Group_project

November 28, 2024

0.1 COMP5721M: Programming for Data Science

0.2 Group project (Coursework 2): Data Analysis Project

1 *Analysis of Indian Online Food Delivery Services*

Give names and emails of group members here:

- Divyanshu Sinha, gkrm7654@leeds.ac.uk
- Venkatraj Venkatesh, gbqh0993@leeds.ac.uk
- Tejaswini Singanamala, bdnh0150@leeds.ac.uk
- Aradhya Khandeparker, lmfx5605@leeds.ac.uk

2 Project Plan

2.1 The Data (10 marks)

In this project, we are working with four datasets from Kaggle competitions and public datasets in the Indian Food Delivery domain. The datasets are as follows: 1. Swiggy Restaurants Dataset of Metro Cities (https://www.kaggle.com/datasets/aniruddhapa/swiggy-restaurants-dataset-of-metro-cities). 2. Restaurants Dataset | Swiggy (https://www.kaggle.com/datasets/ashishjangra27/swiggy-restaurants-dataset). 3. Zomato (https://www.kaggle.com/datasets/rishikeshkonapure/zomato). 4. Online Food Delivery Preferences-Bangalore region (https://www.kaggle.com/datasets/benroshan/online-food-delivery-preferencesbangalore-region/data).

- ***Data Cleaning***

After downloading the datasets, we applied standard cleaning procedures such as:

1. Handling missing values: Columns with missing data were either filled with default values (such as 0 or the column mean) or dropped if the data was irrelevant.
2. Data type correction: Columns were checked and converted to appropriate data types, such as ensuring that date columns were in datetime format and numeric columns were integers or floats.
3. Outlier detection: We examined the data for extreme outliers, and applied techniques like capping or removal if necessary.
4. Normalization and scaling: Some datasets contained numerical values on different scales, so we normalized or scaled them to ensure consistency for later analysis.

- ***Merging Datasets***

The next step involves identifying common columns across the datasets that allow us to merge them. For instance, if one dataset contains Restaurant data with an City and another dataset contains customer demographic information with the same City, this column can serve as a key to merge the two datasets.

- ***Description of the Merged Dataset***

At the end of the merging process, the final dataset contains information about customers, restaurants, order details, and customer feedback. The columns in this dataset include:

1. Order_Id: This column contains a unique Id for every order that a customer has placed on the app. Values are a alphanumeric string.
2. Age : This column contains the age of the customer. Values are numeric ranging from 18 to 60.
3. Gender This column contains the gender of the customer. Values are either Male(M) or Female(F).
4. Marital_Status : This column contains the marital status of the customer. Values are either Single, Married or Prefer not to say.
5. Occupation : This column contains the occupation of the customer. Values are either Student, Employee, Self Employed or House wife.
6. Monthly_Income : This column contains the monthly income of the customer. Values are either in the range No Income, Below Rs.10000, 10001 to 25000, 25001 to 50000 or More than 50000.
7. App_Preference : This column contains data on which app the customer used to order from a restaurant. Values are either Zomato or Swiggy.
8. Restaurant_Name : This column contains the name of the restaurant. Values are a string.
9. City : This column contains the name of the the city where the restaurant is located. Values are a string.
10. Area : This column contains the specific area of the city. Values are a string.
11. Cuisine : This column contains the cuisine that a restaurant serves. Values are a string.
12. Veg/Non-Veg : This column contains the type of food that a restaurant serves. Values are either Veg or Non-Veg.
13. Delivery_Time : This column contains the time it takes to deliver the order to the customer. Values are numeric.
14. Total_Order_Value : This column contains the total amount a customer spends on the order. Values are a string.
15. Meal_Type : This column contains the type of meal a customer orders. Values are either Breakfast, Lunch, Snacks or Dinner.
16. Dish_Liked : This column contains the dish which a customer likes the most. Values are a string.

17. Average_Rating : This column contains Average rating of a restaurant. Values range from 1 to 5.
18. Total_Rating_String : This column contains the total number of ratings for a restaurant. Values are a string.
19. Influence_of_Rating : This column contains whether the rating of a restaurant influence a customer to order from the restaurant. Values are either Yes, No or Maybe.
20. Freshness_of_Food : This column contains the importance of freshness of food. Values are either Important, Very Important, Slightly Important, Moderately Important or Unimportant.
21. Temperature_of_Food : This column contains the importance of temperature of food. Values are either Important, Very Important, Slightly Important, Moderately Important or Unimportant.
22. Taste_of_Food : This column contains the importance of taste of food. Values are either Important, Very Important, Slightly Important, Moderately Important or Unimportant.
23. Quantity_of_Food : This column contains the importance of quantity of food. Values are either Important, Very Important, Slightly Important, Moderately Important or Unimportant.
24. Food_Quality : This column contains the importance of quality of food. Values are either Agree, Strongly Agree, Neutral, Disagree or Strongly Disagree.
25. Poor_Hygiene : This column contains the importance of hygene of restaurant. Values are either Agree, Strongly Agree, Neutral, Disagree or Strongly Disagree.
26. Bad_Past_Experience : This column contains the importance of bad past experience of ordering from the restaurant. Values are either Agree, Strongly Agree, Neutral, Disagree or Strongly Disagree.
27. High_Quality_Of_Package : This column contains the importance of quality of packaging of food provided by the restaurant. Values are either Important, Very Important, Slightly Important, Moderately Important or Unimportant.
28. Late_Delivery : This column contains the importance of timely delivery from the restaurant. Values are either Agree, Strongly Agree, Neutral, Disagree or Strongly Disagree.
29. Long_Delivery_Time : This column contains whether long delivery time causes cancellation of order. Values are either Agree, Strongly Agree, Neutral, Disagree or Strongly Disagree.
30. Delay_of_Delivery_Person_Getting_Assigned : This column contains whether delay of delivery person getting assigned causes cancellation of order. Values are either Agree, Strongly Agree, Neutral, Disagree or Strongly Disagree.
31. Delay_of_Delivery_Person_Picking_Up_Food : This column contains whether delay of delivery person picking up food causes cancellation of order. Values are either Agree, Strongly Agree, Neutral, Disagree or Strongly Disagree
32. Maximum_Wait_Time : This column contains how long a customer can wait for food delivery. Values are either 15 minutes, 30 minutes, 45 minutes, 60 minutes, More than 60 minutes.

33. Less_Delivery_Time : This column contains the importance of less delivery time of deliverign food from the restaurant. Values are either Important, Very Important, Slightly Important, Moderately Important or Unimportant.

34. Wrong_Order_Delivered : This column contains whether wrong order delivered causes cancellation of order. Values are either Agree, Strongly Agree, Neutral, Disagree or Strongly Disagree.

35. Missing_Item : This column contains whether missing item causes cancellation of order. Values are either Agree, Strongly Agree, Neutral, Disagree or Strongly Disagree.

36. Order_Placed_By_Mistake : This column contains whether order placed by mistake causes cancellation of order. Values are either Agree, Strongly Agree, Neutral, Disagree or Strongly Disagree.

- ***Data Quality and Accuracy***

Since the data comes from Kaggle, the quality and accuracy of each dataset can vary. Generally, Kaggle datasets are curated and pre-processed, but it's still important to check the quality of the data post-cleaning. We performed rigorous cleaning steps, ensuring that the columns were consistent, the data types were correct, and any missing or anomalous values were addressed. However, the accuracy of the data ultimately depends on the sources from which it was collected. If the original data collection process had inaccuracies or biases, these would carry over to our dataset. As part of our analysis, we will need to account for this potential uncertainty, particularly when using the data for decision-making.

By carefully merging and cleaning these four datasets, we now have a robust dataset that combines customer, restaurant, and order data. This data allows us to perform detailed analyses, uncover insights, and make data-driven business decisions.

2.2 Project Aim and Objectives (5 marks)

The project focuses on a comprehensive analysis of the Indian food delivery services market, aiming to uncover critical insights into customer behaviour, restaurant performance, and evolving market trends. With the rapid growth of food delivery services, particularly due to technological advancements and the proliferation of smartphones, understanding these dynamics has become increasingly vital for businesses in this sector.

One of the primary objectives of this study is to delve into customer demographics. By analysing variables such as age, income level, and geographic location, the project will identify how different population segments engage with food delivery services. Furthermore, it will explore order patterns and preferences regarding cuisines, highlighting favourites among consumers and the factors that influence their choices. This includes examining feedback and ratings provided by customers, which can reveal the drivers of satisfaction and loyalty in a highly competitive market.

In parallel, the project will evaluate restaurant performance using various metrics such as order volume and revenue generation. This assessment will help identify top-performing establishments and those in need of improvement. Understanding how restaurants respond to customer feedback is crucial for enhancing service quality and overall dining experience.

Additionally, the study will analyse revenue trends, seeking emerging opportunities within the Indian food delivery landscape. A significant focus will be placed on major players in the industry, including Swiggy and Zomato, to understand how they adapt to ever-changing market demands. Insights into pricing strategies, delivery speed, and service quality will provide a clearer picture of consumer preferences and spending habits across different regions. Ultimately, the findings of this project will equip businesses with actionable insights to thrive in a dynamic marketplace.

2.2.1 Specific Objective(s)

- **Objective 1:** *Data Preprocessing and Simulation*

Objective 1 focuses on preparing and simulating data to ensure that the datasets are clean, consistent, and ready for analysis and visualization. This phase is crucial to transform raw data into a structured, usable format, making it suitable for accurate analysis and insights.

- **Objective 2:** *Customer Demographic Analysis of Food Delivery Apps*

Objective 2 aims to identify patterns in customer behavior and how these are shaped by demographic factors. Insights from this analysis will help food delivery platforms tailor their services, marketing strategies, and product offerings to better meet the preferences of diverse customer segments.

- **Objective 3:** *Revenue Analysis of Food Delivery Apps*

Objective 3 helps identify key revenue drivers, uncovering how Swiggy and Zomato can optimize operations, marketing strategies, and partnerships with restaurants to maximize profits. Understanding these dynamics also highlights potential growth opportunities in underserved cities, cuisines, or meal types.

- **Objective 4:** *Customer Feedback Analysis of Food Delivery Apps*

Objective 4 analyzes the performance of top and bottom-rated restaurants across four cities based on hygiene, cancellation, and delivery scores from Swiggy and Zomato. This analysis highlights critical service quality differences, offering insights into areas for improvement and helping platforms enhance customer satisfaction and restaurant partnerships.

2.3 System Design (5 marks)

2.3.1 Architecture

In the system architecture we have 9 steps as described below:

1. Data Collection - The project begins with comprehensive data collection from Kaggle, leveraging four meticulously curated datasets to analyze the Indian Food Delivery Service landscape. These datasets encapsulate a broad spectrum of information including customer preferences, order details, restaurant data, delivery trends, and regional variations. This diverse data provides the foundation for robust and insightful analysis.
2. Data Cleaning - To ensure the integrity and usability of the data, a rigorous data cleaning process is performed. This step includes addressing missing values, correcting inconsistent or erroneous data types, and removing outliers. The goal is to enhance data quality and ensure uniformity across all datasets.

3. Column Selection - Relevant columns are carefully selected from each dataset, aligning with the project's objectives. This targeted approach ensures that the data used contains only the necessary and meaningful information, facilitating efficient analysis.

4. Merge Datasets - Datasets are sequentially merged based on common attributes such as Order_Id, Restaurant_Name, and City. This process enriches the dataset by integrating complementary features from different sources, creating a cohesive and comprehensive dataset.

5. Data Simulation - To address potential data gaps and improve model robustness:

- Synthetic Data Generation is employed, preserving the probability distribution of the original dataset to maintain data integrity.
- Data Augmentation Techniques are applied, enhancing data completeness and ensuring more accurate predictive modeling and analysis.

6. Data Transformation - The dataset undergoes transformation through the application of various aggregation functions:

- sum(), max(), and min() to compute cumulative and extreme values.
- value_counts() for understanding frequency distributions.
- isna() to identify and quantify missing values.

   These transformations are vital for summarizing the dataset, uncovering patterns, and deriving actionable insights.

7. Dataset Successfully Prepared - The final merged and cleaned dataset is prepared for analysis. This step ensures the dataset is complete, consistent, and well-structured, serving as a reliable input for subsequent exploration and modeling.

8. Exploratory Data Analysis (EDA) - Exploratory Data Analysis (EDA) is conducted to understand the dataset's characteristics and underlying patterns. This involves:

- Identifying trends, correlations, and anomalies within the data.
- Using statistical methods and visual techniques to uncover insights, laying the groundwork for advanced analysis.

9. Data Visualization - Visualization tools are employed to present key insights gleaned from the data. A variety of graphical representations such as bar charts, pie charts and heatmaps are created. These visualizations enable stakeholders to intuitively understand the findings and make data-driven decisions.

2.3.2 Processing Modules and Algorithms

1. Data Cleaning and Outlier Removal

Objective: Ensure the dataset is free from inconsistencies and irrelevant entries.

Components: * Pandas: Used for detecting and handling missing values (fillna() and dropna()), removing duplicates, and correcting data types.

2. Data Integration and Merging

Objective: Combine multiple datasets to create a unified and enriched dataset.

Components: * Pandas Merge and Join Functions: Datasets are combined using common columns (e.g., Order_Id, Restaurant_Name, City) to ensure cohesion. * Handling Key Conflicts: Checks for duplicate keys and missing data during merging to maintain data integrity.

3. Feature Engineering and Representation

Objective: Transform raw data into meaningful features that are suitable for analysis and modeling.

Components: * One-Hot Encoding (Pandas): Converts categorical data into numerical format for better compatibility with analysis models. * Data Normalization (NumPy): Scales numerical features to a standard range (e.g., 0 to 1) to eliminate bias caused by varying feature magnitudes. * Synthetic Data Generation: Applies probability distribution techniques to create additional samples, preserving original data patterns.

4. Exploratory Data Analysis and Visualization

Objective: Analyze data to uncover patterns, trends, and correlations, and present insights visually.

Components: * Statistical Aggregation (Pandas): Functions like groupby() and value_counts() are used to derive summary statistics and distributions. * Visualization Libraries: * Matplotlib and Seaborn: Used to create static visualizations like bar charts, heatmaps, and scatter plots. * Plotly: Enables the creation of interactive and dynamic visualizations.

3 Program Code (15 marks)

Importing all the relevant libraries required in this project:

[1]:
```
import pandas as pd
import numpy as np
import kagglehub
from collections import Counter
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import seaborn as sns
```

[2]:
```
# Configuring custom notebook settings
pd.set_option('display.max_rows',500)
pd.set_option('display.max_columns',50)
pd.set_option('display.width',1000)
```

Declaring the path of all required datasets from kaggle:

[3]:
```
# Path of all the required datasets
DATASET_FILE_1 = 'aniruddhapa/swiggy-restaurants-dataset-of-metro-cities'
DATASET_FILE_2 = 'ashishjangra27/swiggy-restaurants-dataset'
DATASET_FILE_3 = 'rishikeshkonapure/zomato'
DATASET_FILE_4 = 'benroshan/online-food-delivery-preferencesbangalore-region'
```

The following function downloads datasets from Kaggle, loads them into pandas dataframes, and prints their details for analysis:

[4]:
```
def read_all_files():
    try:
        # Load all four datasets into a dictionary with corresponding file names
        df_datasets = {
            "DATASET_FILE_1": pd.read_csv(kagglehub.
 ↪dataset_download(DATASET_FILE_1)+'/Swiggy_dataset.csv'),
            "DATASET_FILE_2": pd.read_csv(kagglehub.
 ↪dataset_download(DATASET_FILE_2)+'/swiggy.csv').rename(columns={'cuisine':
 ↪'cuisines'}),
            "DATASET_FILE_3": pd.read_csv(kagglehub.
 ↪dataset_download(DATASET_FILE_3)+'/zomato.csv'),
            "DATASET_FILE_4": pd.read_csv(kagglehub.
 ↪dataset_download(DATASET_FILE_4)+'/onlinedeliverydata.csv'),
        }
    except Exception as e:
        # Raise an exception with a detailed error message if any dataset fails␣
 ↪to load
        raise Exception('While loading the datasets -\nError Msg:',e)
    else:
        # Confirm successful loading and display dataset details
        print("All datasets have been loaded successfully!\nDataset Details:")
        # Display size and a sample of 5 rows for each dataset
        for name, df in df_datasets.items():
            print(f"Size of {name} is {df.shape} and sample 5 rows:")
            print(df.sample(5), "\n")
    return df_datasets
```

Objective 1 - Data Preprocessing and Simulation

Defining methods for Data Preprocessing, Cleaning and Merging:

The following function parses and replaces list-like strings (e.g., [‘Indian’, ‘Chinese’]) in a Cuisine column with formatted strings (Indian, Chinese):

[5]:
```
def remove_list_like_structure(cuisine_list,dataFrame):
    # Iterate through each cuisine in the list
    for cuisine in cuisine_list:
        # Check if the cuisine entry appears to be a list-like structure␣
 ↪(contains square brackets)
        if '[' in cuisine:
            # Remove the surrounding brackets from the string
            temp_cuisine = cuisine[1:-1]
            temp_cuisine_list = [] # Initialize a list to store parsed values
            quote_idx = [-1, -1]  # To store the indices of single quotes
            count = 0  # Counter to keep track of quote pairs

            # Iterate through each item in the stripped string
            for i in range(len(temp_cuisine)):
                # Check for single quotes and capture their indices
                if temp_cuisine[i] == "'" and count <2:
                    quote_idx[count] = i
                    count += 1

                # If two quotes are found, extract the substring and reset the␣
↪counter
                if count > 1:
                    # Extract the substring between the found quote indices
                    temp_cuisine_list.append(temp_cuisine[quote_idx[0] + 1 :␣
↪quote_idx[1]])
                    count = 0  # Reset the count for the next substring

            # Replace the original list-like structure in the dataframe with␣
↪the parsed values
            dataFrame.replace(cuisine,', '.join(temp_cuisine_list),inplace=True)
```

The follwing function counts the occurrences of individual cuisines from a list of comma-separated strings, producing a dictionary of cuisine-frequency pairs:

[6]:
```
def count_occurrence_of_each_cuisine(cuisine_list):
    # Split each cuisine string by commas, trim whitespace, and flatten the␣
↪resulting list
    cuisines = [cuisine.strip() for cuisines in cuisine_list for cuisine in␣
↪cuisines.split(',')]

    # Count occurrences of each cuisine using the Counter class and return as a␣
↪dictionary
    return dict(Counter(cuisines))
```

The following function replaces the cuisine value with the most frequently ordered cuisine for each row in the dataframe, based on occurrence counts:

[7]:
```
def replace_cuisine_value_with_most_ordered(cuisine_list,␣
↪cuisine_occurrence_count, cuisine_col_name, id_col_name):
    # Dictionary to store the mapping of ID to the most ordered cuisine
    modified_cuisine_col_value = {}
    # Iterate through each row of the dataframe
    for _, row in cuisine_list.iterrows():
        # Extract and sort cuisine types from the specified column, removing␣
↪extra spaces
        cuisine_types = [c.strip() for c in sorted(row[cuisine_col_name].
↪split(","))]

        # Determine the most ordered cuisine based on the occurrence count␣
↪dictionary
```

```
        most_ordered_cuisine = max(cuisine_types, key=lambda x:␣
↪cuisine_occurrence_count.get(x, 0))

        # Map the row's ID to the most ordered cuisine
        modified_cuisine_col_value[row[id_col_name]] = most_ordered_cuisine

    # Return the mapping of IDs to their most ordered cuisine
    return modified_cuisine_col_value
```

The following function splits each City column entry into its components based on specific delimiters and returns a dataframe containing the original city entry, area, and the city name:

[8]:
```
def split_city_area(city_entry):
    # Strip any leading or trailing whitespace from the city entry
    city_entry_org = city_entry.strip()

    # Loop through the list of possible separators
    for sep in [',', '&', '/']:
        # Split the city entry using the current separator
        parts = city_entry_org.split(sep)

        # If the split results in exactly two parts, process the entry
        if len(parts) == 2:
            # Create a DataFrame with the extracted information
            return pd.DataFrame({
                'city': [city_entry],           # Original city entry
                'area': [parts[0].strip()],     # First part (area)
                'new_city': [parts[1].strip()]  # Second part (city name)
            })
```

The following function takes a list of rating strings (e.g., “4.5/5”) and converts them into a dictionary where the key is the original rating and the value is the numerical part before the / :

[9]:
```
def modify_rating_value(col_list):
    # Create a dictionary comprehension:
    # For each rating in the input list, split by '/' and take the first part
    return {rate: rate.split('/')[0] for rate in col_list}
```

The following function replaces NaN values in a specified column of a dataframe with random values sampled from the existing data in that column, using their observed frequencies as probabilities:

[10]:
```
def replace_nan_values(dataFrame, column_name):
    # Calculate the normalized value counts (frequencies) of non-NaN entries in␣
↪the column
    value_counts = column_name.dropna().value_counts(normalize = True)

    # Replace NaN values with random choices based on the observed value␣
↪frequencies
```

```
    return column_name.apply(lambda x: x if pd.notna(x) else np.random.
↪choice(value_counts.index, p=value_counts.values))
```

The following function processes a cleaned dataset and handle missing values while preserving the original distribution of data in the columns:

```
[11]: def replace_missing_values(dataset_after_preprocess_cleaned):
          try:
              # Display the count of missing values in each column
              print("Count of Missing values\n",dataset_after_preprocess_cleaned.
↪isna().sum(),'\n')

              # Iterate over columns to replace missing values, excluding 'Order_Id'
              for col in dataset_after_preprocess_cleaned.columns:
                  if col not in ['Order_Id', 'App_Preference']:
                      # Replace NaN values in the column while maintaining original␣
↪data distribution
                      dataset_after_preprocess_cleaned[col] =␣
↪replace_nan_values(dataset_after_preprocess_cleaned,dataset_after_preprocess_cleaned[col])
                  elif col in ['App_Preference']:
                      value_counts = dataset_after_preprocess_cleaned[col].
↪value_counts(normalize = True)
                      dataset_after_preprocess_cleaned[col] =␣
↪dataset_after_preprocess_cleaned[col].apply(lambda x: x if pd.isna(x) else␣
↪np.random.choice(value_counts.index, p=value_counts.values))

              # Replace boolean values in 'Veg/Non-Veg' column with 'Veg' or 'Non-Veg'
              dataset_after_preprocess_cleaned['Veg/Non-Veg'] =␣
↪dataset_after_preprocess_cleaned['Veg/Non-Veg'].replace({True: 'Veg', False:␣
↪"Non-Veg"})
          except Exception as e:
              # Raise an exception if any error occurs during processing
              raise Exception("While replacing the missing values -\nError Msg:", e)
          else:
              # Confirm successful processing and display a sample of the cleaned␣
↪dataset
              print("Missing values has been successfully replaced maintaining␣
↪original distribution percentage. Have a look at final dataset for␣
↪visualization!\n")
              print(f"Size of final dataset file is {dataset_after_preprocess_cleaned.
↪shape} and sample 5 rows:\n")
              print(dataset_after_preprocess_cleaned.sample(5), "\n")

          # Return the cleaned dataset with columns in proper order
          return dataset_after_preprocess_cleaned[['Order_Id', 'Age', 'Gender',␣
↪'Marital_Status', 'Occupation', 'Monthly_Income',␣
↪'App_Preference',        'Restaurant_Name',
        'City',        'Area',        'Cuisine', 'Veg/Non-Veg',␣
↪'Delivery_Time', 'Total_Order_Value', 'Meal_Type', 'Dish_Liked',␣
↪'Average_Rating', 'Total_Rating_String', 'Influence_of_Rating',
        'Freshness_of_Food', 'Temperature_of_Food', 'Taste_of_Food',␣
↪'Quantity_of_Food', 'Food_Quality', 'Poor_Hygiene', 'Bad_Past_Experience',
        'High_Quality_of_Package', 'Late_Delivery', 'Long_Delivery_Time',␣
↪'Delay_of_Delivery_Person_Getting_Assigned',␣
↪'Delay_of_Delivery_Person_Picking_Up_Food',
        'Maximum_Wait_Time', 'Less_Delivery_Time', 'Wrong_Order_Delivered',␣
↪'Missing_Item', 'Order_Placed_by_Mistake']]
```

The following function merges multiple datasets with different structures into a unified format. It maps specific columns from each dataset to a standardized schema and then processes the merged data:

```
[12]: def merge_all_dataset(df_datasets_map):
          try:
              # Initialize a list to hold standardized dataframes
              final_dataset_list = []

              # Iterate over the datasets provided in the dictionary
              for name,df in df_datasets_map.items():
                  # Temporary dataframe for mapping columns
                  df_temp = pd.DataFrame()

                  # Column mapping logic based on dataset name
                  if name == 'DATASET_FILE_1':
                      current_dataset_cols =␣
↪['uuid','name','city','area','avgRating','totalRatingsString','cuisines','deliveryTime','ve
                      final_dataset_columns =␣
↪['Order_Id','Restaurant_Name','City','Area','Average_Rating','Total_Rating_String','Cuisine
↪Non-Veg','App_Preference']

                  elif name == 'DATASET_FILE_2':
                      current_dataset_cols =␣
↪['id','name','new_city','area','rating','rating_count','cuisines',␣
↪'cost','App_Preference']
                      final_dataset_columns =␣
↪['Order_Id','Restaurant_Name','City','Area','Average_Rating','Total_Rating_String','Cuisine

                  elif name == 'DATASET_FILE_3':
                      current_dataset_cols =␣
↪['name','City','location','rate','votes','cuisines','dish_liked','App_Preference']
                      final_dataset_columns =␣
↪['Restaurant_Name','City','Area','Average_Rating','Total_Rating_String','Cuisine',␣
↪'Dish_Liked','App_Preference']
```

```
            elif name == 'DATASET_FILE_4':
                current_dataset_cols = ['Age', 'City', 'Gender', 'Marital␣
↪Status', 'Occupation', 'Monthly Income',  'Medium (P1)', 'Meal(P1)',␣
↪'Influence of rating', 'Freshness ', 'Temperature', 'Good Taste ', 'Good␣
↪Quantity', 'Good Food quality', 'Poor Hygiene', 'Bad past experience', 'High␣
↪Quality of package', 'Late Delivery', 'Long delivery time',  'Delay of␣
↪delivery person getting assigned', 'Delay of delivery person picking up␣
↪food', 'Maximum wait time', 'Less Delivery time', 'Wrong order delivered', ␣
↪'Missing item', 'Order placed by mistake']
                final_dataset_columns = ['Age', 'City', 'Gender',␣
↪'Marital_Status', 'Occupation', 'Monthly_Income',  'App_Preference',␣
↪'Meal_Type', 'Influence_of_Rating', 'Freshness_of_Food',␣
↪'Temperature_of_Food', 'Taste_of_Food', 'Quantity_of_Food', 'Food_Quality',␣
↪'Poor_Hygiene', 'Bad_Past_Experience', 'High_Quality_of_Package',␣
↪'Late_Delivery', 'Long_Delivery_Time',␣
↪'Delay_of_Delivery_Person_Getting_Assigned',␣
↪'Delay_of_Delivery_Person_Picking_Up_Food', 'Maximum_Wait_Time',␣
↪'Less_Delivery_Time', 'Wrong_Order_Delivered',  'Missing_Item',␣
↪'Order_Placed_by_Mistake']

            # Map the columns from the current dataset to the final␣
↪standardized format
            df_temp[final_dataset_columns] = df[current_dataset_cols].copy()
            # Add the processed dataframe to the final list
            final_dataset_list.append(df_temp)

        # Combine all processed dataframes into one unified dataset
        dataset_after_preprocess_cleaned = pd.concat(final_dataset_list,␣
↪ignore_index=True, sort=False)
        # Remove invalid ratings (e.g., '--' as placeholder for missing ratings)
        dataset_after_preprocess_cleaned =␣
↪dataset_after_preprocess_cleaned[~dataset_after_preprocess_cleaned['Average_Rating'].
↪str.contains('--',na=False)]
        # Generate synthetic 'Age' values using a normal distribution clipped␣
↪to the range 18-60
        dataset_after_preprocess_cleaned['Age'] = np.clip(np.random.normal(30,␣
↪8, len(dataset_after_preprocess_cleaned)), 18, 60).astype(int)
    except Exception as e:
        # Handle and raise exceptions with detailed error messages
        print(e)
        raise Exception("While merging the dataset -\nError Msg:", e)
    else:
        # Confirm successful merging and proceed to replace missing value
        print("All the datasets has been successfully Merged. Moving on to␣
↪replace the missing values if any...\n")
```

```
        dataset_after_preprocess_cleaned =␣
↪replace_missing_values(dataset_after_preprocess_cleaned)

    # Return the cleaned and merged dataset
    return dataset_after_preprocess_cleaned
```

The following function performs several data preprocessing tasks on multiple datasets, such as cleaning columns, handling missing values, modifying column values, and merging datasets:

```
[13]: def objective_1_preprocess_cleaning(df_datasets_map):
    try:
        # Loop through the datasets in the provided map
        for name, df in df_datasets_map.items():
            # Processing for DATASET_FILE_1
            if name == 'DATASET_FILE_1':
                # Add 'App_Preference' column with default value 'Swiggy'
                df = df.assign(App_Preference='Swiggy')
                # Remove list-like structures in the 'cuisines' column
                remove_list_like_structure(df['cuisines'],df)
                # Replace the cuisines values with the most ordered ones
                modified_cuisine_col_value =␣
↪replace_cuisine_value_with_most_ordered(df[['id','cuisines']],count_occurrence_of_each_cuis
                # Update 'cuisines' column with the modified values
                df['cuisines'] = df['id'].map(modified_cuisine_col_value).
↪fillna(df['cuisines'])

            # Processing for DATASET_FILE_2
            elif name == 'DATASET_FILE_2':
                # Drop rows with missing values
                df.dropna(inplace=True)
                # Add 'App_Preference' column with default value 'Swiggy'
                df = df.assign(App_Preference='Swiggy')
                # Split the 'city' column into 'city', 'area', and 'new_city'␣
↪and remove duplicates
                df_split_city_area = pd.concat([split_city_area(city) for city␣
↪in df['city']], ignore_index=True)
                df_split_city_area.drop_duplicates(inplace=True)
                # Merge the split city data back into the main dataframe
                df = df.merge(df_split_city_area[['city', 'area', 'new_city']])
                # Replace cuisines with most ordered ones
                modified_cuisine_col_value =␣
↪replace_cuisine_value_with_most_ordered(df[['id','cuisines']],count_occurrence_of_each_cuis
                df['cuisines'] = df['id'].map(modified_cuisine_col_value).
↪fillna(df['cuisines'])

            # Processing for DATASET_FILE_3
            elif name == 'DATASET_FILE_3':
                # Drop rows with missing values
                df.dropna(inplace=True)
                # Add 'City' column with default value 'Bangalore', create new␣
↪'id' and new 'App_Preference' columns
                df = df.assign(City='Bangalore', id=range(1, len(df) + 1),␣
↪App_Preference='Zomato')
                # Replace ratings with modified values
                df['rate'] = df['rate'].replace(modify_rating_value(df['rate']))
                # Modify 'dish_liked' column
                modified_dish_liked_col_value =␣
↪replace_cuisine_value_with_most_ordered(df[['id','dish_liked']],count_occurrence_of_each_cu
                df['dish_liked'] = df['id'].map(modified_dish_liked_col_value).
↪fillna(df['dish_liked'])
                # Replace cuisines with most ordered ones
                modified_cuisine_col_value =␣
↪replace_cuisine_value_with_most_ordered(df[['id','cuisines']],count_occurrence_of_each_cuis
                df['cuisines'] = df['id'].map(modified_cuisine_col_value).
↪fillna(df['cuisines'])

            # Processing for DATASET_FILE_4
            elif name == 'DATASET_FILE_4':
                # Update values in App_Preference to Zomato and add 'City'␣
↪column with default value 'Bangalore'
                df['Medium (P1)'] = 'Zomato'
                df = df.assign(City='Bangalore')

            # Update the dataset map with cleaned DataFrame
            df_datasets_map[name] = df

    except Exception as e:
        # Raise exception with the error message if something goes wrong during␣
↪cleaning and preprocessing
        raise Exception("While cleaning and pre-processing the dataset -\nError␣
↪Msg:",e)
    else:
        # Print message indicating successful cleaning and preprocessing
        print("All the datasets has been successfully cleaned and pre-processed.
↪ Moving on to merger part...\n")

        # Call the merge function to merge the datasets
        dataset_after_preprocess_cleaned = merge_all_dataset(df_datasets_map)

    # Return the merged and preprocessed dataset
    return dataset_after_preprocess_cleaned
```

The following function orchestrates the entire process of loading datasets, preprocessing them, and handling exceptions:

[14]:
```
def objective_1():
    try:
        # Step 1: Read all dataset files into memory using the 'read_all_files'␣
↪function
        df_datasets_map = read_all_files()

        # Step 2: Preprocess and clean the datasets using the␣
↪'objective_1_preprocess_cleaning' function
        dataset_after_preprocess_cleaned =␣
↪objective_1_preprocess_cleaning(df_datasets_map)

    except Exception as e:
        # Raise exception with the error message if something goes wrong
        print('Oops! Something went wrong', e)

    else:
        # Print a success message
        print('Woohoo!!! Program executed successfully.')

    return dataset_after_preprocess_cleaned
```

Objective 2 - Customer Demographic Analysis of Food Delivery Apps

The following function plots a heatmap using seaborn:

[15]:
```
def plot_heatmap(filtered_pivot_no_area, title, ylable, xlable, app_preference,␣
↪color):
    # Set the figure size for the heatmap
    plt.figure(figsize=(14, 8))

    # Create the heatmap with specified data, annotations, color map, and␣
↪formatting
    sns.heatmap(
        filtered_pivot_no_area[app_preference].unstack(level=0),  # Unstacking␣
↪to reshape data for the heatmap
        annot=True, fmt=".0f", cmap=color, linewidths=.5, cbar=True  # Adding␣
↪values, color map, and color bar
    )

    # Set the title and axis labels with specified fonts
    plt.title(title, fontsize=16)
    plt.ylabel(ylable, fontsize=12)
    plt.xlabel(xlable, fontsize=12)

    # Rotate x-axis tick labels for better readability
    plt.xticks(rotation=45, ha='right')

    # Adjust layout for better fitting
    plt.tight_layout()

    # Display the heatmap
    plt.show()

    # Add a blank figure to create spacing after the heatmap
    plt.figure(figsize=(14, 0.5))  # Setting height for spacing
    plt.tight_layout()  # Adjust layout for the blank figure
```

The following class contains two methods to analyze food delivery preferences. The first function visualizes app preferences based on demographic factors, while the second function analyzes city-wise app usage using heatmaps:

[16]:
```
class Objective_2_solution:

    def solution_question_1(self, df_food_delivery):
        try:
            # Define bins for age groups and their corresponding labels
            age_bins = [18, 21, 31, 41, 51, 61]
            age_labels = ['18-20', '21-30', '31-40', '41-50', '51-60']

            # Bin age data into specified groups
            df_food_delivery['Age_Group'] = pd.cut(df_food_delivery['Age'],␣
↪bins=age_bins, labels=age_labels, right=False)

            # Group data by Age_Group, Gender, Marital_Status, and␣
↪App_Preference
            grouped_data = df_food_delivery.groupby(['Age_Group', 'Gender',␣
↪'Marital_Status', 'App_Preference'], observed=False).size().
↪unstack(fill_value=0)

            # Calculate percentages within groups
            grouped_percentages = grouped_data.div(grouped_data.sum(axis=1),␣
↪axis=0) * 100

            # Reset index for easier plotting
            grouped_percentages = grouped_percentages.reset_index()
            grouped_data = grouped_data.reset_index()

            # Define a red-orange color palette for the App_Preference
            colors = sns.color_palette(["#FF5F3D", "#A52A2A"])

            # Create the plot with defined size and layout
            fig, ax = plt.subplots(figsize=(16,25))

            # Calculate x-axis positions with space between groups
            group_gap = 2 # Space between groups
            x_positions = np.arange(len(grouped_percentages['Age_Group'].
↪unique())) * (1 + group_gap)

            # Initialize bar width and position offset
            bar_width = 0.4  # Width of each bar group
            offset = 0

            # Iterate through Gender and Marital_Status combinations to plot␣
↪bars
            unique_combinations = grouped_data[['Gender', 'Marital_Status']].
↪drop_duplicates()
            for idx, (gender, marital_status) in enumerate(unique_combinations.
↪values):
                # Filter data for the specific combination
                subset = grouped_data[(grouped_data['Gender'] == gender) &
                                      (grouped_data['Marital_Status'] == marital_status)]
                subset_percentages =␣
↪grouped_percentages[(grouped_percentages['Gender'] == gender) &
                                                                   ␣
↪(grouped_percentages['Marital_Status'] == marital_status)]
                # Plot bars for each app preference
                for col_idx, order_pref in enumerate(grouped_data.columns[3:]):␣
↪ # Skip demographic columns
                    heights = subset[order_pref]
                    percentages = subset_percentages[order_pref]
                    bars = ax.bar(x_positions + offset, heights, bar_width,
                                  bottom=subset[grouped_data.columns[3:]].iloc[:,␣
↪:col_idx].sum(axis=1), color=colors[col_idx], edgecolor='black')

                    # Annotate bars with percentages
                    for i, (height, pct) in enumerate(zip(heights,␣
↪percentages)):
                        if height > 0:
                            ax.text(x_positions[i] + offset,␣
↪subset[grouped_data.columns[3:]].iloc[i, :col_idx].sum() + height / 2,
                                    f'{pct:.1f}%', ha='center', va='center',␣
↪fontsize=8, color='white')

                # Add legend above each group
                text_offset = 30 # Space above the bars for the text
                for i, x_pos in enumerate(x_positions):
                    ax.text(x_pos + offset, subset[grouped_data.columns[3:]].
↪iloc[i].sum() + text_offset,
                        f'{gender}-{marital_status}', ha='center', va='bottom',␣
↪fontsize=10, fontweight='bold', rotation=90)
                offset += bar_width  # Increment offset for next group
            # Add chart title and axis labels
            ax.set_title('Influence of Age Groups, Gender, and Marital Status␣
↪on App Preference', fontsize=14)
            ax.set_xlabel('Age Group', fontsize=12)
            ax.set_ylabel('Number of Orders', fontsize=12)

            # Set x-axis ticks and labels
            ax.set_xticks(x_positions + (bar_width * (len(unique_combinations)␣
↪- 1)) / 2)
            ax.set_xticklabels(grouped_data['Age_Group'].unique())

            # Add legend for app preferences
            labels_legend = ['Swiggy','Zomato']
            ax.legend(title="App_Preference", bbox_to_anchor=(0.8, 0.95),␣
↪loc='upper left',labels=labels_legend)
            plt.tight_layout()

        except Exception as e:
            # Raise exception with the error message if something goes wrong
            raise Exception("While solving Objective 2 Question 1 -\nError Msg:
↪", e)

        else:
            # Print a success message
            print("Result fetched successfully...")
            plt.show()

    def solution_question_2(self, df_food_delivery):
        try:
            # Filter necessary columns
            detailed_data = df_food_delivery[['City', 'Age', 'Monthly_Income',␣
↪'Marital_Status', 'Occupation', 'App_Preference']]

            # Bin ages into three broader groups
            detailed_data = detailed_data.copy()  # Create a full copy of the␣
↪DataFrame
            detailed_data.loc[:, 'Age_Group'] = pd.cut(detailed_data['Age'],␣
↪bins=[0, 25, 40, 60], labels=['<25', '25-40', '40-60'])

            # Filter data for the top 3 cities and top 2 income groups ("More␣
↪than 50000", "25001 to 50000")
            filtered_data = detailed_data[
            #(detailed_data['Monthly_Income'].isin(['More than 50000', '25001␣
↪to 50000'])) &
```

```
            (detailed_data['City'].isin(detailed_data['City'].value_counts().
↪head(10).index))
            ]

            # Group data for analysis
            filtered_grouped_data_no_area = filtered_data.groupby(
            ['City', 'Age_Group', 'Monthly_Income', 'App_Preference'],␣
↪observed=False
            ).size().reset_index(name='Count')

            # Pivot data for heatmap plotting
            filtered_pivot_no_area = filtered_grouped_data_no_area.pivot_table(
            index=['City', 'Age_Group', 'Monthly_Income'],
            columns='App_Preference',
            values='Count',
            fill_value=0,
            observed=False
            )

        except Exception as e:
            # Raise exception with the error message if something goes wrong
            raise Exception("While solving Objective 2 Question 2 -\nError Msg:
↪", e)

        else:
            # Print a success message
            print("Result fetched successfully...")

            # Generate heatmaps for Swiggy and Zomato preferences
            plot_heatmap(filtered_pivot_no_area,"Swiggy Preferences by City,␣
↪Age Group, and Income","City, Age Group, Income","Swiggy␣
↪Preferences",'Swiggy','Blues')
            plot_heatmap(filtered_pivot_no_area,"Zomato Preferences by City,␣
↪Age Group, and Income","City, Age Group, Income","Zomato␣
↪Preferences",'Zomato','Reds')
```

The following function visualizes the question - How do age groups, gender, and marital status influence customers' order preferences between Swiggy and Zomato?:

```
[17]: def objective_2_Q1(dataset_after_preprocess_cleaned):
          try:
              # Create an instance of the Objective_2_solution class
              obj2 = Objective_2_solution()

              # Call the solution_question_1 method, passing the preprocessed dataset
              obj2.solution_question_1(dataset_after_preprocess_cleaned)
```

```
    except Exception as e:
        # Raise exception with the error message if something goes wrong
        print('Oops! Something went wrong', e)

    else:
        # Print a success message
        print('Woohoo!!! Program executed successfully.')
```

The following function visualizes the question - How do customer app preferences (Swiggy vs. Zomato) vary across the top 10 cities, segmented by income groups and age ranges (<25, 25–40, 40–60, 60+)?:

```
[18]: def objective_2_Q2(dataset_after_preprocess_cleaned):
          try:
              # Create an instance of the Objective_2_solution class
              obj2 = Objective_2_solution()

              # Call the solution_question_2 method, passing the preprocessed dataset
              obj2.solution_question_2(dataset_after_preprocess_cleaned)

          except Exception as e:
              # Raise exception with the error message if something goes wrong
              print('Oops! Something went wrong', e)

          else:
              # Print a success message
              print('Woohoo!!! Program executed successfully.')
```

Objective 3 - Revenue Analysis of Food Delivery Apps

The following class is designed to process food delivery data and generate visualizations that compare revenue and preferences for Swiggy and Zomato:

```
[19]: class Objective_3_solution:

          # Function to process data for a given order preference
          def process_order_preference(self,filtered_data, order_pref, color):
              # Filter data for the specified order preference
              data_pref = filtered_data[filtered_data['App_Preference'] == order_pref]

              # Calculate city revenue breakdown and determine the top 3 cities
              city_breakdown = data_pref.groupby('City')['Total_Order_Value'].sum().
        ↪sort_values(ascending=False)
              top3_cities = city_breakdown[:3]
              others = city_breakdown[3:].sum() # Combine remaining cities into␣
        ↪'Others'
              city_breakdown_final = pd.concat([top3_cities, pd.Series(others,␣
        ↪index=['Others'])])
              city_percentage = (city_breakdown_final / city_breakdown_final.sum()) *␣
        ↪100

              # Determine the most popular city for this preference
              most_popular_city = city_breakdown.idxmax()
              most_popular_city_data = data_pref[data_pref['City'] ==␣
        ↪most_popular_city]

              # Breakdown of meal type revenue and calculate percentages
              meal_type_revenue = most_popular_city_data.
        ↪groupby('Meal_Type')['Total_Order_Value'].sum()
              meal_type_percentage = (meal_type_revenue / meal_type_revenue.sum()) *␣
        ↪100

              # Calculate top 2 cuisines by revenue for each meal type
              meal_cuisine_revenue = most_popular_city_data.groupby(['Meal_Type',␣
        ↪'Cuisine'], observed=False)['Total_Order_Value'].sum().reset_index()
              top_cuisines_by_meal_percentage = (
                  meal_cuisine_revenue.groupby('Meal_Type', observed=False)
                  .apply(lambda x: x.nlargest(2, 'Total_Order_Value')) # Select top 2␣
        ↪cuisines per meal type
                  .reset_index(drop=True)
              )

              # Create a pivot table for cuisine percentage by meal type
              pivot_data = top_cuisines_by_meal_percentage.pivot_table(
                  index='Meal_Type', columns='Cuisine', values='Total_Order_Value',␣
        ↪aggfunc='sum', fill_value=0
              )
              pivot_percentage = pivot_data.div(pivot_data.sum(axis=1), axis=0) * 100

              # Return the computed percentages and associated metadata
              return city_percentage, meal_type_percentage, pivot_percentage, color,␣
        ↪order_pref


          # Function to plot the charts for Swiggy and Zomato
          def plot_order_pref(self,ax_city, ax_meal, ax_cuisine, results, order_pref):
              # Unpack results
              city_percentage, meal_type_percentage, pivot_percentage, _, _ = results

              # Plot city revenue breakdown as a stacked bar chart
              bottom_city = 1
              width = 0.2
              colors = ["#FFC300", "#28A745", "#DC3545", "#6F42C1"] # Define color␣
        ↪palette
```

```
        for i, (city, percentage) in enumerate(city_percentage.items()):
            bottom_city -= percentage / 100
            bar = ax_city.bar(0, percentage / 100, width, bottom=bottom_city,␣
↪label=city, color=colors[i % len(colors)])
            ax_city.bar_label(bar, labels=[f"{percentage:.1f}%"],␣
↪label_type='center', fontsize=8)
        ax_city.set_title(f"City Breakdown: {order_pref}")
        ax_city.legend(loc='upper left', bbox_to_anchor=(1, 1))
        ax_city.axis('off')

        # Plot meal type revenue breakdown
        bottom_meal = 1
        colors = ["#F28500", "#2E8B57", "#FF6347", "#4B0082"] # Define color␣
↪palette
        for i, (meal_type, percentage) in enumerate(meal_type_percentage.
↪items()):
            bottom_meal -= percentage / 100
            bar = ax_meal.bar(0, percentage / 100, width, bottom=bottom_meal,␣
↪label=meal_type, color=colors[i % len(colors)])
            ax_meal.bar_label(bar, labels=[f"{percentage:.1f}%"],␣
↪label_type='center', fontsize=8)
        ax_meal.set_title(f"Meal Breakdown: Bangalore - {order_pref}")
        ax_meal.legend(loc='upper left', bbox_to_anchor=(1, 1))
        ax_meal.axis('off')

        # Plot top 2 cuisines by meal type as a stacked bar chart
        pivot_percentage.plot(kind='bar', stacked=True, ax=ax_cuisine,␣
↪color=['#7DCFB6', '#FF8A80'])
        ax_cuisine.set_title(f"Top 2 Cuisines by Meal Type: {order_pref}")
        ax_cuisine.set_ylabel('Percentage (%)')
        ax_cuisine.set_xlabel('Meal Type')
        ax_cuisine.legend(title='Cuisine', loc='upper left', bbox_to_anchor=(1,␣
↪1))

        # Add percentage labels to the cuisine chart
        for p in ax_cuisine.patches:
            height = p.get_height()
            if height > 0:  # Only label non-zero patches
                width = p.get_width()
                x = p.get_x() + width / 2
                y = p.get_y() + height / 2
                ax_cuisine.text(x, y, f'{height:.1f}%', ha='center',␣
↪va='center', fontsize=10, color='white')


    def solution(self,df_food_delivery):
        try:
            # Data preprocessing: Calculate revenue percentages by order␣
↪preference and city breakdown
            df_food_delivery['Total_Order_Value'] =␣
↪df_food_delivery['Total_Order_Value'].astype(str).str.replace("  ","")
            df_food_delivery['Total_Order_Value'] = pd.
↪to_numeric(df_food_delivery['Total_Order_Value'], errors='coerce')
            filtered_data = df_food_delivery[['City', 'App_Preference',␣
↪'Total_Order_Value', 'Meal_Type', 'Cuisine']].dropna()


            # Calculate revenue percentages by order preference
            order_pref_revenue = filtered_data.
↪groupby('App_Preference')['Total_Order_Value'].sum()
            order_pref_percentage = (order_pref_revenue / order_pref_revenue.
↪sum()) * 100

            # Process data for Swiggy and Zomato
            swiggy_results = self.process_order_preference(filtered_data,␣
↪'Swiggy', '#FF5F3D')
            zomato_results = self.process_order_preference(filtered_data,␣
↪'Zomato', '#A52A2A')

            # Create the figure and grid layout
            fig = plt.figure(figsize=(24, 18))
            gs = gridspec.GridSpec(3, 4, figure=fig, width_ratios=[0.35, 0.4, 0.
↪4, 0.4], height_ratios=[0.1, 0.03, 0.1])

            # Pie chart for order preference on the extreme left
            ax_pie = fig.add_subplot(gs[:, 0])  # Use all rows in the first␣
↪column
            pie_sizes = order_pref_percentage.values
            order_pref_labels = order_pref_percentage.index
            explode = [0.1 if label in ['Swiggy', 'Zomato'] else 0 for label in␣
↪order_pref_labels]
            colors = ['#FF5F3D' if label == 'Swiggy' else '#A52A2A' if label ==␣
↪'Zomato' else 'gray' for label in order_pref_labels]
            angle = -180 * (order_pref_percentage['Swiggy'] / 100)
            wedges, *_ = ax_pie.pie(
                pie_sizes, labels=order_pref_labels, autopct='%1.1f%%',␣
↪startangle=angle,
                explode=explode, wedgeprops={"edgecolor": "black"},␣
↪colors=colors)
            ax_pie.set_title("Revenue by App Preference")

            # Top row for Swiggy
            ax_city_swiggy = fig.add_subplot(gs[0, 1])
            ax_meal_swiggy = fig.add_subplot(gs[0, 2])
            ax_cuisine_swiggy = fig.add_subplot(gs[0, 3])
            self.plot_order_pref(ax_city_swiggy, ax_meal_swiggy,␣
↪ax_cuisine_swiggy, swiggy_results, "Swiggy")

            # Bottom row for Zomato
            ax_city_zomato = fig.add_subplot(gs[2, 1])
            ax_meal_zomato = fig.add_subplot(gs[2, 2])
            ax_cuisine_zomato = fig.add_subplot(gs[2, 3])
            self.plot_order_pref(ax_city_zomato, ax_meal_zomato,␣
↪ax_cuisine_zomato, zomato_results, "Zomato")

            # Remove unused middle row to align pie chart with Swiggy and␣
↪Zomato rows
            fig.delaxes(fig.add_subplot(gs[1, 1]))
            fig.delaxes(fig.add_subplot(gs[1, 2]))
            fig.delaxes(fig.add_subplot(gs[1, 3]))

            # Display the plot
            plt.tight_layout()

        except Exception as e:
            # Raise exception with the error message if something goes wrong
            raise Exception("While solving Objective 3 Question -\nError Msg:",␣
↪e)

        else:
            # Print a success message
            print("Result fetched successfully...")
            plt.show()
```

The following function visualizes the question - How do Swiggy and Zomato compare in terms of user preferences across cities, meal types, and popular cuisines, and how does this influence revenue distribution?:

[20]:
```
def objective_3(dataset_after_preprocess_cleaned):
    try:
        # Create an instance of the Objective_3_solution class
        obj3 = Objective_3_solution()

        # Call the solution method, passing the preprocessed dataset
        obj3.solution(dataset_after_preprocess_cleaned)

    except Exception as e:
        # Raise exception with the error message if something goes wrong
        print('Oops! Something went wrong', e)
    else:
        # Print a success message
        print('Woohoo!!! Program executed successfully.')
```

Objective 4 - Customer Feedback Analysis of Food Delivery Apps

The following class is designed to process food delivery data and generate visualizations that compare customer feedback preferences for Swiggy and Zomato:

```
[21]: class Objective_4_solution:
        def find_top_bottom_restaurants(self, df_food_delivery):

          cities = ['Bangalore', 'Chennai', 'Mumbai', 'Kolkata']

          # Filter the DataFrame to include only rows where the 'City' column matches␣
      ↪one of the cities in your list
          data_filtered = df_food_delivery[df_food_delivery['City'].isin(cities)]

          # Group data by City, Restaurant_Name, and App_Preferred and then pivot the␣
      ↪App_Preferred to create columns for each app
          app_data = data_filtered.groupby(['City', 'Restaurant_Name',␣
      ↪'App_Preference']).size().unstack(fill_value=0)

          # Identify valid restaurants present on both Swiggy and Zomato
          valid_restaurants = app_data[(app_data['Swiggy'] > 0) & (app_data['Zomato']␣
      ↪> 0)].index

          # Filter data to include only those restaurants found on both platforms
          valid_data = data_filtered.set_index(['City', 'Restaurant_Name']).
      ↪loc[valid_restaurants].reset_index()

          # Group the data by city and count the occurrences of each restaurant
          city_restaurant_counts = valid_data.groupby('City')['Restaurant_Name'].
      ↪value_counts()

          # Loop through each city and select the top and bottom restaurants
          # Iterate through cities and their corresponding restaurant data
          top_restaurant = [
              (city, restaurants.idxmax()[1])
              for city, restaurants in city_restaurant_counts.groupby(level=0)]
          # Loop through each city in the grouped data
          # Filter restaurants and extract the bottom restaurant for each city
          bottom_restaurant = [
              (city, restaurants[(restaurants >= 3) & (restaurants <= 6)].idxmin()[1])
              for city, restaurants in city_restaurant_counts.groupby(level=0)
              if not restaurants[(restaurants >= 3) & (restaurants <= 6)].empty
          ]
```

```
    return [top_restaurant, bottom_restaurant]

  # Function to extract data for top and bottom restaurants and include␣
↪app_preference and scores
  def get_restaurant_data(self, restaurant_list, data_source, restaurant_type):
      output_data = []

      for city, restaurant_name in restaurant_list:
          # Filter the valid_data for the specific restaurant and city
          restaurant_data = data_source[(data_source['City'] == city) &␣
↪(data_source['Restaurant_Name'] == restaurant_name)]

          # If data is found, add it to the output list
          if not restaurant_data.empty:
              # Assuming the app_preference is a column in your data
              app_preference = restaurant_data['App_Preference'].values  # Get␣
↪the app preference (Swiggy/Zomato)

              # Collect the scores
              hygiene_score = restaurant_data['Food_Hygiene_Score'].values[0]
              cancellation_score = restaurant_data['Cancellation_Score'].
↪values[0]
              delivery_score = restaurant_data['Delivery_Score'].values[0]

              output_data.append({
                  'City': city,
                  'Restaurant Name': restaurant_name,
                  'Type': restaurant_type,  # Add the restaurant type (Top or␣
↪Bottom)
                  'App Preference': app_preference,
                  'Hygiene Score': hygiene_score,
                  'Cancellation Score': cancellation_score,
                  'Delivery Score': delivery_score
              })
      return output_data

  def plot_graph(self, final_result):
    # Cities in the data
    cities = final_result['City'].unique()

    # Define custom colors for Swiggy and Zomato (using seaborn color palette)
    colors = sns.color_palette(["#FF5F3D", "#A52A2A"])  # Red and Brown shades␣
↪for Swiggy and Zomato

    # Plot setup
    fig, axes = plt.subplots(2, 2, figsize=(22, 14))
    axes = axes.flatten()  # Flatten axes for easier indexing
    fig.suptitle('Top and Bottom Rated Restaurants by City by App Preference',␣
↪fontsize=16)

    # Loop through each city to plot its data
    for i, city in enumerate(cities):
        ax = axes[i]

        # Filter data for the city
        city_data = final_result[final_result['City'] == city]

        # Get scores for top and bottom restaurants
        top_data = city_data[city_data['Type'] == 'Top']
        bottom_data = city_data[city_data['Type'] == 'Bottom']

        # Prepare data for stacked bar plot
        x_labels = [
            f"{top_data['Restaurant Name'].values[0]} (Top)",
            f"{bottom_data['Restaurant Name'].values[0]} (Bottom)"
        ]
        x = np.arange(len(x_labels))  # Create positions for the bars

        # Scores from Swiggy and Zomato for Hygiene, Cancellation, and Delivery
        hygiene_scores = [
            [top_data['Swiggy Hygiene'].values[0], bottom_data['Swiggy␣
↪Hygiene'].values[0]],
            [top_data['Zomato Hygiene'].values[0], bottom_data['Zomato␣
↪Hygiene'].values[0]]
        ]
        cancellation_scores = [
            [top_data['Swiggy Cancellation'].values[0], bottom_data['Swiggy␣
↪Cancellation'].values[0]],
            [top_data['Zomato Cancellation'].values[0], bottom_data['Zomato␣
↪Cancellation'].values[0]]
        ]
        delivery_scores = [
            [top_data['Swiggy Delivery'].values[0], bottom_data['Swiggy␣
↪Delivery'].values[0]],
            [top_data['Zomato Delivery'].values[0], bottom_data['Zomato␣
↪Delivery'].values[0]]
        ]

        # Plot stacked bars with custom colors
        hygiene_bar = ax.bar(
            x - 0.25, hygiene_scores[0], width=0.25, label='Swiggy Hygiene',␣
↪color=colors[0], edgecolor='black'
```

```
        )
        zomato_hygiene_bar = ax.bar(
            x - 0.25, hygiene_scores[1], width=0.25, label='Zomato Hygiene',␣
↪color=colors[1], bottom=hygiene_scores[0], edgecolor='black'
        )

        cancellation_bar = ax.bar(
            x, cancellation_scores[0], width=0.25, label='Swiggy Cancellation',␣
↪color=colors[0], edgecolor='black'
        )
        zomato_cancellation_bar = ax.bar(
            x, cancellation_scores[1], width=0.25, label='Zomato Cancellation',␣
↪color=colors[1], bottom=cancellation_scores[0], edgecolor='black'
        )

        delivery_bar = ax.bar(
            x + 0.25, delivery_scores[0], width=0.25, label='Swiggy Delivery',␣
↪color=colors[0], edgecolor='black'
        )
        zomato_delivery_bar = ax.bar(
            x + 0.25, delivery_scores[1], width=0.25, label='Zomato Delivery',␣
↪color=colors[1], bottom=delivery_scores[0], edgecolor='black'
        )
        # Annotate bars with the actual values inside the blocks
        # For each bar, annotate with its value (Swiggy or Zomato)
        for bar_set, label in zip([hygiene_bar, zomato_hygiene_bar,␣
↪cancellation_bar, zomato_cancellation_bar, delivery_bar,␣
↪zomato_delivery_bar],
                                  ['Swiggy Hygiene', 'Zomato Hygiene', 'Swiggy␣
↪Cancellation', 'Zomato Cancellation', 'Swiggy Delivery', 'Zomato Delivery']):
            for bar in bar_set:
                height = bar.get_height()
                y_position = bar.get_y() + height / 2  # Center of the bar
                value = height  # The actual value to display
                ax.text(bar.get_x() + bar.get_width() / 2, y_position, f'{value:
↪.2f}', ha='center', va='center', color='white', fontsize=10)

        # Add vertical headings for each bar group
        vertical_labels = ['Hygiene', 'Cancellation', 'Delivery']
        offsets = [-0.25, 0, 0.25]  # Position adjustments for each category

        for j, offset in enumerate(offsets):  # Iterate through Hygiene,␣
↪Cancellation, Delivery
            for k, group in enumerate(x):  # Iterate through Top and Bottom␣
↪groups
                # Calculate the maximum height of the bar group for the category
                if j == 0:  # Hygiene
                    max_height = hygiene_scores[0][k] + hygiene_scores[1][k]
                elif j == 1:  # Cancellation
                    max_height = cancellation_scores[0][k] +␣
↪cancellation_scores[1][k]
                elif j == 2:  # Delivery
                    max_height = delivery_scores[0][k] + delivery_scores[1][k]

                # Place the heading just above the maximum height
                ax.text(
                    group + offset,  # Position above the bar group
                    max_height + 0.2,  # Slightly above the tallest bar in the␣
↪group
                    vertical_labels[j],  # Use the correct vertical label
                    rotation=90,
                    ha='center',
                    va='bottom',
                    fontsize=10,
                    color='black'
                )

        # Set x-axis labels and title
        ax.set_xticks(x)
        ax.set_xticklabels(x_labels, rotation=0)  # Set labels with rotation 0
        ax.set_title(city)
        ax.set_ylabel('Score (out of 10)')
        ax.set_ylim(0, 10)

    # Add a simplified legend for the figure (only Swiggy and Zomato)
    handles = [hygiene_bar, zomato_hygiene_bar]  # Swiggy and Zomato bars for␣
↪the first category
    labels = ['Swiggy', 'Zomato']
    fig.legend(handles, labels, loc='upper center', ncol=2, bbox_to_anchor=(0.
↪5, 0.95))

    plt.tight_layout(rect=[0, 0, 1, 0.95])  # Adjust layout for the title and␣
↪legend
    plt.show()

 def solution(self, df_food_delivery):
    try:
      # Define food hygiene and delivery columns
      food_hygiene_columns = ['Poor_Hygiene', 'Food_Quality',␣
↪'Freshness_of_Food', 'Temperature_of_Food', 'Taste_of_Food',␣
↪'Quantity_of_Food']
```

```
    delivery_columns = ['Late_Delivery', 'Long_Delivery_Time',␣
↪'Delay_of_Delivery_Person_Getting_Assigned',␣
↪'Delay_of_Delivery_Person_Picking_Up_Food']
    cancellation_columns = ['Late_Delivery', 'Long_Delivery_Time',␣
↪'Delay_of_Delivery_Person_Getting_Assigned',␣
↪'Delay_of_Delivery_Person_Picking_Up_Food', 'Wrong_Order_Delivered',␣
↪'Missing_Item', 'Order_Placed_by_Mistake']

    # Define mapping for categorical responses if needed
    response_mapping = {
        'Strongly Agree': 5,
        'Agree': 4,
        'Neutral': 3,
        'Disagree': 2,
        'Strongly Disagree': 1,
        "Very Important": 5,
        "Moderately Important": 4,
        "Important": 3,
        "Less Important": 2,
        "Not Important": 1,
        "Slightly Important":3,
        "Unimportant":1
    }
    df_food_delivery[food_hygiene_columns] =␣
↪df_food_delivery[food_hygiene_columns].map(lambda x: response_mapping.get(x,␣
↪x))
    df_food_delivery[delivery_columns] = df_food_delivery[delivery_columns].
↪map(lambda x: response_mapping.get(x, x))
    df_food_delivery[cancellation_columns] =␣
↪df_food_delivery[cancellation_columns].map(lambda x: response_mapping.get(x,␣
↪x))

    # Normalize food hygiene and delivery columns
    df_food_delivery[food_hygiene_columns] =␣
↪df_food_delivery[food_hygiene_columns].apply(pd.to_numeric, errors='coerce')
    df_food_delivery[delivery_columns] = df_food_delivery[delivery_columns].
↪apply(pd.to_numeric, errors='coerce')
    df_food_delivery[cancellation_columns] =␣
↪df_food_delivery[cancellation_columns].apply(pd.to_numeric, errors='coerce')

    # Compute scores
    df_food_delivery['Food_Hygiene_Score'] =␣
↪df_food_delivery[food_hygiene_columns].mean(axis=1)
    df_food_delivery['Delivery_Score'] = df_food_delivery[delivery_columns].
↪mean(axis=1)
```

```
    df_food_delivery['Cancellation_Score'] =␣
↪df_food_delivery[cancellation_columns].mean(axis=1)

    # Assuming 'Top_Restaurant' and 'Bottom_Restaurant' are lists of (city,␣
↪restaurant_name) tuples
    # Get the top restaurant data
    top_restaurant_data = self.get_restaurant_data(self.
↪find_top_bottom_restaurants(df_food_delivery)[0], df_food_delivery, 'Top')

    # Get the bottom restaurant data
    bottom_restaurant_data = self.get_restaurant_data(self.
↪find_top_bottom_restaurants(df_food_delivery)[1], df_food_delivery, 'Bottom')

    # Combine the top and bottom restaurant data into one list
    all_restaurant_data = top_restaurant_data + bottom_restaurant_data

    # Convert the list of dictionaries into a DataFrame
    df = pd.DataFrame(all_restaurant_data)

    # For each restaurant in each city, calculate the percentage distribution␣
↪of app_preference
    # First, we count the occurrences of 'Swiggy' and 'Zomato' for each␣
↪restaurant in the city
    app_preference_distribution = df.explode('App Preference')  # Exploding␣
↪in case of multiple preferences per row

    # Now, calculate the percentage distribution of Swiggy and Zomato
    distribution = app_preference_distribution.groupby(['City', 'Restaurant␣
↪Name', 'App Preference']).size().unstack(fill_value=0)

    # Normalize to get percentages
    distribution_percentage = distribution.div(distribution.sum(axis=1),␣
↪axis=0) * 100

    # Merge the percentage distribution with the hygiene, cancellation, and␣
↪delivery scores
    # First, we group the original df by 'City' and 'Restaurant Name' to get␣
↪the mean of the scores
    score_data = df.groupby(['City', 'Restaurant Name']).agg({
        'Hygiene Score': 'mean',
        'Cancellation Score': 'mean',
        'Delivery Score': 'mean',
        'Type': 'first'  # Ensuring Type column is included (either 'Top' or␣
↪'Bottom')
    }).reset_index()
    # Merge the scores with the app preference percentage distribution
    final_result = distribution_percentage.reset_index().merge(score_data,␣
↪on=['City', 'Restaurant Name'], how='left')

    # Add Swiggy and Zomato Percentage Columns
    final_result['Swiggy Percentage'] = final_result['Swiggy']
    final_result['Zomato Percentage'] = final_result['Zomato']
    final_result.loc[final_result['Type'] == 'Top', ['Hygiene Score',␣
↪'Delivery Score', 'Cancellation Score']] *= 2

    # Define metrics and platforms
    metrics = ['Hygiene', 'Delivery', 'Cancellation']
    platforms = ['Swiggy', 'Zomato']

    # Calculate scores for each platform and metric
    for metric in metrics:
        for platform in platforms:
            final_result[f'{platform} {metric}'] = (
                final_result[f'{platform} Percentage'] / 100
            ) * final_result[f'{metric} Score']

  except Exception as e:
    # Raise exception with the error message if something goes wrong
    raise Exception("While solving Objective 4 Question -\nError Msg:", e)

  else:
    # Print a success message
    print("Result fetched successfully...")
    self.plot_graph(final_result)
```

The following function visualizes the question - How do Swiggy and Zomato ratings for hygiene, cancellation, and delivery performance vary between top and bottom-rated restaurants across major cities, and what insights can be drawn to improve service quality?:

[22]:
```
def objective_4(dataset_after_preprocess_cleaned):
    try:
        # Create an instance of the Objective_4_solution class
        obj4 = Objective_4_solution()

        # Call the solution method, passing the preprocessed dataset
        obj4.solution(dataset_after_preprocess_cleaned)

    except Exception as e:
        # Raise exception with the error message if something goes wrong
        print('Oops! Something went wrong', e)

    else:
        # Print a success message
        print('Woohoo!!! Program executed successfully.')
```

4 Project Outcome (10 + 10 marks)

4.1 Overview of Results

The study's objectives collectively aim to understand and optimize consumer engagement and business strategies for Swiggy and Zomato. Objective 1 is foundational, focusing on data preparation to ensure accuracy and reliability in subsequent analysis. Clean, standardized, and complete datasets are vital for drawing actionable insights, forming the basis for effective decision-making in later objectives.

Objective 2 delves into customer preferences across top cities, analyzing variations by demographic factors such as income, age, gender, and marital status. This segmentation aims to identify patterns that influence platform choice and ordering behavior. The findings from this analysis are expected to guide targeted marketing strategies and enhance service offerings for specific demographic groups.

Objective 3 complements this by examining consumer behavior on both platforms through city-wise trends, meal type preferences, and popular cuisines. This broader market analysis seeks to highlight key factors driving revenue and user engagement, offering insights into meal and cuisine popularity by location and time. Together, these objectives are structured to provide a comprehensive understanding of user behavior and market trends, enabling Swiggy and Zomato to tailor their services effectively to enhance user satisfaction and maximize profitability.

Objective 4 compares top and bottom-rated restaurants across four cities based on hygiene, cancellation, and delivery scores from Swiggy and Zomato. Top-rated restaurants consistently outperform bottom-rated ones, with hygiene being the key differentiator. Swiggy slightly edges Zomato in scores. Delivery reliability is generally weaker for poorly rated restaurants.

4.2 Objective 1: Data Preprocessing and Simulation

Objective 1 focuses on preparing and simulating data to ensure that the datasets are clean, consistent, and ready for analysis and visualization. This phase is crucial to transform raw data into a structured, usable format, making it suitable for accurate analysis and insights.

4.2.1 Explanation of Results

Steps Followed: 1. Data Preprocessing:

The first step involves cleaning the raw data by removing irrelevant or redundant rows and columns that are not necessary for analysis. This includes eliminating missing or inconsistent data points, as well as dropping duplicates. By retaining only the most relevant data, we ensure that the dataset is both manageable and free of noise, which could otherwise distort results.

2. Standardizing Column Formats:

Since the datasets come from different sources, they may have varying formats or naming conventions. In this step, values in key columns, such as ratings, cuisine types, or delivery times, are standardized to maintain uniformity across datasets. Standardizing these columns ensures compatibility, allowing the datasets to be integrated and analyzed together efficiently.

3. Merging Datasets:

After cleaning and standardizing the individual datasets, the next step is to merge them into one consolidated dataset. This involves combining the datasets based on common attributes like restaurant names, cities, and cuisine types. Merging ensures that all relevant data points are unified in a single dataset, which simplifies the analysis of relationships and trends across variables.

4. Handling Missing Values:

Missing data is common in real-world datasets and can lead to biased results. To address this, missing values are imputed based on the existing distribution of data in each column. This ensures that the dataset remains complete and representative without introducing bias, maintaining the integrity of the data.

5. Data Simulation:

In cases where certain features, such as Cuisine or Feedback, are missing or incomplete, data simulation techniques are employed. For example, missing Feedback values can be simulated using a normal distribution, providing realistic estimates. This step enhances the dataset by filling in gaps and ensuring that all variables are populated.

Final Outcome:

The goal of Objective 1 is to transform raw data into a clean, consistent, and complete dataset that is ready for analysis. After preprocessing, standardization, merging, and handling missing values, the resulting dataset is structured and optimized for further analysis. It is now ready for data visualization, ensuring that any insights generated are based on high-quality, reliable data.

4.2.2 Visualisation

[23]:
```
dataset = objective_1()
dataset.sample(5)
```

```
Downloading from
https://www.kaggle.com/api/v1/datasets/download/aniruddhapa/swiggy-restaurants-
dataset-of-metro-cities?dataset_version_number=1…

100%|      | 776k/776k [00:00<00:00, 22.4MB/s]

Extracting files…


Downloading from
https://www.kaggle.com/api/v1/datasets/download/ashishjangra27/swiggy-
restaurants-dataset?dataset_version_number=2…

100%|      | 141M/141M [00:02<00:00, 64.0MB/s]

Extracting files…


Downloading from https://www.kaggle.com/api/v1/datasets/download/rishikeshkonapu
re/zomato?dataset_version_number=1…
```

```
100%|     | 89.0M/89.0M [00:01<00:00, 84.0MB/s]

Extracting files…


Downloading from
https://www.kaggle.com/api/v1/datasets/download/benroshan/online-food-delivery-
preferencesbangalore-region?dataset_version_number=3…

100%|     | 24.0k/24.0k [00:00<00:00, 16.2MB/s]

Extracting files…
All datasets have been loaded successfully!
Dataset Details:
Size of DATASET_FILE_1 is (8691, 18) and sample 5 rows:
     type      id                        name
uuid       city        area avgRating totalRatingsString
cuisines costForTwoStrings  deliveryTime  minDeliveryTime  maxDeliveryTime
address              locality unserviceable    veg  City
6326    F  323459  Sion Restaurant and Stores
0f32097c-aed1-43f3-b6f1-61c1b6929b21     Mumbai        Sion         4        20+
ratings  ['Indian' 'Tandoor' 'Chinese' 'Mughlai']       300 FOR TWO
43               43               43  Shop No. 3 & 4 Sion Mansion 31 SIon Main
Road …      Sion Bhakti park         False  False   NaN
3031    F  126017                Dev kitchen
34c47843-d315-47df-a75c-5bdd8699aa4e      Delhi      Rohini        --    Too Few
Ratings  ['North Indian' 'Chinese' 'Continental']       200 FOR TWO
50               50               50                   Csc 5 Shop no 35 Sector-9
Rohini       Sector-9 Rohini         False   True   NaN
1599    F  297374                  Annapurna
8116af4b-34e7-44f5-bbf3-02159c17d3e4  Bangalore  Srirampura        --    Too Few
Ratings      ['North Indian' 'Tandoor' 'Chinese']       150 FOR TWO
60               60               60   NO. 31/16 1ST MAIN 1ST CROSS 1ST STAGE
Okalip…  1ST STAGE Okalipuram         False  False   NaN
6370    F   31405             Kisan Dosawala
5eb045e8-b344-4932-b549-e4f01aa73180     Mumbai        Sion       4.4        20+
ratings                          ['South Indian']       300 FOR TWO
47               47               47  10/A/1 Sion Sindhi Colony Opp.sach Khand
Darba…        Khand Darbar         False   True   NaN
8446    F  350104               Cake Eggless
cdc4c2b6-34cc-4dcd-8701-85ad288e2bea      Surat       Athwa        --    Too Few
Ratings                                ['Bakery']       200 FOR TWO
52               52               52  51 DEVDIP SOCIETY SARAGAM SHOPPING CENTER
PARL…                Piplod         False  False   NaN

Size of DATASET_FILE_2 is (148541, 11) and sample 5 rows:
            id               name                         city rating
rating_count   cost               cuisines          lic_no
link                                                    address              menu
```

```
93750   474546           Benfish  Central Kolkata,Kolkata    3.5      20+
ratings    500   North Indian,Bengali  12821019001992
https://www.swiggy.com/restaurants/benfish-cen…  Benfish, 1558 RAJDANGA MAIN
ROAD,WARD NO-107,B…  Menu/474546.json
116916  532724     WoodWill Cafe                  Noida-1     --  Too Few
Ratings    300       Beverages,Snacks  22722922000202
https://www.swiggy.com/restaurants/woodwill-ca…             WoodWill Cafe,
SH-01 Sector-46, Noida  Menu/532724.json
88147   497897           Grill B        Kalamassery,Kochi     --  Too Few
Ratings    200    Indian,North Indian         license
https://www.swiggy.com/restaurants/grill-b-kal…  Grill B, 101/A1, ward- 24,
kalamassery circle,…  Menu/497897.json
111540  537013         LASSI BAR            Muzaffarnagar     --  Too Few
Ratings    250             Beverages  22722904000055
https://www.swiggy.com/restaurants/lassi-bar-n…  LASSI BAR, 60/380, AVADH
VIHAR A2Z ROAD  WARD …  Menu/537013.json
123816   10695  Rajee's Pure Veg           Bibwewadi,Pune    4.3     100+
ratings    250  Street Food,Fast Food  11516035000203
https://www.swiggy.com/restaurants/rajees-pure…  Rajee's Pure Veg, 1st Floor,
Raviraj Cru Mall,…   Menu/10695.json

Size of DATASET_FILE_3 is (51717, 17) and sample 5 rows:
                                                     url
address                            name online_order book_table    rate  votes
phone        location       rest_type
dish_liked                                    cuisines approx_cost(for two
people)                                     reviews_list
menu_item listed_in(type)    listed_in(city)
41596  https://www.zomato.com/bangalore/the-bahubali-…  95/3, Gowrasi Arcade,
Doddenekundi, Outer Ring…                   The Bahubali          Yes
Yes  3.4 /5    153                    +91 7337625805    Marathahalli   Casual
Dining              Paratha, Kulcha, Biryani, Dal Makhani
North Indian, Chinese                        800  [('Rated 3.0', "RATED\n  I
went there for a di…                                               []
Dine-out       Marathahalli
47220  https://www.zomato.com/bangalore/ithaca-the-ch…  Chancery Pavilion,
135, Residency Road, Bangalore  Ithaca - The Chancery Pavilion          No
Yes  3.9 /5    312                    +91 9902785566  Residency Road     Fine
Dining  Salads, Sunday Brunch, Cocktails, Watermelon J…  North Indian,
Italian, European, Continental                      2,000  [('Rated 3.0',
'RATED\n  The food was good\nTh…
[]          Buffet     Residency Road
41435  https://www.zomato.com/bangalore/the-waffcake-…  50, Bali Squre, Sai
Baba Temple Road, Kundnaha…            The Waffcake Legacy          Yes
No  4.0 /5    174                   +91 8971283161    Marathahalli  Dessert
Parlor  Waffles, Pancakes, Liti Chokha, Ginger Tea, Po…    Desserts, Street
Food, Sandwich, Beverages                        350  [('Rated 1.0', 'RATED\n
The quantity of the f…  ['Masala Maggi', 'Aloo Sandwich', 'Litti Chokh…
```

```
Desserts       Marathahalli
937    https://www.zomato.com/bangalore/anirams-jp-na…  5th A Cross, Lakshmi
Complex, Manjunatha Colon…                              Aniram's          Yes
No   4.1/5    739  +91 8867760111\r\r\n+91 8867760222        JP Nagar
Delivery  Chilli Chicken, Butter Chicken, Kadhai Paneer,…
Chinese, North Indian                            500  [('Rated 2.0', 'RATED\n  1.5
TBH, gobi chill a…  ['Veg Chinese Combo', 'Chicken Biryani Combo',…
Delivery  Bannerghatta Road
26049  https://www.zomato.com/bangalore/kfc-kammanaha…     4M- 403, 80 Feet
Road, Kammanahalli, Bangalore                                 KFC          Yes
No  3.1 /5    639                      +91 8033994444    Kammanahalli     Quick
Bites  Rice Bowl, Hot Wings, Popcorn Chicken, Crispy …
Burger, Fast Food                          450  [('Rated 4.0', 'RATED\n  One stop
place for al…                                                [|]        Dine-
out       Kammanahalli

Size of DATASET_FILE_4 is (388, 55) and sample 5 rows:
     Age  Gender Marital Status      Occupation   Monthly Income Educational
Qualifications  Family size  latitude  longitude  Pin code        Medium (P1)
Medium (P2)   Meal(P1) Meal(P2)                  Perference(P1)
Perference(P2) Ease and convenient     Time saving More restaurant choices Easy
Payment option More Offers and Discount Good Food quality Good Tracking system
Self Cooking Health Concern  … Long delivery time Delay of delivery person
getting assigned Delay of delivery person picking up food Wrong order delivered
Missing item Order placed by mistake Influence of time          Order Time
Maximum wait time Residence in busy location Google Maps Accuracy Good Road
Condition Low quantity low time Delivery person ability Influence of rating
Less Delivery time High Quality of package       Number of calls
Politeness          Freshness            Temperature          Good Taste
Good Quantity Output  \
158   27    Male        Married        Employee  More than 50000
Ph.D            5   12.9850    77.5533    560010  Food delivery apps   Web
browser     Dinner   Dinner  Non Veg foods (Lunch / Dinner)                 Ice
cream / Cool drinks              Agree        Disagree                Neutral
Disagree                 Neutral          Disagree    Strongly disagree
Agree          Agree  …              Agree                           Strongly
agree                                 Agree                 Agree  Strongly
agree       Strongly disagree              Yes    Anytime (Mon-Sun)
15 minutes                      Agree       Strongly Agree              Agree
Strongly Agree          Strongly Agree                 Yes            Important
Very Important            Important      Very Important      Very Important
Very Important        Very Important      Very Important     No
382   24  Female         Single         Student        No Income
Post Graduate            3   12.9828    77.6131    560042  Food delivery apps
Direct call     Dinner   Dinner  Non Veg foods (Lunch / Dinner)
Ice cream / Cool drinks      Strongly agree          Agree
Agree               Agree                    Agree             Agree
Agree        Agree          Agree  …              Agree
Agree                                    Agree               Neutral
Agree                  Agree               Yes  Weekend (Sat & Sun)
45 minutes                     Agree                Agree               Agree
Neutral                Disagree               Maybe  Moderately Important
Important  Moderately Important  Slightly Important           Important
Moderately Important        Very Important      Very Important    Yes
121   26    Male         Single  Self Employeed   25001 to 50000
Graduate           3   12.9766    77.5993    560001  Food delivery apps   Web
browser      Lunch   Dinner  Non Veg foods (Lunch / Dinner)   Veg foods
(Breakfast / Lunch / Dinner)               Agree           Agree
Agree      Strongly agree          Strongly agree    Strongly agree
Strongly agree     Disagree       Disagree  …             Agree
Disagree                                    Agree              Disagree
Disagree                Disagree                No   Weekdays (Mon-Fri)  More
than 60 minutes                     Agree                Agree      Strongly
Agree                 Agree          Strongly Agree                 No
Important              Important    Slightly Important  Slightly Important
Important             Important             Important            Important    Yes
185   28    Male         Married        Employee  More than 50000
Post Graduate            1   12.9925    77.5633    560021  Food delivery apps
Walk-in  Breakfast    Lunch  Non Veg foods (Lunch / Dinner)   Veg foods
(Breakfast / Lunch / Dinner)               Agree  Strongly agree
Agree      Strongly agree                 Agree            Agree
Strongly agree     Disagree       Disagree  …           Neutral
Neutral                                    Agree              Disagree
Disagree       Strongly disagree              Yes    Anytime (Mon-Sun)
45 minutes                     Agree       Strongly Agree               Agree
Strongly Agree                  Agree                Yes        Very Important
Very Important  Moderately Important           Important           Important
Slightly Important             Important  Slightly Important    Yes
317   29    Male         Single  Self Employeed  More than 50000
Graduate           6   12.8845    77.6036    560076  Food delivery apps   Web
browser     Snacks   Dinner  Non Veg foods (Lunch / Dinner)   Veg foods
(Breakfast / Lunch / Dinner)               Agree           Agree
Neutral            Disagree                  Agree             Agree
Agree     Disagree       Disagree  …             Agree
Neutral                                    Agree              Disagree
Disagree                Disagree              Yes  Weekend (Sat & Sun)  More
than 60 minutes                     Agree              Neutral
Agree              Neutral                  Agree                   No
Moderately Important      Slightly Important  Moderately Important  Slightly
Important  Slightly Important  Moderately Important  Moderately Important
Slightly Important    Yes

                                               Reviews
158  Worst behavior by customer care. They process …
382                                                NIL
121  I have no problem with the delivery service. I…
```

```
185   I love zomato's service \n-from a premium member
317                           Love to order online

[5 rows x 55 columns]
```

```
All the datasets has been successfully cleaned and pre-processed. Moving on to
merger part…

All the datasets has been successfully Merged. Moving on to replace the missing
values if any…

Count of Missing values
 Order_Id                                     23584
Restaurant_Name                                388
City                                             0
Area                                           388
Average_Rating                                 388
Total_Rating_String                            388
Cuisine                                        388
Delivery_Time                                62704
Veg/Non-Veg                                  62706
App_Preference                                   0
Total_Order_Value                            28993
Dish_Liked                                   44923
Age                                              0
Gender                                       67728
Marital_Status                               67728
Occupation                                   67728
Monthly_Income                               67728
Meal_Type                                    67728
Influence_of_Rating                          67728
Freshness_of_Food                            67728
Temperature_of_Food                          67728
Taste_of_Food                                67728
Quantity_of_Food                             67728
Food_Quality                                 67728
Poor_Hygiene                                 67728
Bad_Past_Experience                          67728
High_Quality_of_Package                      67728
Late_Delivery                                67728
Long_Delivery_Time                           67728
Delay_of_Delivery_Person_Getting_Assigned    67728
Delay_of_Delivery_Person_Picking_Up_Food     67728
Maximum_Wait_Time                            67728
Less_Delivery_Time                           67728
Wrong_Order_Delivered                        67728
```

```
Missing_Item                                67728
Order_Placed_by_Mistake                     67728
dtype: int64

Missing values has been successfully replaced maintaining original distribution
percentage. Have a look at final dataset for visualization!

Size of final dataset file is (68116, 36) and sample 5 rows:

       Order_Id          Restaurant_Name       City                   Area
Average_Rating Total_Rating_String       Cuisine  Delivery_Time Veg/Non-Veg
App_Preference Total_Order_Value Dish_Liked  Age  Gender Marital_Status
Occupation   Monthly_Income  Meal_Type Influence_of_Rating   Freshness_of_Food
Temperature_of_Food   Taste_of_Food Quantity_of_Food    Food_Quality
Poor_Hygiene Bad_Past_Experience High_Quality_of_Package Late_Delivery
Long_Delivery_Time Delay_of_Delivery_Person_Getting_Assigned
Delay_of_Delivery_Person_Picking_Up_Food Maximum_Wait_Time    Less_Delivery_Time
Wrong_Order_Delivered       Missing_Item Order_Placed_by_Mistake
126465      NaN                  B-hive  Bangalore  Koramangala 1st Block
4.3                 1559  North Indian           31.0     Non-Veg         Swiggy
  99      Pasta   25    Male         Single    Student        No Income
Breakfast               Maybe      Very Important        Very Important
Important   Very Important           Agree      Neutral              Agree
Very Important       Neutral           Disagree
Neutral                                  Neutral       60 minutes  Moderately
Important               Neutral          Disagree              Disagree
115413      NaN  70 Brigade - Iris Hotel  Bangalore          Brigade Road
3.9                106  North Indian           34.0         Veg         Swiggy
  300      Pasta   37    Male         Single    Student   Below Rs.10000
Snacks                 Yes      Very Important    Slightly Important  Very
Important        Important         Neutral      Neutral            Disagree
Important       Neutral              Agree
Agree                                  Neutral        15 minutes
Important               Neutral     Strongly agree              Disagree
20549    260444            Donne Biryani  Bangalore                   BTM
3.7        500+ ratings       Biryani           48.0         Veg         Swiggy
  100   Pancakes   32    Male         Single    Student  More than 50000
Snacks                 Yes      Very Important        Very Important
Important   Very Important           Agree     Disagree             Neutral
Very Important      Disagree              Agree
Disagree                                   Agree       30 minutes
Important        Strongly agree          Disagree          Strongly agree
90287    507766    Meat and Eat - Mysore     Mysore          Mysore South
3.8          20+ ratings      American           40.0     Non-Veg         Zomato
  150       Roti   26  Female          Single    Student   25001 to 50000
Snacks                 Yes  Slightly Important  Moderately Important
Important   Very Important  Strongly agree      Neutral            Disagree
Important       Neutral              Agree                            Strongly
```

```
agree                          Strongly agree       45 minutes
Important              Disagree  Strongly disagree          Strongly agree
69247    348122            Tibbs frankie  Hyderabad                Attapur
3.9        100+ ratings        Indian          52.0          Veg          Swiggy
 140  Cocktails   40  Female        Married    Student        No Income
Breakfast                 Maybe             Important            Important  Very
Important   Very Important        Disagree        Agree              Neutral
Very Important      Disagree            Neutral
Agree                                   Disagree       15 minutes
Important     Strongly disagree           Disagree                   Agree

Woohoo!!! Program executed successfully.
```

[23]:
```
       Order_Id  Age  Gender Marital_Status Occupation  Monthly_Income
App_Preference     Restaurant_Name       City                   Area
Cuisine Veg/Non-Veg  Delivery_Time Total_Order_Value  Meal_Type       Dish_Liked
Average_Rating Total_Rating_String Influence_of_Rating     Freshness_of_Food
Temperature_of_Food   Taste_of_Food      Quantity_of_Food       Food_Quality
Poor_Hygiene Bad_Past_Experience High_Quality_of_Package      Late_Delivery
Long_Delivery_Time Delay_of_Delivery_Person_Getting_Assigned
Delay_of_Delivery_Person_Picking_Up_Food Maximum_Wait_Time    Less_Delivery_Time
Wrong_Order_Delivered       Missing_Item Order_Placed_by_Mistake
13373     48076   31     Male          Single    Student       No Income
Swiggy     Lot Like Crepes  Bangalore            Koramangala      Desserts
Veg            52.0              600     Snacks              Pasta             4.5
50+ ratings               Maybe    Slightly Important      Very Important
Important              Important     Strongly agree             Agree
Disagree          Very Important              Agree              Agree
Strongly agree                                   Agree       45 minutes
Moderately Important              Disagree            Disagree
Disagree
31146    404931    38  Female        Married   Employee  10001 to 25000
Swiggy              Foodhub    Chennai               Annanagar       Chinese
Non-Veg           37.0              350      Lunch           Paratha
4.1         50+ ratings                 Yes  Moderately Important      Very
Important  Very Important        Very Important           Neutral
Agree            Disagree                Important           Disagree
Agree                                  Disagree
Agree        15 minutes  Moderately Important                 Agree     Strongly
agree                Disagree
119012      NaN   30     Male          Single    Student  10001 to 25000
Swiggy  Asha Sweets Centre  Bangalore                JP Nagar   Street Food
Veg            57.0              800  Breakfast         Panipuri             3.9
42                 Yes         Very Important           Important  Very Important
Important              Agree  Strongly disagree           Disagree
Unimportant              Agree     Strongly agree
Disagree                                    Agree        60 minutes    Slightly
```

```
Important        Strongly agree  Strongly disagree                Neutral
121788      NaN   18    Male        Married    Student  Below Rs.10000
Swiggy       Desi Flavours  Bangalore  Koramangala 1st Block  North Indian
Non-Veg          68.0              200  Breakfast  Chicken Biryani
3.9                121                Yes    Slightly Important      Very
Important  Very Important        Very Important     Strongly agree
Agree              Neutral              Important              Agree
Agree                            Strongly agree
Agree        45 minutes  Moderately Important       Strongly agree
Disagree                  Agree
16129    108069   27    Male         Single    Student       No Income
Swiggy       Yumlane Pizza  Bangalore            Indiranagar     Beverages
Veg           54.0              300      Lunch              Beer            4.0
100+ ratings                 Yes          Unimportant          Important
Important  Moderately Important  Strongly disagree  Strongly disagree
Disagree              Important  Strongly disagree            Agree
Agree                                   Agree       30 minutes
Important                 Agree          Disagree               Disagree
```

4.3 Objective 2: Customer Demographic Analysis of Food Delivery Apps

Objective 2 aims to identify patterns in customer behavior and how these are shaped by demographic factors. Insights from this analysis will help food delivery platforms tailor their services, marketing strategies, and product offerings to better meet the preferences of diverse customer segments.

Question 1. How do age groups, gender, and marital status influence customers' order preferences between Swiggy and Zomato?

This question seeks to uncover demographic trends in food delivery app usage.

4.3.1 Explanation of Results

- Key Insight:

  Swiggy forms the base of each bar (lower section), while Zomato is the upper section. The larger size of Swiggy's base across most categories indicates that Swiggy is the preferred platform for orders.

- Overall Insights
  - Swiggy consistently leads in most age and gender categories, especially among younger and middle-aged groups (18-40 years).
  - Zomato has competitive performance among older married individuals (51-60 years) and some smaller subcategories, such as those who "prefer not to say" within specific demographics.
  - Single individuals, regardless of gender or age, lean heavily toward Swiggy, which may indicate stronger brand loyalty or appeal among this group.
  - Married females across most age brackets show a closer split in preferences compared to other segments.

- General Insights:
  - Age Group Trends:
    - Customers aged 21–30 exhibit the highest engagement across both platforms, suggesting this is the primary demographic for food delivery services.
    - Older age groups (41–60) show lower order volumes, highlighting a diminishing preference for online food delivery as age increases.
  - Gender and Marital Status Patterns:
    - Male Singles consistently show the highest order preferences, particularly for Swiggy, across all age groups. For instance, in the 21–30 age group, male singles contribute significantly, with Swiggy preferred by ~65%.
    - Female Singles also prefer Swiggy in most cases and exhibit similar behaviour compared to their male counterparts.
    - Married customers (both genders) show slightly reduced engagement compared to singles but still lean towards Swiggy.
    - Customers who selected "Prefer not to say" generally have smaller contributions to the data pool, with Swiggy seeing a relatively higher proportion in this category whereas Zomato has higher proportion in Male 51-60 age group.
- Specific Observations:
  - Age Group 18-20 : Swiggy is the preferred platform for all subcategories. Single females and single males show almost equal preference percentages for Swiggy at ~65%. Zomato lags behind by a huge margin of around 35%. Among those who "prefer not to say," males at ~67% lean toward Swiggy more than females at ~64%. Married individuals in this age group also prefer Swiggy, with males at ~67% slightly outpacing females at ~65%.
  - Age Group 21-30 : This is the largest contributing age group, with Swiggy consistently leading across subcategories. Single females at ~64% and single males at ~65% show similar preference like married individuals for Swiggy. Those who "prefer not to say" are notably inclined toward Swiggy, with females at ~65% and males at ~61%.
  - Age Group 31-40 : Swiggy continues to dominate in all subcategories. Among single and married individuals, both genders show steady preferences for Swiggy, typically above 65%.
  - Age Group 41-50 : Swiggy maintains its lead among all age groups. Zomato performs relatively better among married females at ~40% but still lags behind whereas male who prefer not to say have the least percentage at ~31%. The preference gap widens slightly in favor of Swiggy as users express more flexibility in their marital status.
  - Age Group 51-60 : A few exceptions emerge where Zomato surpasses Swiggy: Among those who "prefer not to say," males at 100% lean toward Zomato more than females at 25%. Zomato also performs relatively better among married male at ~41% whereas married female have the least percentage at ~29%.
- Recommendations
  - Zomato should focus on marketing efforts on older age groups (51-60) and married individuals, where its performance is relatively stronger. Target campaigns toward females who "prefer not to say" across age categories, leveraging its competitive percentages.

Address gaps in younger segments (18-40), especially among singles, to capture more market share.
  - Swiggy, on the other hand, should continue leveraging its broad appeal among younger and single demographics to solidify its dominance.

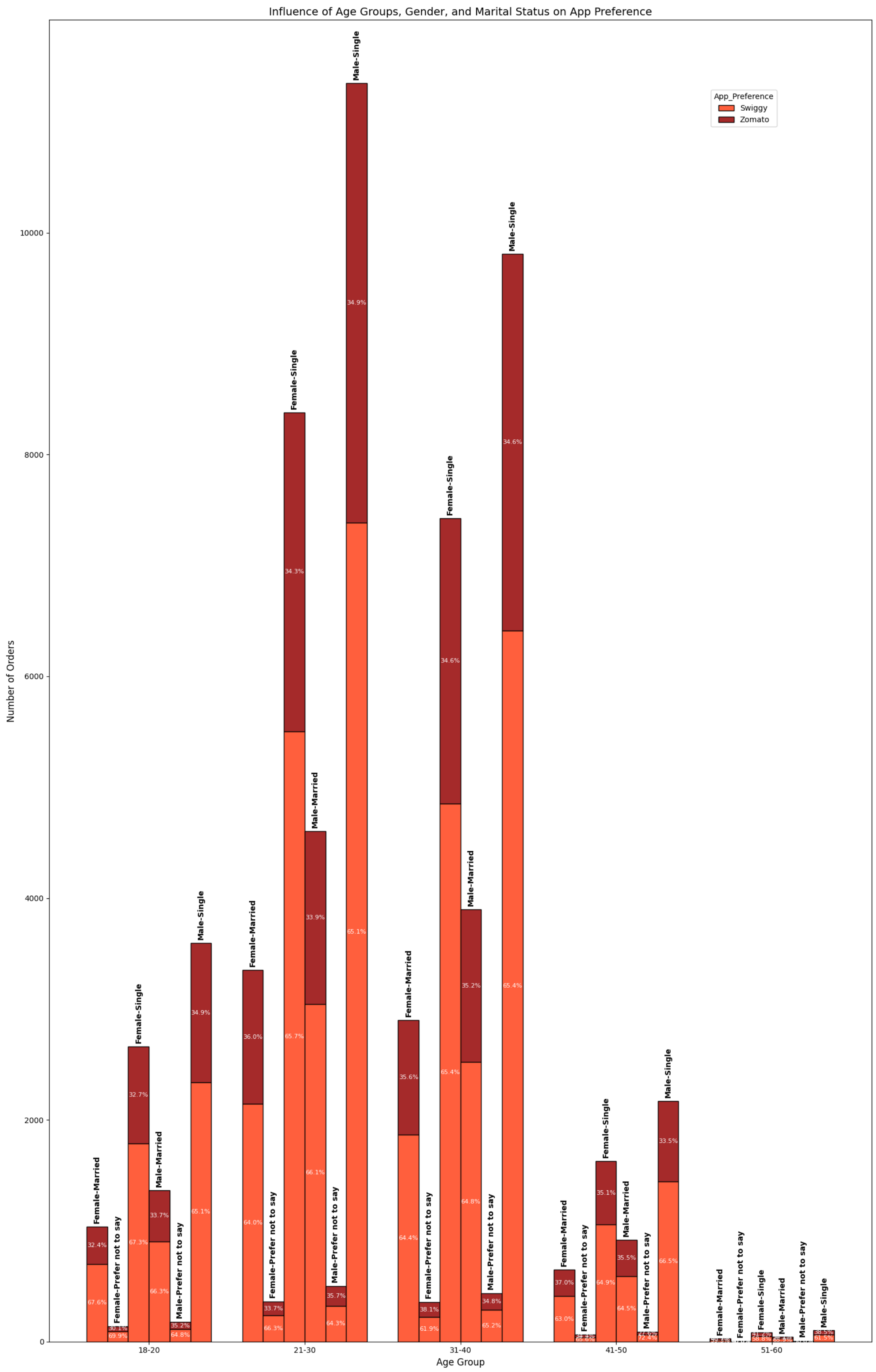
4.3.2 Visualisation

Description of the Graph:

The bar chart represents the distribution of order preferences between Swiggy and Zomato based on the variables of age groups, gender, and marital status. Each cluster of bars corresponds to a specific age group (18–20, 21–30, 31–40, 41–50, 51–60), subdivided further by gender (Male/Female) and marital status (Single/Married/Prefer not to say). The darker shade (red) represents Zomato preferences, while the lighter shade (orange) indicates Swiggy preferences. Each bar is annotated with the percentage of customers favoring a specific app.

[24]:
```
objective_2_Q1(dataset)
```

```
Result fetched successfully…
```


Influence of Age Groups, Gender, and Marital Status on App Preference
Number of Orders
Age Group
18-20
21-30
31-40
41-50
51-60
App_Preference
Swiggy
Zomato
Female-Married
Female-Prefer not to say
Female-Single
Male-Married
Male-Prefer not to say
Male-Single

```
Woohoo!!! Program executed successfully.
```

Question 2. How do customer app preferences (Swiggy vs. Zomato) vary across the top 10 cities, segmented by income groups and age ranges (<25, 25–40, 40–60, 60+)?

This question aims to explore geographic and economic influences on app choice.

4.3.3 Explanation of Results

- Swiggy Preferences (First Heatmap)

  The Swiggy heatmap illustrates the number of Swiggy orders distributed across combinations of age groups, income ranges, and top 10 cities.

  - General Observations:
    - The highest preference for Swiggy is observed among 25–40-year-olds with No Income, especially in Bangalore, followed by cities like Chennai and Hyderabad.
    - Across all cities, younger customers (<25 and 25–40) contribute significantly to Swiggy orders, irrespective of their income levels.
    - Customers in the 40–60 age range and above demonstrate reduced Swiggy usage across all cities and income levels, though Bangalore and Chennai maintain a relatively stronger presence.
  - City-Specific Insights:
    - Bangalore: Swiggy is the most popular in this city across all demographics, with the maximum number of orders coming from 25–40-year-olds with No Income.
    - Chennai and Hyderabad: These cities follow Bangalore in Swiggy orders, with stronger participation from 25–40-year-olds with No Income groups.
    - Delhi, Kolkata and Pune: Moderate Swiggy usage is seen in these cities, with lower-income groups in the 25–40 range leading the preference.
    - Mumbai and Ahmedabad: Swiggy usage is relatively low in these cities, across all age and income groups.
  - Age and Income Trends:
    - No Income (Across Age Groups): This segment has the highest Swiggy preference, particularly among younger customers (<25 and 25–40 years old).
    - Higher Income Groups (Above Rs. 50,000): Swiggy usage drops significantly among higher income groups, regardless of age or city, except in Bangalore.

- Zomato Preferences (Second Heatmap)

  The Zomato heatmap illustrates the number of Zomato orders distributed across combinations of age groups, income ranges, and top 10 cities.

  - General Observations:
    - Zomato also shows its strongest preference among 25–40-year-olds with No Income, with Bangalore being the leading city.
    - Zomato usage is slightly lower than Swiggy for all age and income groups.

        * Orders from the 40–60 age group are sparse, similar to Swiggy.
    - City-Specific Insights:
        * Bangalore: Zomato is most popular here, particularly among 25–40-year-olds with No Income and Rs. 25,001–50,000 income group.
        * Chennai, Hyderabad, and Delhi: These cities also show strong Zomato preference, especially among the No Income group and lower income brackets (<Rs. 25,000).
        * Pune and Kolkata: Usage is moderate for Zomato, similar to Swiggy, with orders concentrated in the 25–40 range and lower income categories.
        * Mumbai and Ahmedabad: These cities report comparatively fewer orders on Zomato, with limited variation across demographics.
    - Age and Income Trends:
        * No Income (Across Age Groups): Zomato's strongest preference comes from this group, particularly among 25–40-year-olds.
        * Middle Income Groups (Rs. 10,001–50,000): Zomato performs slightly on par with Swiggy among this demographic, especially in cities like Kolkata and Hyderabad.
        * Higher Income Groups (Above Rs. 50,000): Similar to Swiggy, Zomato usage decreases significantly in this category, with Bangalore being an exception.
- Comparison of Swiggy and Zomato Preferences
    - Overall Strengths:
        * Both platforms dominate among 25–40-year-olds with No Income, with Swiggy slightly leading in cities like Bangalore and Chennai.
        * Zomato is relatively more balanced across middle income groups in certain cities like Kolkata and Hyderabad.
    - City-Specific Dominance:
        * Swiggy: Leads in Bangalore, Chennai, and Hyderabad across most age and income groups.
        * Zomato: Strong in Bangalore but slightly more competitive in Kolkata and Hyderabad for middle-income users.
    - Age and Income Trends:
        * Both platforms have reduced participation from older age groups (>40).
        * No Income customers dominate usage for both apps, followed by lower income groups. Higher income customers are less likely to use these services.

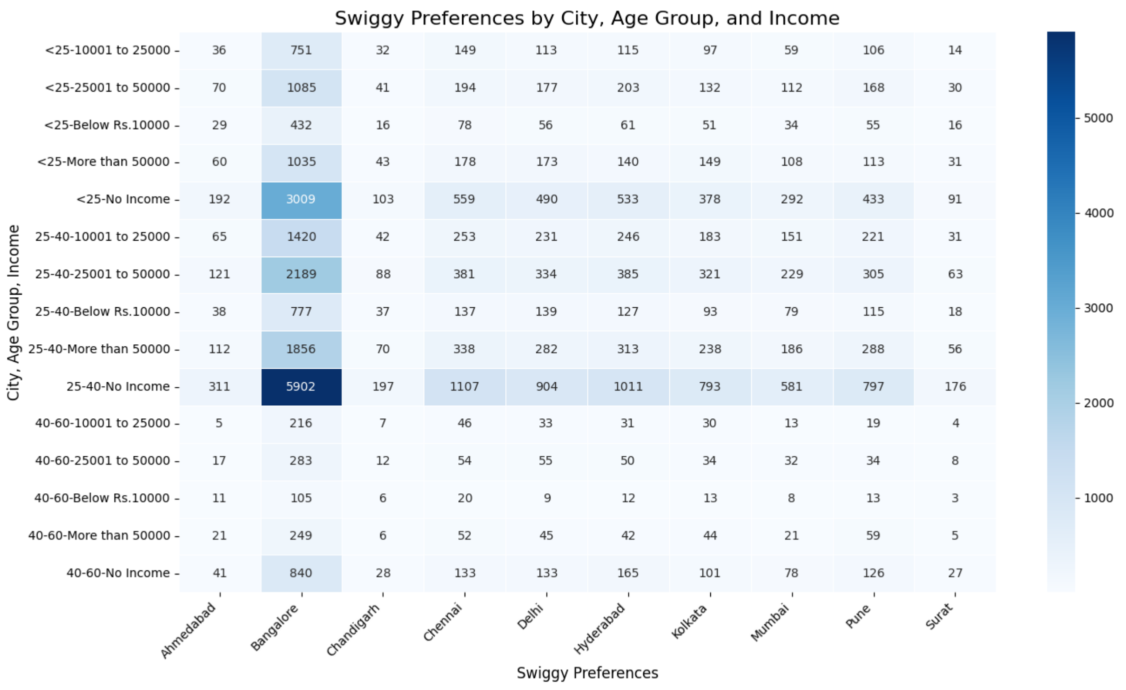
4.3.4 Visualisation

Description of the heatmaps:

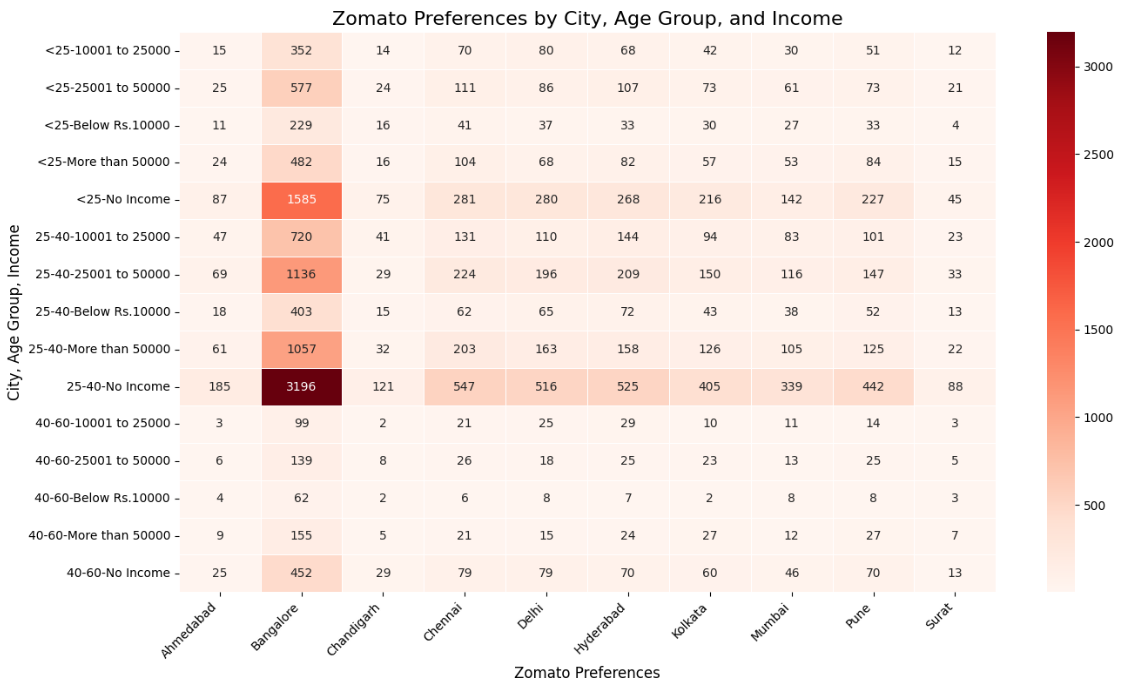
The heatmaps represents the number of orders distributed across combinations of age groups, income ranges, and top 10 cities distribution for Swiggy and Zomato respectively.

[25]:
```
objective_2_Q2(dataset)
```

```
Result fetched successfully…
```



```
<Figure size 1400x50 with 0 Axes>
```



```
Woohoo!!! Program executed successfully.
```

```
<Figure size 1400x50 with 0 Axes>
```

4.4 Objective 3: Revenue Analysis of Food Delivery Apps

Objective 3 helps identify key revenue drivers, uncovering how Swiggy and Zomato can optimize operations, marketing strategies, and partnerships with restaurants to maximize profits. Understanding these dynamics also highlights potential growth opportunities in underserved cities, cuisines, or meal types.

Question 1. How do Swiggy and Zomato compare in terms of user preferences across cities, meal types, and popular cuisines, and how does this influence revenue distribution?

4.4.1 Explanation of Results

- Revenue by App Preference

  The pie chart in the bottom-left corner clearly shows Swiggy holds a significant lead in revenue share at ~65%, compared to Zomato's ~34%. This suggests Swiggy enjoys a dominant market presence, possibly driven by higher order volumes, better retention, or more frequent usage.

- City Breakdown:

  Both Swiggy and Zomato exhibit very similar user distribution across cities, as depicted in the stacked bar charts:

  - Bengaluru is the top city for both platforms, contributing ~45% on both Swiggy and Zomato of the total orders.
  - Other major cities like Chennai and Delhi or Hyderabad (depending on app) collectively account for 15%–17%, showing minor variation between the two platforms.
  - A large portion (~39%) of the user base comes from Other cities, indicating the importance of expanding operations and improving service coverage in non-metro regions.

- Meal Breakdown in Bangalore:

  Focusing on Bangalore, which is the top city contributing to the revenue, the meal type distribution is remarkably consistent for both platforms:

  - Lunch and snacks dominate, with lunch accounting for ~30% and snacks ~31% of orders.
  - Dinner makes up ~23%, while breakfast contributes the least, around 14%.

  These trends highlight user preference for quick and convenient meal types during busy hours of the day.

- Top 2 Cuisines by Meal Type:

  - Both Swiggy and Zomato show a clear preference for North Indian cuisine across all meal types, comprising over 75% of the orders.
  - Chinese cuisine remains a secondary choice, contributing 21%–26% depending on the meal type and app. These trends demonstrate the dominance of North Indian dishes as the primary choice regardless of time of day.

Conclusion:

The analysis highlights key overlaps and differences in consumer preferences between the two platforms.:

- Revenue Dominance: Swiggy significantly outpaces Zomato in revenue generation, holding a decisive 65.1% share.
- City-Wise Similarities: Both platforms rely heavily on Bengaluru, followed by "Others," for a major portion of their user base.
- Meal Type Consistency: User behavior across meal types (breakfast, lunch, snacks and dinner) remains largely uniform between the two platforms, emphasizing similar demand patterns.
- Cuisine Preferences: North Indian cuisine consistently leads across all meal types, with Chinese cuisine as the secondary choice.

These insights provide a comprehensive view of consumer behavior, helping stakeholders identify growth opportunities and refine platform-specific strategies.

4.4.2 Visualisation

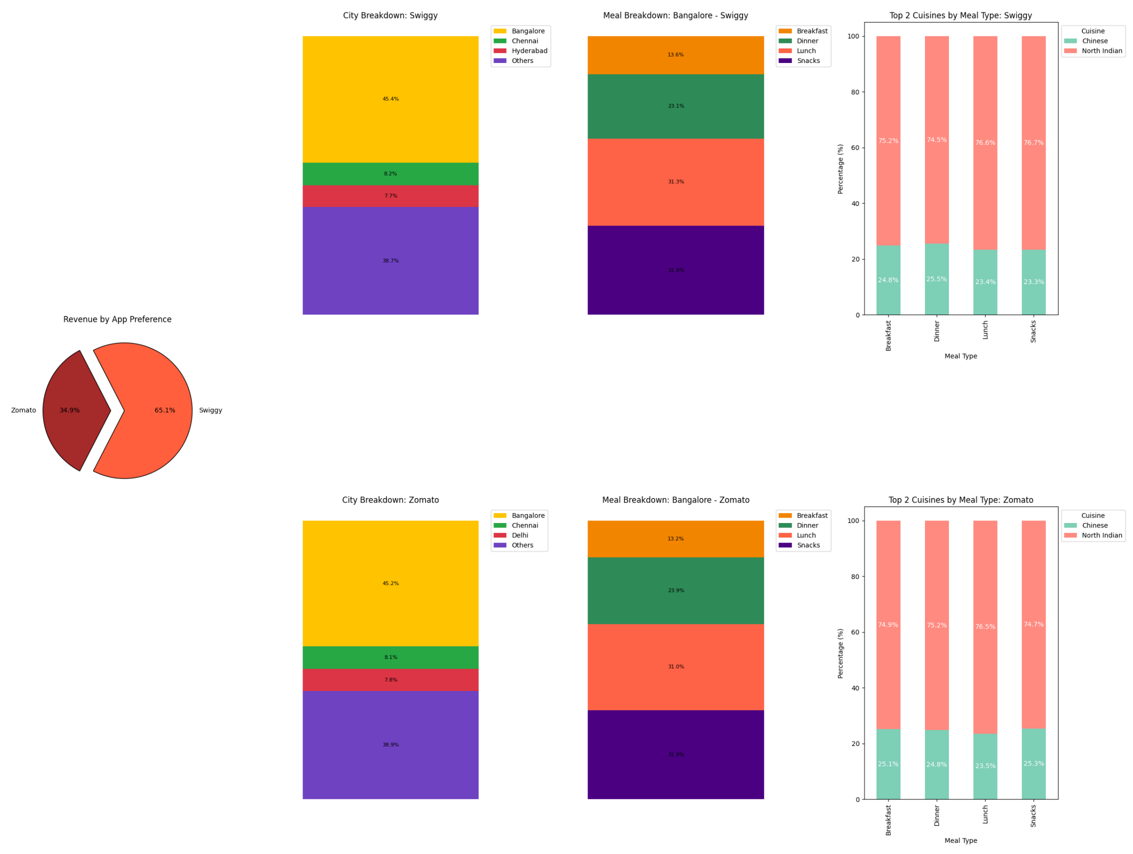
Description of the graph:

The analysis explores user behavior for Swiggy and Zomato across three dimensions: city-wise preferences, meal type distribution in the top city, and popular cuisines for the top ordered meal-type. The comparison also highlights how these preferences impact revenue distribution between the platforms.

[26]:
```
objective_3(dataset)
```

```
<ipython-input-19-1f2e22231de5>:27: DeprecationWarning: DataFrameGroupBy.apply
operated on the grouping columns. This behavior is deprecated, and in a future
version of pandas the grouping columns will be excluded from the operation.
Either pass `include_groups=False` to exclude the groupings or explicitly select
the grouping columns after groupby to silence this warning.
  .apply(lambda x: x.nlargest(2, 'Total_Order_Value')) # Select top 2 cuisines
per meal type
<ipython-input-19-1f2e22231de5>:27: DeprecationWarning: DataFrameGroupBy.apply
operated on the grouping columns. This behavior is deprecated, and in a future
version of pandas the grouping columns will be excluded from the operation.
Either pass `include_groups=False` to exclude the groupings or explicitly select
the grouping columns after groupby to silence this warning.
  .apply(lambda x: x.nlargest(2, 'Total_Order_Value')) # Select top 2 cuisines
per meal type

Result fetched successfully…
```



```
Woohoo!!! Program executed successfully.
```

4.5 Objective 4: Customer Feedback Analysis of Food Delivery Apps

Objective 4 compares top and bottom-rated restaurants across four cities based on hygiene, cancellation, and delivery scores from Swiggy and Zomato. Top-rated restaurants consistently outperform bottom-rated ones, with hygiene being the key differentiator. Swiggy slightly edges Zomato in scores. Delivery reliability is generally weaker for poorly rated restaurants.

Qustion 1. How do Swiggy and Zomato ratings for hygiene, cancellation, and delivery performance vary between top and bottom-rated restaurants across major cities, and what insights can be drawn to improve service quality?

4.5.1 Explanation of Results

- Key Observations:
  - Bangalore:
    - ∗ The top-rated restaurant, Onesta, has significantly higher scores across all metrics than the bottom-rated restaurant, #refuel.
    - ∗ Swiggy scores are marginally higher than Zomato's for both restaurants in most categories.

    * For the bottom-rated restaurant, delivery and hygiene scores are notably low.

  - Chennai:
    * The top-rated Domino's Pizza displays moderately high and balanced scores for both platforms, while the bottom-rated AB Food Paradise shows relatively poor scores.
    * Zomato scores are significantly higher than Swiggy for bottom-rated restaurant.
    * Delivery and cancellation scores for the bottom-rated restaurant are the most significant gaps.

  - Kolkata:
    * The top-rated Wow! Momo achieves strong ratings across all categories compared to the bottom-rated 6 Ballygunje Place, whose scores are substantially lower.
    * Swiggy scores for the top-rated and bottom-rated restaurant are marginally higher than Zomato.
    * Delivery emerges as the weakest metric for the bottom-rated restaurant.

  - Mumbai:
    * Both the top-rated Apsara Ice Creams and bottom-rated 1441 Pizzeria display closer scores compared to other cities, suggesting a narrower quality gap.
    * While hygiene and cancellation scores differ slightly, delivery scores for both restaurants remain relatively consistent.

- General Trends:
  - Hygiene is the key differentiator between top and bottom-rated restaurants, showing a significant gap in scores.
  - Cancellation and delivery scores are generally lower across all restaurants, especially for bottom-rated ones, suggesting these as potential areas for improvement.
  - Swiggy scores tend to be marginally higher for top-rated restaurants, but bottom-rated ones exhibit similar scores across both platforms.
  - Cities like Chennai and Mumbai exhibit narrower score differences, indicating relatively consistent standards, while Bangalore and Kolkata show larger gaps in quality between top and bottom performers.

This detailed analysis highlights actionable insights into restaurant performance and consumer expectations, focusing on service quality enhancement for bottom-rated establishments.

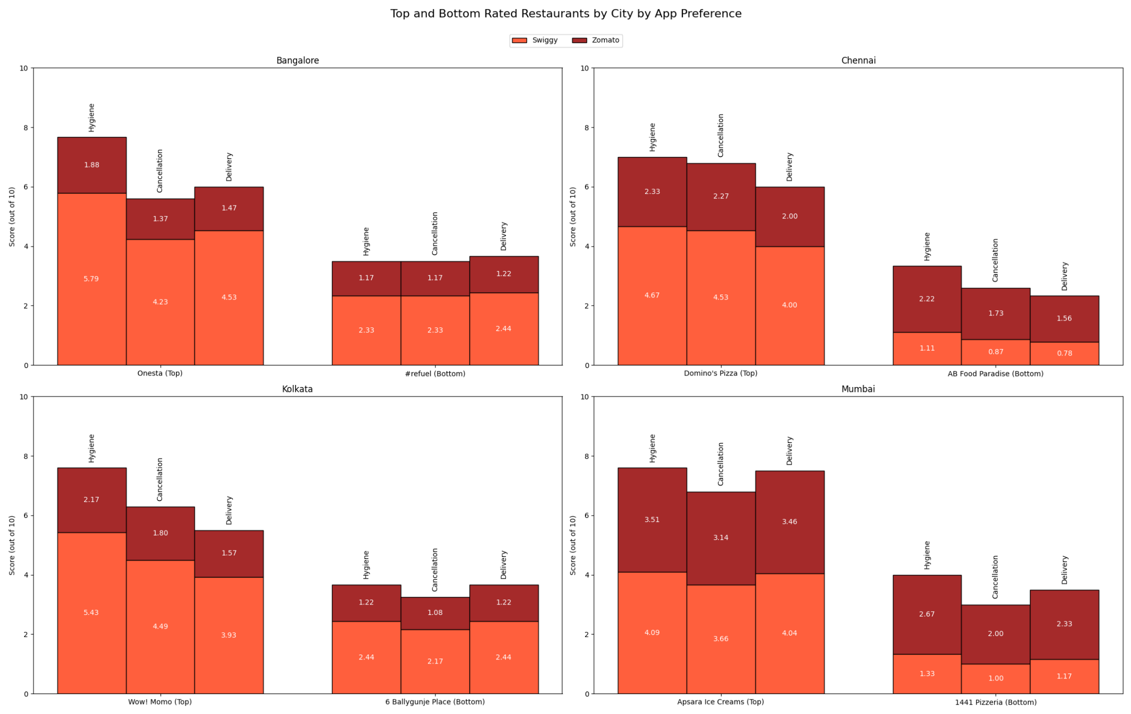
4.5.2 Visualisation

Description of the graph:

The graph compares the top and bottom-rated restaurants in four cities — Bangalore, Chennai, Kolkata, and Mumbai — based on scores for hygiene, cancellation, and delivery reliability as rated on two food delivery platforms, Swiggy and Zomato. Each subplot represents a city and displays the comparative ratings for one top-rated and one bottom-rated restaurant, with scores out of 10.

[27]:
```
objective_4(dataset)
```

```
Result fetched successfully…
```



```
Woohoo!!! Program executed successfully.
```

5 Conclusion (5 marks)

5.0.1 Achievements

In conclusion, the analysis successfully identifies key consumer behavior trends and preferences across Swiggy and Zomato, providing valuable insights for strategic decision-making. Swiggy emerges as the dominant platform, particularly among younger, single demographics, and commands a significant 65% share of revenue, reflecting its broader market appeal. Zomato, while trailing in overall preferences, performs competitively among older and married individuals, as well as in middle-income groups in select cities. The heatmap analysis highlights Bengaluru as the leading city for both platforms, with lunch and snacks being the most popular meal types. Both platforms show a consistent preference for North Indian cuisine.

Additionally, we compared top and bottom-rated restaurants across four cities based on hygiene, cancellation, and delivery scores from Swiggy and Zomato. Top-rated restaurants consistently outperform bottom-rated ones, with hygiene emerging as the key differentiator. Swiggy slightly edges Zomato in scores, although both platforms exhibit weaker delivery reliability for poorly rated restaurants.

These findings not only illuminate demographic-driven preferences but also highlight opportunities for both platforms to refine their marketing strategies and expand their reach. The insights gathered pave the way for more targeted and effective service offerings, positioning both Swiggy and Zomato for continued growth and market optimization.

5.0.2 Limitations

- In this project, our analysis focused exclusively on the two most popular online food delivery platforms in India, Zomato and Swiggy. However, it is important to note that there are numerous other similar apps operating across the country, many of which are either region-specific or tailored to individual restaurants.

- Another limitation of this study is that we examined data from only 25 cities in India. Given that both Zomato and Swiggy operate on a pan-India basis, our findings may not fully capture the broader trends and dynamics of these platforms nationwide.

5.0.3 Future Work

- Prioritize factors such as freshness, temperature, and food quantity to enhance predictive accuracy and improve overall customer satisfaction.
- Analyse the preferred payment modes (e.g., digital wallets, cash on delivery, cards) and their correlation with customer satisfaction to identify actionable insights.
- Examine trends over weeks, months, or seasons, such as the rising popularity of specific dishes during holidays or festivals, to better understand customer preferences.
- Identify patterns that can help reduce delivery times and costs by optimizing routes and improving coordination between restaurants and delivery staff.
- Develop predictive models to forecast peak order times, customer demand patterns, and the likelihood of specific cuisines being ordered based on historical data, seasonality, and external factors such as festivals or weather conditions.
- Implement machine learning models for dynamic pricing optimization, offering personalized discounts based on order frequency, customer loyalty, and demand fluctuations (e.g., during peak hours or special promotions).
- Leverage geospatial data analysis (GIS) to optimize delivery routes, reduce delivery times, and minimize fuel consumption by considering traffic patterns, weather conditions, and the geographical distribution of customers.
- Design an interactive dashboard that allows users to dynamically adjust values and visualize the impact on key metrics in real time.