# Pizza Sales Data Analysis Project

***(Individual Project)***

**T.M. Chamila Piumantha Thennakoon**

**University of Colombo**

**06/22/2025**

# Executive Summary

This project analyzes a fictional pizza sales dataset to identify trends in revenue, customer ordering behavior over time, and popular products. Tools used include Python (with Pandas, Matplotlib, and Seaborn) for data cleaning and visualization, MySQL for relational database management, and Power BI for dashboard creation. Key insights include peak sales hours, top-selling days of the week and months of the year, most profitable pizza types, and revenue patterns over time.

# Contents

# Table of figures



# Objectives

- Understand customer purchasing behavior over time.
- Identify best-selling pizzas and peak order times.
- Enhance revenue by analyzing customer purchasing behavior.
- Generate visual insights for business decisions.
- Discuss business recommendations using visual insights

## Dataset Overview

- **About the dataset**

The dataset represents one year's worth of sales data from a fictional pizza restaurant. It includes detailed information on customer orders such as the date and time of each transaction, the specific pizzas ordered, their type and size, quantity, price, and the ingredients used. The dataset is divided into four relational tables: orders, order_details, pizzas, and pizza_types. This structure allows for a comprehensive analysis of customer behavior, sales performance, and product popularity over time. The data is well-suited for time series analysis, revenue tracking, and identifying key business trends.

- **Tables**

➢ **Orders (No. of rows = 21350)**

- ❖ *Table name - orders*
- ❖ *Attributes – order_id , date, time*
- ❖ *Primary key – order_id*
- ❖ *Foreign keys – none*
- ❖ *Sample table…*

| | order_id | date | time |
|---|---|---|---|
| 0 | 1 | 2015-01-01 | 11:38:36 |
| 1 | 2 | 2015-01-01 | 11:57:40 |
| 2 | 3 | 2015-01-01 | 12:12:28 |
| 3 | 4 | 2015-01-01 | 12:16:31 |
| 4 | 5 | 2015-01-01 | 12:21:30 |

*Figure 1: Orders Table*

- **Order details (No. of rows = 48620)**
  - *Table name – order_details*
  - *Attributes – order_details_id , order_id , pizza_id , quantity*
  - *Primary key – order_details_id*
  - *Foreign keys – order_id , pizza_id*
  - *Sample table…*

| | order_details_id | order_id | pizza_id | quantity |
|---|---|---|---|---|
| 0 | 1 | 1 | hawaiian_m | 1 |
| 1 | 2 | 2 | classic_dlx_m | 1 |
| 2 | 3 | 2 | five_cheese_l | 1 |
| 3 | 4 | 2 | ital_supr_l | 1 |
| 4 | 5 | 2 | mexicana_m | 1 |

*Figure 2: Order details table*

- **Pizzas (No. of rows = 96)**
  - *Table name – pizzas*
  - *Attributes – pizza_id , pizza_type_id , size , price*
  - *Primary key – pizza_id*
  - *Foreign keys – pizza_type_id*
  - *Sample table…*

| | pizza_id | pizza_type_id | size | price |
|---|---|---|---|---|
| 0 | bbq_ckn_s | bbq_ckn | S | 12.75 |
| 1 | bbq_ckn_m | bbq_ckn | M | 16.75 |
| 2 | bbq_ckn_l | bbq_ckn | L | 20.75 |
| 3 | cali_ckn_s | cali_ckn | S | 12.75 |
| 4 | cali_ckn_m | cali_ckn | M | 16.75 |

*Figure 3: Pizza table*

- **Pizza types (No. of rows = 32)**
  - *Table name – pizza_types*
  - *Attributes – pizza_type_id , name , category , ingredients*
  - *Primary key – pizza_type_id*
  - *Foreign keys – none*
  - *Sample table…*

| | pizza_type_id | name | category | ingredients |
|---|---|---|---|---|
| 0 | bbq_ckn | The Barbecue Chicken Pizza | Chicken | Barbecued Chicken, Red Peppers, Green Peppers,... |
| 1 | cali_ckn | The California Chicken Pizza | Chicken | Chicken, Artichoke, Spinach, Garlic, Jalapeno ... |
| 2 | ckn_alfredo | The Chicken Alfredo Pizza | Chicken | Chicken, Red Onions, Red Peppers, Mushrooms, A... |
| 3 | ckn_pesto | The Chicken Pesto Pizza | Chicken | Chicken, Tomatoes, Red Peppers, Spinach, Garli... |
| 4 | southw_ckn | The Southwest Chicken Pizza | Chicken | Chicken, Tomatoes, Red Peppers, Red Onions, Ja... |

*Figure 4: Pizza type table*

## Tools and technologies

- Python (Google Colab) – For data cleaning and visualization
  - Libraries – Pandas, Matplotlib, Seaborn
- MySQL (via Xampp) – For Relational Database Management
- Power BI – For create dashboards

## Data cleaning preparation

Under data cleaning following points were checked in each table.

