Restaurants & Online Food Ordering Analysis

Introduction:

Our project is focused on analyzing customer behavior and restaurant performance within the context of online food ordering. This report aims to extract actionable insights from the dataset, with a primary focus on restaurants, customer information, and order details.

Dataset Selection:

The chosen dataset revolves around Online food Ordering featuring five distinct files: Restaurants.csv, RestaurantMenu.csv, OrderLeads.csv, customer.csv, cartitems.scv. This dataset provides a comprehensive look into customer meal choices, order values. It proves to be an ideal choice for our analytical endeavors, offering invaluable insights into consumer actions and habits.

Data Source Link: Online food ordering

Dataset Attributes:

Restaurants.csv:

Id (integer): Unique identifier for each restaurant
Name (string): Name of the restaurant
Score (float): Actual rating on a scale of 5
Rating (float): Number of reviews given by customers
Category (string): Categorization of the restaurant
Price Range (varchar): Indicates the affordability of the restaurant, where $ means affordable, $$ is costly, and an increase in $ signs indicates a higher price range
State Code (string): Code representing the state where the restaurant is located.

Restaurant_Menu.csv:

Restaurant id (Integer): Unique identifier for each restaurant
Category (varchar): Category of the food item on the menu
Name (varchar): Name of the food item
Description (varchar): Description of the food item
Price (varchar): Price of the food item.

Order leads.csv:

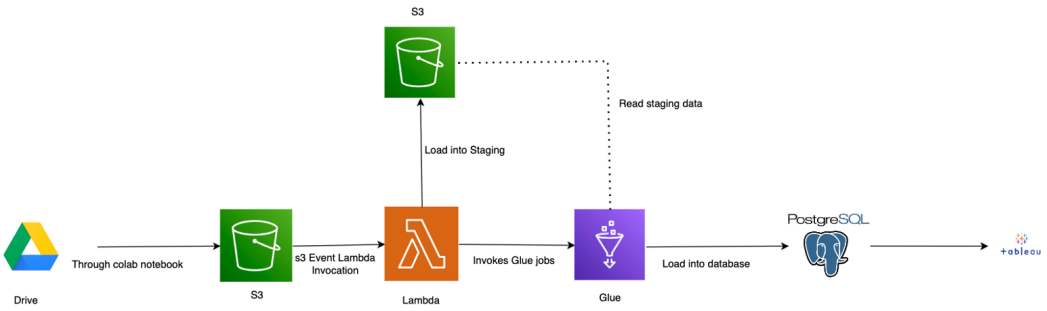
Order id (varchar): Unique identifier for each order
Date (date): Date when the order was placed
Order value (integer): Total value of the order

Converted (boolean): Indicates whether the order was placed successfully or not
Customer id (varchar): Unique identifier for each customer.

Customer_info.csv:
First Name (varchar): First name of the customer
Last Name (varchar): Last name of the customer
Email (varchar): Email address of the customer
Phone (varchar): Phone number of the customer
Address (varchar): Address of the customer
Gender (string): Gender of the customer
Age (integer): Age of the customer
Is Married (boolean): Indicates whether the customer is married or not.

Cart items.csv:
Order id (varchar): Unique identifier for each order
Restaurant id (integer): Unique identifier for each restaurant
Food item (varchar): Name of the food item
Quantity (integer): Quantity of the food item ordered.

Data Pipeline Implementation:



Data Ingestion:
Source: Raw data files are uploaded into an Amazon S3 bucket, providing a scalable and cost-effective storage solution.
—Uploading data into s3 bucket

