Case Study: Umami Kitchen

Jordin Huang & Hannah Lin

I. Proposal

With many meal kit subscription services focusing on Western cuisine, this company has stepped up to address the growing demand for Asian cuisine. Umami Kitchen's Asian meal kit subscription service seeks to provide customers with delicious, authentic, and restaurant-quality Asian dishes at the comfort of their own homes. With pre-portioned ingredients and easy-to-follow recipes, these kits feature authentic recipes and high-quality ingredients, often not found at local grocery stores.

Influencers are key to promoting the meal kit service, using their social media platforms to showcase the convenience and variety of asian cuisine recipes the kit provides. This project will model how the company integrates customer preferences, supplier coordination, and influencer marketing to provide a sustainable, authentic, and high-quality meal kit subscription service.

Theory for Asian Meal Kit Subscription

The company specializes in delivering Asian meal kits through a subscription model. Each meal kit must be designed to cater to diverse customer preferences and dietary needs (e.g. vegan, gluten-free). To manage this; the company needs to record essential information about meals, customers, influencers, suppliers and orders.

For each meal kit (PRODUCT), the company must store: MEAL KIT ID to uniquely identify the kit, NAME for the dish, TYPE OF CUISINE to categorize the meals, RECIPE providing step-by-step instructions, FOOD ALLERGENS/DIETARY RESTRICTIONS to accommodate various health needs, SERVING SIZE recorded to ensure the kit means the serving requirements,INGREDIENTS list what is included in the kit, AVAILABILITY of each meal kit should be
noted to either be continuous or seasonal, CUSTOMER RATING (anonymous) will help asses meal popularity and quality.

The company needs to maintain a database of CUSTOMER. For each customer, it should record CUSTOMER ID to uniquely identify them, NAME, PHONE NUMBER, EMAIL, DELIVERY ADDRESS, BILLING INFORMATION should be securely stored, ALLERGIES/PREFERENCES, REFERRAL CODE for new customers (first order) to track marketing effectiveness, SUBSCRIPTION PLAN numbers of meal and servings per week, DELIVERY FREQUENCY weekly, biweekly, monthly, ORDER HISTORY .Influencers play a key role in promoting the meal kits, and the company needs to track their

information as well. Each influencer will have a NAME, INFLUENCER ID, GENDER, AGE, DOB recorded. The SOCIAL MEDIA ACCOUNTS will be monitored, along with the NUMBER OF FOLLOWERS they have. Each influencer will be assigned a REFERRAL CODE for tracking purposes, and their COMMISSION RATE for successful subscriptions generated through their promotions must be documented.

The company will also maintain a database of SUPPLIER. For each supplier, it will store a SUPPLIER ID, NAME, PRODUCT PRICES, ADDRESS, DELIVERY SCHEDULES/LEAD TIMES for product delivery should be recorded to ensure timely fulfillment of orders.
Each purchase needs to have a unique ORDER ID to track each transaction. Additionally, the system will have to record the CUSTOMER ID to link the order to the respective customer. The ORDER TIME, DELIVERY DATE, STATUS, and TRACKING DELIVERY INFORMATION are important pieces of logistic information to ensure each order is fulfilled in a timely fashion and the customer is informed of the order's progress. The PRODUCTS PURCHASED, QUANTITY , and PRODUCT PRICE are used for fulfillment and to provide a clear breakdown to the customer and system of what has been ordered.