- Null values
- Fixed data types (date/time)
- Duplicates

There were no any null or duplicate values in all 4 tables.

- **Datetime Format Conversion**

As an example, the orders table was examined for data quality issues. The following code was used to check for null values, correct data types, and identify duplicate records.

```
print(df1.isnull().sum())
print("Duplicates :",df1.duplicated().sum())
print(df1.dtypes)
```

**output :**

```
order_id    0
date        0
time        0
dtype: int64
Duplicates : 0
order_id     int64
date        object
time        object
dtype: object
```

**Interpretation :**

The orders table does not contain any null values or duplicate rows. However, there is an issue with the data types of the date and time columns. Both columns are initially stored as object types, which is not suitable for time-based analysis. Therefore, they need to be converted to proper datetime format before proceeding with the analysis. The following code was used to perform this conversion.

```
df1['date'] = pd.to_datetime(df1['date'])
df1['time'] = pd.to_datetime(df1['time'], format='%H:%M:%S',
errors='coerce')
print(df1.dtypes)
```

**output :**

```
order_id             int64
date        datetime64[ns]
time        datetime64[ns]
dtype: object
```

**Interpretation :**

As shown in the output, the date and time columns have been successfully converted to datetime format. During the conversion of the time column, the parameter **errors='coerce'** was used to handle any invalid or incorrectly formatted time values. Normally, **pd.to_datetime()** would raise an error if a value does not match the specified format **('%H:%M:%S')**. However, by setting **errors='coerce',** such values are safely converted to **NaT** (Not a Time), preventing the code from breaking and allowing the analysis to continue smoothly.

## Exploratory Data Analysis

The following section presents data visualizations along with key insights derived from them.

- **Pizza category distribution.**

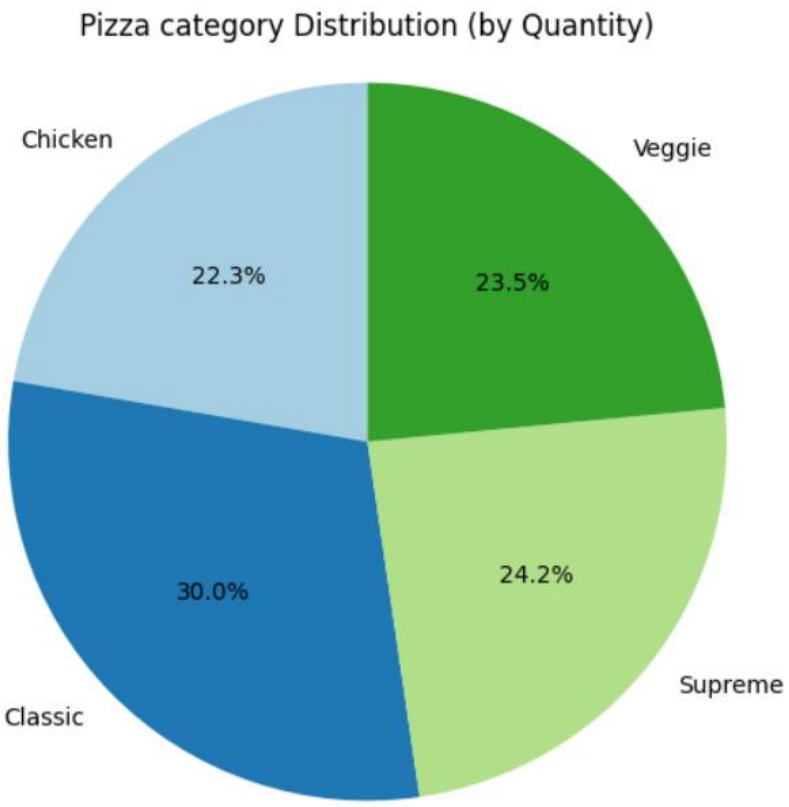


*Figure 5: Pizza category distribution*

Above pie chart shows the percentages of each pizza category. **Classic pizzas** are the most popular category, accounting for **30.0%** of total sales by quantity. **Supreme pizzas** come next with **24.2%**, followed closely by **Veggie pizzas** at **23.5%**. **Chicken pizzas** have the lowest share, making up **22.3%** of total quantity sold. Classic pizzas lead customer preference and might be the most consistent revenue driver. The relatively balanced distribution suggests that all categories are in demand.

