Swiggy Sales Analytics Project - SQL Server

Project Overview

This project analyzes a dataset of 197,430 food delivery records from Swiggy, spanning states, cities, restaurants, categories, and dishes across India. Key performance metrics include Total Orders (197,430), Total Revenue (53.01 Million INR), Average Dish Price (268.51 INR), and Average Rating (4.34). The structured approach involves data cleaning and validation for quality assurance, dimensional modeling via a Star Schema for optimized querying, and KPI development to derive actionable insights on sales trends, location performance, food metrics, customer spending patterns, and ratings distribution.

SQL Server Connection Setup

In [1]:
```
import pyodbc
import pandas as pd
from sqlalchemy import create_engine, text
import urllib
```

In [2]:
```
#Creating the Engine
server = r'DESKTOP-FNTJ\SQLEXPRESS'
database = 'Swiggy Database'

# Encode connection string for SQLAlchemy
params = urllib.parse.quote_plus(
    f'DRIVER={{ODBC Driver 17 for SQL Server}};SERVER={server};DATABASE={database};Trusted_Co
)

# Create SQLAlchemy engine
engine = create_engine(f"mssql+pyodbc:///?odbc_connect={params}")
```

Data Cleaning & Validation

In [3]:
```
# Null Check:  Identifying missing values in key columns.
null_check_query = """
SELECT
    SUM(CASE WHEN State IS NULL THEN 1 ELSE 0 END) AS null_state,
    SUM(CASE WHEN City IS NULL THEN 1 ELSE 0 END) AS null_city,
    SUM(CASE WHEN Order_Date IS NULL THEN 1 ELSE 0 END) AS null_order_date,
    SUM(CASE WHEN Restaurant_Name IS NULL THEN 1 ELSE 0 END) AS null_restaurant,
    SUM(CASE WHEN Location IS NULL THEN 1 ELSE 0 END) AS null_location,
    SUM(CASE WHEN Category IS NULL THEN 1 ELSE 0 END) AS null_category,
    SUM(CASE WHEN Price_INR IS NULL THEN 1 ELSE 0 END) AS null_price,
    SUM(CASE WHEN Rating IS NULL THEN 1 ELSE 0 END) AS null_rating,
    SUM(CASE WHEN Rating_Count IS NULL THEN 1 ELSE 0 END) AS null_rating_count
FROM swiggy_data;
"""

# Execute query using the SQLAlchemy engine
null_check_df = pd.read_sql(null_check_query, engine)

# Display results
null_check_df
```

Out[3]:

	null_state	null_city	null_order_date	null_restaurant	null_location	null_category	null_price	null_ra
0	0	0	0	0	0	0	0	

In [4]:

```
# Blank/Empty String Check: Identify rows where key fields are empty strings
empty_string_query = """
SELECT *
FROM swiggy_data
WHERE State = ''
   OR City = ''
   OR Restaurant_Name = ''
   OR Location = ''
   OR Category = ''
   OR Dish_Name = ''
   OR Price_INR = ''
   OR Rating = ''
   OR Rating_Count = '';
"""

# Execute the query using the SQLAlchemy engine
empty_string_df = pd.read_sql(empty_string_query, engine)

# Display results
empty_string_df
```

Out[4]:

	State	City	Order_Date	Restaurant_Name	Location	Category	Dish_Name
0	Karnataka	Bengaluru	2025-06-29	Anand Sweets & Savouries	Rajarajeshwari Nagar	Snack	Butter Murukku-200gm
1	Karnataka	Bengaluru	2025-03-13	Srinidhi Sagar Deluxe	Kengeri	Recommended	Mix Raitha
2	Karnataka	Bengaluru	2025-07-08	Srinidhi Sagar Deluxe	Kengeri	Recommended	Srinidhi Sagar Special
3	Karnataka	Bengaluru	2025-04-13	Srinidhi Sagar Deluxe	Kengeri	Recommended	Pista
4	Karnataka	Bengaluru	2025-08-03	Srinidhi Sagar Deluxe	Kengeri	North Indian Gravy	Kaju Masla
...	...	...	...	...	...	...	...
79075	Sikkim	Gangtok	2025-01-25	Mama's Kitchen	Gangtok	Momos	Soya cheese chilli momo ...
79076	Sikkim	Gangtok	2025-07-02	Mama's Kitchen	Gangtok	Momos	Kurkure momo fried ...
79077	Sikkim	Gangtok	2025-03-25	Mama's Kitchen	Gangtok	Momos	Chilli cheese momo
79078	Sikkim	Gangtok	2025-03-26	Mama's Kitchen	Gangtok	Momos	Veg Momos (8 Pc)
79079	Sikkim	Gangtok	2025-03-27	Mama's Kitchen	Gangtok	Momos	Soya Momo

79080 rows × 10 columns

In [5]:

```python
# Duplicate Detection: Identify rows that are duplicated across critical columns
duplicate_query = """
SELECT
    State, City, Order_Date, Restaurant_Name, Location, Category, Dish_Name, Price_INR, Ratin
    COUNT(*) AS CNT
FROM swiggy_data
GROUP BY State, City, Order_Date, Restaurant_Name, Location, Category, Dish_Name, Price_INR, R
HAVING COUNT(*) > 1;
"""

# Execute the query using the SQLAlchemy engine
duplicate_df = pd.read_sql(duplicate_query, engine)

# Display results
duplicate_df
```

Out[5]:

State	City	Order_Date	Restaurant_Name	Location	Category	Dish_Name	Price_INR	Rating	Ratir

In [6]:
```
# Duplicate Removal: Remove duplicate records while retaining one unique row per business key
duplicate_removal_query = """
WITH CTE AS (
    SELECT *,
           ROW_NUMBER() OVER (
               PARTITION BY
                   State, City, Order_Date, Restaurant_Name, Location,
                   Category, Dish_Name, Price_INR, Rating, Rating_Count
               ORDER BY (SELECT NULL)
           ) AS rn
    FROM swiggy_data
)
DELETE FROM CTE
WHERE rn > 1;
"""

# Execute the duplicate removal query
with engine.begin() as connection:
    connection.execute(text(duplicate_removal_query))
```

Dimensional Modeling (Star Schema)

I use a Star Schema: dimension tables store attributes, the fact table stores metrics, and foreign keys link them for efficient analysis.

Dimension Tables

In [37]:
```
# dim_date : Stores date breakdowns for time-based analysis.
create_dim_date_query = """
IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='dim_date' AND xtype='U')
BEGIN
    CREATE TABLE dim_date (
        date_id INT IDENTITY(1,1) PRIMARY KEY,
        Full_Date DATE,
        Year INT,
        Month INT,
        Month_Name VARCHAR(20),
        Quarter INT,
        Day INT,
        Week INT
    );
END
"""

with engine.begin() as conn:
    conn.execute(text(create_dim_date_query))
```

In [38]:
```
# dim_location : Stores geographic details.
create_dim_location_query = """
IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='dim_location' AND xtype='U')
BEGIN
    CREATE TABLE dim_location (
        location_id INT IDENTITY(1,1) PRIMARY KEY,
        State VARCHAR(100),
        City VARCHAR(100),
        Location VARCHAR(200)
    );
END
"""

# Execute the table creation query
```

```python
with engine.begin() as connection:
    connection.execute(text(create_dim_location_query))
```

In [39]:
```python
# dim_restaurant : Stores restaurant names.
create_dim_restaurant_query = """
IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='dim_restaurant' AND xtype='U')
BEGIN
    CREATE TABLE dim_restaurant (
        restaurant_id INT IDENTITY(1,1) PRIMARY KEY,
        Restaurant_Name VARCHAR(200)
    );
END
"""

# Execute the table creation query
with engine.begin() as connection:
    connection.execute(text(create_dim_restaurant_query))
```

In [40]:
```python
# dim_category : Stores cuisine categories.
create_dim_category_query = """
IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='dim_category' AND xtype='U')
BEGIN
    CREATE TABLE dim_category (
        category_id INT IDENTITY(1,1) PRIMARY KEY,
        Category VARCHAR(200)
    );
END
"""

# Execute the table creation query
with engine.begin() as connection:
    connection.execute(text(create_dim_category_query))
```

In [41]:
```python
# dim_dish : Stores dish names.
create_dim_dish_query = """
IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='dim_dish' AND xtype='U')
BEGIN
    CREATE TABLE dim_dish (
        dish_id INT IDENTITY(1,1) PRIMARY KEY,
        Dish_Name VARCHAR(200)
    );
END
"""

# Execute the table creation query
with engine.begin() as connection:
    connection.execute(text(create_dim_dish_query))
```

Fact Table: fact_swiggy_orders

Central table with metrics and foreign keys.

In [42]:
```python
# Fact Table: fact_swiggy_orders
create_fact_swiggy_orders_query = """
IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='fact_swiggy_orders' AND xtype='U')
BEGIN
    CREATE TABLE fact_swiggy_orders (
        order_id INT IDENTITY(1,1) PRIMARY KEY,
        date_id INT,
        Price_INR DECIMAL(10,2),
        Rating DECIMAL(4,2),
```

```
        RatingCount INT,
        location_id INT,
        restaurant_id INT,
        category_id INT,
        dish_id INT,
        FOREIGN KEY (date_id) REFERENCES dim_date(date_id),
        FOREIGN KEY (location_id) REFERENCES dim_location(location_id),
        FOREIGN KEY (restaurant_id) REFERENCES dim_restaurant(restaurant_id),
        FOREIGN KEY (category_id) REFERENCES dim_category(category_id),
        FOREIGN KEY (dish_id) REFERENCES dim_dish(dish_id)
    );
END
"""

# Execute the table creation query
with engine.begin() as connection:
    connection.execute(text(create_fact_swiggy_orders_query))
```

Populate Dimension Tables

In [13]:
```
# dim_date :
insert_dim_date_query = """
INSERT INTO dim_date (FULL_DATE, Year, Month, Month_Name, Quarter, Day, Week)
SELECT DISTINCT
    Order_Date,
    YEAR(Order_Date),
    MONTH(Order_Date),
    DATENAME(MONTH, Order_Date),
    DATEPART(QUARTER, Order_Date),
    DAY(Order_Date),
    DATEPART(WEEK, Order_Date)
FROM swiggy_data
WHERE Order_Date IS NOT NULL;
"""

# Execute the insert query
with engine.begin() as connection:
    connection.execute(text(insert_dim_date_query))
```

In [14]:
```
# dim_location :
insert_dim_location_query = """
INSERT INTO dim_location (State, City, Location)
SELECT DISTINCT State, City, Location
FROM swiggy_data;
"""

# Execute the insert query
with engine.begin() as connection:
    connection.execute(text(insert_dim_location_query))
```

In [15]:
```
# dim_restaurant :
insert_dim_restaurant_query = """
INSERT INTO dim_restaurant (Restaurant_Name)
SELECT DISTINCT Restaurant_Name
FROM swiggy_data;
"""

# Execute the insert query
with engine.begin() as connection:
    connection.execute(text(insert_dim_restaurant_query))
```

In [16]:
```
# dim_category :
insert_dim_category_query = """
INSERT INTO dim_category (Category)
SELECT DISTINCT Category
FROM swiggy_data;
"""

# Execute the insert query
with engine.begin() as connection:
    connection.execute(text(insert_dim_category_query))
```

In [17]:
```
# dim_dish :
insert_dim_dish_query = """
INSERT INTO dim_dish (Dish_Name)
SELECT DISTINCT Dish_Name
FROM swiggy_data;
"""

# Execute the insert query
with engine.begin() as connection:
    connection.execute(text(insert_dim_dish_query))
```

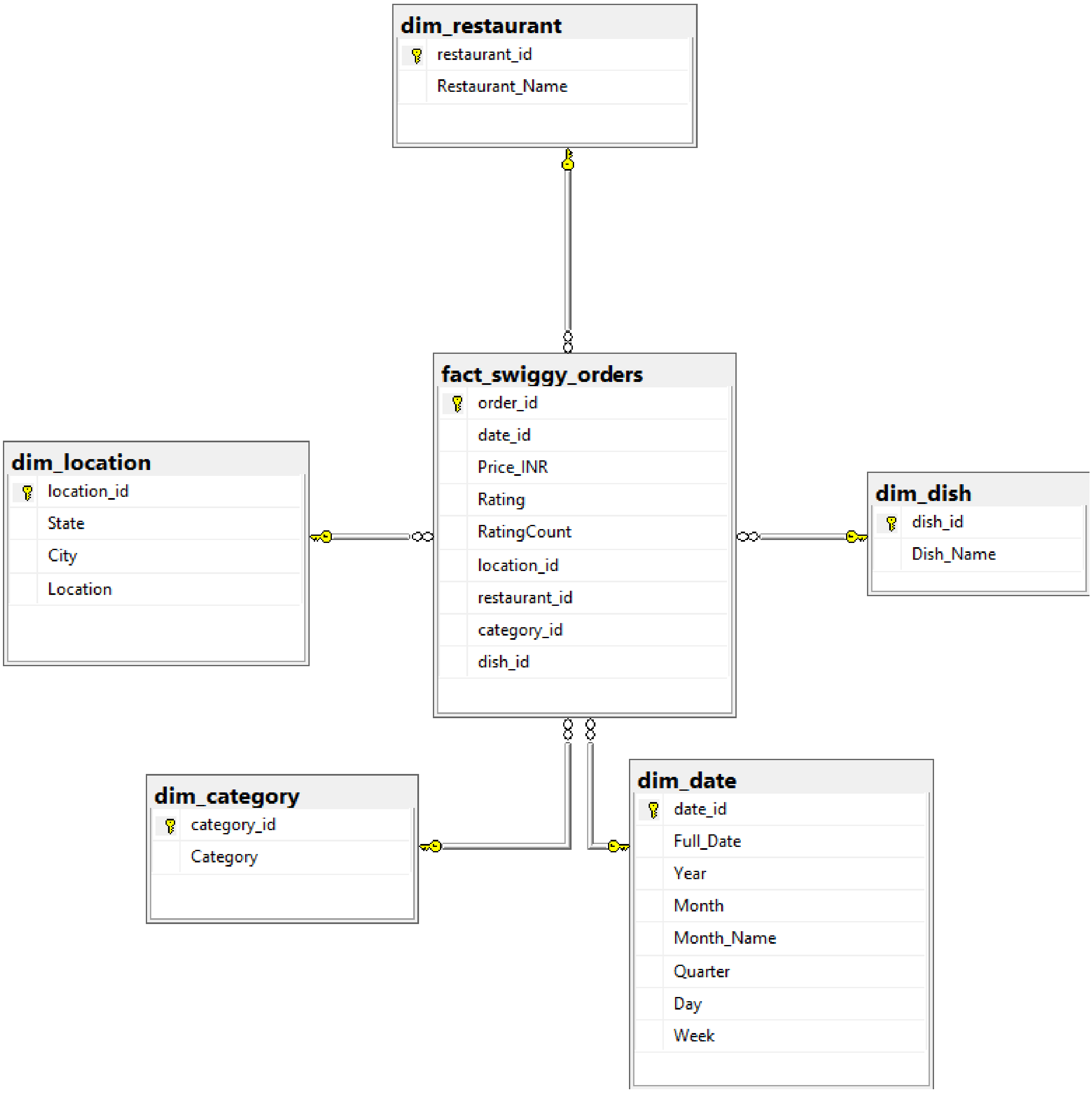
Populate Fact Table

In [43]:
```
# Populated the fact table by mapping each order to its corresponding dimension keys and captu
#such as price, rating, and rating count. This completes the star schema for analytical queri

insert_fact_swiggy_orders_query = """
INSERT INTO fact_swiggy_orders (
    date_id, Price_INR, Rating, RatingCount, location_id, restaurant_id, category_id, dish_id
)
SELECT
    dd.date_id,
    s.Price_INR,
    s.Rating,
    s.Rating_Count,
    dl.location_id,
    dr.restaurant_id,
    dc.category_id,
    dsh.dish_id
FROM swiggy_data AS s
JOIN dim_date AS dd ON dd.FULL_DATE = s.Order_Date
JOIN dim_location AS dl ON dl.State = s.State AND dl.City = s.City AND dl.Location = s.Locatic
JOIN dim_restaurant AS dr ON dr.Restaurant_Name = s.Restaurant_Name
JOIN dim_category AS dc ON dc.Category = s.Category
JOIN dim_dish AS dsh ON dsh.Dish_Name = s.Dish_Name;
"""

# Execute the insert query
with engine.begin() as connection:
    connection.execute(text(insert_fact_swiggy_orders_query))
```

ERD Diagram: Star Schema

The Star Schema connects the fact table to dimensions via foreign keys. (Refer to attached image1.png for visual representation of the ERD.)

In [19]:
```
from IPython.display import Image, display

display(Image(filename="Swiggy_ERD.png"))
```



KPI Development - Basic and Advanced metrics

In [20]:

```
# Total Orders. Calculated the total number of orders from the fact table to get an overall me
total_orders_query = """
SELECT COUNT(*) AS Total_Orders
FROM fact_swiggy_orders;
"""

# Execute the query and load results into a DataFrame
total_orders_df = pd.read_sql(text(total_orders_query), engine)
total_orders_df
```

Out[20]:

	Total_Orders
0	6514233

In [21]:

```
# Total Revenue (INR Million). Calculated the total revenue in millions of INR to understand t
total_revenue_query = """
SELECT FORMAT(SUM(CONVERT(FLOAT, Price_INR)) / 1000000, 'N2') + ' INR Millions' AS Total_Rever
FROM fact_swiggy_orders;
"""
```

```
total_revenue_df = pd.read_sql(text(total_revenue_query), engine)
total_revenue_df
```

Out[21]:

	Total_Revenue
0	1,749.09 INR Millions

In [22]:

```
# Average Dish Price. Computed the average dish price to understand typical customer spending
avg_dish_price_query = """
SELECT FORMAT(AVG(CONVERT(FLOAT, Price_INR)), 'N2') + ' INR' AS Average_Dish_Price
FROM fact_swiggy_orders;
"""

avg_dish_price_df = pd.read_sql(text(avg_dish_price_query), engine)
avg_dish_price_df
```

Out[22]:

	Average_Dish_Price
0	268.50 INR

In [23]:

```
# Average Rating. Calculated the average rating across all orders to measure customer satisfa
avg_rating_query = """
SELECT AVG(Rating) AS Average_Rating
FROM fact_swiggy_orders;
"""

avg_rating_df = pd.read_sql(text(avg_rating_query), engine)
avg_rating_df
```

Out[23]:

	Average_Rating
0	4.341577

Deep-Dive Business Analysis

Date-Based Analysis

In [24]:

```
# Monthly Order Trends. Calculated total orders per month to analyze monthly order patterns ar
monthly_order_trends_query = """
SELECT
    d.Year, d.Month, d.Month_Name,
    COUNT(*) AS Total_Orders
FROM fact_swiggy_orders AS f
JOIN dim_date d ON f.date_id = d.date_id
GROUP BY d.Year, d.Month, d.Month_Name
ORDER BY COUNT(*) DESC;
"""

monthly_order_trends_df = pd.read_sql(text(monthly_order_trends_query), engine)
monthly_order_trends_df
```

Out[24]:

	Year	Month	Month_Name	Total_Orders
0	2025	1	January	837969
1	2025	8	August	832491
2	2025	5	May	831204
3	2025	7	July	822888
4	2025	4	April	811272
5	2025	3	March	805200
6	2025	6	June	804606
7	2025	2	February	768603

In [25]:

```
# Quarterly Order Trends. Calculated total orders per quarter to track seasonal trends and qu
quarterly_order_trends_query = """
SELECT
    d.Year, d.Quarter,
    COUNT(*) AS Total_Orders
FROM fact_swiggy_orders AS f
JOIN dim_date d ON f.date_id = d.date_id
GROUP BY d.Year, d.Quarter
ORDER BY COUNT(*) DESC;
"""

quarterly_order_trends_df = pd.read_sql(text(quarterly_order_trends_query), engine)
quarterly_order_trends_df
```

Out[25]:

	Year	Quarter	Total_Orders
0	2025	2	2447082
1	2025	1	2411772
2	2025	3	1655379

In [26]:

```
# Year-Wise Growth. Calculated total orders per year to evaluate annual growth and long-term
yearly_growth_query = """
SELECT
    d.Year,
    COUNT(*) AS Total_Orders
FROM fact_swiggy_orders AS f
JOIN dim_date d ON f.date_id = d.date_id
GROUP BY d.Year
ORDER BY COUNT(*) DESC;
"""

yearly_growth_df = pd.read_sql(text(yearly_growth_query), engine)
yearly_growth_df
```

Out[26]:

	Year	Total_Orders
0	2025	6514233

In [27]:

```
# Day-of-Week Patterns. Calculated total orders by weekday to identify patterns in customer o
day_of_week_query = """
SELECT
    DATENAME(WEEKDAY, d.Full_Date) AS Day_Name,
```

```
    COUNT(*) AS Total_Orders
FROM fact_swiggy_orders f
JOIN dim_date d ON f.date_id = d.date_id
GROUP BY DATENAME(WEEKDAY, d.Full_Date), DATEPART(WEEKDAY, d.Full_Date)
ORDER BY DATEPART(WEEKDAY, d.Full_Date);
"""

day_of_week_df = pd.read_sql(text(day_of_week_query), engine)
day_of_week_df
```

Out[27]:

	Day_Name	Total_Orders
0	Sunday	939477
1	Monday	909744
2	Tuesday	904629
3	Wednesday	933372
4	Thursday	938850
5	Friday	933372
6	Saturday	954789

Location-Based Analysis

In [28]:

```
# Top 10 Cities by Order Volume. I identify the ten cities with the highest order counts to u
top_cities_query = """
SELECT TOP 10
    l.City,
    COUNT(*) AS Total_Orders
FROM fact_swiggy_orders f
JOIN dim_location l ON l.location_id = f.location_id
GROUP BY l.City
ORDER BY COUNT(*) DESC;
"""

top_cities_df = pd.read_sql(text(top_cities_query), engine)
top_cities_df
```

Out[28]:

	City	Total_Orders
0	Bengaluru	662376
1	Mumbai	346731
2	Hyderabad	340164
3	Jaipur	339405
4	Lucknow	336336
5	New Delhi	336303
6	Ahmedabad	335775
7	Chandigarh	331980
8	Kolkata	331452
9	Chennai	331386

In [29]:
```
# Revenue Contribution by States. Calculated total revenue per state to assess which states co
state_revenue_query = """
SELECT TOP 10
    l.State,
    SUM(f.Price_INR) AS Total_Revenue
FROM fact_swiggy_orders f
JOIN dim_location l ON l.location_id = f.location_id
GROUP BY l.State
ORDER BY COUNT(*) DESC;
"""

state_revenue_df = pd.read_sql(text(state_revenue_query), engine)
state_revenue_df
```

Out[29]:

	State	Total_Revenue
0	Karnataka	1.800443e+08
1	Maharashtra	9.951392e+07
2	Telangana	9.971467e+07
3	Rajasthan	8.259351e+07
4	Uttar Pradesh	1.028729e+08
5	Delhi	9.336296e+07
6	Gujarat	9.291270e+07
7	Punjab	9.256473e+07
8	West Bengal	8.785305e+07
9	Tamil Nadu	8.720562e+07

Food Performance

In [30]:
```
# Top 10 Restaurants by Orders. I identify the top ten restaurants by revenue to analyze high
top_restaurants_query = """
SELECT TOP 10
    r.Restaurant_Name,
    SUM(f.Price_INR) AS Total_Revenue
FROM fact_swiggy_orders f
JOIN dim_restaurant r ON r.restaurant_id = f.restaurant_id
GROUP BY r.Restaurant_Name
ORDER BY COUNT(*) DESC;
"""

top_restaurants_df = pd.read_sql(text(top_restaurants_query), engine)
top_restaurants_df
```

Out[30]:

	Restaurant_Name	Total_Revenue
0	McDonald's	1.103010e+08
1	KFC	1.401002e+08
2	Burger King	6.271710e+07
3	Pizza Hut	7.039777e+07
4	Domino's Pizza	6.048852e+07
5	LunchBox - Meals and Thalis	3.633765e+07
6	Baskin Robbins - Ice Cream Desserts	2.839953e+07
7	Faasos - Wraps, Rolls & Shawarma	2.574710e+07
8	Olio - The Wood Fired Pizzeria	4.068012e+07
9	The Good Bowl	2.222032e+07

In [31]:

```
# Top Categories: I calculated total orders per cuisine category to identify popular food type
top_categories_query = """
SELECT
    c.Category,
    COUNT(*) AS Total_Orders
FROM fact_swiggy_orders f
JOIN dim_category c ON f.category_id = c.category_id
GROUP BY c.Category
ORDER BY Total_Orders DESC;
"""

top_categories_df = pd.read_sql(text(top_categories_query), engine)
top_categories_df
```

Out[31]:

	Category	Total_Orders
0	Recommended	795201
1	Desserts	118206
2	Main Course	98439
3	Beverages	88506
4	BURGERS	83754
...	...	...
4725	Meethe Me Kuch	33
4726	Mega burrito	33
4727	MINERAL WATER	33
4728	Mini burger (sliders)	33
4729	Miles Meal 1	33

4730 rows × 2 columns

In [32]:

```
# Most Ordered Dishes. I calculated order counts per dish to find the most popular menu items
most_ordered_dishes_query = """
SELECT
```

```
    d.Dish_Name,
    COUNT(*) AS Order_Count
FROM fact_swiggy_orders f
JOIN dim_dish d ON f.dish_id = d.dish_id
GROUP BY d.Dish_Name
ORDER BY Order_Count DESC;
"""

most_ordered_dishes_df = pd.read_sql(text(most_ordered_dishes_query), engine)
most_ordered_dishes_df
```

Out[32]:

	Dish_Name	Order_Count
0	Veg Fried Rice	10593
1	Choco Lava Cake	9999
2	Jeera Rice	8745
3	Paneer Butter Masala	8646
4	French Fries	8184
...	...	...
53983	Vegetable Minestrone Soup	33
53984	Vegetable Momo	33
53985	VEGETABLE MOMO FRIED	33
53986	Vegetable Momo In Chilli Gravy	33
53987	VEGETABLE MOMO STEAM	33

53988 rows × 2 columns

Cuisine Performance (Orders + Avg Rating)

I calculate total orders and average ratings per category to understand both popularity and customer satisfaction by cuisine type.

In [33]:
```
# Cuisine Performance (Orders + Avg Rating)
category_analysis_query = """
SELECT
    c.Category,
    COUNT(*) AS Total_Orders,
    AVG(f.Rating) AS Avg_Rating
FROM fact_swiggy_orders f
JOIN dim_category c ON f.category_id = c.category_id
GROUP BY c.Category
ORDER BY Total_Orders DESC;
"""

category_analysis_df = pd.read_sql(text(category_analysis_query), engine)
category_analysis_df
```

Out[33]:

	Category	Total_Orders	Avg_Rating
0	Recommended	795201	4.321782
1	Desserts	118206	4.371635
2	Main Course	98439	4.310191
3	Beverages	88506	4.368717
4	BURGERS	83754	4.324940
...	...	...	...
4725	Meethe Me Kuch	33	3.800000
4726	Mega burrito	33	4.400000
4727	MINERAL WATER	33	4.400000
4728	Mini burger (sliders)	33	4.400000
4729	Miles Meal 1	33	4.400000

4730 rows × 3 columns

Customer Spending Insights

In [34]:

```
# Orders by Price Ranges. Categorized orders into price buckets to understand spending patter
price_bucket_query = """
SELECT
    CASE
        WHEN CONVERT(FLOAT, Price_INR) < 100 THEN 'Under 100'
        WHEN CONVERT(FLOAT, Price_INR) BETWEEN 100 AND 199 THEN '100-199'
        WHEN CONVERT(FLOAT, Price_INR) BETWEEN 200 AND 299 THEN '200-299'
        WHEN CONVERT(FLOAT, Price_INR) BETWEEN 300 AND 399 THEN '300-399'
        WHEN CONVERT(FLOAT, Price_INR) BETWEEN 400 AND 499 THEN '400-499'
        ELSE '500+'
    END AS Price_Range,
    COUNT(*) AS Total_Orders
FROM fact_swiggy_orders
GROUP BY
    CASE
        WHEN CONVERT(FLOAT, Price_INR) < 100 THEN 'Under 100'
        WHEN CONVERT(FLOAT, Price_INR) BETWEEN 100 AND 199 THEN '100-199'
        WHEN CONVERT(FLOAT, Price_INR) BETWEEN 200 AND 299 THEN '200-299'
        WHEN CONVERT(FLOAT, Price_INR) BETWEEN 300 AND 399 THEN '300-399'
        WHEN CONVERT(FLOAT, Price_INR) BETWEEN 400 AND 499 THEN '400-499'
        ELSE '500+'
    END
ORDER BY Total_Orders DESC;
"""

price_bucket_df = pd.read_sql(text(price_bucket_query), engine)
price_bucket_df
```

Out[34]:

	Price_Range	Total_Orders
0	100-199	1854237
1	200-299	1800711
2	300-399	1028808
3	Under 100	884235
4	500+	534336
5	400-499	411906

Ratings Analysis : Computed the distribution of ratings to evaluate customer satisfaction levels across all orders.

In [35]:

```
# Ratings Distribution
ratings_distribution_query = """
SELECT
    Rating,
    COUNT(*) AS Rating_Count
FROM fact_swiggy_orders
GROUP BY Rating
ORDER BY COUNT(*) DESC;
"""

ratings_distribution_df = pd.read_sql(text(ratings_distribution_query), engine)
ratings_distribution_df
```

Out[35]:

	Rating	Rating_Count
0	4.4	2826186
1	4.3	452034
2	4.6	357720
3	4.5	328218
4	5.0	310233
5	4.7	299937
6	4.8	290697
7	4.2	271062
8	4.1	251427
9	4.9	188529
10	4.0	176418
11	3.9	132693
12	3.8	130878
13	3.7	89463
14	3.6	66330
15	3.5	62139
16	3.4	44220
17	3.3	42174
18	3.2	32868
19	3.0	24420
20	3.1	23133
21	2.8	18216
22	2.9	18117
23	2.7	14388
24	2.6	10428
25	2.5	9339
26	2.0	8151
27	2.4	7722
28	2.3	7491
29	2.2	6897
30	2.1	5313
31	1.5	2112
32	1.8	1815
33	1.9	1650
34	1.6	990

	Rating	Rating_Count
35	1.7	825