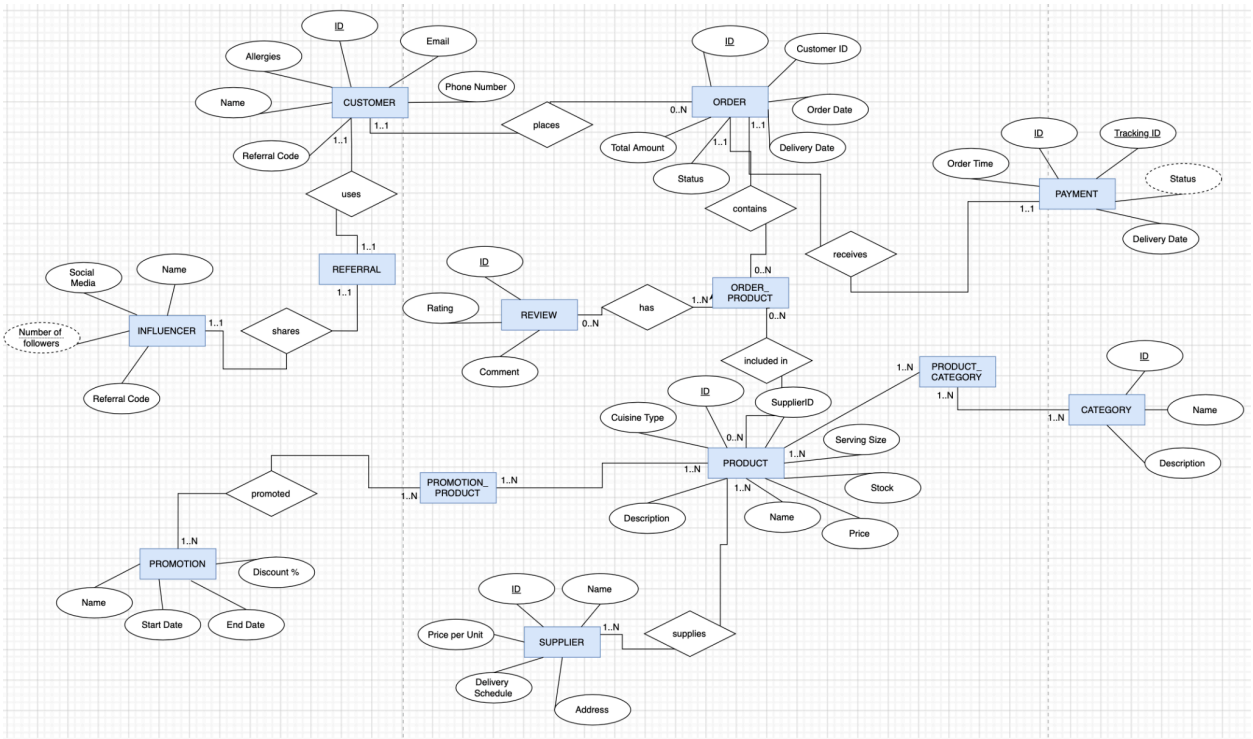
The process of tracking the company's financial metrics is very important. The company should monitor SUBSCRIPTION REVENUE generated from the meal kit service, as it is the primary source of income. Additionally, it is important to account for costs such as SUPPLIER COSTS, which include expenses associated with procuring less common and high-quality products from suppliers Another cost that needs to be recorded are INFLUENCER COMMISSION PAYOUTS, for influencers' promotional efforts promoting the meal kit. Lastly, SHIPPING AND DELIVERY COSTS. Oftentimes, if customers are first time subscribers, the company provides free shipping for their order to encourage sign-ups.

Other Requirements:

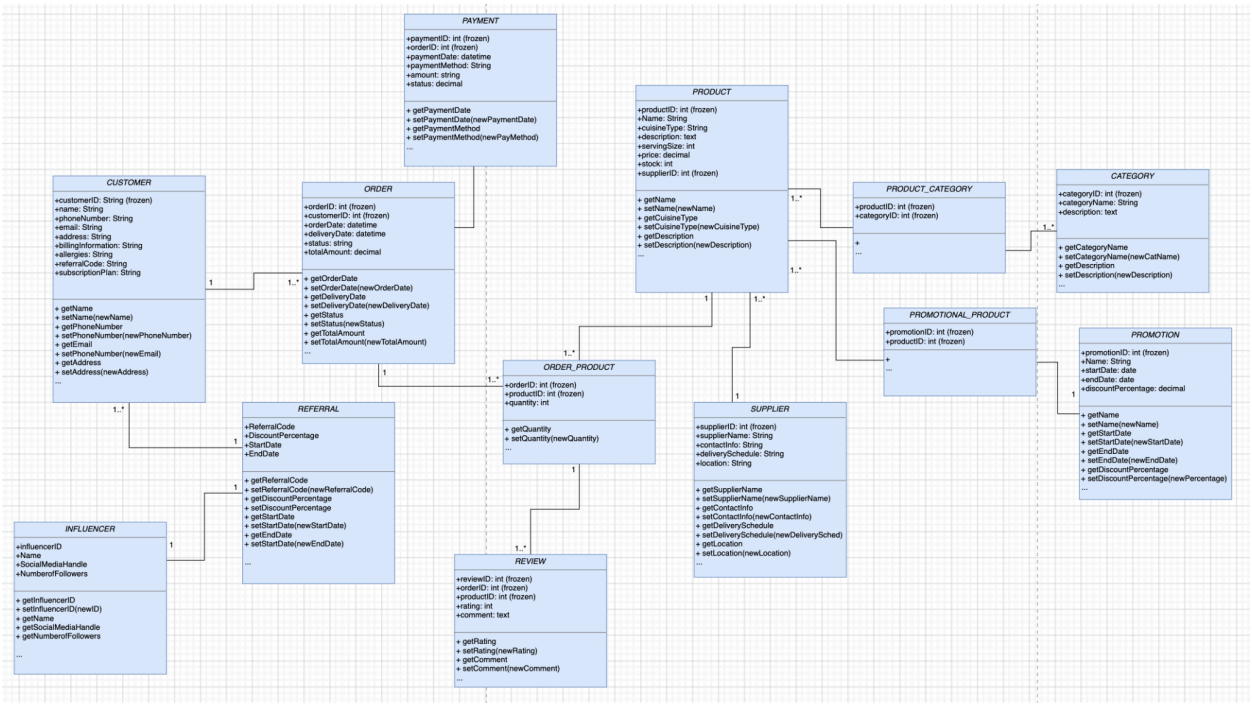
- A meal kit can be provided by multiple but at least one supplier; a supplier can provide ingredients for multiple but at least one meal kit
- A customer can only subscription to one meal plan at a time; the meal plan must be subscribed to by only one customer
- A customer can have zero and infinite meal kit deliveries; each delivery must belong to one customer
- A customer can skip zero or infinite deliveries
- A customer can have zero or infinite influencers who refer them; an influencer can refer multiple but at least one customer
- A customer can leave zero to infinite reviews for meal kits; a meal kit can have multiple
- An influencer can promote zero to infinite meal kits; a meal kit can be promoted by multiple but at least one influencer

