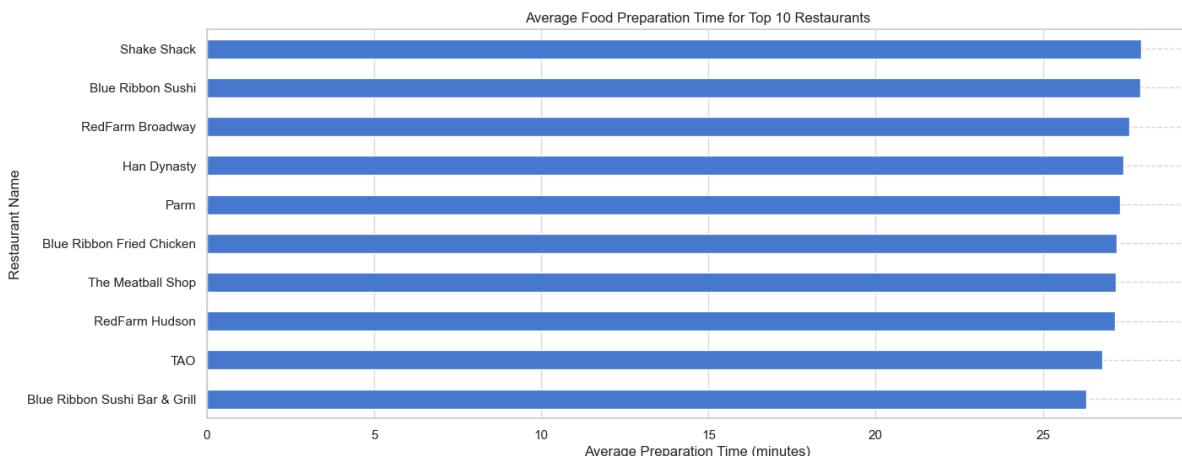
### 🔍 Insights:

Food preparation time distribution indicates that all restaurants show consistency in preparing food. & Seems like all of them following conventional steps to prepare food & hence having almost similar set of duration, likewise.

In [124...]

### # Plot 2 (Horizontal Bar Plot)

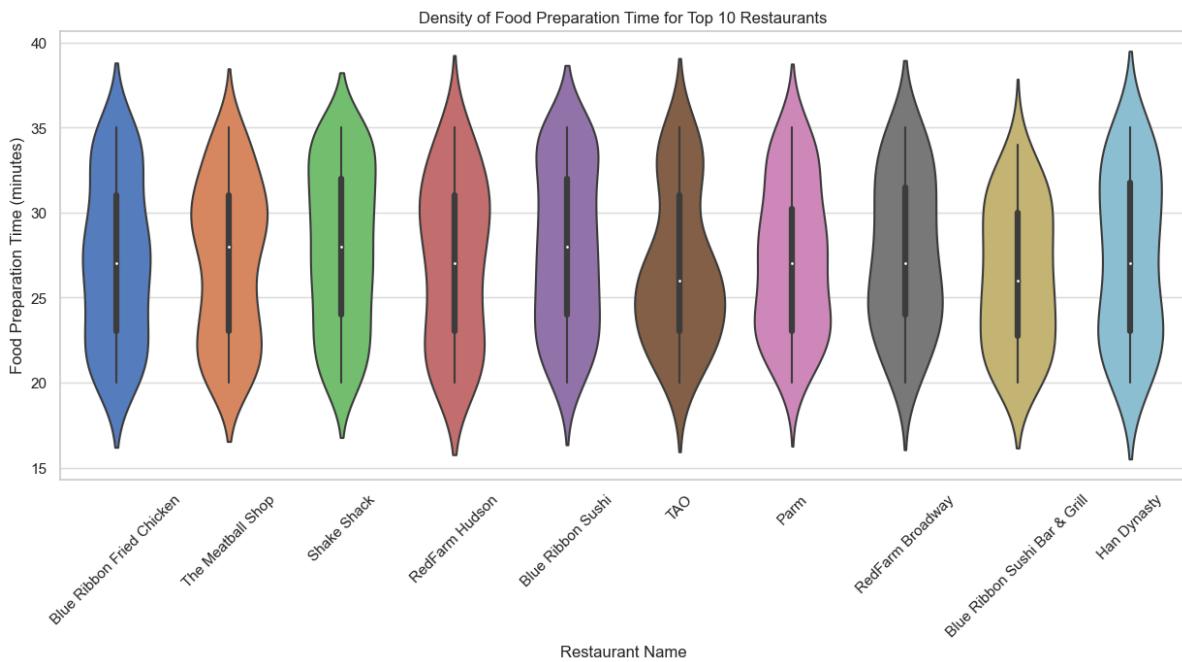
```
restaurant_avg_prep_time = filtered_df.groupby('restaurant_name')['food_prep_time'].mean()
# plot (horizontal bars)
restaurant_avg_prep_time.plot(kind='barh')
# aesthetics
plt.title('Average Food Preparation Time for Top 10 Restaurants')
plt.xlabel('Average Preparation Time (minutes)')
plt.ylabel('Restaurant Name')
# note: need to provide grid property here manually
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



In [125...]

### # Plot 3

```
sns.violinplot(data=filtered_df, x='restaurant_name', y='food_preparation_time')
plt.title('Density of Food Preparation Time for Top 10 Restaurants')
plt.xlabel('Restaurant Name')
plt.ylabel('Food Preparation Time (minutes)')
plt.xticks(rotation=45)
plt.show()
```

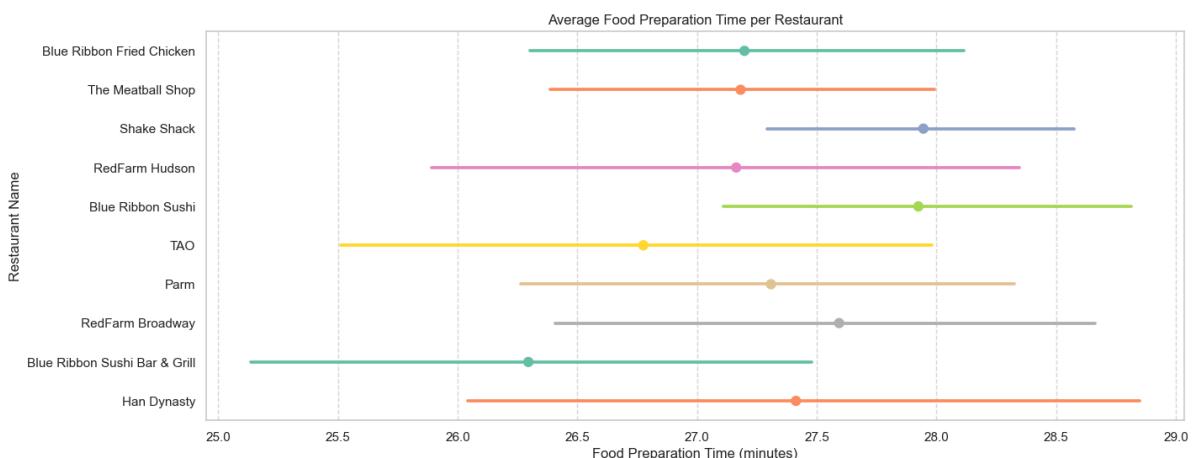


In [126]: # Plot 4

```

sns.pointplot(
    data=filtered_df,
    x="food_preparation_time",
    y="restaurant_name",
    palette="Set2",
    join=False,
    markers="o",
    linestyles="--",
)
plt.title("Average Food Preparation Time per Restaurant")
plt.xlabel("Food Preparation Time (minutes)")
plt.ylabel("Restaurant Name")
plt.grid(axis="x", linestyle="--", alpha=0.7)
plt.show()

```



In [127]:

```

# get stats for food preparation time for top 10 restaurants
restaurant_food_preparation_time = filtered_df.groupby('restaurant_name')[['food_preparation_time']].mean()

```

Out [127]:

| restaurant_name               | count | mean      | std      | min  | 25%   | 50%  | 75%   | max  |
|-------------------------------|-------|-----------|----------|------|-------|------|-------|------|
| Blue Ribbon Fried Chicken     | 96.0  | 27.197917 | 4.740908 | 20.0 | 23.00 | 27.0 | 31.00 | 35.0 |
| Blue Ribbon Sushi             | 119.0 | 27.924370 | 4.758863 | 20.0 | 24.00 | 28.0 | 32.00 | 35.0 |
| Blue Ribbon Sushi Bar & Grill | 44.0  | 26.295455 | 4.100933 | 20.0 | 22.75 | 26.0 | 30.00 | 34.0 |
| Han Dynasty                   | 46.0  | 27.413043 | 4.833108 | 20.0 | 23.00 | 27.0 | 31.75 | 35.0 |
| Parm                          | 68.0  | 27.308824 | 4.351213 | 20.0 | 23.00 | 27.0 | 30.25 | 35.0 |
| RedFarm Broadway              | 59.0  | 27.593220 | 4.464877 | 20.0 | 24.00 | 27.0 | 31.50 | 35.0 |
| RedFarm Hudson                | 55.0  | 27.163636 | 4.728807 | 20.0 | 23.00 | 27.0 | 31.00 | 35.0 |
| Shake Shack                   | 219.0 | 27.945205 | 4.744549 | 20.0 | 24.00 | 28.0 | 32.00 | 35.0 |
| TAO                           | 49.0  | 26.775510 | 4.431259 | 20.0 | 23.00 | 26.0 | 31.00 | 35.0 |
| The Meatball Shop             | 132.0 | 27.181818 | 4.596411 | 20.0 | 23.00 | 28.0 | 31.00 | 35.0 |

In [128...]

```
# Just to see prep time in sorted order refering this
restaurant_avg_prep_time
```

Out [128]:

| restaurant_name               | avg_prep_time |
|-------------------------------|---------------|
| Blue Ribbon Sushi Bar & Grill | 26.295455     |
| TAO                           | 26.775510     |
| RedFarm Hudson                | 27.163636     |
| The Meatball Shop             | 27.181818     |
| Blue Ribbon Fried Chicken     | 27.197917     |
| Parm                          | 27.308824     |
| Han Dynasty                   | 27.413043     |
| RedFarm Broadway              | 27.593220     |
| Blue Ribbon Sushi             | 27.924370     |
| Shake Shack                   | 27.945205     |

Name: food\_preparation\_time, dtype: float64

### Observation:

1. **Shake Shack** has the highest average food preparation time of approximately 27.95 minutes.
2. **Blue Ribbon Sushi Bar & Grill** has the lowest average food preparation time of approximately 26.30 minutes.
3. The interquartile range (IQR) for most restaurants is narrow, suggesting that the majority of orders are prepared within a similar time frame.
4. The maximum food preparation time for all top 10 restaurants is 35 minutes, while the minimum is 20 minutes.
5. **TAO** and **Blue Ribbon Sushi Bar & Grill** have the lowest average preparation times, indicating potentially more efficient kitchen operations.

Overall, the data indicates that the top 10 restaurants maintain a high level of consistency in their food preparation times, with some variability due to order complexity or volume. These observations suggest that while there is some variability in food preparation times among the top 10 restaurants, the differences are not substantial. Most restaurants have a standardized process that ensures consistent preparation times, which can lead to predictable and reliable service for customers.

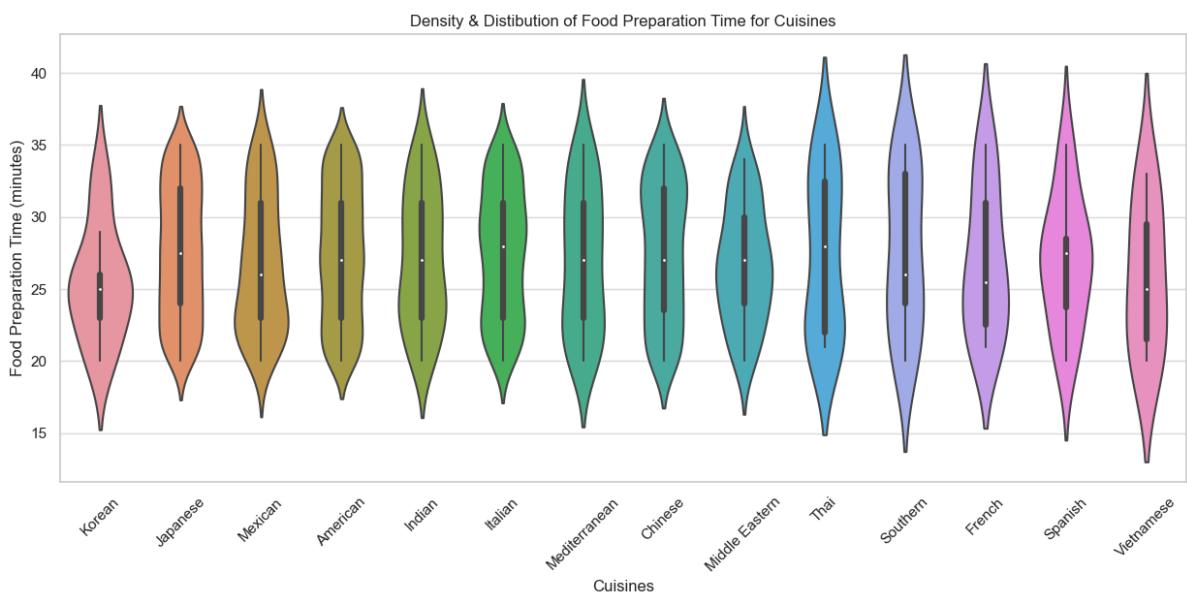
## 8. Cuisine Type & Food Preparation Time

(Potential insights into restaurant operations based on cuisine-specific demands)

```
In [129]: x = 'cuisine_type'
y = 'food_preparation_time'
df[x].nunique(), df[y].nunique()
```

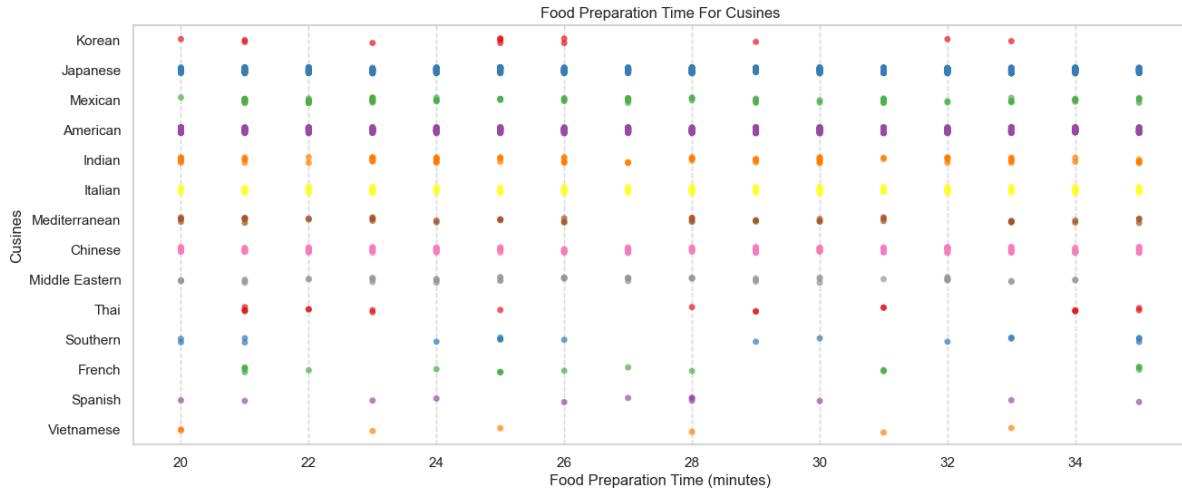
Out[129]: (14, 16)

```
In [130]: # Plot 1
sns.violinplot(data=df, x='cuisine_type', y='food_preparation_time', inner='white')
plt.title('Density & Distribution of Food Preparation Time for Cuisines')
plt.xlabel('Cuisines')
plt.ylabel('Food Preparation Time (minutes)')
plt.xticks(rotation=45)
plt.show()
```



```
In [131]: # Plot 2
```

```
# Plot the strip plot
sns.stripplot(
    data=df,
    x='food_preparation_time',
    y='cuisine_type',
    jitter=True,
    alpha=0.7,
    palette='Set1',
)
plt.title("Food Preparation Time For Cusines")
plt.xlabel("Food Preparation Time (minutes)")
plt.ylabel("Cusines")
plt.grid(axis="x", linestyle="--", alpha=0.7)
plt.show()
```



```
In [132]: # Get Descriptive Stats about food preparation time for each cuisine type
cuisine_prep_time_stats = df.groupby('cuisine_type')['food_preparation_time'].describe()
```

Out[132]:

| cuisine_type          | count | mean      | std      | min  | 25%   | 50%  | 75%  | max  |
|-----------------------|-------|-----------|----------|------|-------|------|------|------|
| <b>American</b>       | 584.0 | 27.440068 | 4.658474 | 20.0 | 23.00 | 27.0 | 31.0 | 35.0 |
| <b>Chinese</b>        | 215.0 | 27.511628 | 4.758525 | 20.0 | 23.50 | 27.0 | 32.0 | 35.0 |
| <b>French</b>         | 18.0  | 26.888889 | 5.039712 | 21.0 | 22.50 | 25.5 | 31.0 | 35.0 |
| <b>Indian</b>         | 73.0  | 27.109589 | 4.620491 | 20.0 | 23.00 | 27.0 | 31.0 | 35.0 |
| <b>Italian</b>        | 298.0 | 27.483221 | 4.519474 | 20.0 | 23.00 | 28.0 | 31.0 | 35.0 |
| <b>Japanese</b>       | 470.0 | 27.510638 | 4.604496 | 20.0 | 24.00 | 27.5 | 32.0 | 35.0 |
| <b>Korean</b>         | 13.0  | 25.461538 | 3.971049 | 20.0 | 23.00 | 25.0 | 26.0 | 33.0 |
| <b>Mediterranean</b>  | 46.0  | 27.000000 | 4.908043 | 20.0 | 23.00 | 27.0 | 31.0 | 35.0 |
| <b>Mexican</b>        | 77.0  | 26.727273 | 4.604450 | 20.0 | 23.00 | 26.0 | 31.0 | 35.0 |
| <b>Middle Eastern</b> | 49.0  | 26.673469 | 4.012417 | 20.0 | 24.00 | 27.0 | 30.0 | 34.0 |
| <b>Southern</b>       | 17.0  | 27.588235 | 5.523346 | 20.0 | 24.00 | 26.0 | 33.0 | 35.0 |
| <b>Spanish</b>        | 12.0  | 26.916667 | 4.501683 | 20.0 | 23.75 | 27.5 | 28.5 | 35.0 |
| <b>Thai</b>           | 19.0  | 27.315789 | 5.498006 | 21.0 | 22.00 | 28.0 | 32.5 | 35.0 |
| <b>Vietnamese</b>     | 7.0   | 25.714286 | 5.154748 | 20.0 | 21.50 | 25.0 | 29.5 | 33.0 |

```
In [133]: restaurant_prep_time = df.groupby('cuisine_type')['food_preparation_time'].mean()
restaurant_prep_time
```

```
Out[133]: cuisine_type
Korean           25.0
Vietnamese       25.0
French           25.5
Mexican          26.0
Southern          26.0
American          27.0
Chinese           27.0
Indian            27.0
Mediterranean    27.0
Middle Eastern    27.0
Japanese          27.5
Spanish           27.5
Italian           28.0
Thai               28.0
Name: food_preparation_time, dtype: float64
```

### **Observation:**

- The average preparation times for American, Chinese, Italian, and Japanese cuisines are quite similar.
- The standard deviations are also similar, indicating consistent preparation times within each cuisine type.
- The minimum and maximum preparation times are the same across all cuisines, suggesting a standardized range for food preparation.
- The interquartile ranges (IQR) are narrow, indicating that most orders are prepared within a similar time frame for each cuisine.
- The median preparation times are close to the average times, further indicating consistency in preparation times.

These observations suggest that the preparation times for different cuisines are fairly consistent, with only slight variations. This consistency can lead to predictable and reliable service for customers, regardless of the cuisine type.

## 9. Cost of Order & Food Preparation Time

(This can help identify if there's a relationship between the cost of an order and the time taken to prepare the food.) For example, do higher-cost orders generally require more preparation time?

```
In [134...]: x = 'cost_of_the_order'
y = 'food_preparation_time'
```

```
In [135...]: # Get correlation between 2 variables
df[x].corr(df[y])
```

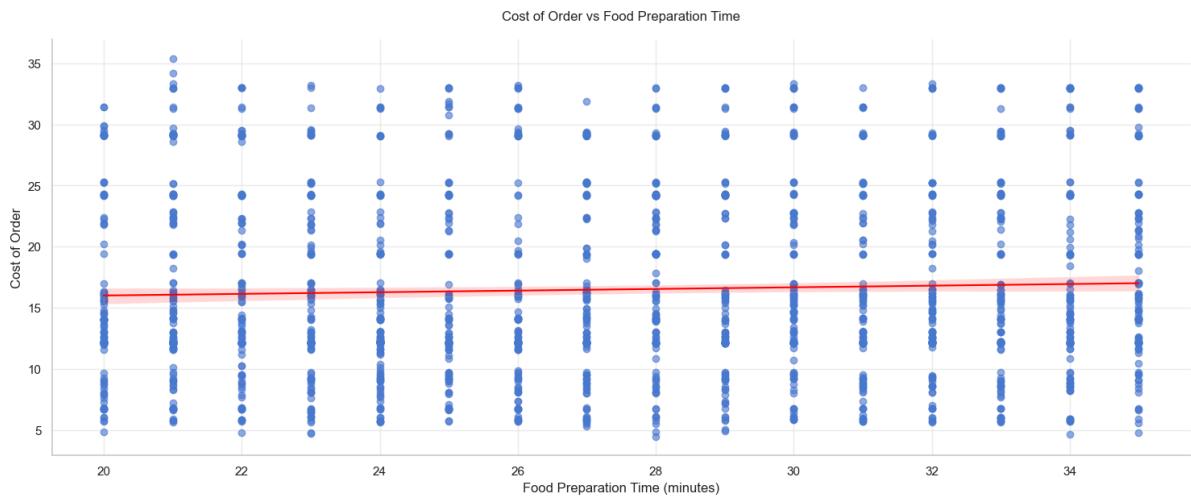
```
Out[135]: 0.04152747282774475
```

```
In [136...]: df[[x, y]].corr()
```

|                       | cost_of_the_order | food_preparation_time |
|-----------------------|-------------------|-----------------------|
| cost_of_the_order     | 1.000000          | 0.041527              |
| food_preparation_time | 0.041527          | 1.000000              |

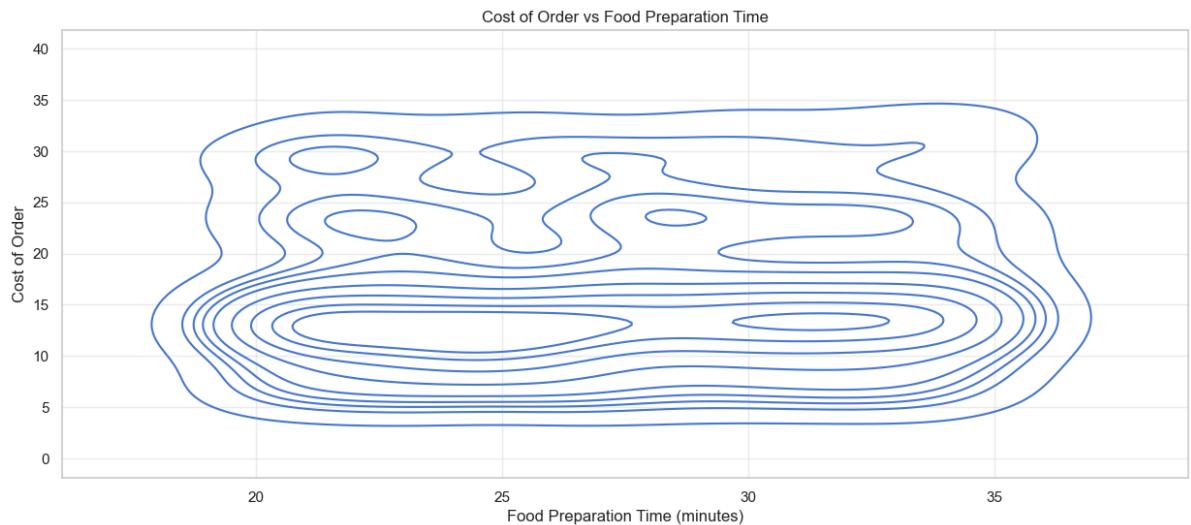
In [137...]

```
sns.lmplot(data=df,
            x='food_preparation_time',
            y='cost_of_the_order',
            height=6, aspect=2.5,
            scatter_kws={'alpha': 0.6}, # Transparency for scatter points
            line_kws={'linewidth': 1.5, 'color': 'red'}) # Thicker regression line
plt.title('Cost of Order vs Food Preparation Time', y=1.03)
plt.xlabel('Food Preparation Time (minutes)')
plt.ylabel('Cost of Order')
plt.grid(alpha=0.3)
plt.show()
```



In [138...]

```
sns.kdeplot(
    data=df,
    x="food_preparation_time",
    y="cost_of_the_order",
)
plt.title("Cost of Order vs Food Preparation Time")
plt.xlabel("Food Preparation Time (minutes)")
plt.ylabel("Cost of Order")
plt.grid(alpha=0.3)
plt.show()
```



Insights:

At high level, it seems that the preparation time for food items does not strongly influence their cost. i.e High-cost items do not necessarily take longer to prepare, nor do low-cost items tend to be quicker.

Hence maybe cuisine type & restaurant policies, might play a more significant role in determining cost.

## 10. Cuisine Type, Cost of Order, Day of Week

This can reveal demand & patterns for specific cuisines on different days.

```
In [139...]: x = 'cuisine_type'
y = 'cost_of_the_order'
z = 'day_of_the_week'
```

```
In [140...]: cost_of_cuisine_by_day = df.groupby([x, z])[y].mean()
cost_of_cuisine_by_day
```

```
Out[140]:   cuisine_type  day_of_the_week
American      Weekday      15.308343
                  Weekend     16.731735
Chinese       Weekday      17.147692
                  Weekend     16.036442
French        Weekday      20.866000
                  Weekend     19.381538
Indian         Weekday      17.162500
                  Weekend     16.800816
Italian        Weekday      16.890110
                  Weekend     16.211449
Japanese       Weekday      16.368074
                  Weekend     16.278925
Korean          Weekday      12.180000
                  Weekend     14.332727
Mediterranean Weekday      14.040000
                  Weekend     16.102500
Mexican         Weekday      16.555833
                  Weekend     17.103962
Middle Eastern Weekday      19.832941
                  Weekend     18.282812
Southern        Weekday      19.226667
                  Weekend     19.340909
Spanish         Weekday      12.130000
                  Weekend     19.618182
Thai            Weekday      15.462500
                  Weekend     20.206667
Vietnamese      Weekday      11.916667
                  Weekend     13.607500
Name: cost_of_the_order, dtype: float64
```

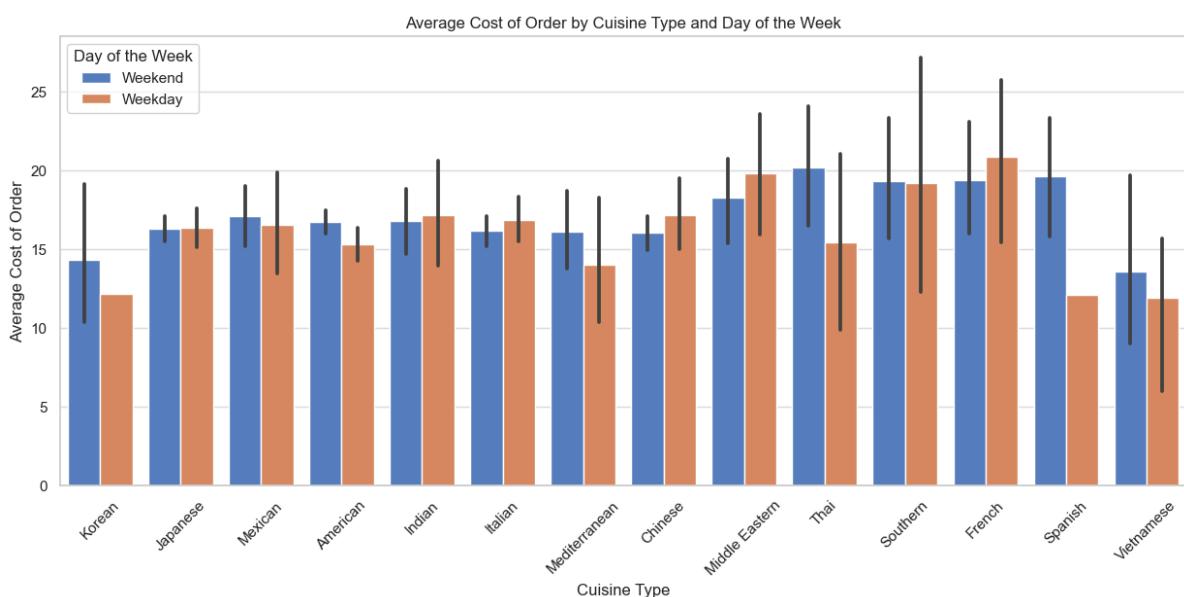
```
In [141...]: # convert to table form by unstack
# NOTE: second level (ie day_of_the_week) will become column
cost_of_cuisine_by_day_tbl = cost_of_cuisine_by_day.unstack()
cost_of_cuisine_by_day_tbl
```

Out [141]: day\_of\_the\_week Weekday Weekend

| cuisine_type   | Weekday   | Weekend   |
|----------------|-----------|-----------|
| American       | 15.308343 | 16.731735 |
| Chinese        | 17.147692 | 16.036442 |
| French         | 20.866000 | 19.381538 |
| Indian         | 17.162500 | 16.800816 |
| Italian        | 16.890110 | 16.211449 |
| Japanese       | 16.368074 | 16.278925 |
| Korean         | 12.180000 | 14.332727 |
| Mediterranean  | 14.040000 | 16.102500 |
| Mexican        | 16.555833 | 17.103962 |
| Middle Eastern | 19.832941 | 18.282812 |
| Southern       | 19.226667 | 19.340909 |
| Spanish        | 12.130000 | 19.618182 |
| Thai           | 15.462500 | 20.206667 |
| Vietnamese     | 11.916667 | 13.607500 |

In [142...]

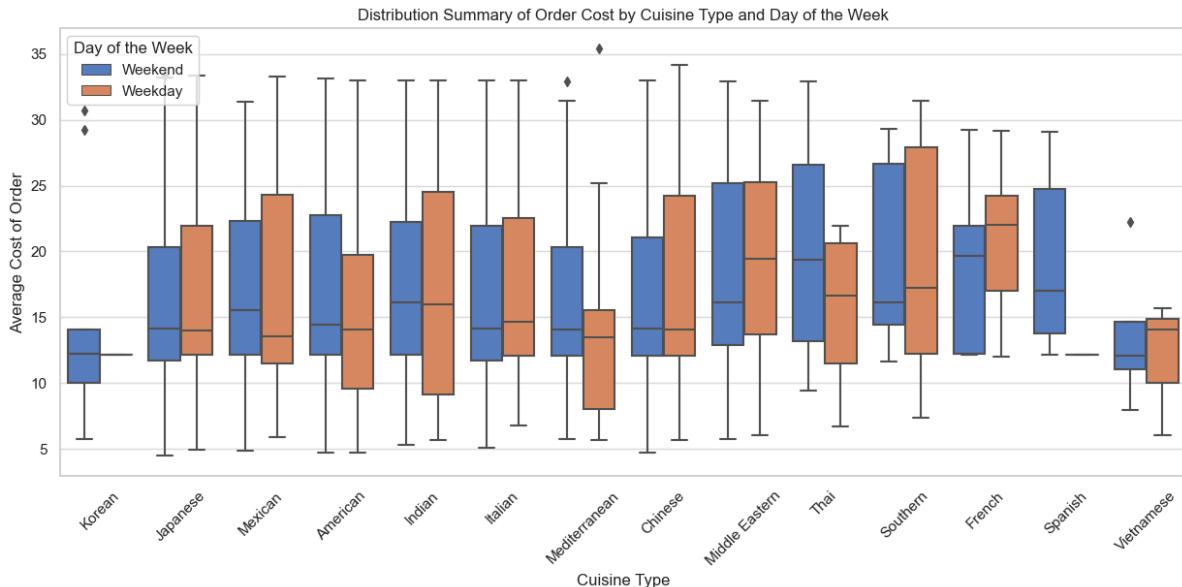
```
# Plot 1
sns.barplot(data=df, x=x, y=y, hue=z)
plt.title('Average Cost of Order by Cuisine Type and Day of the Week')
plt.xlabel('Cuisine Type')
plt.ylabel('Average Cost of Order')
plt.xticks(rotation=45)
plt.legend(title='Day of the Week', loc='upper left')
plt.show()
```



In [143...]

```
# Plot 2
sns.boxplot(data=df, x=x, y=y, hue=z)
plt.title('Distribution Summary of Order Cost by Cuisine Type and Day of the Week')
plt.xlabel('Cuisine Type')
plt.ylabel('Average Cost of Order')
plt.xticks(rotation=45)
```

```
plt.legend(title='Day of the Week', loc='upper left')
plt.show()
```



In [144]: spanish\_weekday\_orders = df[(df['cuisine\_type'] == 'Spanish') & (df['day\_of\_the\_week'] == 'W')]

Out [144]:

|      | order_id | customer_id | restaurant_name | cuisine_type | cost_of_the_order | day_of_the_week |
|------|----------|-------------|-----------------|--------------|-------------------|-----------------|
| 1138 | 1477023  | 99288       | Boqueria        | Spanish      | 12.13             | W               |

In [145]: korean\_weekday\_orders = df[(df['cuisine\_type'] == 'Korean') & (df['day\_of\_the\_week'] == 'W')]

Out [145]:

|      | order_id | customer_id | restaurant_name         | cuisine_type | cost_of_the_order | day_of_the_week |
|------|----------|-------------|-------------------------|--------------|-------------------|-----------------|
| 1449 | 1477461  | 241719      | Kori Restaurant and Bar | Korean       | 12.18             | W               |
| 1859 | 1476790  | 141732      | Cho Dang Gol            | Korean       | 12.18             | W               |

🔍 Insights:

- There is only 1 Spanish weekday order
- There are only 2 Orders for Korean Dishes on Weekdays

**People in New-York don't like to consume Spanish & Korean dishes during weekdays**

🔍 Observation:

- Most cuisines have higher average costs on weekends compared to weekdays, indicating that customers are generally willing to spend more on food orders during weekends.
- French, Middle Eastern, and Thai cuisines have the highest average costs, suggesting that these cuisines are generally more expensive.
- The variation in average costs across different cuisines and days of the week provides insights into customer spending behavior and preferences.

## 11. Ratings, Cost of Order & Food Preparation Time

(Determine how preparation time and cost impact customer ratings. The goal is to explore correlations and trends, which can highlight key drivers of customer satisfaction.)

```
In [146...]: x = 'cost_of_the_order'
y = 'preparation_time'
z = 'rating'
```

```
In [147...]: # Plot
g = sns.jointplot(
    data=df,
    x="cost_of_the_order",
    y="food_preparation_time",
    kind="reg", # Regression plot
    height=8, # Figure size
    space=0.5, # Space between main plot and marginal plots
    marginal_kws={"bins": 20, "fill": True}, # Styling marginal histograms
    color="teal",
)
plt.suptitle(
    "Relationship Between Cost of Order and Food Preparation Time", y=1.02,
)
g.figure.set_figwidth(15)
g.figure.set_figheight(6)
plt.show()
```



### 🔍 Insights:

The distribution of food preparation times is fairly uniform, with no significant peaks or concentrations at any particular time range. This indicates that preparation times are consistent across all orders, with no noticeable trend or bias toward faster or slower preparations.

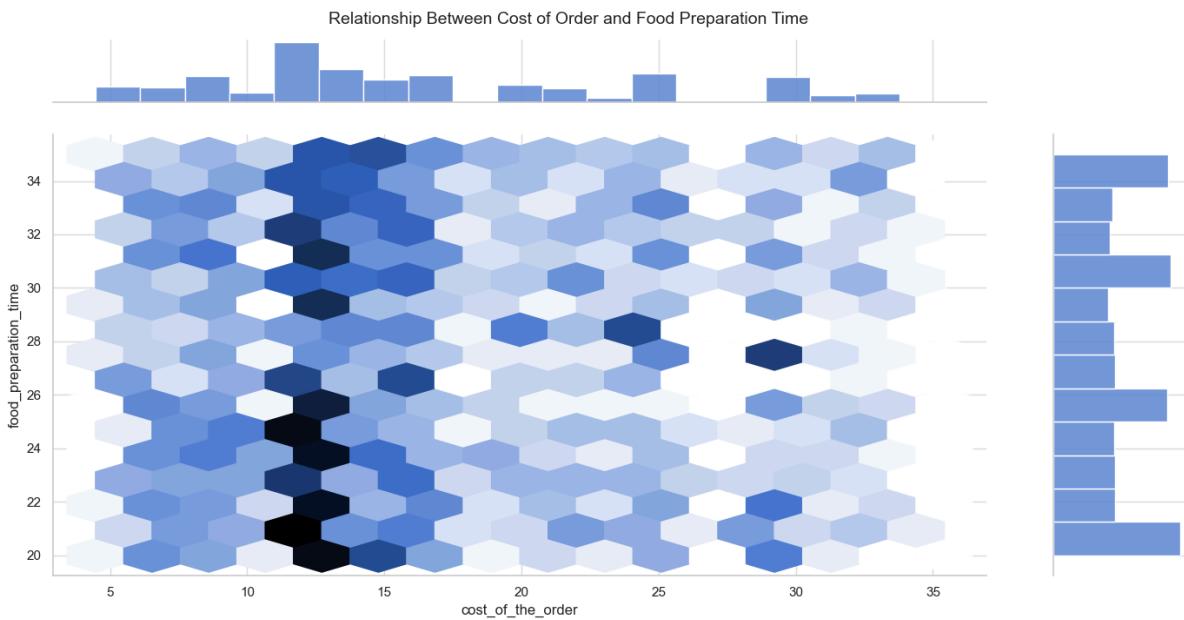
The distribution of order costs is right-skewed, meaning that most orders are relatively inexpensive, with fewer orders having significantly higher costs. This suggests that while the majority of customers are making lower-cost orders, there are a few large or multiple-item orders that are driving the higher cost values.

```
In [148...]: g = sns.jointplot(
    data=df,
    x="cost_of_the_order",
```

```

y="food_preparation_time",
kind="hex", # Scatter plot
height=7, # Figure size
space=0.5, # Space between main plot and marginal plots
)
plt.suptitle(
    "Relationship Between Cost of Order and Food Preparation Time", y=1.02,
)
g.figure.set_figwidth(15)
g.figure.set_figheight(7)
plt.show()

```



### 🔍 Insights:

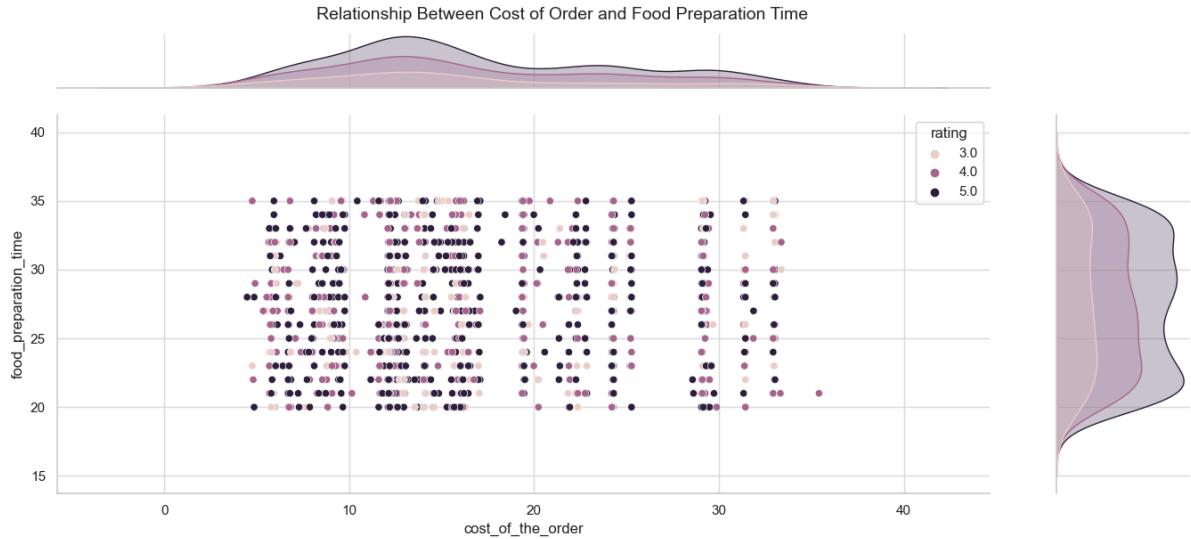
The hex plot reveals that the majority of orders, especially the higher frequency ones, tend to have lower costs and relatively shorter food preparation times. However, there are some instances of low-cost orders that took significantly longer to prepare, suggesting that while most orders follow this trend, there are exceptions with longer preparation times for certain cost-effective items.

In [149]:

```

# Plot
g = sns.jointplot(
    data=df,
    x="cost_of_the_order",
    y="food_preparation_time",
    kind="scatter", # Scatter plot
    height=6, # Figure size
    hue="rating",
    space=0.5, # Space between main plot and marginal plots
    color="teal",
)
plt.suptitle(
    "Relationship Between Cost of Order and Food Preparation Time", y=1.02,
)
g.figure.set_figwidth(15)
g.figure.set_figheight(6)
plt.show()

```



### 🔍 Insights:

The scatter plot reveals that orders across all ratings are spread out across both preparation time and order cost. This indicates that there's no clear pattern linking the rating to either the time taken for preparation or the cost of the order. In other words, the ratings don't seem to depend on how long the food takes to prepare or how much it costs. Note: here we are ignoring the orders where ratings are not provided

Orders with a rating of 5 show some unusual patterns compared to those rated 3 and 4. Specifically, these orders tend to have a higher or more varied cost and food preparation time, which isn't as consistent in the lower-rated orders. This suggests that while customers may give a perfect rating, these orders might involve more complex or expensive meals, leading to less predictability in how much they cost or how long they take to prepare

In [150...]

```
# Plot
sns.swarmplot(
    data=df,
    x='rating',
    y='food_preparation_time',
    hue='rating',
    dodge=True,
)
plt.title('Food Preparation Time vs Rating')
plt.xlabel('Rating')
plt.ylabel('Food Preparation Time')
plt.legend(title='Rating')
plt.show()
```



### 🔍 Insights:

We can see that many people have rated 5 regardless of Food Preparation Time. Also similar behavior is seen for rating 4 too

In [151...]

```
# Plot
g = sns.FacetGrid(df, col='rating', col_wrap=3, height=4, aspect=1.2, sharex=True)
g.map(sns.scatterplot, 'cost_of_the_order', 'food_preparation_time', alpha=0.5)
g.set_titles("Rating: {col_name}")
g.set_axis_labels("Cost of the Order", "Food Preparation Time")
g.figure.suptitle('Cost vs Food Preparation Time by Rating', y=1.02, fontsize=14)
plt.show()
```



### 🔍 Observation:

- People ratings 4 or 5 to an order seems to place more low cost or nominal cost order.
- People rating 3 seems not to have concerns primarily with cost but other aspects of order.

Hence From the scatter plot, it appears that customers who rated their orders 4 or 5 tend to order lower-cost items, regardless of the preparation time. On the other hand, customers who rated their orders with a 3 seem to prioritize factors other than cost, indicating that they may be less concerned with the price and more focused on other aspects of their order, such as quality or service.

In [152...]

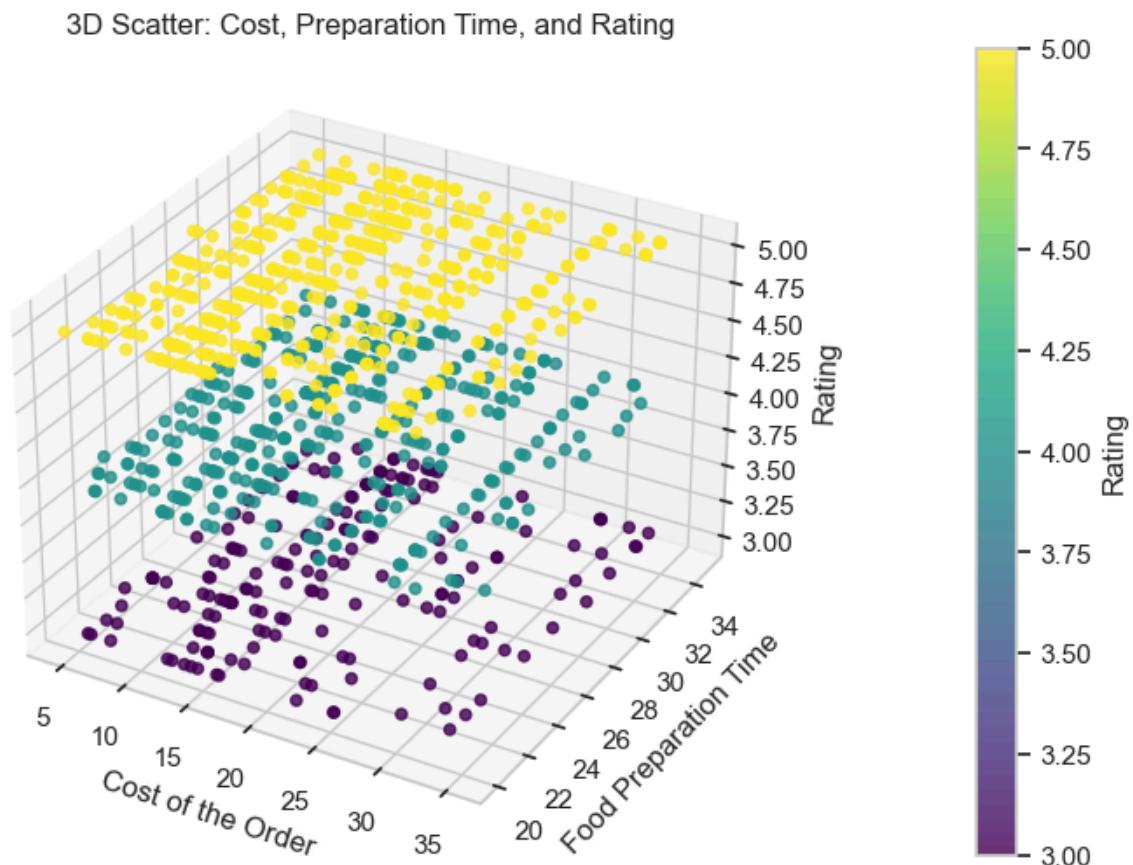
```
# Plot
fig = plt.figure(figsize=(15, 6))
ax = fig.add_subplot(111, projection='3d')
```

```
# Scatter plot
sc = ax.scatter(
    df['cost_of_the_order'],
    df['food_preparation_time'],
    df['rating'],
    c=df['rating'], cmap='viridis', alpha=0.8
)

# Axis labels
ax.set_xlabel('Cost of the Order')
ax.set_ylabel('Food Preparation Time')
ax.set_zlabel('Rating')

# Colorbar
cbar = plt.colorbar(sc, ax=ax, pad=0.1)
cbar.set_label('Rating')

plt.title('3D Scatter: Cost, Preparation Time, and Rating')
plt.show()
```



### 🔍 Insights:

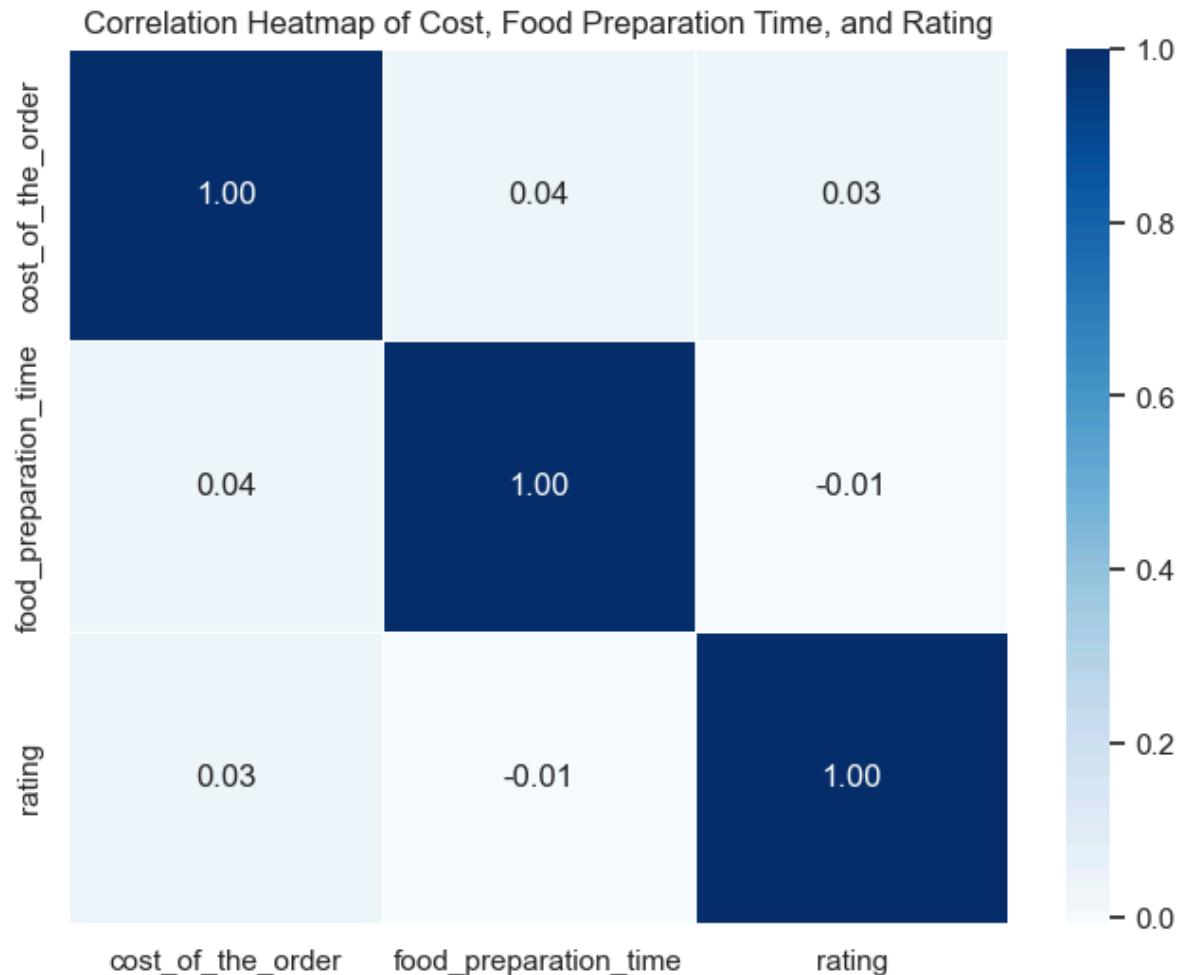
In the 3D plot, the points representing orders with a 5-star rating are more evenly distributed across both food preparation time and cost, indicating a uniform pattern in customer choices. In contrast, the 3-star rating plane shows a noticeable gap between the clustered points, suggesting that these customers might have more varied preferences or that there is a specific factor influencing their decisions differently than those who gave higher ratings.

In [153...]

```
# Plot

# Calculate correlation matrix
corr_matrix = df[['cost_of_the_order', 'food_preparation_time', 'rating']].c
```

```
# Plot heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='Blues', fmt='.2f', linewidths=0.5)
plt.title('Correlation Heatmap of Cost, Food Preparation Time, and Rating')
plt.show()
```



### 🔍 Observation:

The correlations between cost, preparation time, and rating are all very weak, indicating that these factors are largely independent of each other. This suggests that other factors, such as food quality, service, and customer expectations, may play a more significant role in determining customer satisfaction and ratings.

In [154...]: # Left Blank Intentionally !! (For Gap)

In [155...]: # Snity check

```
# Seems good as rating column is of numeric type (as converted)
df.head()
```

Out [155]:

|   | order_id | customer_id | restaurant_name           | cuisine_type | cost_of_the_order | day_of_the_week |
|---|----------|-------------|---------------------------|--------------|-------------------|-----------------|
| 0 | 1477147  | 337525      | Hangawi                   | Korean       | 30.75             | Weekend         |
| 1 | 1477685  | 358141      | Blue Ribbon Sushi Izakaya | Japanese     | 12.08             | Weekend         |
| 2 | 1477070  | 66393       | Cafe Habana               | Mexican      | 12.23             | Weekend         |
| 3 | 1477334  | 106968      | Blue Ribbon Fried Chicken | American     | 29.20             | Weekend         |
| 4 | 1478249  | 76942       | Dirty Bird to Go          | American     | 11.59             | Weekend         |

**Question 13:** The company wants to provide a promotional offer in the advertisement of the restaurants. The condition to get the offer is that the restaurants must have a rating count of more than 50 and the average rating should be greater than 4. Find the restaurants fulfilling the criteria to get the promotional offer. [3 marks]

In [156]:

```
# ? Assumption - at this point column 'rating' has no missing values & it's
# 2. Group by restaurant name and calculate the count of ratings and the average rating
restaurant_stats = df.groupby('restaurant_name')['rating'].agg(['count', 'mean'])

# 3. Filter restaurants based on the criteria: count > 50 and mean > 4
eligible_restaurants = restaurant_stats[(restaurant_stats['count'] > 50) &
                                         (restaurant_stats['mean'] > 4)]

# 4. Display the eligible restaurants
tableit_df(eligible_restaurants, show_index=True)
```

| restaurant_name           | count | mean               |
|---------------------------|-------|--------------------|
| Blue Ribbon Fried Chicken | 64.0  | 4.328125           |
| Blue Ribbon Sushi         | 73.0  | 4.219178082191781  |
| Shake Shack               | 133.0 | 4.2781954887218046 |
| The Meatball Shop         | 84.0  | 4.511904761904762  |

In [157]:

```
# ! Printing raw as well (as if above tabular doesn't work i.e missing module)
eligible_restaurants
```

Out [157]:

| restaurant_name           | count | mean     |
|---------------------------|-------|----------|
| Blue Ribbon Fried Chicken | 64    | 4.328125 |
| Blue Ribbon Sushi         | 73    | 4.219178 |
| Shake Shack               | 133   | 4.278195 |
| The Meatball Shop         | 84    | 4.511905 |

In [158]:

```
print('Eligible Restaurants for Promotional Offer:')
eligible_restaurants.index.to_list()
```

Eligible Restaurants for Promotional Offer:

```
Out[158]: ['Blue Ribbon Fried Chicken',
 'Blue Ribbon Sushi',
 'Shake Shack',
 'The Meatball Shop']
```

 **Observation:**

The following restaurants meet the criteria of having more than 50 ratings and an average rating greater than 4:

1. **Blue Ribbon Fried Chicken** - avg rating: 4.33 (64 ratings)
2. **Blue Ribbon Sushi** - avg rating: 4.22 (73 ratings)
3. **Shake Shack** – avg rating: 4.28 (133 ratings)
4. **The Meatball Shop** – avg rating: 4.51 (84 ratings)

These restaurants are eligible for the promotional offer as they consistently receive high ratings from a significant number of customers.

**Question 14:** The company charges the restaurant 25% on the orders having cost greater than 20 dollars and 15% on the orders having cost greater than 5 dollars. Find the net revenue generated by the company across all orders. [3 marks]

```
In [159...]  
def calculate_revenue(row):  
    ''' Calculate revenue based on cost of order'''  
    # ? NOTE : the unit of currency is dollar  
    match row['cost_of_the_order']:  
        case cost if cost > 20: # more than 20 dollar  
            return cost * 0.25  
        case cost if cost > 5: # more than 5 dollar  
            return cost * 0.15  
        case _: # less than or equal to 5 dollar  
            return 0  
  
# Apply the function to each row and calculate the total revenue  
orders_revenue = df.apply(calculate_revenue, axis=1)  
  
# Calculate total net revenue generated  
total_revenue = orders_revenue.sum()  
  
# Display the total revenue  
print(f"Total Net Revenue Generated by the Company across all orders is: ${total_revenue}")
```

Total Net Revenue Generated by the Company across all orders is: \$6166.30

 **Observation:**

Total Net Revenue Generated by the Company across all orders is: \$6166.30

**Question 15:** The company wants to analyze the total time required to deliver the food. What percentage of orders take more than 60 minutes to get delivered from the time the order is placed? (The food has to be prepared and then delivered.) [2 marks]

```
In [160...]  
# Calculate total time = (food preparation time + delivery time)  
order_total_delivery_time = df['food_preparation_time'] + df['delivery_time']
```

```
# Find percentage of orders taking more than 60 minutes
percentage_orders_above_60min = (order_total_delivery_time > 60).mean() * 100

# Result
print(f"Percentage of orders taking more than 60 minutes from the time the order is placed is: {percentage_orders_above_60min}%")
```

Percentage of orders taking more than 60 minutes from the time the order is placed is: 10.54%

#### 🔍 Observation:

Percentage of orders taking more than 60 minutes from the time the order is placed is: 10.54%

**Question 16:** The company wants to analyze the delivery time of the orders on weekdays and weekends. How does the mean delivery time vary during weekdays and weekends? [2 marks]

```
In [161]: mean_delivery_time_by_day = df.groupby('day_of_the_week')[['delivery_time']].mean()
tableit(mean_delivery_time_by_day)

+-----+-----+
| delivery_time | value |
+=====+=====+
| Weekday       | 28.34 |
+-----+-----+
| Weekend        | 22.47 |
+-----+-----+
```

```
In [162]: # ! Printing raw as well (in case above tabular doesn't work i.e missing module)
mean_delivery_time_by_day
```

```
Out[162]: day_of_the_week
Weekday    28.340037
Weekend    22.470022
Name: delivery_time, dtype: float64
```

#### 🔍 Observation:

The mean delivery time on weekdays is 28.34 minutes, whereas on weekends it is 22.47 minutes. This shows that delivery times are generally longer on weekdays compared to weekends, with a difference of approximately 5.87 minutes. This could indicate that there are more delays or possibly higher demand for deliveries during weekdays.

## Conclusion and Recommendations

```
In [163]: # Just an analysis for curiosity of Indian Dishes
indian_orders = (df['cuisine_type'] == 'Indian').sum()
indian_orders_pct = indian_orders / total_orders * 100
format_pct(indian_orders_pct)
```

```
Out[163]: '3.85%'
```

**Question 17:** What are your conclusions from the analysis? What recommendations would you like to share to help improve the business? (You can use cuisine type and feedback ratings to drive your business recommendations.) [6 marks]

## Conclusions:

### Customer Preferences

- Certain cuisines, such as American and Japanese, are highly popular, especially on weekends.
- Vietnamese and Thai cuisines are the least sought after.
- Spanish cuisine is mostly ordered on weekends, with less than 10% contribution on weekdays.

### Order Ratings

- A significant number of orders remain unrated, highlighting a potential gap in customer feedback that could be valuable for understanding satisfaction and improving services
- Customers who rate their orders tend to give higher ratings.
- Ratings are not heavily influenced by the cost or delivery time of an order.
- Delivery time consistency does not strongly impact customer ratings. i.e Whether the delivery time is slightly longer or shorter, customers tend to give similar ratings.

### *Order and Delivery Patterns*

- Most orders are prepared within 30 minutes, with preparation times being fairly consistent across restaurants
- 25% of deliveries are completed in 20 minutes or less, and half are delivered within 25 minutes.
- Despite more orders on weekends, delivery times are shorter compared to weekdays, possibly due to traffic or operational factors.

### Customer Behavior

- The majority of customers are one-time orderers, with only 12.5% placing more than two orders.
- A small segment of repeat customers accounts for the highest order count, with one customer ordering 13 times.
- The customer repeat ratio is relatively low, indicating a potential area for improvement in customer retention and loyalty strategies.

### Cuisines and Spending

- Southern dishes cater to both low-spenders and high-spenders, suggesting a diverse menu for varying budgets.
- Korean and Vietnamese dishes are typically more affordable compared to other cuisines.
- Top restaurants specialize in one or two cuisines rather than offering a wide variety.

### *Geographical Insights*

- People in New-York don't like to consume Spanish & Korean dishes during weekdays.

- In New York, Indian dishes appear to be less frequently ordered from restaurants, indicating lower demand compared to other cuisines.
- It can be guessed that New York may experience considerable traffic, as the delivery times remain relatively high even on weekdays despite the lower order volumes.

## **Business Implications**

- The consistent preparation time across orders indicates standardized processes in restaurants.
- A promotional campaign, such as discounts for delayed deliveries, could attract more customers.
- Addressing weekday delivery delays, possibly due to traffic (or other aspects/reasons), could enhance customer satisfaction.

## **Recommendations:**

### **Cuisine Focus**

#### Target Popular Cuisines

- Restaurants should focus on promoting American, Japanese, Italian, and Chinese cuisines, especially during peak weekends when demand is high.
- Given that Spanish cuisine is mostly ordered on weekends, restaurants could consider offering weekend-specific promotions or special menus to boost weekday orders as well.
- Vietnamese and Thai cuisines, being less sought after, might benefit from creative marketing strategies or value additions to increase demand.

### **Delivery Time & Efficiency**

#### Enhance Delivery Speed, esp during weekdays

- Most orders are prepared in a consistent timeframe, so focusing on improving delivery times can significantly enhance customer experience and reduce overall wait time.
- Since most orders are delivered within 20-30 minutes, maintaining or improving this consistent performance is crucial for customer satisfaction.
- Moreover, As quick delivery is increasingly becoming a key factor in customer satisfaction, the company should focus on improving the overall delivery experience to stay competitive
- For weekends, when orders are higher, optimizing staffing and resources will help manage the increased volume while ensuring faster delivery.

### **Operational Adjustments**

#### Weekend Order Management

- Since more orders are placed on weekends, restaurants should consider increasing their staff and resources during this time to handle the higher volume efficiently.

- Offering weekend-specific promotions or discounts could attract more customers and help manage the demand effectively.
- Encourage advance orders and plan for optimized delivery schedules during weekends to maintain consistency and customer satisfaction.

## Customer Retention & Feedback

### Boosting Repeat Customers

- Since most customers place only one order, there is an opportunity to increase customer retention through loyalty programs, exclusive offers, or promotions aimed at encouraging repeat orders.
- Targeting the small group of high-frequency customers with special offers could further strengthen their loyalty and increase order frequency.

### Encouraging Ratings and Feedback

- The presence of many "Not Given" ratings suggests an opportunity to encourage customers to provide feedback. Restaurants could initiate campaigns to request feedback post-purchase, offering incentives such as discounts on future orders
- Since ratings are not heavily influenced by the cost of orders, focusing on non-cost-related aspects like food presentation, freshness, and delivery accuracy could improve customer satisfaction and increase ratings.

## Promotional Campaigns

### Cuisine-Specific Campaigns

- For cuisines with varying demand (such as Spanish), running targeted campaigns during weekdays can help spread demand more evenly across the week. Special discounts or themed dishes could help drive weekday orders
- With Italian and Chinese cuisines following closely behind American and Japanese in popularity, **combo deals** or promotions during peak times could further drive sales and balance demand across different cuisines

## Enhancing Customer Experience

### Small Gestures to Improve Ratings

- Focus on improving aspects of customer experience like food quality, presentation, and delivery speed. Offering small gestures such as complimentary items or personalized notes could enhance the overall experience and potentially improve ratings without impacting average order costs.

⚡ By focusing on these strategic areas, businesses can not only attract more customers but also improve customer satisfaction, retention, and operational efficiency, leading to sustained growth

In [ ]: