# Data Analysis Project Documentation

## Project Overview

### Objective

* Analyze restaurant sales data to uncover:
  + Trends in revenue
  + Customer behavior and demographics
  + Operational insights to improve efficiency

### Tools Used

* **MySQL**: For data storage, cleaning, and querying
* **Excel**: For initial exploration and storage of raw data
* **Tableau**: For interactive dashboards

### Dataset Description

* **File**: “single\_restaurant\_data\_dirty.csv”
* **Source**: Simulated dataset with intentionally introduced issues
* **Columns**:
  + **Date**: The date of the order
  + **Order\_ID**: Unique identifier of each order
  + **Item\_Name**: The name of the menu item ordered
  + **Category**: The category of the menu item
  + **Quantity**: The number of items ordered
  + **Price**: Price per item
  + **Total\_Sales**: Quantity × Price
  + **Payment\_Method**: Payment method used
  + **Customer\_ID**: Unique customer identifier
  + **Customer\_Age**: Age of the customer
  + **Customer\_Gender**: Gender of the customer
  + **Visit\_Type**: Dine-in, Takeout, or Delivery
  + **Order\_Time**: Time the order was placed
  + **Wait\_Time\_Mins**: Wait time in minutes
  + **Day\_of\_the\_Week**: Day of the week
  + **Meal\_Period**: Breakfast, Lunch, or Dinner
  + **Day\_Type**: Weekday, Weekend, or Holiday

## Dataset Import

### Steps Taken

1. **Created a Database:**

```sql

CREATE DATABASE restaurant\_data\_analysis;  
 USE restaurant\_data\_analysis;  
 ```

1. **Created a Table:**

```sql

CREATE TABLE restaurant\_data (

Date DATE,

Order\_ID VARCHAR(255),

Item\_Name VARCHAR(255),

Category VARCHAR(255),

Quantity INT NULL,

Price DECIMAL(10,2),

Total\_Sales DECIMAL(10,2),

Payment\_Method VARCHAR(255),

Customer\_ID VARCHAR(255),

Customer\_Age INT NULL,

Customer\_Gender VARCHAR(255),

Visit\_Type VARCHAR(255),

Order\_Time TIME,

Wait\_Time\_Mins INT NULL,

Day\_of\_the\_Week VARCHAR(255),

Meal\_Period VARCHAR(255),

Day\_Type VARCHAR(255)

);  
 ```

1. **Created a Temporary Table:**

```sql

CREATE TABLE restaurant\_data\_temp (

Date VARCHAR(255),

Order\_ID VARCHAR(255),

Item\_Name VARCHAR(255),

Category VARCHAR(255),

Quantity VARCHAR(255),

Price DECIMAL(10,2),

Total\_Sales DECIMAL(10,2),

Payment\_Method VARCHAR(255),

Customer\_ID VARCHAR(255),

Customer\_Age VARCHAR(255),

Customer\_Gender VARCHAR(255),

Visit\_Type VARCHAR(255),

Order\_Time VARCHAR(255),

Wait\_Time\_Mins VARCHAR(255),

Day\_of\_the\_Week VARCHAR(255),

Meal\_Period VARCHAR(255),

Day\_Type VARCHAR(255)

);

```

* Created a temporary table (restaurant\_data\_temp) to store the raw data as it came from the CSV file. This allowed me to import the blank or invalid fields before transferring the data to the main table

1. **Used Table Data Import Wizard to import csv into restaurant\_data\_temp table**
   * Selected the file “single\_restaurant\_data\_dirty.csv”
   * Mapped columns to restaurant\_data\_temp schema
   * Ensured correct delimiters and data handling settings
2. **Verified import was successful by comparing number of rows between csv and table:**

```sql

SELECT COUNT(\*)

FROM restaurant\_data\_temp;

```

### Challenges Encountered

* Initially, I attempted to use the Load Data Local Infile command but ran into persistent access and data type mapping issues. To address these:
  + I tried the Command-Line Client to execute the import but ran into challenges with date and column formats
  + Finally, I used the Data Import Wizard, adjust the schema to use VARCHAR(255) for problematic columns like Quantity and Date. This approach ensured the raw data could be successfully imported for further cleaning

## Data Cleaning

### Steps Taken

1. **Replaced empty or invalid fields with NULL in the Quantity, Customer\_Age, and Wait\_Time\_Mins columns:**

```sql

UPDATE restaurant\_data\_temp

SET Quantity = NULL

WHERE Quantity = '' OR Quantity IS NULL;

UPDATE restaurant\_data\_temp

SET Customer\_Age = NULL

WHERE Customer\_Age = '' OR Customer\_Age IS NULL;

UPDATE restaurant\_data\_temp

SET Wait\_Time\_Mins = NULL

WHERE Wait\_Time\_Mins = '' OR Wait\_Time\_Mins IS NULL;

```

1. **Transferred Data with NULLS into Main Table:**

```sql

INSERT INTO restaurant\_data (

Date, Order\_ID, Item\_Name, Category, Quantity, Price, Total\_Sales, Payment\_Method, Customer\_ID, Customer\_Age, Customer\_Gender,

Visit\_Type, Order\_Time, Wait\_Time\_Mins, Day\_of\_the\_Week, Meal\_Period, Day\_Type

)

SELECT

STR\_TO\_DATE(Date, '%Y-%m-%d'), -- Convert date strings to DATE format

Order\_ID,

Item\_Name,

Category,

CAST(Quantity AS SIGNED), -- Convert cleaned strings to INT

Price,

Total\_Sales,

Payment\_Method,

Customer\_ID,

CAST(Customer\_Age AS SIGNED), -- Convert cleaned strings to INT

Customer\_Gender,

Visit\_Type,

STR\_TO\_DATE(Order\_Time, '%H:%i:%s'), -- Convert time strings to TIME format

CAST(Wait\_Time\_Mins AS SIGNED), -- Convert cleaned strings to INT

Day\_of\_the\_Week,

Meal\_Period,

Day\_Type

FROM restaurant\_data\_temp;

```

1. **Handling Duplicate Order IDs**
   * Checked for duplicate Order IDs:

```sql

SELECT Order\_ID, COUNT(\*) AS Duplicate\_Count

FROM restaurant\_data

GROUP BY Order\_ID

HAVING COUNT(\*) > 1;

```

* + Added a unique row identifier
    - Since there were duplicate Order IDs, I added a row\_id column to distinguish between duplicate records:

```sql

ALTER TABLE restaurant\_data

ADD COLUMN row\_id INT

AUTO\_INCREMENT

PRIMARY KEY;

```

* + Deleted Duplicates keeping 1 copy:

```sql

DELETE t1

FROM restaurant\_data t1

JOIN restaurant\_data t2

ON t1.Order\_ID = t2.Order\_ID

AND t1.row\_id > t2.row\_id;

```

1. **Handling NULL values** 
   * Replaced blank values in Item\_Name with “Unknown Item”:

```sql

UPDATE restaurant\_data

SET Item\_Name = "Unknown Item"

WHERE Item\_Name = '' OR Item\_Name IS NULL;

```

* + Identified missing values:

```sql

SELECT

SUM(CASE WHEN Quantity IS NULL THEN 1 ELSE 0 END) AS Null\_Quantity,

SUM(CASE WHEN Customer\_Age IS NULL THEN 1 ELSE 0 END) AS Null\_Customer\_Age,

SUM(CASE WHEN Wait\_Time\_Mins IS NULL THEN 1 ELSE 0 END) AS Null\_Wait\_Time

FROM restaurant\_data;

```

* + Replaced NULL values in Quantity with median quantity

```sql

SELECT COUNT(\*) AS Total\_Rows

FROM restaurant\_data

WHERE Quantity IS NOT NULL;

SELECT Quantity

FROM restaurant\_data

WHERE Quantity IS NOT NULL

ORDER BY Quantity

LIMIT 1 OFFSET 3912;

UPDATE restaurant\_data

SET Quantity = 3

WHERE Quantity IS NULL;

```

* + Replaced NULL values in Customer\_Age with median age

```sql

WITH AgeOrdered AS (

SELECT Customer\_Age,

ROW\_NUMBER() OVER (ORDER BY Customer\_Age) AS row\_num,

COUNT(\*) OVER () AS total\_rows

FROM restaurant\_data

WHERE Customer\_Age IS NOT NULL

)

SELECT Customer\_Age

FROM AgeOrdered

WHERE row\_num = FLOOR((total\_rows - 1) / 2) + 1;

UPDATE restaurant\_data

SET Customer\_Age = 41

WHERE Customer\_Age IS NULL;

```

* + Replaced NULL values in Wait\_Time\_Mins with average wait time

```sql

SELECT ROUND(AVG(Wait\_Time\_Mins), 0) AS Avg\_Wait\_Time

FROM restaurant\_data

WHERE Wait\_Time\_Mins IS NOT NULL;

UPDATE restaurant\_data

SET Wait\_Time\_Mins = 16

WHERE Wait\_Time\_Mins IS NULL;

```

* + Mean vs Median
    - I chose to use Median for replacing NULLs in Quantity and Customer Age because it protects from outliers and skewed distribution more effectively
    - I chose to use Mean for replacing NULLS in Wait\_Time\_Mins because the wait times follow more of a normal distribution

1. **Standardizing Categorical Data**

* After handling missing values, I checked for inconsistencies in categorical columns. Payment\_Method and Visit\_Type had no issues. Customer\_Gender was categorized into “Male”, “Female”, “unknown”, and “Prefer not to answer”. I merged the “unknown” values into “Prefer not to answer” for better consistency:

```sql

UPDATE restaurant\_data

SET Customer\_Gender = 'Prefer not to answer'

WHERE Customer\_Gender = 'unknown';

```

1. **Fix Incorrect Total\_Sales Values**
   * I noticed some of the Total\_Sales values didn’t make sense mathematically. So I updated the Total\_Sales column to reflect the correct equation:

```sql

UPDATE restaurant\_data

SET Total\_Sales = Quantity \* Price

WHERE Total\_Sales <> (Quantity \* Price);

```

1. **Final Data Quality Check**
   * Before proceeding to analysis, I performed one last quality check to confirm data integrity
     + Verified no NULLs in key columns:

```sql

SELECT

SUM(CASE WHEN Item\_Name = '' OR Item\_Name IS NULL THEN 1 ELSE 0 END) AS Null\_Item\_Name,

SUM(CASE WHEN Quantity IS NULL THEN 1 ELSE 0 END) AS Null\_Quantity,

SUM(CASE WHEN Price IS NULL THEN 1 ELSE 0 END) AS Null\_Price,

SUM(CASE WHEN Customer\_Age IS NULL THEN 1 ELSE 0 END) AS Null\_Customer\_Age,

SUM(CASE WHEN Wait\_Time\_Mins IS NULL THEN 1 ELSE 0 END) AS Null\_Wait\_Time

FROM restaurant\_data;

```

* + - Verified no duplicates remain:

```sql

SELECT Order\_ID, COUNT(\*)

FROM restaurant\_data

GROUP BY Order\_ID

HAVING COUNT(\*) > 1;

```

* + - Verified no negative or zero prices:

```sql

SELECT \* FROM restaurant\_data WHERE Price <= 0;

```

### Summary of Data Cleaning

* Duplicates removed
* NULL values fixed
* Categorical data standardized
* Final quality check completed

## Exploratory Data Analysis

### General Overview

* To understand the dataset, I first gathered key statistics:

```sql

SELECT

COUNT(\*) AS Total\_Orders,

COUNT(DISTINCT Order\_ID) AS Unique\_Orders,

COUNT(DISTINCT Customer\_ID) AS Unique\_Customers,

SUM(Total\_Sales) AS Total\_Revenue,

AVG(Total\_Sales) AS Avg\_Order\_Value

FROM restaurant\_data;

```

* + Findings:
    - Total Orders: 4,097
    - Unique Orders: 4,097
    - Unique Customers: 3,311
    - Total Revenue: $114,412
    - Average Order Value: $27.93

### Sales Performance Analysis

* Top ten best selling items:

```sql

SELECT

Item\_Name,

SUM(Quantity) AS Total\_Quantity\_Sold,

SUM(Total\_Sales) AS Total\_Revenue

FROM restaurant\_data

GROUP BY Item\_Name

ORDER BY Total\_Quantity\_Sold DESC

LIMIT 10;

```

* + Findings:
    - Top 5 most popular items by quantity sold were: Burger (1,741), Pizza (1,669), Salad (1,571), Tea (1,485), and Beer (1,394)
* Revenue breakdown by category:

```sql

SELECT

Category,

SUM(Total\_Sales) AS Total\_Revenue

FROM restaurant\_data

GROUP BY Category

ORDER BY Total\_Revenue DESC;

```

* + Findings: The highest revenue-generating category by far is Main Course with $67,608, followed by Alcoholic Drinks ($18,384), Appetizer ($14,733), and Beverages ($13,687)
* Monthly Sales Trends:

```sql

SELECT

DATE\_FORMAT(Date, '%Y-%m') AS Month,

SUM(Total\_Sales) AS Monthly\_Revenue

FROM restaurant\_data

GROUP BY Month

ORDER BY Month;

```

* + Findings:
    - Highest sales months are April ($11,197), February ($10,413), and September ($10,094)
    - Lowest sales months are June ($7,255), November ($8,451), and March ($8,846)

### Customer Behavior Analysis

* Top Customers:

```sql

SELECT

Customer\_ID,

SUM(Total\_Sales) AS Total\_Spending,

COUNT(Order\_ID) AS Total\_Orders

FROM restaurant\_data

GROUP BY Customer\_ID

ORDER BY Total\_Spending DESC

LIMIT 10;

* + Findings: Highest spenders spent a total of $320, $318, and $304 respectively. Not many frequent customers
* Age Group Analysis:

```sql

SELECT

CASE

WHEN Customer\_Age BETWEEN 18 AND 25 THEN '18-25'

WHEN Customer\_Age BETWEEN 26 AND 35 THEN '26-35'

WHEN Customer\_Age BETWEEN 36 AND 50 THEN '36-50'

ELSE '50+'

END AS Age\_Group,

COUNT(DISTINCT Customer\_ID) AS Customers,

COUNT(Order\_ID) AS Orders,

SUM(Total\_Sales) AS Total\_Spending,

AVG(Total\_Sales) AS Avg\_Spending\_Per\_Order

FROM restaurant\_data

GROUP BY Age\_Group;

```

* + Findings: Average spending per order is pretty even across age groups, but the 36-50 and 50+ age groups are the most frequent customers
* Order Method (Dine-in, Delivery, Takeout) Analysis:

```sql

SELECT

Visit\_Type,

COUNT(Order\_ID) AS Total\_Orders,

SUM(Total\_Sales) AS Total\_Revenue

FROM restaurant\_data

GROUP BY Visit\_Type

ORDER BY Total\_Revenue DESC;

```

* + Findings: Orders are fairly evenly distributed among the order methods, but Takeout has the most with 1,427, followed by Dine-in (1,377) and Delivery (1,293)

### Operational Insights

* Average Wait Time by Meal Period:

```sql

SELECT

Meal\_Period,

AVG(Wait\_Time\_Mins) AS Avg\_Wait\_Time

FROM restaurant\_data

GROUP BY Meal\_Period

ORDER BY Avg\_Wait\_Time DESC;

```

* + Findings: The Lunch period had the longest average wait times (19 minutes), followed by Breakfast (14 minutes) and Dinner (14 minutes)
* Busiest Days of the Week

```sql

SELECT

Day\_of\_the\_Week,

COUNT(Order\_ID) AS Total\_Orders,

SUM(Total\_Sales) AS Total\_Revenue

FROM restaurant\_data

GROUP BY Day\_of\_the\_Week

ORDER BY Total\_Orders DESC;

```

* + Findings:
    - The busiest days are Tuesday (619 orders), Wednesday (613 orders), and Friday (612 orders)
    - The slowest days are Saturday (532 orders), Thursday (566 orders), and Sunday (574 orders)
* Peak Order Hours

```sql

SELECT

HOUR(Order\_Time) AS Order\_Hour,

COUNT(Order\_ID) AS Total\_Orders

FROM restaurant\_data

GROUP BY Order\_Hour

ORDER BY Total\_Orders DESC;

```

* + Findings:
    - The busiest order times are between 5PM and 9PM
    - The slowest order times are between 6AM and 11AM

## Business Insights and Recommendations

* **Peak Hours and Staffing:**
  + **Insights**: Busiest time is between 5PM and 9PM
  + **Recommendations**: Increase staffing levels during peak hours and introduce promotions for the slower times (happy hour). Consider opening later.
* **Menu Optimization:**
  + **Insights**: Burger, Pizza, and Salad are the most popular menu items
  + **Recommendations**: Bundle top-selling items to maximize sales. Consider revising the menu to remove or rebrand underperforming items
* **Customer Loyalty and Retention:**
  + **Insights**: While most customers visit only once or twice a year, there are opportunities to encourage repeat visits through targeted promotions
  + **Recommendations**: Offer discounts on future visits and implement seasonal deals for milestones such as birthdays, anniversaries, and holidays.
* **Order Method Strategy:**
  + **Insights**: Takeout receives the highest number of orders, closely followed by dine-in and delivery
  + **Recommendations**: Partner with food delivery services to increase reach, providing exclusive online discounts
* **Wait Time Optimization:**
  + **Insights**: Lunch has the longest wait times (19 minutes), while breakfast and dinner are around 14 minutes
  + **Recommendations**: Evaluate kitchen operations and prep efficiency during lunch hours. Consider pre-prepped ingredients and additional staffing
* **Daily Promotions:**
  + **Insights**: Busiest days are Tuesday, Wednesday, and Friday, while Saturday and Sunday have fewer orders
  + **Recommendations**: Launch weekend promotions (brunch specials, family discounts) to boost traffic on the weekends

## Dashboard and Visualizations

### Objective: The Tableau dashboard was designed to provide a clear, interactive view of restaurant performance, making it easy to identify key business insights. The visualizations help analyze trends in revenue, order behavior, peak hours, and customer preferences.

**Dashboard Overview: The dashboard consists of 5 key visualizations:**

* Monthly Revenue Trends
* Order Type Distribution
* Peak Order Hours
* Top-Selling Menu Items
* Average Wait Times

**Interactive Features:**

* The dashboard is interactive, allowing users to explore different aspects of the data dynamically:
  + Clicking on Order Type filters all charts to show only that category
  + Hovering over the bar charts and heatmaps reveals detailed order counts
  + Filters for Meal Period, Day of the Week, and Customer Age are available for further analysis