II. Conceptual Modeling

A. EER Diagram



B. UML Diagram



III. Conceptual Model to Relational Model

Customer(CustomerID (PK), Name, Email, PhoneNumber, Address, Allergies, Subscription Plan, ReferralCode (FK))

- **CustomerID**: Primary key for customers. Unique ID for each customer. NOT NULL
- **Name**: Full name of the customer.
- **Email**: Email address of the customer.
- **PhoneNumber**: Contact phone number of the customer.
- **Address**: Residential address of the customer.
- **Allergies**: Details of allergies (if any) for the customer.
- **SubscriptionPlan**: The type of subscription plan the customer is on.
- **ReferralCode**: Foreign key linking to a referral code.

Influencer(InfluencerID (PK), Name, Social Media Handle, Number of Followers, Referral Code (FK))

- **InfluencerID**: Primary key for influencers. Unique ID for each influencer. NOT NULL
- **Name**: Full name of the influencer.
- **SocialMediaHandle**: Social media username/handle of the influencer.
- **NumberOfFollowers**: Total number of followers the influencer has.
- **ReferralCode**: Foreign key linking to a referral code.

Order(OrderID (PK), CustomerID(FK), Order Date, Delivery Date, Status, Total Amount)

- **OrderID**: Primary key for orders. Unique ID for each order. NOT NULL
- **CustomerID**: Foreign key linking to the customer placing the order. NOT NULL.
- **OrderDate**: Date when the order was placed.
- **DeliveryDate**: Date when the order is expected to be delivered.
- **Status**: Current status of the order (e.g., pending, delivered).
- **TotalAmount**: Total cost of the order.

Payment(PaymentID (PK), OrderID (FK), PaymentDate, PaymentMethod, Amount, Status)

- **PaymentID**: Primary key for payments. Unique ID for each payment. NOT NULL
- **OrderID**: Foreign key linking to the associated order. NOT NULL.
- **PaymentDate**: Date when the payment was made.
- **PaymentMethod**: Method of payment used (e.g., credit card, PayPal).
- **Amount**: Amount paid.
- **Status**: Payment status (e.g., successful, failed).

Category(CategoryID (PK), Category Name, Description)

- **CategoryID**: Primary key for categories. Unique ID for each category. NOT NULL
- **CategoryName**: Name of the category.
- **Description**: Description of the category.

Product(ProductID (PK), Name, Cuisine Type, Description, Serving Size, Price, Stock, SupplierID (FK))

- **ProductID**: Primary key for products. Unique ID for each product. NOT NULL
- **Name**: Name of the product.
- **CuisineType**: Type of cuisine the product belongs to.
- **Description**: Description of the product.
- **ServingSize**: Suggested serving size for the product.
- **Price**: Price of the product.
- **Stock**: Current stock of the product.
- **SupplierID**: Foreign key linking to the supplier providing the product. NOT NULL

Product_Category(ProductID (FK), CategoryID (FK))

- **ProductID**: Foreign key linking to the product. NOT NULL
- **CategoryID**: Foreign key linking to the category. NOT NULL

Supplier(SupplierID(PK), Name, Contact Info, Delivery Schedule, Location)

- **SupplierID**: Primary key for suppliers. Unique ID for each supplier. NOT NULL
- **Name**: Name of the supplier
- **ContactInfo**: Contact details of the supplier.
- **DeliverySchedule**: Regular delivery schedule of the supplier.
- **Location:** Location of the supplier.

OrderProduct(OrderID (PK), ProductID, Quantity)

- **OrderID**: Foreign key linking to the associated order. NOT NULL
- **ProductID**: Foreign key linking to the product in the order. NOT NULL
- **Quantity**: Quantity of the product ordered.

Promotion(PromotionID (PK), Name, Start Date, End Date, Discount Percentage)

- **PromotionID**: Primary key for promotions. Unique ID for each promotion. NOT NULL
- **Name**: Name of the promotion.
- **StartDate**: Date when the promotion starts.
- **EndDate**: Date when the promotion ends.
- **DiscountPercentage**: Percentage discount offered by the promotion.

Promotion_Product(PromotionID (FK), ProductID (FK))

- **PromotionID**: Foreign key linking to the promotion. NOT NULL

- **ProductID**: Foreign key linking to the product in the promotion. NOT NULL

Order_Product(OrderID (FK), ProductID (FK), Quantity)

- **OrderID**: Foreign key linking to the associated order. NOT NULL
- **ProductID**: Foreign key linking to the product in the order. NOT NULL
- **Quantity**: Quantity of the product ordered.