- **Pizza sales by category and size**

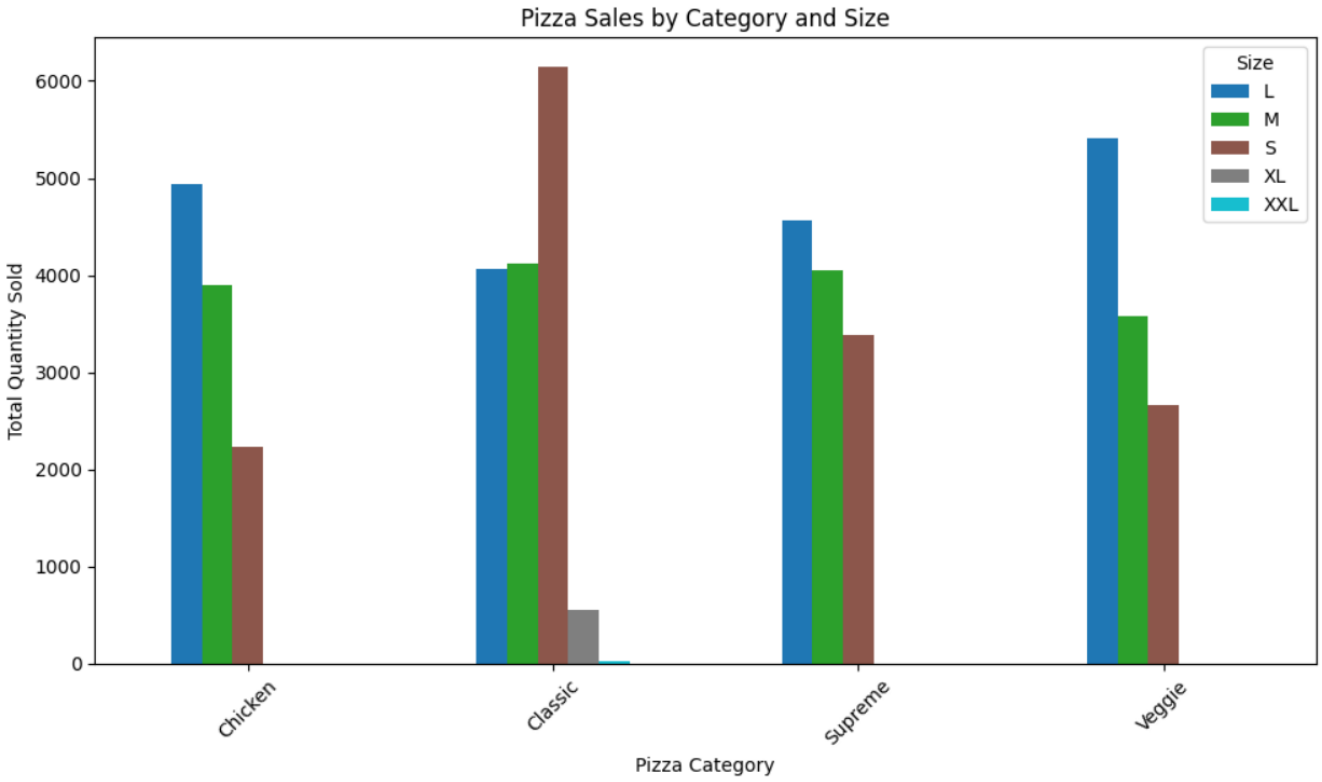


*Figure 6: Pizza sales by category and size*

Above clustered bar chart shows how pizza sales are distributed across different **categories** and **sizes** based on total quantity sold. **Large (L)** pizzas are the most popular size across all categories. **XL** and **XXL** pizzas have significantly lower sales, with XXL being almost negligible. **Classic pizzas** dominate the **Small (S)** size segment, with over 6,000 units sold—much higher than any other category.

- **Pizza revenue by category and size**

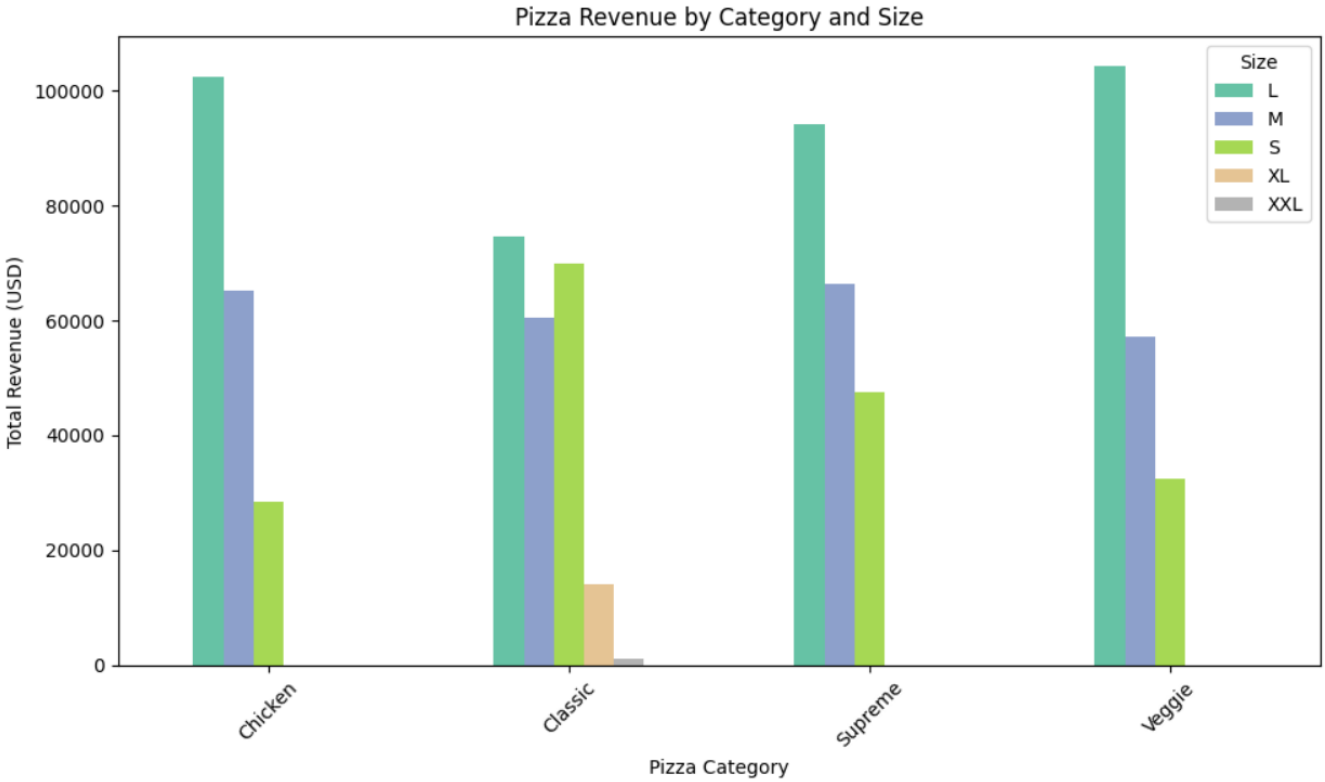


*Figure 7: Pizza revenue by category and size*

Above clustered bar graph presents the total revenue (in USD) for different **pizza categories** (Chicken, Classic, Supreme, Veggie) across **pizza sizes.** In every pizza category, the **Large (L)** size contributes the highest revenue. Especially in **Veggie and Chicken**, the revenue from L size is over **$100,000**. Small Size Revenue is high for **Classic** (close to L and M). Although Classic Small pizzas have the highest sales volume across all categories, Large pizzas contribute the most to total revenue.

- **Total pizza quantity sold by hour of the day.**

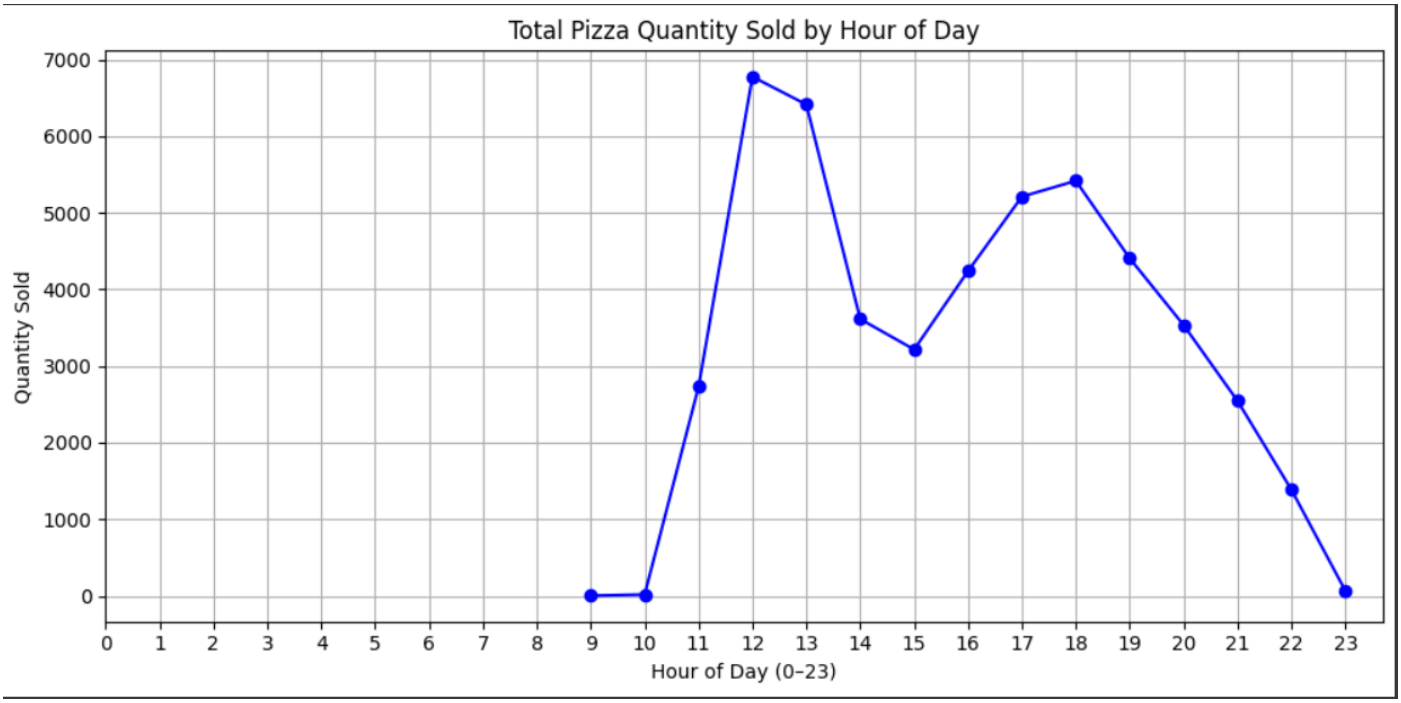


*Figure 8: Total pizza quantity sold by hour of day*

This line chart shows how pizza sales vary across the 24-hour day (0–23 hours). We can notice a **sharp increase** begins at **11:00 AM**. **Peak sales occur at 12:00 PM (noon)** with approximately **6,800 pizzas sold.** There is a noticeable drop from **2:00 PM (3,600)** to **3:00 PM (3,200)**. This suggests a typical post-lunch slowdown. Chart shows another rise in sales from **4:00 PM to 6:00 PM**, peaking again around **6:00 PM (5,400–5,500 pizzas)**. This is likely due to **dinner-time demand**.

- **Total pizza quantity sold by day of the week**

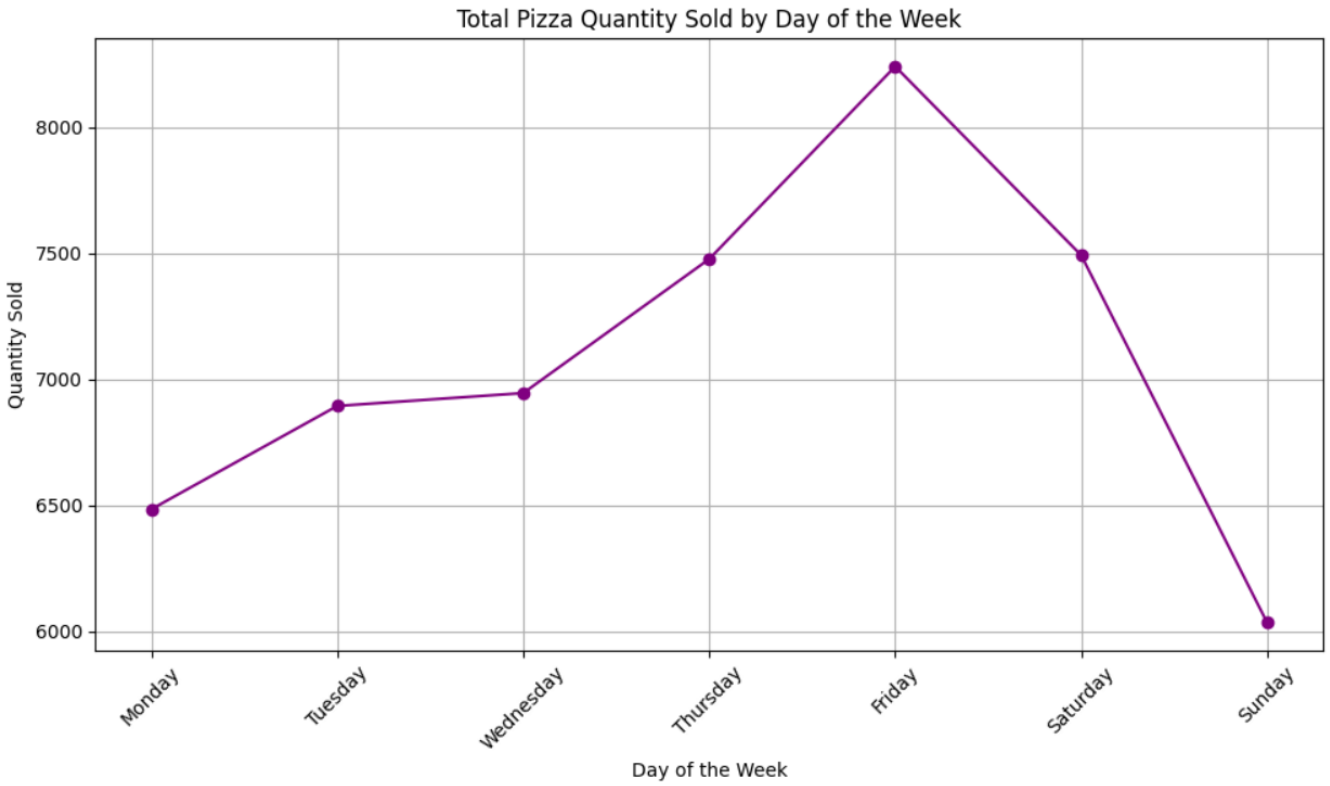


*Figure 9: Total pizza quantity sold by day of the week*

This line chart shows the total number of pizzas sold on each day from **Monday to Sunday**. **Friday is the highest sales day**, suggesting strong demand going into the weekend (likely due to social gatherings, parties, or takeout culture). Sunday is the **lowest sales day.** We can assume both lunch and dinner peaks are probably stronger on Fridays**.**

## Database design and MySQL analysis

- **ER Diagram**

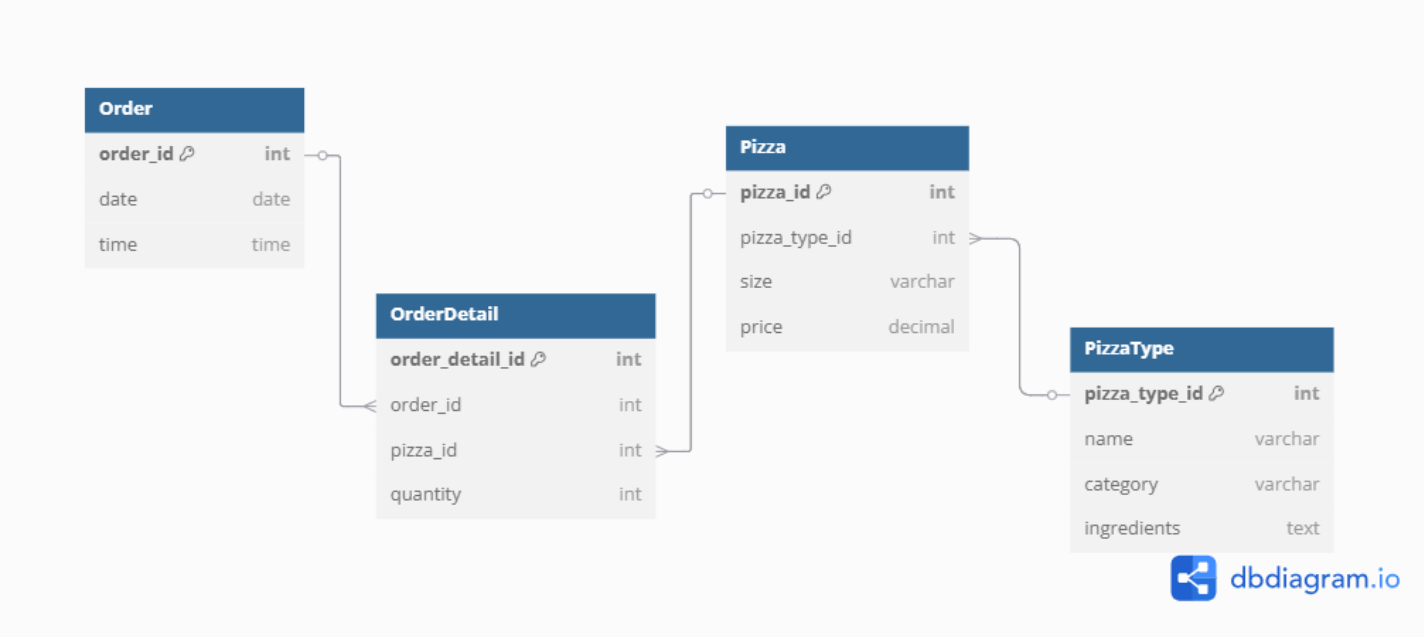


*Figure 10: ER Diagram for pizza sales data*

- **Relationships between tables**

  - **Order → Order Detail**: One-to-many (an order has many order details)
  - **Pizza → Order Detail**: One-to-many (each pizza can appear in many order details)
  - **Pizza Type → Pizza**: One-to-many (one pizza type can correspond to multiple pizzas with different sizes or prices)

- **Sample SQL queries**

  - **Total sales revenue (USD)**

**Code :**

```
--Total sales revenue

SELECT ROUND(SUM(od.quantity * p.price), 2) AS total_revenue
FROM order_details od
JOIN pizzas p ON od.pizza_id = p.pizza_id;
```

**Output :**

| total_revenue |
|---|
| 817860.05 |

  - **Top 5 selling pizzas**

**Code :**

```
-- Top selling pizzas

SELECT pizza_id, COUNT(*) AS total_orders
FROM order_details
GROUP BY pizza_id
ORDER BY total_orders DESC;
```

**Output :**

| pizza_id | total_orders ▼ 1 |
|---|---|
| big_meat_s | 1811 |
| thai_ckn_l | 1365 |
| five_cheese_l | 1359 |
| four_cheese_l | 1273 |
| classic_dlx_m | 1159 |

- **Sales by pizza size**

## Code :

```
-- Sales by pizza size

SELECT pizzas.size, SUM(order_details.quantity) AS total
FROM order_details LEFT JOIN pizzas
ON pizzas.pizza_id = order_details.pizza_id GROUP BY pizzas.size;
```

