Data Analysis Internship Project Report

Connect with me on LinkedIn

Image Source: Freepik

Restaurant Data Analysis Insights

In this project, I analyzed a restaurant dataset to uncover key trends, insights, and actionable recommendations for optimizing business strategies. The dataset included crucial information such as restaurant names, locations, cuisines, price ranges, ratings, and service availability (e.g., online delivery and table booking). The analysis was performed using **Python**, **SQL**, and various data visualization **tools**.

Here's a summary of the key tasks and **tools** used:

- **Data Cleaning and Storage:** Cleaned the dataset and stored it in a **MySQL database**.

- **Insights Extraction:** Extracted valuable insights using **SQL queries** and **Python libraries** like **Pandas** and **NumPy**.
- **Visualization:** Presented data visually using **Matplotlib**, **Seaborn**, **Plotly**, and **Folium** for geographic insights.
- **Actionable Insights:** Provided data-driven recommendations to enhance restaurant operations.

About Me ....

I am Bhushan Gawali, a detail-oriented Data Analyst based in Nashik, Maharashtra. I possess expertise in **SQL**, **Python**, **Power BI**, and **Advanced Excel**, specializing in ETL processes, data cleaning, and statistical analysis. My skills include database management, data visualization, and extracting actionable insights from complex datasets. I am proficient in Python libraries like **Pandas**, **NumPy**, **Matplotlib**, and **Seaborn** for data analysis and visualization. With a proven track record of improving decision-making through data-driven strategies, I am passionate about leveraging data to drive business growth.

Dataset Overview

The dataset contains detailed information on various restaurants, including location, cuisine type, price ranges, ratings, and service availability (e.g., online delivery or table booking). Below is a detailed breakdown of each column in the dataset:

Dataset Columns:

- **restaurant_id:** A unique identifier for each restaurant.
- **restaurant_name:** Name of the restaurant.
- **country_code:** Code representing the country where the restaurant is located.
- **city:** The city where the restaurant operates.
- **address:** The full address of the restaurant.
- **locality:** The general locality where the restaurant is situated.
- **locality_verbose:** A more detailed description of the locality.
- **longitude:** Longitude coordinate for the restaurant's location.
- **latitude:** Latitude coordinate for the restaurant's location.
- **cuisines:** The type of cuisines served by the restaurant (e.g., Japanese, French).
- **average_cost_for_two:** The average cost for two people dining at the restaurant.
- **currency:** The currency used for the price (e.g., Pula, Dollar).
- **has_table_booking:** Indicates whether the restaurant accepts table bookings (Yes/No).
- **has_online_delivery:** Indicates whether the restaurant offers online delivery (Yes/No).
- **is_delivering_now:** Indicates if the restaurant is currently delivering (Yes/No).
- **switch_to_order_menu:** Whether the restaurant has switched to an online order menu (Yes/No).
- **price_range:** Categorized price range of the restaurant (from 1 to 4, with 1 being the lowest and 4 the highest).
- **aggregate_rating:** The overall rating of the restaurant, as given by customers.
- **rating_color:** The color representing the rating (e.g., Dark Green for Excellent).
- **rating_text:** Text description of the rating (e.g., Excellent, Good).

- **votes:** The number of customer votes received for the restaurant.

This dataset provides a wealth of information that can be used to analyze trends in restaurant services, customer preferences, and pricing.

In [ ]:

🌟 Table of Contents 🌟

1. Data Cleaning and Storage in MySQL 🗄️

A. Data Cleaning 🧹

B. Data Transformation 🔄

- 4️⃣ **Inserting Data**

2. Insights Extraction Using SQL and Python 📊

Level 1 Tasks 🥇

- 🍽️ **Task 1: Top Cuisines**
- 🌆 **Task 2: City Analysis**
- 💲 **Task 3: Price Range Distribution**
- 📦 **Task 4: Online Delivery**

Level 2 Tasks 🥈

- 🍕 **Task 1: Cuisine Combination**
- 🗺️ **Task 2: Geographic Analysis**
- 🍴 **Task 3: Restaurant Chains**

Level 3 Tasks 🥉

- 🗳️ **Task 2: Votes Analysis**
- 💲 **Task 3: Price Range vs. Online Delivery and Table Booking**

1. Data Cleaning and Storage

IN MYSQL

In this section, I concentrated on the data cleaning process and data transformation in MySQL to ensure high-quality data for analysis. This involved several key tasks:

A. Data Cleaning

I implemented methods to clean and standardize strings, ensuring consistency across the dataset. I also corrected data types for critical columns to facilitate accurate calculations. Handling missing values was a priority, where I either removed or imputed data, ensuring the dataset's integrity.

Table of Contents

1. Importing Required Libraries

In [94]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
from unidecode import unidecode
import warnings

# Ignore warnings
warnings.filterwarnings('ignore')
```

In [95]:
```
pd.set_option('display.max_columns', None)
```

Table of Contents

2. Load the Data

In [96]:
```
# Load the datase
df = pd.read_csv(r"C:\Users\BHUSHAN\Downloads\All Certificates\Cognifyz Technology\
df
```

Out[96]:

	Restaurant ID	Restaurant Name	Country Code	City	Address	Locality	Loc Ver
0	6317637	Le Petit Souffle	162	Makati City	Third Floor, Century City Mall, Kalayaan Avenu...	Century City Mall, Poblacion, Makati City	Century Mall, Pobla Makati M
1	6304287	Izakaya Kikufuji	162	Makati City	Little Tokyo, 2277 Chino Roces Avenue, Legaspi...	Little Tokyo, Legaspi Village, Makati City	Little To Legaspi Vil Makati
2	6300002	Heat - Edsa Shangri-La	162	Mandaluyong City	Edsa Shangri-La, 1 Garden Way, Ortigas, Mandal...	Edsa Shangri-La, Ortigas, Mandaluyong City	Edsa Shang Ort Mandalu City,
3	6318506	Ooma	162	Mandaluyong City	Third Floor, Mega Fashion Hall, SM Megamall, O...	SM Megamall, Ortigas, Mandaluyong City	SM Mega Ort Mandalu City, Man
4	6314302	Sambo Kojin	162	Mandaluyong City	Third Floor, Mega Atrium, SM Megamall, Ortigas...	SM Megamall, Ortigas, Mandaluyong City	SM Mega Ort Mandalu City, Man
...	...	...	...	...	...	...	
9546	5915730	Namlı Gurme	208	��stanbul	Kemanke�� Karamustafa Pa��a Mahallesi, Rıhtım ...	Karak�_y	Karak ��sta
9547	5908749	Ceviz A��acı	208	��stanbul	Ko��uyolu Mahallesi, Muhittin ��st�_nda�� Cadd...	Ko��uyolu	Ko��u ��sta
9548	5915807	Huqqa	208	��stanbul	Kuru�_e��me Mahallesi, Muallim Naci Caddesi, N...	Kuru�_e��me	Kuru�_e�� ��sta
9549	5916112	A���k Kahve	208	��stanbul	Kuru�_e��me Mahallesi, Muallim Naci Caddesi, N...	Kuru�_e��me	Kuru�_e�� ��sta
9550	5927402	Walter's Coffee Roastery	208	��stanbul	Cafea��a Mahallesi, Bademaltı Sokak, No 21/B, ...	Moda	M ��sta

9551 rows × 21 columns

Table of Contents

3. Standardizing Column Names

In [97]:
```
# Re-standardize column names after re-loading
df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_')
```

Table of Contents

4. Initial Data Exploration

In [98]:
```
df.shape
```

Out[98]:
```
(9551, 21)
```

In [99]:
```
# Display basic information about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9551 entries, 0 to 9550
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   restaurant_id         9551 non-null   int64  
 1   restaurant_name       9551 non-null   object 
 2   country_code          9551 non-null   int64  
 3   city                  9551 non-null   object 
 4   address               9551 non-null   object 
 5   locality              9551 non-null   object 
 6   locality_verbose      9551 non-null   object 
 7   longitude             9551 non-null   float64
 8   latitude              9551 non-null   float64
 9   cuisines              9542 non-null   object 
 10  average_cost_for_two  9551 non-null   int64  
 11  currency              9551 non-null   object 
 12  has_table_booking     9551 non-null   object 
 13  has_online_delivery   9551 non-null   object 
 14  is_delivering_now     9551 non-null   object 
 15  switch_to_order_menu  9551 non-null   object 
 16  price_range           9551 non-null   int64  
 17  aggregate_rating      9551 non-null   float64
 18  rating_color          9551 non-null   object 
 19  rating_text           9551 non-null   object 
 20  votes                 9551 non-null   int64  
dtypes: float64(3), int64(5), object(13)
memory usage: 1.5+ MB
```

Table of Contents

5. Check for duplicates

In [100…
```
# Check for duplicates
print("Number of duplicate rows:", df.duplicated().sum())
```

```
Number of duplicate rows: 0
```

Table of Contents

6. Handling Missing Values

In [101…

```
# Check for missing values
df_missing = df.isnull().sum()
print("Missing values in each column after re-loading:")
df_missing
```

```
Missing values in each column after re-loading:
```

Out[101]:

```
restaurant_id           0
restaurant_name         0
country_code            0
city                    0
address                 0
locality                0
locality_verbose        0
longitude               0
latitude                0
cuisines                9
average_cost_for_two    0
currency                0
has_table_booking       0
has_online_delivery     0
is_delivering_now       0
switch_to_order_menu    0
price_range             0
aggregate_rating        0
rating_color            0
rating_text             0
votes                   0
dtype: int64
```

In [102…

```
# Drop rows where the 'cuisines' column has NaN values inplace
df.dropna(subset=['cuisines'], inplace=True)
```

In [103…

```
# Verify that the null values were removed
print(df.cuisines.isnull().sum())
```

```
0
```

Table of Contents

7. Cleaning and Standardizing Strings

Data cleaning and standardization is the process of identifying and correcting inaccuracies or inconsistencies in data to improve its quality and usability. This involves normalizing character encodings, correcting typographical errors, removing or replacing unwanted characters, and ensuring consistent formatting across datasets.

In [104…

```
import re
from unidecode import unidecode

def clean_and_correct_string(s):
    # Normalize unicode characters to ASCII (e.g., convert Turkish characters to Er
    s = unidecode(s)
    # Replace unwanted characters (_ . -) with spaces
    s = re.sub(r'[_\.\-]+', ' ', s)
    # Replace multiple spaces with a single space
    s = re.sub(r'\s+', ' ', s)
    # Remove any leading/trailing whitespace
    s = s.strip()
    # Capitalize each word
```

```
    return ' '.join(word.capitalize() for word in s.split())

# List of columns to clean in the DataFrame
columns_to_clean = ['restaurant_name', 'city', 'address', 'locality', 'locality_ver

# Apply the cleaning function to each column in the list
for column in columns_to_clean:
    df[column] = df[column].apply(clean_and_correct_string)
```

Table of Contents

8. Handling Data Types

In [105…

```
df.dtypes
```

Out[105]:

```
restaurant_id             int64
restaurant_name          object
country_code              int64
city                     object
address                  object
locality                 object
locality_verbose         object
longitude               float64
latitude                float64
cuisines                 object
average_cost_for_two      int64
currency                 object
has_table_booking        object
has_online_delivery      object
is_delivering_now        object
switch_to_order_menu     object
price_range               int64
aggregate_rating        float64
rating_color             object
rating_text              object
votes                     int64
dtype: object
```

In [106…

```
# Convert appropriate columns to their correct data types
df['restaurant_id'] = df['restaurant_id'].astype(int)
df['country_code'] = df['country_code'].astype(int)
df['average_cost_for_two'] = df['average_cost_for_two'].astype(float)
df['price_range'] = df['price_range'].astype(int)
df['aggregate_rating'] = df['aggregate_rating'].astype(float)
df['votes'] = df['votes'].astype(int)
```

In [107…

```
# Convert categorical columns
categorical_columns = ['cuisines','country_code', 'currency', 'has_table_booking',
for column in categorical_columns:
    df[column] = df[column].astype('category')
```

Table of Contents

9. Currency Conversion: Converting Restaurant Costs to Indian Rupees (INR)

In [108…

```
# Define a dictionary mapping each currency to its exchange rate in INR
currency_to_inr = {
    'Botswana Pula(P)': 6.1,
```

```python
    'Brazilian Real(R$)': 17.5,
    'Dollar($)': 83.0,
    'Emirati Diram(AED)': 22.6,
    'Indian Rupees(Rs.)': 1,  # INR to INR, so conversion rate is 1
    'Pounds(£)': 102.5,
    'Qatari Rial(QR)': 22.7,
    'Rand(R)': 4.5,
    'Sri Lankan Rupee(LKR)': 0.26,
    'Turkish Lira(TL)': 3.1
}

# Function to convert the average cost based on currency
def convert_to_inr(row):
    return row['average_cost_for_two'] * currency_to_inr.get(row['currency'], 1)

# Apply the conversion
df['average_cost_in_inr'] = df.apply(convert_to_inr, axis=1)
```

Table of Contents

10. Final Data Quality Check

In [109…

```python
# Display summary statistics
df.describe()
```

Out[109]:

	restaurant_id	longitude	latitude	average_cost_for_two	price_range	aggregate_ratin
count	9.542000e+03	9542.000000	9542.000000	9542.000000	9542.000000	9542.00000
mean	9.043301e+06	64.274997	25.848532	1200.326137	1.804968	2.66523
std	8.791967e+06	41.197602	11.010094	16128.743876	0.905563	1.51658
min	5.300000e+01	-157.948486	-41.330428	0.000000	1.000000	0.00000
25%	3.019312e+05	77.081565	28.478658	250.000000	1.000000	2.50000
50%	6.002726e+06	77.192031	28.570444	400.000000	2.000000	3.20000
75%	1.835260e+07	77.282043	28.642711	700.000000	2.000000	3.70000
max	1.850065e+07	174.832089	55.976980	800000.000000	4.000000	4.90000

In [110…

```python
# Check final data types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 9542 entries, 0 to 9550
Data columns (total 22 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   restaurant_id         9542 non-null   int32
 1   restaurant_name       9542 non-null   object
 2   country_code          9542 non-null   category
 3   city                  9542 non-null   object
 4   address               9542 non-null   object
 5   locality              9542 non-null   object
 6   locality_verbose      9542 non-null   object
 7   longitude             9542 non-null   float64
 8   latitude              9542 non-null   float64
 9   cuisines              9542 non-null   category
 10  average_cost_for_two  9542 non-null   float64
 11  currency              9542 non-null   category
 12  has_table_booking     9542 non-null   category
 13  has_online_delivery   9542 non-null   category
 14  is_delivering_now     9542 non-null   category
 15  switch_to_order_menu  9542 non-null   object
 16  price_range           9542 non-null   int32
 17  aggregate_rating      9542 non-null   float64
 18  rating_color          9542 non-null   object
 19  rating_text           9542 non-null   object
 20  votes                 9542 non-null   int32
 21  average_cost_in_inr   9542 non-null   float64
dtypes: category(6), float64(5), int32(3), object(8)
memory usage: 1.3+ MB
```

In [111…

```
# Replace NaN with None
df.replace({np.nan: None}, inplace=True)
```

B. Data Transformation

Once the data was cleaned, I transformed it into a structured format suitable for storage. I utilized MySQL to create a database and tables, effectively organizing the data. This preparation allowed for seamless extraction of insights using SQL and Pandas. Additionally, the cleaned dataset was optimized for visualization, enhancing the ability to present findings effectively.

Table of Contents

1. Database Connection

In [112…

```
# Import Required Libraries
import mysql.connector
```

```
# Establish a connection to the MySQL database
connection = mysql.connector.connect(
    host='localhost',  # e.g., 'localhost'
    user='root',  # your MySQL username
    password='Bhushan148',  # your MySQL password
    database='project'  # the database where you want to insert data
)

cursor = connection.cursor()
cursor
```

Out[112]: <mysql.connector.cursor_cext.CMySQLCursor at 0x201aeae9010>

In [113…

```
# Check connection properly work or not
pd.read_sql_query("SHOW TABLES", connection)
```

Out[113]:

	Tables_in_project
0	co2_emissions
1	financial_loan
2	mini_project
3	restaurants
4	supply_chain
5	table1
6	table2

Table of Contents

2. Creating Tables

In [114…

```
# Drop the table if it exists, then create it
cursor.execute("DROP TABLE IF EXISTS Restaurants")

# Create the table in the MySQL database
create_table_query = """
CREATE TABLE Restaurants (
    restaurant_id INT PRIMARY KEY,
    restaurant_name VARCHAR(255) NOT NULL,
    country_code VARCHAR(10) NOT NULL,
    city VARCHAR(100) NOT NULL,
    address VARCHAR(255),
    locality VARCHAR(100),
    locality_verbose VARCHAR(255),
    longitude DECIMAL(10, 6),
    latitude DECIMAL(10, 6),
    cuisines VARCHAR(255),
    average_cost_for_two INT,
    currency VARCHAR(50),
    has_table_booking ENUM('Yes', 'No') DEFAULT 'No',
    has_online_delivery ENUM('Yes', 'No') DEFAULT 'No',
    is_delivering_now ENUM('Yes', 'No') DEFAULT 'No',
    switch_to_order_menu ENUM('Yes', 'No') DEFAULT 'No',
    price_range INT,
    aggregate_rating DECIMAL(3, 2),
    rating_color VARCHAR(20),
    rating_text VARCHAR(50),
```

```
    votes INT,
    average_cost_in_inr DECIMAL(8, 2)
);
"""
cursor.execute(create_table_query)
print("Table created successfully.")
```

```
Table created successfully.
```

Table of Contents

3. Verifying Table Creation

In [115…

```
# Check if the table was created successfully
pd.read_sql_query("SHOW TABLES", connection)
```

Out[115]:

	Tables_in_project
0	co2_emissions
1	financial_loan
2	mini_project
3	restaurants
4	supply_chain
5	table1
6	table2

Table of Contents

4. Inserting Data

In [116…

```
# SQL Insert query
columns = ', '.join(df.columns)  # Convert column names to a string
placeholders = ', '.join(['%s'] * len(df.columns))  # Create %s placeholders for ea
insert_query = f"INSERT INTO Restaurants ({columns}) VALUES ({placeholders})"

# Insert each row from the DataFrame
try:
    for index, row in df.iterrows():
        data_tuple = tuple(row)  # Convert row to a tuple
        cursor.execute(insert_query, data_tuple)  # Execute the insert query

    # Commit changes
    connection.commit()
    print("Data inserted successfully.")

except mysql.connector.Error as err:
    print(f"Error: {err}")
    connection.rollback()  # Rollback in case of error
```

```
Data inserted successfully.
```

2. Insights Extraction

... USING SQL AND PYTHON

In this section, I tackled all levels and tasks by leveraging SQL and Python. My approach involved utilizing SQL for data extraction and manipulation, while employing **Pandas** and **NumPy** for data analysis. Additionally, I integrated visualization tools like **Matplotlib**, **Seaborn**, and **Plotly** to create meaningful insights. For geographical data visualization, I utilized **Folium**, enhancing the overall understanding of the data. This comprehensive process enabled me to extract valuable insights effectively throughout the internship at **Cognifyz Technologies**.

Level 1 Tasks

⬆ Table of Contents

Task 1: Top Cuisines

`➜ 1. Determine the top three most common cuisines in the dataset.`
`➜ 2. Calculate the percentage of restaurants that serve each of the top cuisines.`

1. Determine the top three most common cuisines in the dataset.

In [118…

```
# Split the cuisines and count occurrences
cuisines_series = df['cuisines'].str.split(', ').explode()
top_cuisines = cuisines_series.value_counts().head(3)
top_cuisines
```

Out[118]:

```
cuisines
North Indian    3960
Chinese         2735
Fast Food       1986
Name: count, dtype: int64
```

2. Calculate the percentage of restaurants that serve each of the top cuisines.

In [119…

```
# Step 1: Split the 'cuisines' column in the standardized df DataFrame and expand i
expanded_cuisines = df['cuisines'].str.split(', ').explode()

# Step 2: Calculate total number of restaurants
total_restaurants = len(df)

# Step 3: Count occurrences of each cuisine and calculate percentage
top_cuisines = expanded_cuisines.value_counts()
percentages = (top_cuisines / total_restaurants) * 100

# Step 4: Round the percentages to two decimals and append '%' sign
percentages = percentages.round(2).astype(str) + ' %'

# Display the formatted percentages for the top three cuisines
percentages.head(3)
```

Out[119]:

```
cuisines
North Indian     41.5 %
Chinese         28.66 %
Fast Food       20.81 %
Name: count, dtype: object
```

📊 **Key Insights**

- **North Indian Cuisine** is the most popular, with nearly **42%** of restaurants offering it.
- **Chinese** cuisine ranks second, served in around **29%** of restaurants.
- **Fast Food** follows in third place, available in approximately **21%** of restaurants.

⬆️ Table of Contents

Task 2: City Analysis

➜ 1. Identify the city with the highest number of restaurants in the dataset.
➜ 2. Calculate the average rating for restaurants in each city.

→ 3. Determine the city with the highest average rating.

1. Identify the city with the highest number of restaurants in the dataset.

In [120…

```
# City with highest number of restaurants
pd.read_sql_query(
"""SELECT city as City, COUNT(*) AS "Total Restaurants"
FROM restaurants
GROUP BY city
ORDER BY "Total Restaurants" DESC
LIMIT 1;
"""
,connection)
```

Out[120]:

	City	Total Restaurants
0	New Delhi	5473

2. Calculate the average rating for restaurants in each city.

In [121…

```
# Average rating for restaurants in each city
pd.read_sql_query(
"""SELECT city as City, avg(aggregate_rating) as "Average Rating"
FROM restaurants
GROUP BY city
order by avg(aggregate_rating) desc
limit 5;
"""
,connection)
```

Out[121]:

	City	Average Rating
0	Inner City	4.900000
1	Quezon City	4.800000
2	Makati City	4.650000
3	Pasig City	4.633333
4	Mandaluyong City	4.625000

3. Determine the city with the highest average rating.

In [122…