Referral(ReferralCode (PK), Discount Percentage, Start Date, End Date)

- **PromotionID**: Primary key for promotions. Unique ID for each promotion. NOT NULL
- **Name**: Name of the promotion.
- **StartDate**: Date when the promotion starts.
- **EndDate**: Date when the promotion ends.
- **DiscountPercentage**: Percentage discount offered by the promotion.

Review(ReviewID (PK), OrderID (FK), ProductID (FK), Rating, Comment)

- **PromotionID**: Foreign key linking to the promotion. NOT NULL
- **ProductID**: Foreign key linking to the product in the promotion. NOT NULL

IV. MySQL and NoSQL Implementation

SQL Implementations

1. Retrieve all products priced above $10 (Simple)

```
SELECT ProductID, Name, Price, Stock
FROM PRODUCT
WHERE Price > 10.00;
```

ProductID	Name	Price	Stock
4	Sushi Platter	14.99	20

2. Count the total number of orders and calculate the average order amount (Aggregate)

```
SELECT COUNT(OrderID) AS TotalOrders, AVG(TotalAmount) AS AverageOrderAmount
FROM `ORDER`;
```

TotalOrders	AverageOrderAmount
35	30.608571

3. List all orders along with customer names (Inner Join)

```
SELECT o.OrderID, c.Name AS CustomerName, o.OrderDate, o.TotalAmount
FROM `ORDER` o
INNER JOIN CUSTOMER c ON o.CustomerID = c.CustomerID;
```

OrderID	CustomerName	OrderDate	TotalAmount
1	Alice Wong	2024-01-01 10:00:00	29.97
2	Bob Lee	2024-01-03 15:00:00	14.99
3	Charlie Kim	2024-01-04 09:30:00	18.97
4	Diana Singh	2024-01-05 11:00:00	9.99
5	Ethan Chen	2024-01-06 12:45:00	14.99
6	Alice Wong	2024-02-01 14:00:00	45.50
7	Bob Lee	2024-02-03 11:00:00	20.75
8	Charlie Kim	2024-02-04 16:30:00	35.99
9	Diana Singh	2024-02-05 12:45:00	29.49
10	Ethan Chen	2024-02-06 10:15:00	15.99
11	Grace Lin	2024-02-07 09:00:00	40.25
12	Henry Yu	2024-02-08 18:30:00	19.49
13	Isla Tan	2024-02-09 20:15:00	50.00
14	Jack Lim	2024-02-10 08:45:00	28.99
15	Karen Lau	2024-02-11 13:00:00	38.99

4. List all customers and their orders (including customers who have not placed an order) (Outer Join)

```
SELECT
    c.CustomerID,
    c.Name AS CustomerName,
    o.OrderID,
    o.OrderDate,
    o.TotalAmount
FROM CUSTOMER c
LEFT JOIN `ORDER` o ON c.CustomerID = o.CustomerID;
```

CustomerID	CustomerName	OrderID	OrderDate	TotalAmount
7	Henry Yu	32	2024-02-28 20:45:00	22.99
8	Isla Tan	13	2024-02-09 20:15:00	50.00
8	Isla Tan	23	2024-02-19 20:00:00	60.00
8	Isla Tan	33	2024-03-01 21:30:00	58.50
9	Jack Lim	14	2024-02-10 08:45:00	28.99
9	Jack Lim	24	2024-02-20 07:30:00	33.75
9	Jack Lim	34	2024-03-02 09:15:00	30.00
10	Karen Lau	15	2024-02-11 13:00:00	38.99
10	Karen Lau	25	2024-02-21 14:15:00	44.99
10	Karen Lau	35	2024-03-03 11:00:00	50.00
11	Leo Zhang	NULL	NULL	NULL
12	Mia Chen	NULL	NULL	NULL
13	Noah Tan	NULL	NULL	NULL

5. Find customers who have spent more than the average order total. (Nested)

```sql
SELECT
    c.CustomerID,
    c.Name AS CustomerName,
    SUM(o.TotalAmount) AS TotalSpent
FROM CUSTOMER c
JOIN `ORDER` o ON c.CustomerID = o.CustomerID
GROUP BY c.CustomerID, c.Name
HAVING SUM(o.TotalAmount) > (
    SELECT AVG(TotalAmount)
    FROM `ORDER`
);
```

CustomerID	CustomerName	TotalSpent
1	Alice Wong	124.72
2	Bob Lee	73.74
3	Charlie Kim	128.21
4	Diana Singh	62.97
5	Ethan Chen	108.97
6	Grace Lin	110.50
7	Henry Yu	66.97
8	Isla Tan	168.50
9	Jack Lim	92.74
10	Karen Lau	133.98

6. Find all orders where the total amount is greater than the average order total (Correlated)

```sql
SELECT c.CustomerID, c.Name
FROM CUSTOMER c
WHERE EXISTS (
    SELECT 1
    FROM `ORDER` o
    WHERE o.CustomerID = c.CustomerID
);
```

OrderID	TotalAmount
6	45.50
8	35.99
11	40.25
13	50.00
15	38.99
18	32.50
20	42.99
23	60.00
24	33.75
25	44.99
28	40.75
30	35.00
31	45.25
33	58.50
35	50.00

7. List all customers who have placed at least one order (Exists)

```sql
SELECT c.CustomerID, c.Name
FROM CUSTOMER c
WHERE EXISTS (
    SELECT 1
    FROM `ORDER` o
    WHERE o.CustomerID = c.CustomerID
);
```

CustomerID	Name
1	Alice Wong
2	Bob Lee
3	Charlie Kim
4	Diana Singh
5	Ethan Chen
6	Grace Lin
7	Henry Yu
8	Isla Tan
9	Jack Lim
10	Karen Lau

8. Combine two queries: a list of customer names and a list of influences names (Union)

```sql
SELECT Name AS PersonName, 'Customer' AS Role
FROM CUSTOMER
UNION
SELECT Name AS PersonName, 'Influencer' AS Role
FROM INFLUENCER;
```

PersonName	Role
Elizabeth Lin	Customer
Elijah Ho	Customer
Aria Wu	Customer
Carter Wong	Customer
Scarlett Zh...	Customer
Alice Chan	Influencer
John Smith	Influencer
Emily Davis	Influencer
Michael Brown	Influencer
Sophia Wilson	Influencer

9. Retrieve each product's total sales (quantity ordered) using a subquery in SELECT clause

```sql
SELECT p.ProductID, p.Name, p.Price,
    (SELECT SUM(op.Quantity)
     FROM ORDER_PRODUCT op
     WHERE op.ProductID = p.ProductID) AS TotalQuantitySold
FROM PRODUCT p;
```

ProductID	Name	Price	TotalQuantitySold
1	Spring Rolls	4.99	5
2	Pad Thai	9.99	4
3	Miso Soup	3.99	5
4	Sushi Platter	14.99	5
5	Kimchi Fried Rice	7.99	5
6	Pho	8.99	3
7	Dim Sum Platter	8.99	4
8	Mango Sticky Rice	7.99	4
9	Bubble Tea	4.99	6
10	Satay Skewers	6.99	8

MongoDB NoSql

1. Find all customers with a specific allergy

```
db.customer.find({ Allergies: "Peanuts" });
```


```
{
  _id: ObjectId('675a980406256948393b4837'),
  CustomerID: 6,
  Name: 'Grace Lin',
  Email: 'grace@example.com',
  PhoneNumber: 1122334455,
  Address: '222 Cedar St, Kuala Lumpur',
  Allergies: 'Peanuts',
  SubscriptionPlan: 'Monthly',
  ReferralCode: 'REF001'
}
{
  _id: ObjectId('675a980406256948393b483f'),
  CustomerID: 14,
  Name: 'Olivia Ho',
  Email: 'olivia@example.com',
  PhoneNumber: 1122331122,
  Address: '444 Orange Rd, Tokyo',
  Allergies: 'Peanuts',
  SubscriptionPlan: 'Weekly',
  ReferralCode: 'REF001'
}
{
  _id: ObjectId('675a980406256948393b4847'),
  CustomerID: 22,
  Name: 'Hannah Tan',
  Email: 'hannah.tan@example.com',
  PhoneNumber: 7236176110,
  Address: '45 Sakura Ln, Kuala Lumpur',
  Allergies: 'Peanuts',
  SubscriptionPlan: 'Weekly',
  ReferralCode: 'REF003'
```

2. Count the Number of Customers for Each Subscription Plan

```
db.customer.aggregate([
  { $group: { _id: "$SubscriptionPlan", count: { $sum: 1 } } },
  { $sort: { count: -1 } }
]);
```

```
{
  _id: 'Monthly',
  count: 22
}
{
  _id: 'Weekly',
  count: 19
}
{
  _id: 'Bi-Weekly',
  count: 9
}
```

3. Find the total sales per customer for delivered orders

```
db.order.aggregate([
  {
    $match: { Status: "Delivered" } // Filter only delivered orders
  },
  {
    $group: {
      _id: "$CustomerID", // Group by CustomerID
      totalSpent: { $sum: "$TotalAmount" }, // Sum up the total amount spent
      totalOrders: { $count: {} } // Count the number of delivered orders
    }
  },
  {
    $lookup: { // Join with the customers collection
      from: "customer",
      localField: "_id",
      foreignField: "CustomerID",
      as: "customerDetails"
    }
  },
  {
    $unwind: "$customerDetails" // Flatten the joined customer details
  },
  {
    $project: { // Select and format the output fields
      _id: 0,
      CustomerID: "$_id",
      Name: "$customerDetails.Name",
      Email: "$customerDetails.Email",
      TotalSpent: "$totalSpent",
      TotalOrders: "$totalOrders"
    }
  },
  {
    $sort: { TotalSpent: -1 } // Sort by total spent in descending order
  }
]);
```

```
CustomerID: 8,
Name: 'Isla Tan',
Email: 'isla@example.com',
TotalSpent: 168.5,
TotalOrders: 3

CustomerID: 10,
Name: 'Karen Lau',
Email: 'karen@example.com',
TotalSpent: 133.98000000000002,
TotalOrders: 3

CustomerID: 3,
Name: 'Charlie Kim',
Email: 'charlie@example.com',
TotalSpent: 128.21,
TotalOrders: 4

CustomerID: 6,
Name: 'Grace Lin',
Email: 'grace@example.com',
TotalSpent: 110.5,
TotalOrders: 3
```

V. Access to Database via Python

To be able to obtain access to our MySQL database we need to install and import pymysql and os libraries. Pymysql is used to create a connection between python and a MySQL database. The os library provides the ability to interact with one's operating system. The remaining packages, numpy, pandas, seaborn, and matplotlib.pyplot are used for data analysis and visualization.

[1]:
```
#!pip install mysql-connector-python
```

```
Collecting mysql-connector-python
  Downloading mysql_connector_python-9.0.0-py2.py3-none-any.whl (372 kB)
     |                | 372 kB 2.2 MB/s eta 0:00:01
Installing collected packages: mysql-connector-python
Successfully installed mysql-connector-python-9.0.0
```

[118]:
```
#!pip install pymysql
```

```
Requirement already satisfied: pymysql in ./opt/anaconda3/lib/python3.8/site-
packages (1.1.1)
```

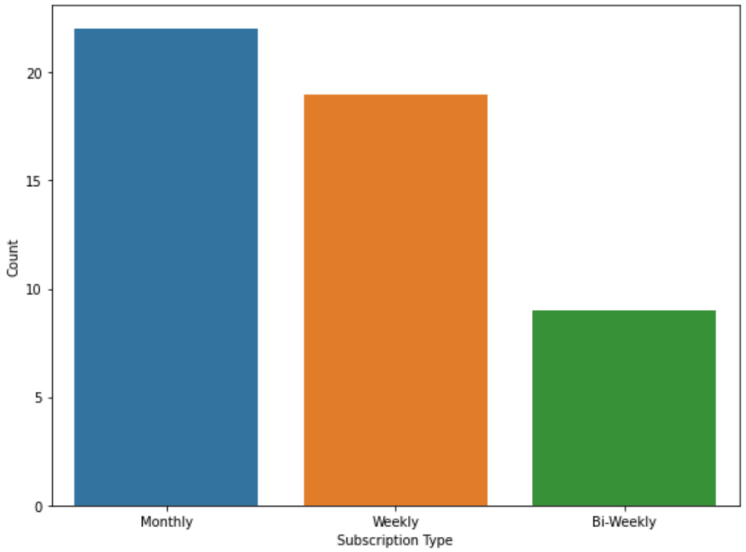
[89]:
```
#import necessary packages
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import mysql.connector
import pymysql
import os
```

[101]:
```
#connect to mySQL database
host= os.getenv('Local')
connection= pymysql.connect(
    host=host,
    port=int (3306),
    user= "root",
    password='3Ku5(7osEvSk',
    database="uk_database",
    charset='utf8mb4')
```

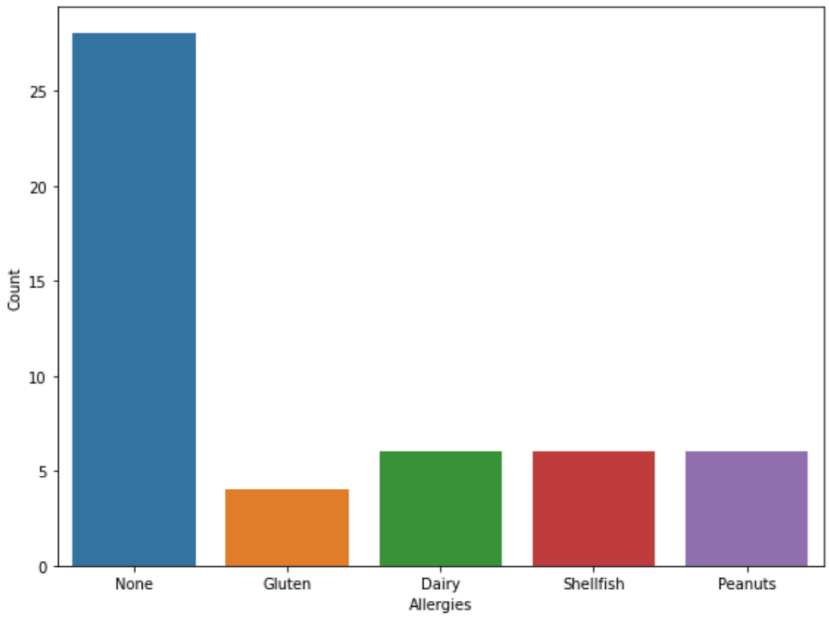
[102]:
```
customer = pd.read_sql('SELECT * FROM uk_database.customer',connection)
```

Something we wanted to explore was the most popular subscription plan amongst customers. As we can see in the figure below the most popular plan is the Monthly subscription. However, a larger number of customers have the Weekly or Bi-Weekly subscription, which indicates that customers prefer more frequent deliveries and possibly reflects that they prefer more flexible plans. This information helps identify potential areas for targeted promotions or adjustment to the subscription plans.

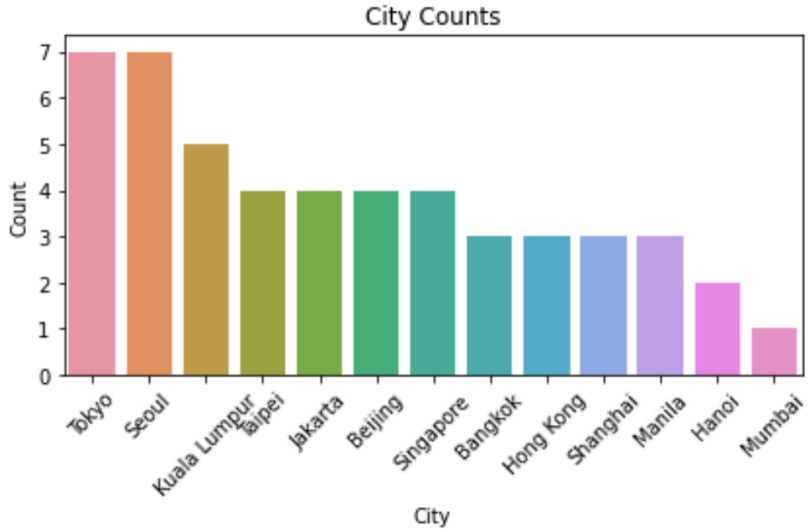
[107]:
```
#Explore most popular subscription plan
plt.figure(figsize=(8, 6))

sns.countplot(x='SubscriptionPlan', data=customer)

plt.tight_layout()
plt.xlabel('Subscription Type')
plt.ylabel('Count')
plt.show()
```



Below is the distribution of allergies among customers. The majority of customers do not report having any allergies which means a large proportion of customers can consume general offerings without any restrictions. Allergies such as gluten, dairy, shellfish, and peanuts are less common, but noticeable. This highlights the need to have clear allergen labeling to accommodate their needs. It also may indicate a niche group the company could cater to. While the majority do not have allergies, offering allergen-free options can help appeal to a wider consumer base.



Consumer geographical data can be crucial for businesses as it provides insight into where their customers are located and helps companies like Umami Kitchen make data-driven decisions to improve their business and increase profit. It appears that Umami Kitchen's customers are located primarily in Asian countries. Top cities are Tokyo and Seoul and less popular are Hanoi and Mumbai. If Umami Kitchen wants to expand their customer base in their top cities they can have more marketing and campaigns in Tokyo or Seoul. Hanoi and Mumbai have the least amount of customers, which presents an opportunity for Umami Kitchen to investigate the reasons behind the low customer numbers. It all depends on what their goals as a business are.



Knowing the status of orders is crucial for a business. If there is an uptick of cancelled orders, it indicates that there potentially is an internal issue that needs to be addressed. For Umami Kitchen's case majority of orders are being delivered with only a small amount of cancellations.

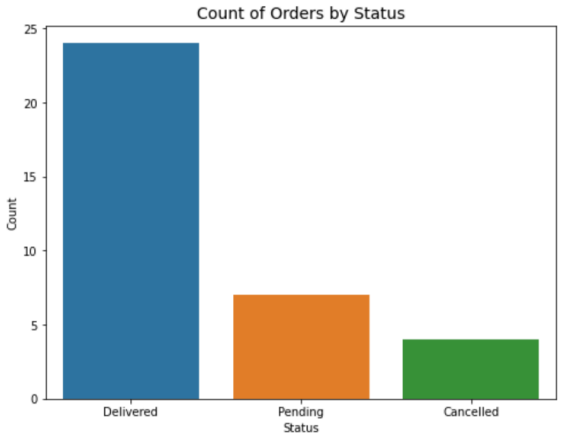
In [13]:
```
order = pd.read_sql('SELECT * FROM uk_database.order',connection)
```

In [14]:
```
order.head()
```

Out[14]:

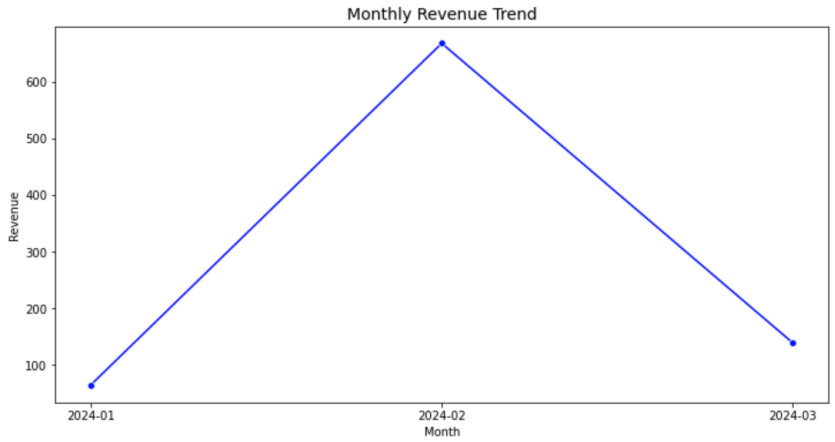
	OrderID	CustomerID	OrderDate	DeliveryDate	Status	TotalAmount
0	1	1	2024-01-01 10:00:00	2024-01-02	Delivered	29.97
1	2	2	2024-01-03 15:00:00	2024-01-05	Pending	14.99
2	3	3	2024-01-04 09:30:00	2024-01-06	Delivered	18.97
3	4	4	2024-01-05 11:00:00	2024-01-06	Cancelled	9.99
4	5	5	2024-01-06 12:45:00	2024-01-08	Delivered	14.99

In [15]:
```
# Plot the count of orders by status

plt.figure(figsize=(8, 6))
sns.countplot(data=order, x='Status')
plt.title('Count of Orders by Status', fontsize=14)
plt.xlabel('Status')
plt.ylabel('Count')
plt.show()
```



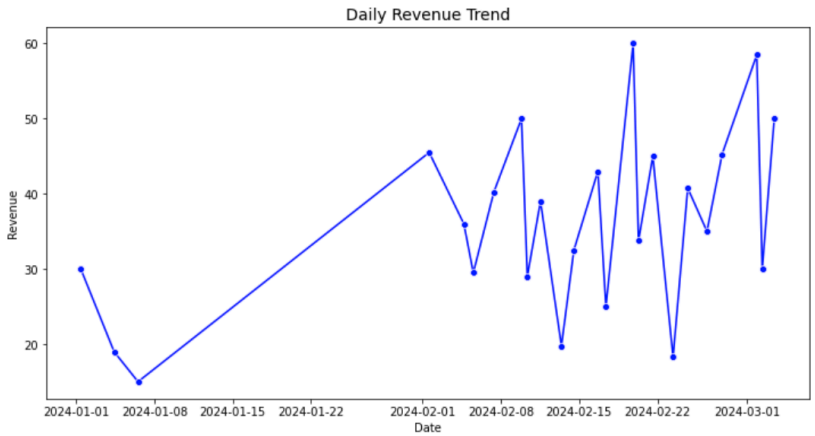
We explored the revenue for Umami Kitchen. Businesses typically do monthly or quarterly summaries, but because our data mainly consists of orders from February 2024 and some from January and March 2024 we aren't able to receive much insight. Ideally we would have data from the entire year to analyze.

In [16]:
```
# Extract month
order['OrderMonth'] = order['OrderDate'].dt.to_period('M').astype(str)

# Calculate monthly revenue
monthly_revenue = order[order['Status'] == 'Delivered'].groupby('OrderMonth')['TotalAmount'].sum().reset_index()

# Plot
plt.figure(figsize=(12, 6))
sns.lineplot(data=monthly_revenue, x='OrderMonth', y='TotalAmount', marker='o', color='blue')
plt.title('Monthly Revenue Trend', fontsize=14)
plt.xlabel('Month')
plt.ylabel('Revenue')
plt.show()
```



We also analyzed the daily trends of orders and observed significant variation in the data. There are multiple spikes in the number of orders placed, indicating periods of heightened activity. These spikes may be due to specific events, promotions, or seasonal factors driving increased customer activity. Further investigation to identify the underlying causes would be beneficial in determining future business strategies to increase profit.

In [17]:
```
daily_revenue = order[order['Status'] == 'Delivered'].groupby('OrderDate')['TotalAmount'].sum().reset_index()

# Plot
plt.figure(figsize=(12, 6))
sns.lineplot(data=daily_revenue, x='OrderDate', y='TotalAmount', marker='o', color='blue')
plt.title('Daily Revenue Trend', fontsize=14)
plt.xlabel('Date')
plt.ylabel('Revenue')
plt.show()
```



Conclusion

Project Outcomes:

- We were able successfully design and implement relational database in MySQL and NoSQL structure in MongoDB
- Effectively migrated structured SQL data into MongoDB collections, while maintaining the integrity of relationships using ObjectID fields and references
- Implement our MySQL into Python and executed series of queries to showcase dynamic data insights
- We queried complex relationships between different tables/collections using joins in SQL and $lookup in MongoDB

Shortcomings:

- MongoDB doesn't enforce foreign key relationships; which could result in orphaned data if the reference documents are deleted
- In SQL, the database can face performance issues with large joins and aggregate queries over time
- In the future, we would like to integrate visualization tools like Tableau, Power BI to demonstrate project insights