## Output :

| size | total |
|---|---|
| L | 18956 |
| M | 15635 |
| S | 14403 |
| XL | 552 |
| XXL | 28 |

- **Monthly sales**

## Code :

```
-- monthly sales

SELECT DATE_FORMAT(o.date, '%Y-%m') AS month,
SUM(od.quantity) AS total_sales
FROM order_details od
JOIN orders o ON od.order_id = o.order_id
JOIN pizzas p ON od.pizza_id = p.pizza_id
GROUP BY month
ORDER BY month;
```

## Output :

| month ▲ 1 | total_sales |
|---|---|
| 0000-00 | 0 |
| 2015-01 | 4232 |
| 2015-02 | 3961 |
| 2015-03 | 4261 |
| 2015-04 | 4151 |
| 2015-05 | 4328 |
| 2015-06 | 4107 |
| 2015-07 | 4392 |
| 2015-08 | 4168 |
| 2015-09 | 3890 |
| 2015-10 | 3883 |
| 2015-11 | 4266 |
| 2015-12 | 3935 |

- **Sales by category**

## Code :

```
-- Sales by category

SELECT pizza_types.category AS category,
SUM(order_details.quantity) AS total_sales
FROM order_details
JOIN pizzas  ON order_details.pizza_id = pizzas.pizza_id
JOIN pizza_types  ON pizza_types.pizza_type_id = pizzas.pizza_type_id
GROUP BY category
ORDER BY category;
```

## Output :

| category ▲ 1 | total_sales |
|---|---|
| Chicken | 11050 |
| Classic | 14888 |
| Supreme | 11987 |
| Veggie | 11649 |

## Power BI dashboard

- **How to connect MySQL to Power BI**

To integrate the MySQL database hosted on **phpMyAdmin** via **XAMPP** with Power BI, the **MySQL ODBC** Connector was first installed to enable communication between the two platforms. After ensuring the MySQL service was running in the XAMPP Control Panel, Power BI's "**Get Data**" feature was used to select the MySQL database option. The server was set to **127.0.0.1:3306**, and the appropriate database credentials (username and password) were provided. (3306 is the default port used by MySQL) Upon successful connection, the required tables were selected from the navigator pane and loaded into Power BI for data visualization and analysis.

**Model view of the data :**


*Figure 11: Model view*

- **Creating New measure for Total revenue**

  - **What is a New measure ?**

A **measure** is a DAX (Data analysis expressions. It is a **formula language** used in **Power BI**, **Power Pivot**, and **Analysis Services** to create **Calculated columns Measure Custom tables**) formula that computes results dynamically **based on filters** (e.g., by category, by date, etc.). Without the measure, Power BI might just show the **sum of prices** or **sum of quantity**, which is not meaningful for total revenue. The measure allows us to correctly compute **line totals (quantity × price)** for every row, and **aggregate them by category, size, etc.**

  - **Why we need a new measure for Total revenue?**

In Our Pizza Sales Analysis project, we want to analyze **total revenue** (i.e., total sales amount) by category, size, or other dimensions.

But the problem is:

- We have **quantity** in one table (order_details).
- We have **price** in another table (pizzas).
- There is **no existing column** that gives the **total value of each order line** (quantity × price).

Power BI **does not do this calculation automatically** across related tables in visuals.

- **How to create the measure?**

Go to Home tab and the new measure. Then type,

```
TotalSales =
SUMX
      (order_details,order_details[quantity] * RELATED(pizzas[price]))
```

**Explanation:**

SUMX(...): A row-by-row iterator function that goes through each row in order_details

RELATED(pizzas[price]): Pulls the price column from the related pizzas table into the row context of order_details.

- **Visual types used**
  - Slicer – Date, Category, Size, Pizza name
  - Cad – Total revenue
  - Bar chart – Total quantity by size, Total revenue by size
  - Line chart – Total quantity by month, Total revenue by month

- **Dashboard**

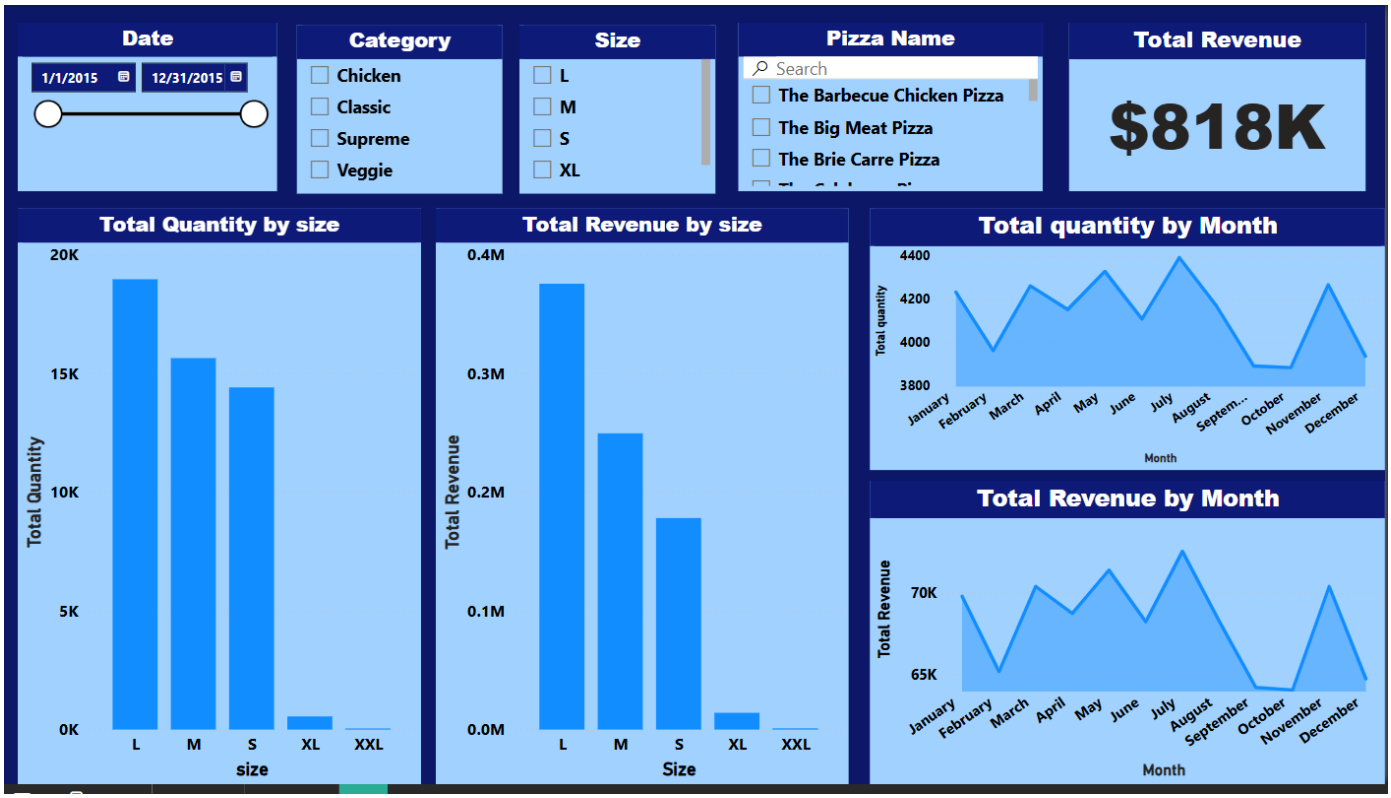


*Figure 12: Screen shot of the Dashboard*

**About the dashboard :**

This dashboard allows users to interactively explore pizza sales data by customizing various filters. Users can select a specific **date range** to focus on sales within a chosen period. They can filter results by **pizza category** (such as Chicken, Classic, Supreme, or Veggie), **pizza size** (S, M, L, XL, XXL), and even by specific **pizza names** using the searchable list. As users adjust these filters, the dashboard dynamically updates key metrics—including **total revenue**, **sales quantity**, and **monthly trends**—to reflect the selected criteria. This enables targeted analysis and easy comparison across different dimensions of the data.

## Insights and business recommendations

- Inventory planning should prioritize Large, Medium, and Small sizes—especially for Classic and Veggie pizzas.
- L size is the bestseller and should be prioritized in marketing and promotions.
- Feature bestsellers and most profitable pizzas at the top of the menu or in highlighted sections.
- Consider promotions or deals around 2–4 PM to boost off-peak sales.
- Staffing and inventory should be increased toward the weekend, especially Friday.
- Marketing can target Friday evenings and Saturday afternoons for maximum impact.
- Lunch Combos can target the 12 PM peak.
- Dinner Deals (like family packs) for the 6 PM peak, especially Friday and Saturday.

## Conclusion

In this project, we successfully analyzed pizza sales data to uncover key business insights such as best-selling pizzas, peak sales hours, and revenue contributions by size and category. Using SQL, we efficiently extracted and joined data from multiple tables. Python helped us clean the data and generate meaningful visualizations, while Power BI enabled us to create interactive dashboards for dynamic exploration. Through this analysis, we learned how customer preferences vary by time and product, and identified opportunities for menu optimization and promotional strategies.

## What can be done next ?

- **Customer Segmentation**: Collect customer data and analyze buying patterns by customer type or region.
- **Predictive Modeling**: Implement forecasting models to predict future sales based on time series trends.

## References


- (Kaggle, n.d.) https://www.kaggle.com/datasets/mysarahmadbhat/pizza-place-sales
- Google. (n.d.). *Google Colaboratory*. https://colab.research.google.com/
- Holistics Software. (n.d.). *dbdiagram.io*. https://dbdiagram.io/