```
# City with the highest average rating
pd.read_sql_query(
    """
    SELECT city AS City, AVG(aggregate_rating) AS "Average Rating"
    FROM restaurants
    GROUP BY city
    ORDER BY AVG(aggregate_rating) DESC
    LIMIT 1
    """,
    connection
)
```

Out[122]:

	City	Average Rating
0	Inner City	4.9

📊 **Key Insights**

- **City with the Most Restaurants**: **New Delhi** leads with the highest number of restaurants, **5,473**.
- **Highest Average Rating**: The **Inner City** has the highest average restaurant rating of **4.9**.

⬆️ Table of Contents

Task 3: Price Range Distribution

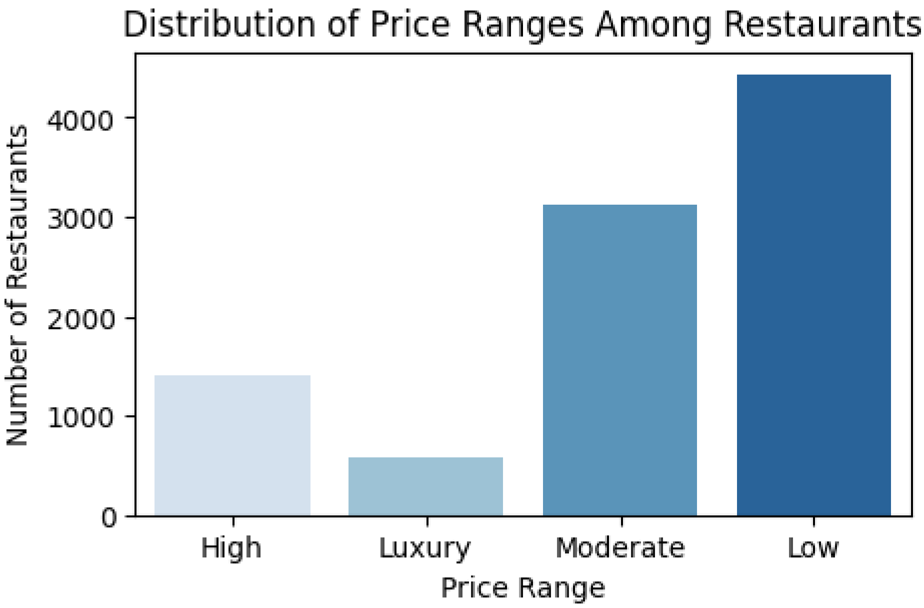
→ 1. Create a histogram or bar chart to visualize the distribution of price ranges among the restaurants.
→ 2. Calculate the percentage of restaurants in each price range category.

1. Create a histogram or bar chart to visualize the distribution of price ranges among the restaurants.

In [123…

```
import seaborn as sns
import matplotlib.pyplot as plt

# Mapping of price ranges to labels
price_mapping = {
    1: 'Low',
    2: 'Moderate',
    3: 'High',
    4: 'Luxury'
}

# Replace numerical price ranges with the corresponding labels
df['price_range_label'] = df['price_range'].map(price_mapping)

# Set the size of the figure
plt.figure(figsize=(5, 3))

# Create the bar plot using Seaborn
sns.countplot(data=df, x='price_range_label', palette='Blues')

# Add title and labels
plt.title('Distribution of Price Ranges Among Restaurants')
plt.xlabel('Price Range')
plt.ylabel('Number of Restaurants')
```

```
# Show the plot
plt.show()
```



2. Calculate the percentage of restaurants in each price range category.

In [124…

```
# Create a mapping of price ranges
price_mapping = {
    1: 'Low',
    2: 'Moderate',
    3: 'High',
    4: 'Luxury'
}

# Replace the price_range numeric values with corresponding words
df['price_range_'] = df['price_range'].map(price_mapping)

# Count occurrences of each price range
price_counts = df['price_range_'].value_counts()

# Calculate the total number of restaurants
total_restaurants = len(df)

# Calculate percentages
percentages = (price_counts / total_restaurants) * 100

# Round the percentages to two decimals
percentages = percentages.round(2)

# Displaying the result in a DataFrame format
percentage_df = pd.DataFrame({
    'Price Range': price_counts.index,
    'Number of Restaurants': price_counts.values,
    'Percentage': percentages.values
})

# Adding a '%' sign to the percentage column for display
percentage_df['Percentage'] = percentage_df['Percentage'].astype(str) + '%'

# Display the DataFrame
percentage_df
```

Out[124]:

	Price Range	Number of Restaurants	Percentage
0	Low	4438	46.51%
1	Moderate	3113	32.62%
2	High	1405	14.72%
3	Luxury	586	6.14%

Key Insights

- **The majority of restaurants** are in the Low and Moderate price ranges, with **46.5%** and **32.6%** of restaurants in each.
- **14.7%** of restaurants fall in the High price range.
- Only **6.1%** of restaurants are in the Luxury category.

Table of Contents

Task 4: Online Delivery

→ 1. Determine the percentage of restaurants that offer online delivery.
→ 2. Compare the average ratings of restaurants with and without online delivery.

1. Determine the percentage of restaurants that offer online delivery.

In [125…

```
# Calculate percentage of restaurants offering online delivery
online_delivery_count = df['has_online_delivery'].value_counts(normalize=True) * 1(
online_delivery_count
```

Out[125]:

```
has_online_delivery
No     74.313561
Yes    25.686439
Name: proportion, dtype: float64
```

2. Compare the average ratings of restaurants with and without online delivery.

In [126…

```
# Average ratings based on online delivery availability
avg_rating_online = df.groupby('has_online_delivery')['aggregate_rating'].mean()
avg_rating_online
```

Out[126]:

```
has_online_delivery
No     2.463517
Yes    3.248837
Name: aggregate_rating, dtype: float64
```

📊 Key Insights

- Only **25.7%** of restaurants offer online delivery.
- Restaurants with online delivery have a higher average rating of **3.25**, compared to **2.46** for those without this option.

Level 2 Tasks

Level 2 Tasks

⬆️ Table of Contents

Task 1: Restaurant Ratings

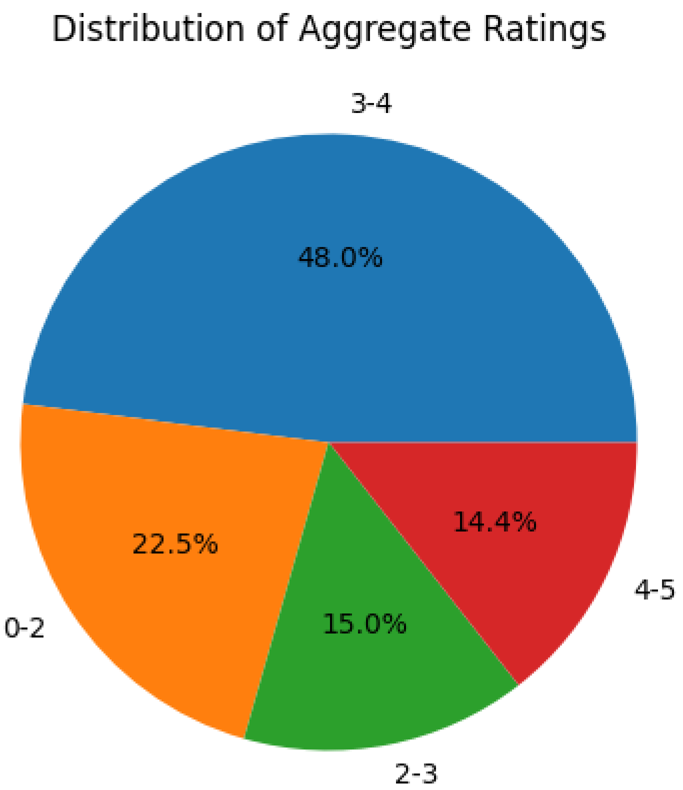
→ 1. Analyze the distribution of aggregate ratings and determine the most common rating range.
→ 2. Calculate the average number of votes received by restaurants.

1. Analyze the distribution of aggregate ratings and determine the most common rating range.

In [127…

```
# Execute the SQL query and read the data into a DataFrame
x = pd.read_sql_query(
    """
    SELECT
        CASE
            WHEN aggregate_rating < 2 THEN '0-2'
            WHEN aggregate_rating >= 2 AND aggregate_rating < 3 THEN '2-3'
            WHEN aggregate_rating >= 3 AND aggregate_rating < 4 THEN '3-4'
            WHEN aggregate_rating >= 4 AND aggregate_rating <= 5 THEN '4-5'
            ELSE 'Other'
        END AS rating_range,
        COUNT(*) AS restaurant_count
    FROM
        restaurants
    GROUP BY
        rating_range
    ORDER BY
        restaurant_count DESC;
    """,
    connection
```

```
)

# Plotting using Pandas with reduced figure size
x.set_index('rating_range')['restaurant_count'].plot(
    kind='pie',
    autopct='%1.1f%%',
    figsize=(6, 5),  # Reduced size
    title='Distribution of Aggregate Ratings'
)
plt.ylabel('')  # Hide the y-label
plt.show()
```



2. Calculate the average number of votes received by restaurants.

In [128…

```
# Assuming 'df' is your DataFrame containing restaurant data
average_votes = df['votes'].mean()
print("Average number of votes:", average_votes)
```

```
Average number of votes: 156.7720603647034
```

📊 Key Insights

- **48%** of restaurants have ratings between 3 and 4.
- The average number of votes received by restaurants is **157**.

Table of Contents

Task 2: Cuisine Combination

→ 1. Identify the most common combinations of cuisines in the dataset.
→ 2. Determine if certain cuisine combinations tend to have higher ratings.

1. Identify the most common combinations of cuisines in the dataset.

In [129…

```
# Count combinations of cuisines
cuisine_combinations = df['cuisines'].value_counts()
print(cuisine_combinations.head())
```

```
cuisines
North Indian             936
North Indian, Chinese    511
Chinese                  354
Fast Food                354
North Indian, Mughlai    334
Name: count, dtype: int64
```

2. Determine if certain cuisine combinations tend to have higher ratings.

In [130…

```
# Assuming 'df' is your DataFrame containing restaurant data
avg_rating_combination = df.groupby('cuisines')['aggregate_rating'].mean().reset_ir

# Sort the results to see which cuisine combinations have the highest average ratir
avg_rating_combination = avg_rating_combination.sort_values(by='aggregate_rating',

# Display the top cuisine combinations with their average ratings
avg_rating_combination.head()
```

Out[130]:

	cuisines	aggregate_rating
1062	Italian, Deli	4.9
949	Hawaiian, Seafood	4.9
93	American, Sandwich, Tea	4.9
683	Continental, Indian	4.9
796	European, Asian, Indian	4.9

📊 Key Insights

- The most common cuisine combination is **North Indian and Chinese**, served by **511** restaurants.
- Some combinations, such as **Italian and Deli** and **Hawaiian and Seafood**, have a perfect average rating of **4.9**.

Table of Contents

Task 3: Geographic Analysis

➜ 1. Plot the locations of restaurants on a map using longitude and latitude coordinates.
➜ 2. Identify any patterns or clusters of restaurants in specific areas.

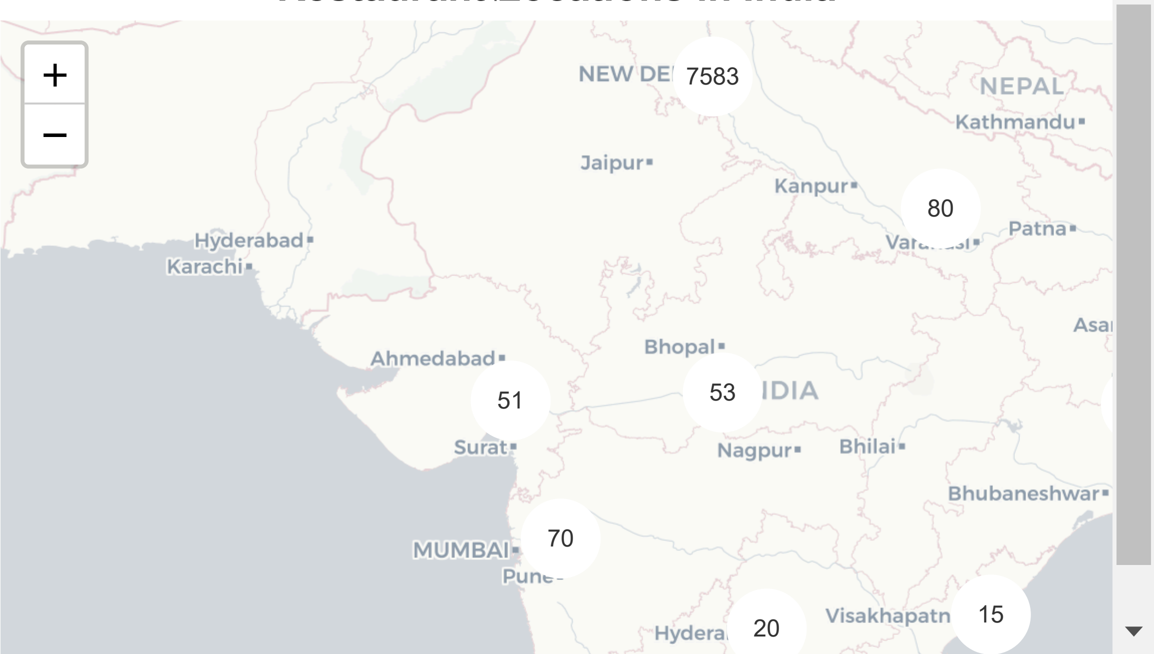
1. Plot the locations of restaurants on a map using longitude and latitude coordinates.

In [131…

```
import folium
from folium.plugins import MarkerCluster

# Create a basic Folium map centered at a specific latitude and longitude
m = folium.Map(location=[20.5937, 78.9629], zoom_start=5, tiles='CartoDB positron')

# Create a MarkerCluster to cluster the markers
marker_cluster = MarkerCluster().add_to(m)

# Loop through the dataframe and add a marker for each restaurant
for idx, row in df.iterrows():
    folium.Marker(
        location=[row['latitude'], row['longitude']],
        popup=f"<b>{row['restaurant_name']}</b><br>{row['address']}<br>{row['city']
        tooltip=row['restaurant_name'],  # Show name on hover
        icon=folium.Icon(color='blue', icon='info-sign')  # Customize marker icon c
    ).add_to(marker_cluster)

# Add a simple title on the map (you could also use HTML for more formatting)
title_html = '''
             <h3 align="center" style="font-size:20px"><b>Restaurant Locations in I
             '''
m.get_root().html.add_child(folium.Element(title_html))

# Save the map to an HTML file (or you can display it inline in Jupyter)
m.save('restaurants_map.html')

# Display the map (if in a Jupyter notebook, you can use m)
m
```

Out[131]: Make this Notebook Trusted to load map: File -> Trust Notebook

Restaurant Locations in India



2. Identify any patterns or clusters of restaurants in specific areas.

In [132…

```
import folium
from folium.plugins import MarkerCluster
from sklearn.cluster import KMeans
import pandas as pd

# Assume 'df' is your DataFrame with latitude and longitude columns

# Step 1: Select the number of clusters (k) for K-means
kmeans = KMeans(n_clusters=5)  # You can choose any value of k
df['cluster'] = kmeans.fit_predict(df[['latitude', 'longitude']])

# Step 2: Create the map
m = folium.Map(location=[20.5937, 78.9629], zoom_start=5, tiles='CartoDB positron')

# Step 3: Add a MarkerCluster
marker_cluster = MarkerCluster().add_to(m)

# Step 4: Plot restaurant locations with clusters
for idx, row in df.iterrows():
    folium.Marker(
        location=[row['latitude'], row['longitude']],
        popup=f"<b>{row['restaurant_name']}</b><br>{row['address']}<br>{row['city']
        tooltip=row['restaurant_name'],
        icon=folium.Icon(color='blue', icon='info-sign')
    ).add_to(marker_cluster)

# Step 5: Visualize cluster centers (optional)
for i, cluster_center in enumerate(kmeans.cluster_centers_):
    folium.Marker(
        location=[cluster_center[0], cluster_center[1]],
        popup=f"Cluster {i+1} Center",
        icon=folium.Icon(color='red', icon='star')
    ).add_to(m)

# Add title to the map
title_html = '''
             <h3 align="center" style="font-size:20px"><b>Restaurant Locations with
             '''
m.get_root().html.add_child(folium.Element(title_html))
```

```python
# Save the map to an HTML file
m.save('restaurant_clusters_map.html')

# Display the map
m
```

Out[132]: Make this Notebook Trusted to load map: File -> Trust Notebook

Restaurant Locations with Clusters

📊 Key Insights

- The largest cluster of restaurants is in **India (New Delhi)**, with over **8,000+** restaurants.
- **North America** has a significant number, with around **400+** restaurants.
- **Africa** has **550+** restaurants, while **Australia** has **120+**.
- **South America** has the fewest restaurants, with just **60+**.

⬆ Table of Contents

Task 4: Restaurant Chains

→ 1. Identify if there are any restaurant chains present in the dataset.
→ 2. Analyze the ratings and popularity of different restaurant chains.

1. Identify if there are any restaurant chains present in the dataset.

In [133…

```
x = pd.read_sql_query(
    """
SELECT
    restaurant_name,
    COUNT(DISTINCT city) AS num_cities,
    COUNT(DISTINCT address) AS num_locations
FROM
    restaurants
GROUP BY
    restaurant_name
HAVING
    COUNT(DISTINCT address) > 1 OR COUNT(DISTINCT city) > 1
ORDER BY
    num_locations DESC
limit 5;

    """,
    connection
)
x
```

Out[133]:

	restaurant_name	num_cities	num_locations
0	Cafe Coffee Day	5	83
1	Domino's Pizza	7	79
2	Subway	5	63
3	Green Chick Chop	4	51
4	Mcdonald's	7	48

2. Analyze the ratings and popularity of different restaurant chains.

In [134…

```
x = pd.read_sql_query(
    """
        SELECT
            restaurant_name,
            AVG(aggregate_rating) AS avg_rating,
            SUM(votes) AS total_votes,
            COUNT(restaurant_id) AS num_branches
        FROM
            restaurants
        GROUP BY
            restaurant_name
        HAVING
            count(city)>5
        ORDER BY
            avg_rating DESC, total_votes DESC
        limit 5;

    """,
    connection
)
x
```

Out[134]:

	restaurant_name	avg_rating	total_votes	num_branches
0	Ab's Absolute Barbecues	4.833333	16551.0	6
1	Farzi Cafe	4.366667	10098.0	6
2	Barbeque Nation	4.353846	28142.0	26
3	Mocha	4.185714	3111.0	7
4	Tgi Friday's	3.850000	4357.0	6

📊 **Key Insights**

- The largest cluster of restaurants is in **India (New Delhi)**, with over **7,500** restaurants.
- **North America** has a significant number, with around **410** restaurants.
- **Africa** has **557** restaurants, while **Australia** has **126**.
- **South America** has the fewest restaurants, with just **60**.

Level 3 Tasks

Level 3 Tasks

⬆️ Table of Contents

Task 2: Votes Analysis

→ 1. Identify the restaurants with the highest and lowest number of votes.
→ 2. Analyze if there is a correlation between the number of votes and the rating of a restaurant.

1. Identify the restaurants with the highest and lowest number of votes.

In [135…

```
# Identify restaurants with the highest and lowest number of votes
highest_votes = df.nlargest(5, 'votes')[['restaurant_name', 'votes', 'aggregate_rat
highest_votes
```

Out[135]:

	restaurant_name	votes	aggregate_rating
728	Toit	10934	4.8
735	Truffles	9667	4.7
3994	Hauz Khas Social	7931	4.3
2412	Peter Cat	7574	4.3
739	Ab's Absolute Barbecues	6907	4.6

In [ ]:

In [136…

```
lowest_votes = df.nsmallest(5, 'votes')[['restaurant_name', 'votes', 'aggregate_rat
lowest_votes
```

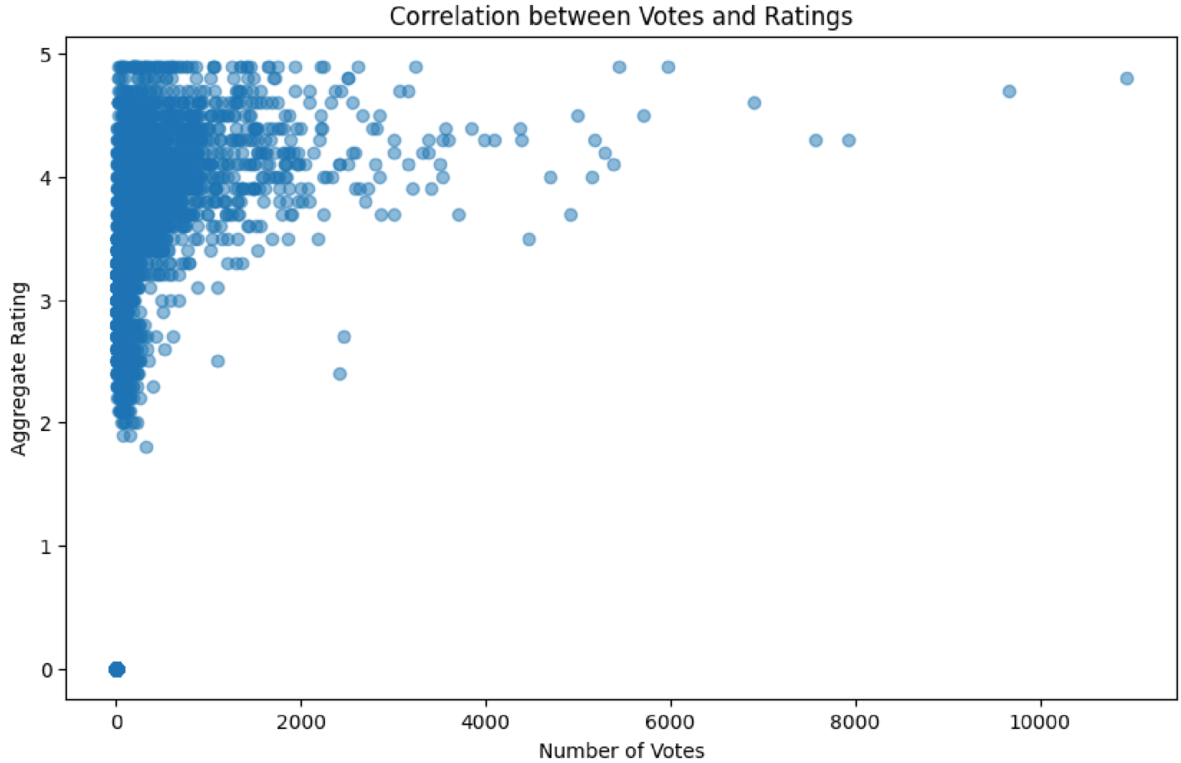
Out[136]:

	restaurant_name	votes	aggregate_rating
69	Cantinho Da Gula	0	0.0
874	The Chaiwalas	0	0.0
879	Fusion Food Corner	0	0.0
880	Punjabi Rasoi	0	0.0
887	Baskin Robbin	0	0.0

2. Analyze if there is a correlation between the number of votes and the rating of a restaurant.

In [137…

```
# Calculate correlation between votes and ratings
correlation = df['votes'].corr(df['aggregate_rating'])

# Create scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(df['votes'], df['aggregate_rating'], alpha=0.5)
plt.xlabel('Number of Votes')
plt.ylabel('Aggregate Rating')
plt.title('Correlation between Votes and Ratings')
plt.show()
```



📊 **Key Insights**

- Restaurants like **Toit** and **Truffles** have the most votes and strong ratings.
- On the other hand, places like **Cantinho da Gula** and **The Chaiwalas** have zero votes and a rating of **0.0**.
- This suggests a general correlation: more votes usually mean a more reliable and higher rating.
- The correlation coefficient of **0.31** indicates a weak positive correlation between votes and ratings. While higher ratings tend to attract more votes, the relationship isn't very strong.

In [ ]:

In [ ]: