SWIGGY

Swiggy Case Study

Presented by: Nikhil Take.

Data Cleaning

SWIGGY

```sql
UPDATE Customers
SET email = 'Not Provided'
WHERE email IS NULL;

UPDATE Customers
SET phone_number = 'Unknown'
WHERE phone_number IS NULL;

UPDATE Customers
SET address = 'Not Available'
WHERE address IS NULL;

UPDATE Feedback
SET rating = 3
WHERE rating IS NULL;

DELETE FROM Orders
WHERE customer_id IS NULL;
```

🛠 **Step 1:** Data Cleaning Before analyzing the Swiggy database, the first step was to clean the data to ensure accuracy and consistency.
Here's what was done:

✅ **Handled Missing Values**:Replaced NULL emails with "Not Provided"Updated missing phone numbers with "Unknown"Filled empty addresses with "Not Available"

✅ **Removed Incomplete Records:**Deleted orders where customer_id was NULL to maintain data integrity.

✅ **Standardized Data:**Assigned a default rating of 3 for missing feedback to avoid errors in analysis.With clean and structured data, we can now move forward with insightful analysis! 🚀 .

1.Display all customers who live in 'Delhi'.

SWIGGY

```sql
SELECT *
FROM Customers
WHERE city = 'Delhi';
```

customer_id	name	email	phone_number	city	address
2	Rohini Verma	rohini.verma@yahoo.com	9823456789	Delhi	B-23, Saket
5	Manish Kumar	Not Provided	9834567890	Delhi	D-45, Lajpat Nagar
18	Sonali Mishra	Not Provided	9878345678	Delhi	N-54, Karol Bagh

🔍 Explanation:

1️⃣ WHERE city = 'Delhi' → Filters records to show only customers from Delhi.

2️⃣ SELECT * → Retrieves all columns of matching customers.

2. Find the average rating of all restaurants in 'Mumbai'.

SWIGGY

```sql
SELECT AVG(rating) AS average_rating
FROM Restaurants
WHERE city = 'Mumbai';
```

average_rating
4.300000

🔍 **Explanation:** 1️⃣ **WHERE city = 'Mumbai'** → Filters restaurants located in Mumbai.
2️⃣ **AVG(rating)** → Calculates the average rating of those restaurants.
3️⃣ **AS average_rating** → Renames the output column for better readability.

✅ **Result:** The average rating of all restaurants in Mumbai is 4.3 ⭐.

3. List all customers who have placed at least one order.

SWIGGY

```sql
SELECT DISTINCT c.*
FROM Customers c
JOIN Orders o ON c.customer_id = o.customer_id;
```

customer_id	name	email	phone_number	city	address
1	Amit Sharma	amit.sharma@gmail.com	9876543210	Mumbai	A-12, Andheri West
2	Rohini Verma	rohini.verma@yahoo.com	9823456789	Delhi	B-23, Saket
3	Rajesh Gupta	rajesh.gupta@gmail.com	Unknown	Mumbai	C-34, Colaba
4	Sneha Mehta	sneha.mehta@gmail.com	9812345678	Pune	Not Available
5	Manish Kumar	Not Provided	9834567890	Delhi	D-45, Lajpat Nagar
6	Priya Singh	priya.singh@hotmail.com	9845678901	Kolkata	E-56, Salt Lake
7	Vikas Reddy	vikas.reddy@gmail.com	9878901234	Chennai	F-67, T. Nagar
8	Anjali Patel	anjali.patel@yahoo.com	9890123456	Ahmedabad	Not Available
9	Suresh Nair	Not Provided	9801234567	Bangalore	G-78, Koramangala
10	Kavita Deshmukh	kavita.deshmukh@gmail.com	Unknown	Pune	H-89, Hinjewadi
11	Vivek Bhatt	vivek.bhatt@gmail.com	9810987654	Indore	I-90, Vijay Nagar
12	Meera Joshi	Not Provided	9823567890	Jaipur	J-11, Malviya Nagar
13	Pankaj Jain	pankaj.jain@yahoo.com	9834012345	Surat	Not Available
14	Nidhi Saxena	nidhi.saxena@gmail.com	9845123789	Lucknow	K-21, Gomti Nagar
15	Ashok Kumar	ashok.kumar@yahoo.com	Unknown	Chennai	L-32, Anna Nagar
16	Deepa Rao	deepa.rao@gmail.com	9856123456	Bangalore	Not Available
17	Karan Kapoor	karan.kapoor@gmail.com	9867234567	Noida	M-43, Sector 62
18	Sonali Mishra	Not Provided	9878345678	Delhi	N-54, Karol Bagh
19	Arjun Desai	arjun.desai@yahoo.com	Unknown	Mumbai	O-65, Bandra
20	Shweta Bansal	shweta.bansal@gmail.com	9889456789	Bhopal	Not Available
21	Rahul Chatterjee	rahul.chatterjee@gmail.com	9890567890	Kolkata	P-76, New Alipore
22	Neha Kaushik	neha.kaushik@yahoo.com	Unknown	Lucknow	Q-87, Aliganj
23	Ravi Singh	ravi.singh@gmail.com	9801678901	Mumbai	R-98, Andheri East

🔍 Explanation: 1 JOIN Orders o ON c.customer_id = o.customer_id → Connects customers with their orders.
2 SELECT DISTINCT c.* → Ensures each customer appears only once, even if they placed multiple orders.
✨ Now, you get a list of all customers who have placed at least one order! 🚀

4. Display the total number of orders placed by each customer.

```sql
SELECT c.customer_id, c.name, COUNT(o.order_id) AS total_orders
FROM Customers c
JOIN Orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id, c.name
ORDER BY total_orders DESC;
```

customer_id	name	total_orders
5	Manish Kumar	4
2	Rohini Verma	3
3	Rajesh Gupta	3
6	Priya Singh	3
7	Vikas Reddy	3
8	Anjali Patel	3
14	Nidhi Saxena	3
15	Ashok Kumar	3
18	Sonali Mishra	3
1	Amit Sharma	2
4	Sneha Mehta	2
10	Kavita Desh...	2
11	Vivek Bhatt	2
12	Meera Joshi	2
13	Pankaj Jain	2
16	Deepa Rao	2
19	Arjun Desai	2
20	Shweta Bansal	2
21	Rahul Chatte...	2
22	Neha Kaushik	2
23	Ravi Singh	2
9	Suresh Nair	1
17	Karan Kapoor	1

🔍 **Explanation:**

1. JOIN Orders o ON c.customer_id = o.customer_id → Links customers with their orders.
2. COUNT(o.order_id) AS total_orders → Counts the total number of orders placed by each customer.
3. GROUP BY c.customer_id, c.name → Groups results by each customer to get individual order counts.
4. ORDER BY total_orders DESC → Sorts customers by order count in descending order, showing the most active customers first.

5. Find the total revenue generated by each restaurant.

```
FROM Restaurants r
JOIN Orders o ON r.restaurant_id = o.restaurant_id
GROUP BY r.restaurant_id, r.name
ORDER BY total_revenue DESC;
```

SWIGGY

restaurant_id	name	total_revenue
3	Biryani House	5300.00
19	Awadhi Zaika	4150.00
12	Flavours of Bengal	4050.00
10	Andhra Spice	4050.00
4	Curry Pot	3200.00
13	South Treat	2950.00
9	Gujarat Express	2550.00
17	Chaat Junction	2150.00
7	Coastal Delight	2100.00
15	Rajasthani Rasoi	2100.00
18	Maharashtrian Ma...	2050.00
8	Veggie Delight	1600.00
14	The Great Indian ...	1600.00
2	Tandoori Flames	1200.00
1	Spice of India	1100.00
16	Kerala Kitchen	950.00
11	Punjabi Tadka	900.00
6	Royal Biryani	650.00
5	Taste of Punjab	600.00

🔍 **Explanation:**

1. **JOIN Orders o ON r.restaurant_id = o.restaurant_id →** Links the Restaurants table with the Orders table to match each restaurant with its orders.
2. **SUM(o.total_amount) AS total_revenue →** Calculates the total revenue for each restaurant by summing up all order amounts.
3. **GROUP BY r.restaurant_id, r.name →** Groups the results by restaurant ID and restaurant name to compute the revenue for each restaurant individually.
4. **ORDER BY total_revenue DESC →** Sorts the results in descending order, displaying the highest-earning restaurants first.

6. Find the top 5 restaurants with the highest average rating.

SWIGGY

```sql
SELECT restaurant_id, name, rating AS average_rating
FROM Restaurants
ORDER BY rating DESC
LIMIT 5;
```

restaurant_id	name	average_rating
3	Biryani House	4.80
22	Paradise Biryani	4.80
30	Lucknowi Nawabi	4.70
6	Royal Biryani	4.70
12	Flavours of Bengal	4.60

🔍 Explanation:

1. SELECT restaurant_id, name, rating → Retrieves the restaurant ID, name, and rating from the Restaurants table.
2. FROM Restaurants → Specifies that we are selecting data from the Restaurants table.
3. ORDER BY rating DESC → Sorts the restaurants by highest rating first, ensuring the best-rated restaurants appear at the top.
4. LIMIT 5 → Restricts the output to only the top 5 restaurants with the highest ratings.

7. Display all customers who have never placed an order.

SWIGGY

```sql
SELECT c.customer_id, c.name
FROM Customers c
LEFT JOIN Orders o ON c.customer_id = o.customer_id
WHERE o.order_id IS NULL;
```

customer_id	name
24	Sonal Kaur
25	Vivek Malhotra
26	Divya Iyer
27	Rakesh Yadav
28	Mona Sharma
29	Sudha Pillai
30	Gaurav Khanna

🔍 **Explanation:** 1 LEFT JOIN Orders o ON c.customer_id = o.customer_id → Retrieves all customers and matches them with their orders (if any).

2 WHERE o.order_id IS NULL → Filters out customers who do not have any matching orders, meaning they have never placed an order.

3 SELECT c.customer_id, c.name → Displays only the customer ID and name.

8. Find the number of orders placed by each customer in 'Mumbai'.

SWIGGY

```
FROM Customers c
JOIN Orders o ON c.customer_id = o.customer_id
WHERE c.city = 'Mumbai'
GROUP BY c.customer_id, c.name
ORDER BY total_orders DESC;
```

customer_id	name	total_orders
3	Rajesh Gupta	3
1	Amit Sharma	2
19	Arjun Desai	2
23	Ravi Singh	2

🔍 Explanation: 1 JOIN Orders o ON c.customer_id = o.customer_id → Links customers with their orders.

2 WHERE c.city = 'Mumbai' → Filters only customers from Mumbai.

3 COUNT(o.order_id) AS total_orders → Counts the total number of orders placed by each customer.

4 GROUP BY c.customer_id, c.name → Groups the result by each customer.

5 ORDER BY total_orders DESC → Sorts customers by highest number of orders first.

9. Display all orders placed in the last 30 days..

SWIGGY

```sql
SELECT order_id, customer_id, restaurant_id, total_amount, order_date
FROM Orders
WHERE order_date >= CURDATE() - INTERVAL 30 DAY
ORDER BY order_date DESC;
```

	order_id	customer_id	restaurant_id	total_amount	order_date
*	NULL	NULL	NULL	NULL	NULL

🔍 Explanation:

1 WHERE order_date >= CURDATE() - INTERVAL 30 DAY → This filter only includes orders with an order_date within the last 30 days. Orders with dates older than this period, such as those from 2024, are not included in the results.

2 Older Dates vs. NULL Values:Older Dates: Orders placed in 2024 have valid dates, but since they don't fall within the last 30 days, they won't appear in the query output.NULL Values: If order_date is NULL, it indicates that no date was recorded. Such rows also won't satisfy the filter condition and thus won't be shown.

3 ORDER BY order_date DESC → The query sorts the results so that the most recent orders appear first.

10. List all delivery partners who have completed more than 1 delivery

SWIGGY

```sql
SELECT dp.partner_id, dp.name, COUNT(od.order_id) AS deliveries_completed
FROM DeliveryPartners dp
JOIN OrderDelivery od ON dp.partner_id = od.partner_id
GROUP BY dp.partner_id, dp.name
HAVING COUNT(od.order_id) > 1;
```

partner_id	name	deliveries_completed
4	Suresh Reddy	6
5	Anita Desai	4
2	Ravi Kumar	5
6	Rajesh Gupta	4
3	Priya Patel	3
9	Sneha Iyer	2
1	Amit Sharma	2
7	Sonia Agarwal	3
12	Reena Rao	2
8	Vikram Singh	2
13	Mohit Saini	2
16	Ritika Sharma	2

🔍 Explanation: 1 JOIN OrderDelivery od ON dp.partner_id = od.partner_id → Links each delivery partner from the DeliveryPartners table with their delivery records in the OrderDelivery table.

2 COUNT(od.order_id) AS deliveries_completed → Counts the total number of deliveries each partner has completed.

3 GROUP BY dp.partner_id, dp.name → Groups the results by each delivery partner to calculate the count individually.

4 HAVING COUNT(od.order_id) > 1 → Filters the groups to display only those partners who have completed more than one delivery.

11. Find the customers who have placed orders on exactly three different days.

SWIGGY

```
FROM Customers c
JOIN Orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id, c.name
HAVING COUNT(DISTINCT DATE(o.order_date)) = 3;
```

customer_id	name
2	Rohini Verma
6	Priya Singh
8	Anjali Patel
14	Nidhi Saxena
15	Ashok Kumar
18	Sonali Mishra

🔍 Explanation:

1️⃣ JOIN Orders o ON c.customer_id = o.customer_id → Links each customer with their orders.

2️⃣ GROUP BY c.customer_id, c.name → Groups the results by customer, so we calculate order counts for each individual customer.

3️⃣ COUNT(DISTINCT DATE(o.order_date)) → Counts the number of unique days (ignoring time) on which each customer placed an order.

4️⃣ HAVING COUNT(DISTINCT DATE(o.order_date)) = 3 → Filters to include only those customers who placed orders on exactly three different days.

12.Find the delivery partner who has worked with the most different customers.

SWIGGY

```sql
SELECT dp.partner_id, dp.name, COUNT(DISTINCT o.customer_id) AS different_customers
FROM DeliveryPartners dp
JOIN OrderDelivery od ON dp.partner_id = od.partner_id
JOIN Orders o ON od.order_id = o.order_id
GROUP BY dp.partner_id, dp.name
ORDER BY different_customers DESC
LIMIT 1;
```

partner_id	name	different_customers
4	Suresh Reddy	6

🔍 Explanation:

1️⃣ JOIN OrderDelivery od ON dp.partner_id = od.partner_id → Connects each delivery partner with their delivery records.

2️⃣ JOIN Orders o ON od.order_id = o.order_id → Links the deliveries to the corresponding orders to access customer information.

3️⃣ COUNT(DISTINCT o.customer_id) AS different_customers → Counts the number of unique customers each partner has served.

4️⃣ GROUP BY dp.partner_id, dp.name → Groups the data by each delivery partner.

5️⃣ ORDER BY different_customers DESC → Sorts the results to show the partner with the highest number of different customers at the top.

6️⃣ LIMIT 1 → Returns only the delivery partner with the most different customers..

13. Identify customers who have the same city and have placed orders at the same restaurants, but on different dates.

```sql
SELECT DISTINCT
    c1.customer_id AS customer1,
    c1.name AS customer1_name,
    c2.customer_id AS customer2,
    c2.name AS customer2_name,
    c1.city,
    o1.restaurant_id
FROM Orders o1
JOIN Orders o2
    ON o1.restaurant_id = o2.restaurant_id
    AND DATE(o1.order_date) <> DATE(o2.order_date)
JOIN Customers c1
    ON o1.customer_id = c1.customer_id
JOIN Customers c2
    ON o2.customer_id = c2.customer_id
WHERE c1.city = c2.city;
```

customer1	customer1_name	customer2	customer2_name	city	restaurant_id
2	Rohini Verma	2	Rohini Verma	Delhi	13
11	Vivek Bhatt	11	Vivek Bhatt	Indore	4
12	Meera Joshi	12	Meera Joshi	Jaipur	15
5	Manish Kumar	18	Sonali Mishra	Delhi	3
5	Manish Kumar	5	Manish Kumar	Delhi	3
18	Sonali Mishra	5	Manish Kumar	Delhi	3
19	Arjun Desai	23	Ravi Singh	Mumbai	8
23	Ravi Singh	19	Arjun Desai	Mumbai	8

🔍 Explanation: 1️⃣ JOIN Orders o2 ON o1.restaurant_id = o2.restaurant_id AND DATE(o1.order_date) <> DATE(o2.order_date)Matches Orders: This self-join on the Orders table finds pairs of orders made at the same restaurant.Different Dates: The condition DATE(o1.order_date) <> DATE(o2.order_date) ensures the orders occurred on different dates.
2️⃣ JOIN Customers c1 ON o1.customer_id = c1.customer_id & JOIN Customers c2 ON o2.customer_id = c2.customer_idLink Customers: These joins connect the orders to their respective customers, letting us access customer details (ID, name, city).
3️⃣ WHERE c1.city = c2.citySame City: This condition filters the results to include only pairs of customers who are from the same city.
4️⃣ SELECT DISTINCT ...Distinct Pairs: The query selects distinct pairs of customers (customer1 and customer2) along with their common city and the restaurant where they both placed orders on different dates..

Thank You for Viewing My Presentation!