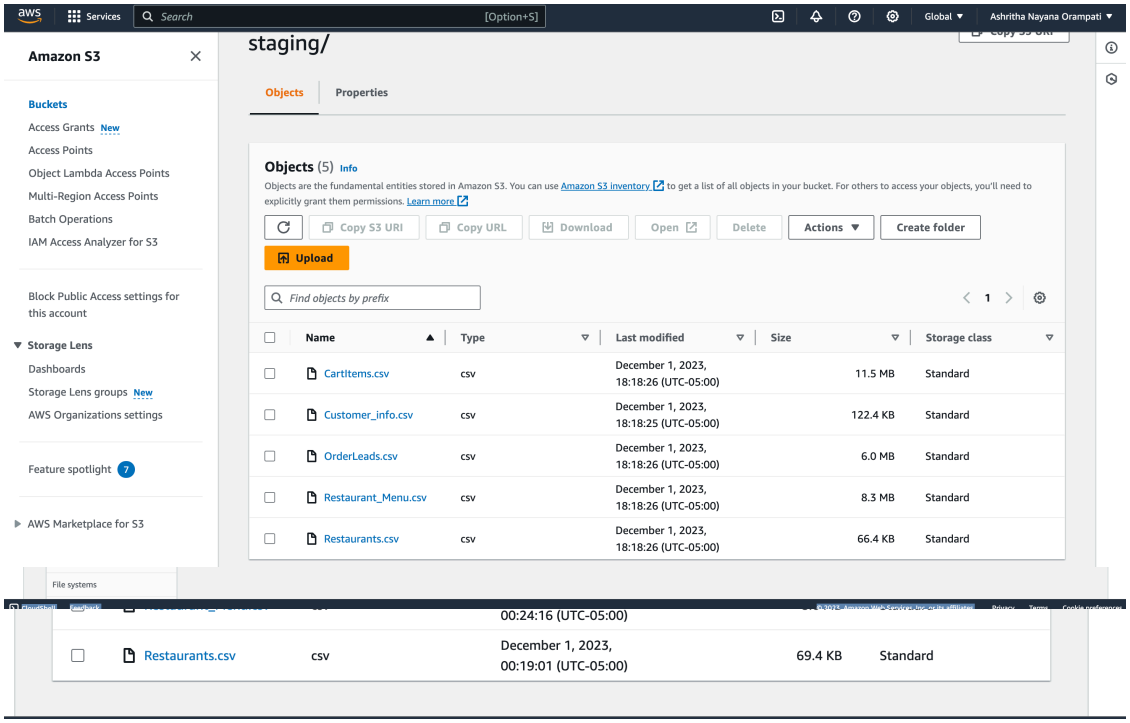
```
import boto3

file_list = ['CartItems.csv','Customer_info.csv','OrderLeads.csv','Restaurants.csv','Restaurant_Menu.csv']
# file_list = ['CartItems.csv']


aws_access_key = 'AKIA2NNK5YAT5RVQRFT3'
aws_secret_key = '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'
region_name = 'us-east-2'

for file in file_list:
  bucket_name = 'project-dw-ashritha'
  local_file_path = '/content/drive/MyDrive/DataWarehousingProject/UberEats/' + file
  s3_file_key = f'rawdata/{file}'

  # Create an S3 client
  s3 = boto3.client('s3', aws_access_key_id=aws_access_key, aws_secret_access_key=aws_secret_key, region_name=region_name)

  # Upload the file to S3
  s3.upload_file(local_file_path, bucket_name, s3_file_key)

  print(f'File uploaded to S3: s3://{bucket_name}/{s3_file_key}')
```

```
File uploaded to S3: s3://project-dw-ashritha/rawdata/CartItems.csv
File uploaded to S3: s3://project-dw-ashritha/rawdata/Customer_info.csv
File uploaded to S3: s3://project-dw-ashritha/rawdata/OrderLeads.csv
File uploaded to S3: s3://project-dw-ashritha/rawdata/Restaurants.csv
File uploaded to S3: s3://project-dw-ashritha/rawdata/Restaurant_Menu.csv
```

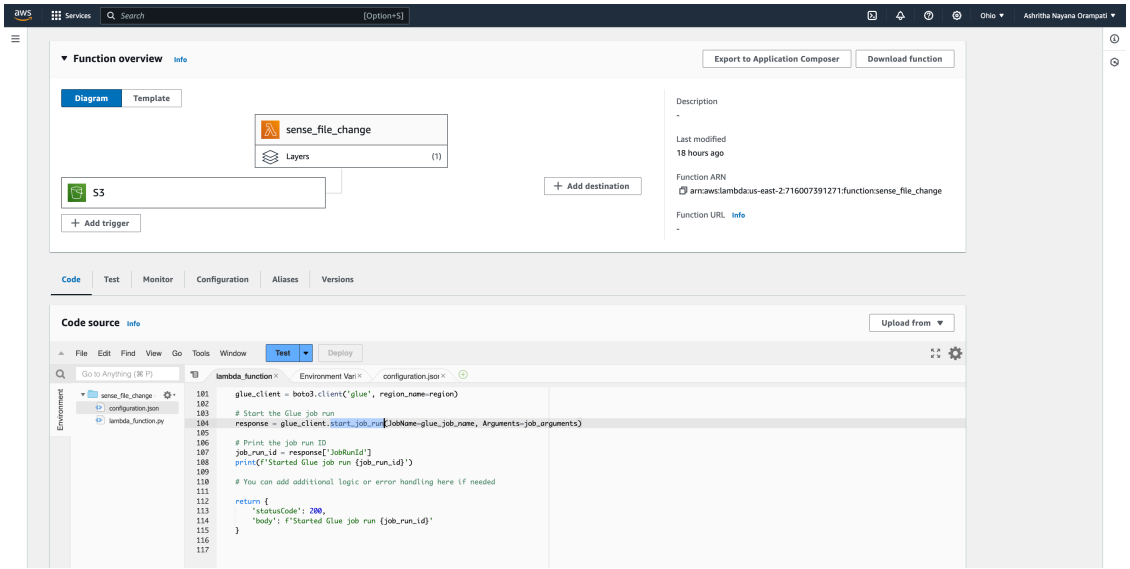
— Uploaded into S3 -rawdata folder

Lambda Event Trigger: An AWS Lambda function is configured to be triggered by events in the S3 bucket. This Lambda function acts as a starting point for the data pipeline.
—Automated Lambda invocation

—-Lambda places the intermediate transformed files into staging folder in s3 bucket which are picked by ETL jobs in the next step



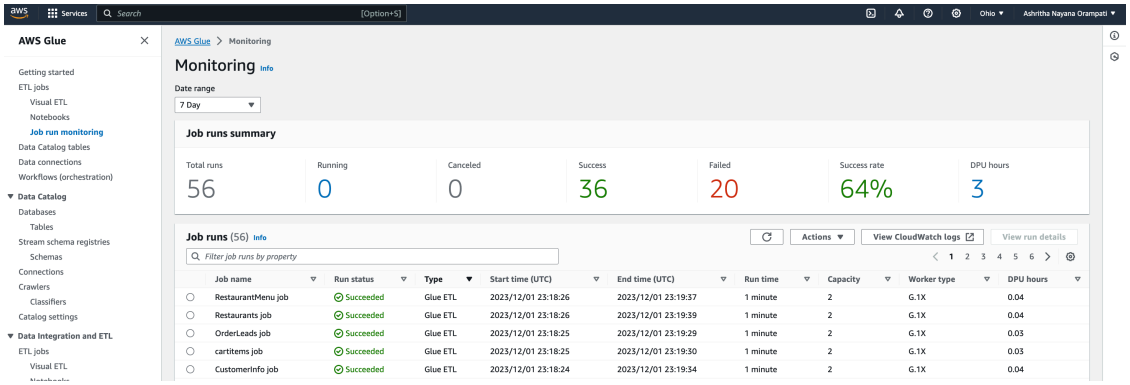
—-A glue ETL script is being triggered for each file by the same Lambda function



Data Transformation:

Glue ETL Job: The Lambda function, upon trigger, invokes an AWS Glue ETL (Extract, Transform, Load) job. This Glue job is responsible for performing various transformations on the raw data, including standardizing date formats, handling null values, and extracting relevant features. Glue provides a serverless environment for scalable and distributed ETL operations.

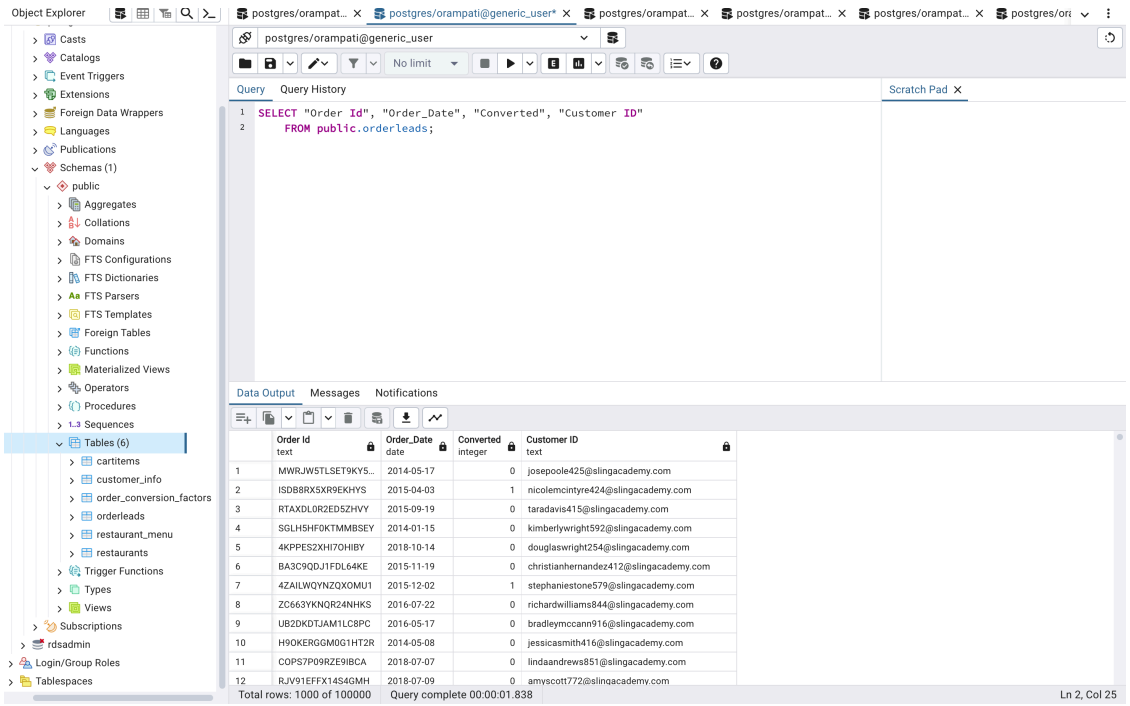
—-Created 5 different jobs for each dataset that has transformations relevant to that dataset and standardize into consumption format



Data Storage:

Target Database: The transformed data is loaded into a PostgreSQL database. The choice of PostgreSQL offers a relational database structure, ensuring data integrity and providing a robust foundation for analytical queries.

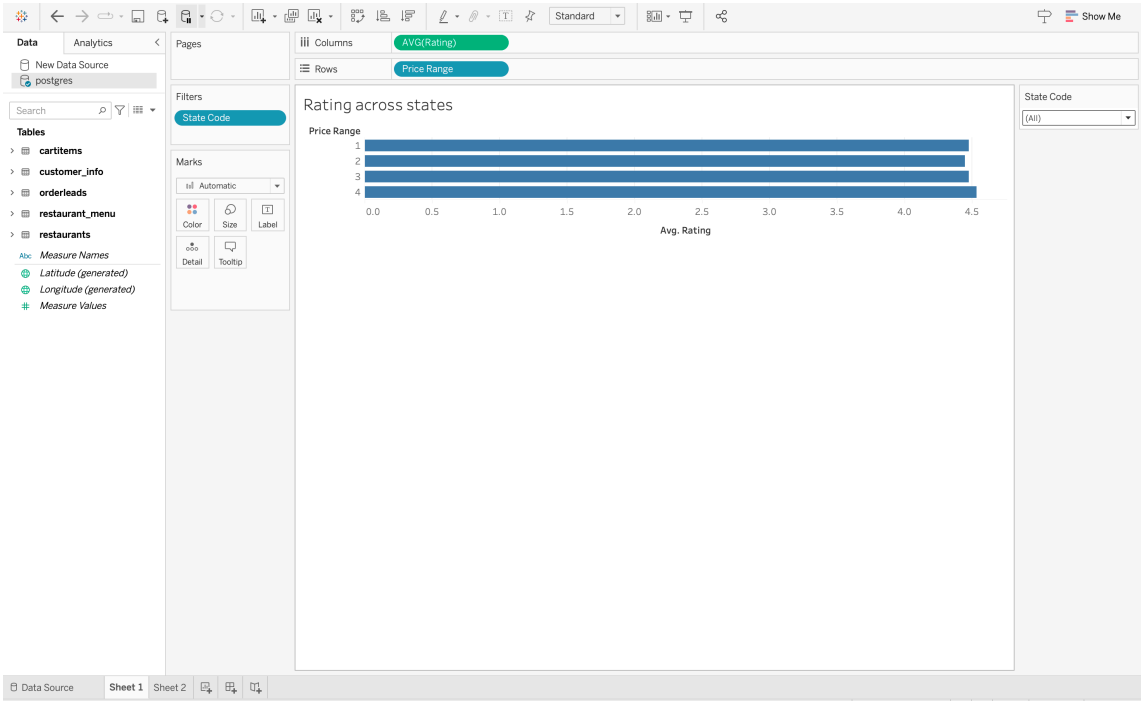
— then the transformed datasets are stored into PostgreSQL database



Connection to Tableau:

Tableau Integration: The PostgreSQL database is connected to Tableau, a powerful data visualization tool. Tableau is configured to pull data directly from the PostgreSQL database, enabling real-time access to the latest insights.

—PostgreSQL connected to Tableau



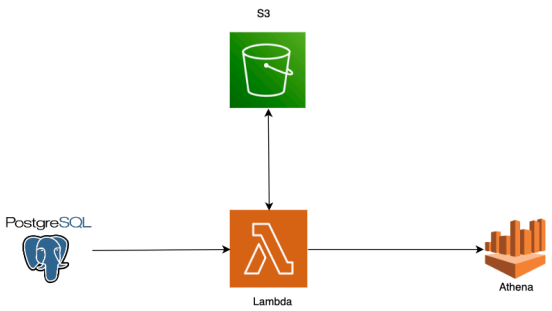
Automation and Orchestration:

Lambda for Automation: The Lambda function not only triggers the Glue ETL job but also serves as a component for automation. It ensures that the data pipeline is regularly updated by responding to new data uploads in the S3 bucket.

Scalability:

Glue for ETL Scalability: AWS Glue's serverless architecture allows the ETL process to scale automatically based on demand. It efficiently handles large datasets and adapts to varying workloads.

AWS Athena Integration with PostgreSQL using Lambda and Connector



Background:
Amazon Athena does not have direct support for connecting to PostgreSQL databases. To bridge this gap, a custom solution was implemented using AWS Lambda, which acts as an intermediary between Athena and PostgreSQL. This solution involves packaging a PostgreSQL connector within a Lambda function along with the necessary configurations.

Solution Overview:

- Lambda Function:
  - A serverless AWS Lambda function was created to act as a connector between Athena and PostgreSQL.
- PostgreSQL Connector:
  - A PostgreSQL connector was included in the Lambda function to facilitate communication with the PostgreSQL database.
- Configuration:
  - The Lambda function was configured with the necessary connection details, including PostgreSQL host, port, database name, credentials, and any other required parameters.
- Query Execution Flow:
  - When a query is submitted to Athena that involves data from the PostgreSQL database, the associated Lambda function is triggered.
  - The Lambda function, upon invocation, establishes a connection to the PostgreSQL database using the embedded connector.
  - It executes the SQL query on the PostgreSQL database and retrieves the result set.
  - The result set is then returned to Athena, allowing the query to be successfully executed.

Implementation Steps:

1. PostgreSQL Connector Integration:
   - The PostgreSQL connector (e.g., JDBC driver) was integrated into the Lambda deployment package.
2. Lambda Function Configuration:

- The Lambda function was configured with the required environment variables, including PostgreSQL connection details, credentials, and any other parameters.

3. IAM Role Configuration:
   - An AWS Identity and Access Management (IAM) role was created for the Lambda function, granting it the necessary permissions to access resources like Athena and PostgreSQL.
4. Lambda Triggers:
   - The Lambda function was associated with appropriate triggers, such as an Athena query execution event.
5. Testing and Validation:
   - The solution was thoroughly tested to ensure seamless data retrieval from PostgreSQL to Athena.

Benefits:

- Seamless Integration:
  - Achieved seamless integration between Athena and PostgreSQL, overcoming the absence of direct support.
- Serverless Architecture:
  - Leveraged the serverless architecture of Lambda, eliminating the need for provisioning and managing servers.
- Secure and Scalable:
  - Ensured secure access to PostgreSQL by configuring IAM roles, and the solution scales automatically based on demand.
- Flexible Querying:
  - Provided the ability to execute complex queries involving data from both Athena and PostgreSQL in a unified manner.

The implemented Lambda function with the embedded PostgreSQL connector serves as a robust solution for integrating Amazon Athena with PostgreSQL. This solution enhances the capabilities of Athena by allowing seamless querying of data residing in PostgreSQL, thus providing a unified and efficient analytical environment.

Analysis Dimensions:

Restaurant Performance:
Explore the distribution of restaurant ratings, the frequency of reviews, and the correlation between price range and customer satisfaction.

Customer Segmentation:
Analyze customer demographics, such as age, gender, and marital status. Identify patterns in customer behavior, including preferred restaurants and order quantities.

Menu Analysis:
Examine the popularity of different food categories, identify top-selling items, and analyze pricing strategies for optimal revenue.

Order Conversion:
Investigate factors influencing order conversion, such as order value, date trends, and customer information.

Cart Abandonment:
Explore instances where customers add items to the cart but do not convert orders. Gain insights into potential issues and improve the ordering process.

Analysis using SQL:

Order conversion

1)Factors Influencing Order Conversion
SELECT converted, AVG(price_val) AS avg_order_value, COUNT(DISTINCT order_id) AS total_orders
FROM ORDER_CONVERSION_FACTORS
GROUP BY converted;

#	converted	avg_order_value	total_orders
1	1	9.68	13659
2	0	9.67	66070

2)Date Trends
SELECT EXTRACT(MONTH FROM Order_Date) as order_month, COUNT(DISTINCT order_id) AS total_orders
FROM ORDER_CONVERSION_FACTORS
GROUP BY EXTRACT(MONTH FROM Order_Date)
ORDER BY total_orders desc ;

#	order_month	total_orders
1	12	6917
2	5	6782
3	1	6770
4	3	6768
5	7	6759
6	10	6685
7	8	6658
8	11	6618
9	9	6535
10	4	6526
11	6	6515
12	2	6196

3)Customer Information Impact--loss ratio in different categories

```
WITH overall_factors_temp AS (
   WITH gender_tmp AS (
      SELECT
         'gender' AS factor,
         gender,
         SUM(CASE WHEN converted = 1 THEN quantity * price_val ELSE 0 END) AS
gained_value,
         SUM(CASE WHEN converted = 0 THEN quantity * price_val ELSE 0 END) AS
lost_value
      FROM
         ORDER_CONVERSION_FACTORS
      GROUP BY
         gender
   ),
   married_tmp AS (
      SELECT
         'is_married' AS factor,
         CAST(is_married as VARCHAR) as is_married,
         SUM(CASE WHEN converted = 1 THEN quantity * price_val ELSE 0 END) AS
gained_value,
         SUM(CASE WHEN converted = 0 THEN quantity * price_val ELSE 0 END) AS
lost_value
```

```
    FROM
      ORDER_CONVERSION_FACTORS
    GROUP BY
      is_married
  ),
  age_tmp AS (
    SELECT
      'age_category' AS factor,
      age_category,
      SUM(CASE WHEN converted = 1 THEN quantity * price_val ELSE 0 END) AS
gained_value,
      SUM(CASE WHEN converted = 0 THEN quantity * price_val ELSE 0 END) AS
lost_value
    FROM
      ORDER_CONVERSION_FACTORS
    GROUP BY
      age_category
  )
  SELECT
    factor,
    gender AS factors_values,
    ((lost_value * 100) / (gained_value + lost_value)) AS lost_rate_overall
  FROM
    gender_tmp
  UNION ALL
  SELECT
    factor,
    is_married AS factors_values,
    ((lost_value * 100) / (gained_value + lost_value)) AS lost_rate_overall
  FROM
    married_tmp
  UNION ALL
  SELECT
    factor,
    age_category AS factors_values,
    ((lost_value * 100) / (gained_value + lost_value)) AS lost_rate_overall
  FROM
    age_tmp
)

SELECT * FROM overall_factors_temp order by lost_rate_overall desc;
```

#	factor	factors_values	lost_rate_overall
1	age_category	46-55	83.67
2	is_married	true	83.33
3	age_category	66-75	83.25
4	age_category	56-65	83.05
5	age_category	36-45	82.74
6	gender	female	82.74
7	gender	male	82.71
8	is_married	false	82.48
9	age_category	26-35	82.44
10	age_category	15-25	82.04
11	age_category	76-85	81.05

4)loss ratio in states

```
with temp_state as (SELECT
        'state' AS factor,
        state,
        SUM(CASE WHEN converted = 1 THEN quantity * price_val ELSE 0 END) AS
gained_value,
        SUM(CASE WHEN converted = 0 THEN quantity * price_val ELSE 0 END) AS
lost_value
    FROM
        ORDER_CONVERSION_FACTORS
    GROUP BY
        state)
SELECT
  factor,
  state AS factors_values,
  ((lost_value * 100) / (gained_value + lost_value)) AS lost_rate_overall
FROM
  temp_state order by lost_rate_overall desc limit 5;
```

#	factor	factors_values	lost_rate_overall
1	state	Arizona	86.85
2	state	Florida	85.95
3	state	Marshall Islands	85.93
4	state	Rhode Island	85.44
5	state	Wisconsin	85.24

5)Loss Ratio by food category

```
with temp_category as (SELECT
        'category' AS factor,
         category,
        SUM(CASE WHEN converted = 1 THEN quantity * price_val ELSE 0 END) AS
gained_value,
        SUM(CASE WHEN converted = 0 THEN quantity * price_val ELSE 0 END) AS
lost_value
    FROM
        ORDER_CONVERSION_FACTORS
    GROUP BY
        category)
SELECT
   factor,
   category AS factors_values,
   case when (lost_value!=0 and gained_value!=0) then ((lost_value * 100) / (gained_value +
lost_value))    ELSE null END AS lost_rate_overall
FROM
   temp_category order by lost_rate_overall desc limit 5
```

#	factor	factors_values	lost_rate_overall
1	category	Batter Cod	99.43
2	category	Chicken Speciality Pizza	99.14
3	category	Local Specials	98.89
4	category	Two Mexican Tacos with Rice & Beans	98.71
5	category	Combination Dinners	98.58

Menu Analysis

--Top selling items wrt revenue generated

1) Most value generated food item at each restaurant

```
with final_tmp as (
        with rest_tmp as (
                SELECT
                c."Restaurant Id" AS restaurant_id,
                        c."Food Item" AS food_item,
                        sum(COALESCE(c.quantity,0) * COALESCE(r."price_val",0)) as
total_value_rest_level
                FROM cartitems c
                  INNER JOIN orderleads o ON c."Order Id" = o."Order Id"
                  INNER JOIN restaurant_menu r ON c."Restaurant Id" = r."restaurant_id"
                and c."Food Item" = r.name
```

```
        WHERE o."Converted" = 1
        GROUP BY
          c."Restaurant Id", c."Food Item")
          Select restaurant_id, food_item, total_value_rest_level,
          row_number() OVER(PARTITION BY restaurant_id
                ORDER BY total_value_rest_level desc) as rank from rest_tmp)
select restaurant_id, food_item, total_value_rest_level from final_tmp where rank =1 order by
total_value_rest_level desc;
```

#	restaurant_id	food_item	total_value_rest_level
1	485	UPPAbaby® VISTA V2 Stroller in Greyson	10999.90
2	422	Strawberry Cake	5400.00
3	706	Sundae Party For 20-25	4509.18
4	551	Jus Relaxing Recess Kit	4189.38
5	737	75 Tenders	3799.62
6	124	Traditional Wings	3589.00
7	766	Family Fiesta Pack	3443.18
8	669	Lobster Mac and Cheese	3136.00
9	486	16 Count Assorted Minis	3022.60
10	708	Pita Family Meal	3019.50

2)TOP 5 value generated food item across all restaurants

```
with final_tmp as (
        with rest_tmp as (
                SELECT
                        c."Food Item" AS food_item,
                        sum(COALESCE(c.quantity,0) * COALESCE(r."price_val",0)) as
total_value_across_all
                FROM cartitems c
                  INNER JOIN orderleads o ON c."Order Id" = o."Order Id"
                  INNER JOIN restaurant_menu r ON c."Restaurant Id" = r."restaurant_id"
                  and c."Food Item" = r.name
                WHERE o."Converted" = 1
                GROUP BY c."Food Item")
                  Select food_item, total_value_across_all,
                  row_number() OVER(ORDER BY total_value_across_all desc) as rank
                                                        from rest_tmp)
select food_item, total_value_across_all from final_tmp where rank <=5 order by
total_value_across_all desc;
```

#	food_item	total_value_across_all
1	UPPAbaby® VISTA V2 Stroller in Greyson	10999.90
2	Traditional Wings	5983.30
3	Strawberry Cake	5887.34
4	BRITAX® B-Lively™/B-Safe® Gen2™ Single Travel System in Greystone	5849.87
5	Lobster Mac and Cheese	5056.00

3)Pricing Strategies:

```
SELECT
   rm.category,
   AVG(rm.price_val) AS avg_item_price
FROM
   restaurant_menu rm
GROUP BY
   rm.category;
```

#	category	avg_item_price
1	Pastries	3.16
2	Hot Dishes	18.16
3	Bundles	28.70
4	Top Sellers	7.99
5	Ben & Jerry's	7.99
6	Seafood	14.83

4)Popularity of Food Categories wrt number of orders

```
SELECT
   rm.category,
   COUNT(DISTINCT ci."Order Id") AS order_count
FROM
   restaurant_menu rm
LEFT JOIN
   cartitems ci ON rm.restaurant_id = ci."Restaurant Id"
GROUP BY
   rm.category
ORDER By order_count DESC;
```

#	category	order_count
1	Picked for you	72644
2	Sides	29145
3	Beverages	27918
4	Desserts	25944
5	Drinks	21226
6	Salads	20216
7	Appetizers	18785

Customer Segmentation:

1)Demographic Analysis

```
SELECT age_category, gender, is_married, COUNT(DISTINCT order_id) AS order_count
FROM ORDER_CONVERSION_FACTORS
GROUP BY age_category, gender, is_married;
```

#	age_category	gender	is_married	order_count
1	46-55	male	true	1748
2	76-85	male	true	865
3	56-65	male	false	4112
4	26-35	female	false	4907
5	76-85	female	false	1917
6	36-45	male	true	1895
7	26-35	male	true	1708
8	66-75	female	true	1537
9	66-75	male	true	1662
10	15-25	male	false	3033

2)Preferred Restaurants and Order Quantities

```
with customer_orders AS (
   SELECT ol."Order Id", ol."Order_Date", ol."Converted", ol."Customer ID" as cust_id,
ci."Restaurant Id" as restaurant_id, ci."Food Item",ci.quantity as quantity
   FROM orderleads ol INNER JOIN cartitems ci ON ol."Order Id" = ci."Order Id")
SELECT cust_id, COUNT(DISTINCT restaurant_id) AS unique_restaurants, SUM(quantity) AS
total_order_quantity
FROM customer_orders
GROUP BY cust_id;
```

#	cust_id	unique_restaurants	total_order_quantity
1	michaellopez745@slingacademy.com	95	1632
2	jessicacollins591@slingacademy.com	77	1173
3	michaelphelps198@slingacademy.com	85	1275
4	paigebrady311@slingacademy.com	72	1145
5	reneesmith495@slingacademy.com	97	1413
6	colleengarza896@slingacademy.com	75	1207
7	aprilcooper368@slingacademy.com	84	1245

3)Customer segmentaion by age

```
WITH cust_orders AS (
   SELECT
        ol."Order Id" as order_id, ct.quantity , rm.price_val,
      CASE
        WHEN age BETWEEN 15 AND 25 THEN '15-25'
        WHEN age BETWEEN 26 AND 35 THEN '26-35'
        WHEN age BETWEEN 36 AND 45 THEN '36-45'
        WHEN age BETWEEN 46 AND 55 THEN '46-55'
        WHEN age BETWEEN 56 AND 65 THEN '56-65'
        WHEN age BETWEEN 66 AND 75 THEN '66-75'
        WHEN age BETWEEN 76 AND 85 THEN '76-85'
        ELSE 'Other'
      END AS age_category
                FROM customer_info ci JOIN orderleads ol ON ci.email = ol."Customer ID"
                                                        JOIN cartitems ct ON ct."Order Id"
= ol."Order Id"
                                                        JOIN restaurant_menu rm ON
rm.restaurant_id = ct."Restaurant Id"
)
SELECT age_category, COUNT(DISTINCT(order_id)) as total_orders,
SUM(quantity * price_val) as total_value,
SUM(quantity * price_val)/COUNT(DISTINCT(order_id)) as avg_value_per_order
FROM cust_orders
group by age_category order by total_orders desc;
```

#	cust_id	unique_restaurants	total_order_quantity
1	michaellopez745@slingacademy.com	95	1632
2	jessicacollins591@slingacademy.com	77	1173
3	michaelphelps198@slingacademy.com	85	1275
4	paigebrady311@slingacademy.com	72	1145
5	reneesmith495@slingacademy.com	97	1413
6	colleengarza896@slingacademy.com	75	1207
7	aprilcooper368@slingacademy.com	84	1245

Restaurant Performance

1)Distribution of Ratings

```
SELECT rating, COUNT(*) AS rating_count
FROM ORDER_CONVERSION_FACTORS
GROUP BY rating;
```

#	rating	rating_count
1	4.1	9577
2	3.7	1170
3	3.6	976
4	5.0	3088
5	4.3	20503
6	4.7	22877
7	3.8	2270
8	3.4	280

2)frequrency of reviews

```
SELECT rating, AVG(reviews) AS avg_reviews
FROM restaurants
GROUP BY rating;
```

3)Correlation Between Price Range and Customer Satisfaction

```
SELECT price_val, AVG(rating) AS avg_rating
FROM ORDER_CONVERSION_FACTORS
GROUP BY price_val;
```

#	price_val	avg_rating
1	2.19	4.449789029535871
2	7.99	4.422096881220927
3	12.00	4.4483677298311575
4	2.89	4.529758713136728
5	3.99	4.418017018128055
6	5.15	4.4075075075074945
7	22.89	4.53314917127072
8	9.40	4.501379310344833
9	11.95	4.4749244712991105
10	12.74	4.385185185185186

Cart Abandonment

1)Instances of Cart Abandonment

```
SELECT COUNT(DISTINCT "Order Id") AS abandoned_orders
FROM cartitems
WHERE "Order Id" NOT IN (SELECT DISTINCT "Order Id" FROM orderleads WHERE
"Converted"=1);
```

#	abandoned_orders
1	76169

```
SELECT COUNT(DISTINCT "Order Id") AS total_orders_placed
FROM cartitems
WHERE "Order Id" NOT IN (SELECT DISTINCT "Order Id" FROM orderleads WHERE
“Converted"=0);
```

#	total_orders_placed
1	15726

Dashboard

Key Performance Indicators(KPI’s)

1. Number of Restaurants in Each State (Map):
KPI Name: State-wise Restaurant Density
Explanation:

- This KPI visually represents the density of restaurants across different states using a map.
- It helps gauge market saturation and potential expansion opportunities in specific regions.
- Higher density indicates a strong presence, while lower density may suggest opportunities for growth.

2. Number of Orders by Each Age Category (Pie Chart):
KPI Name: Order Distribution by Age
Explanation:

- This KPI illustrates the distribution of orders across different age categories using a pie chart.
- It provides insights into customer demographics and helps in tailoring marketing strategies.
- Higher percentages in specific age categories indicate key customer segments contributing to order volumes.

3. Price Value in Each Year by State (TreeMap):
KPI Name: Revenue Distribution Over Time and Geography
Explanation:

- This KPI showcases the distribution of revenue over different years and states using a TreeMap.
- It aids in understanding revenue trends and identifying states contributing the most.

- Allows for strategic decision-making based on revenue performance over time and geographical regions.

4. Most Valued Category (Bar Chart):
KPI Name: Top-Performing Menu Category
Explanation:
- This KPI highlights the most valued category from the restaurant menu using a bar chart.
- It identifies the category contributing the most to overall revenue.
- Enables targeted marketing efforts and strategic planning around the highest-performing menu category.

5. Top 10 Sold Food Items (Bubble Chart):
KPI Name: Top-Selling Food Items
Explanation:
- This KPI focuses on the top 10 sold food items using a bubble chart.
- It identifies popular food items and their corresponding quantities.
- Supports inventory management and marketing strategies by emphasizing high-demand items.

Online Food Ordering Analysis Dashboard: