End-to-End ETL Pipeline: MySQL to BI Tool Integration

Project Overview

> **Objective**

- Build a scalable ETL pipeline that:
- Extracts transactional restaurant data from SQL
- Transforms it into business-ready analytics datasets
- Outputs a single consolidated CSV
- Distributes the file via email / JS / Google Apps Script
- Visualizes insights in Looker Studio

> **Key Business Questions Answered**

- Least & most ordered items (with categories)
- Highest spending orders and item composition
- Peak and low order times
- Cuisine/category performance for menu optimization

In [35]:
```
import mysql.connector # we use this to connect with the sql
import pandas as pd
import warnings
warnings.filterwarnings("ignore")

conn = mysql.connector.connect(
    host="localhost",
    user="analytics_user",
    password="Analytics@123",
    database="restaurant_db",
    port=3306
)

cursor = conn.cursor() # we use this execuate manually

print("Connected to MySQL")
```

```
Connected to MySQL
```

In [36]:
```
# Lets bring the order_details table as it is

cursor.execute("SELECT * FROM order_details;")

rows = cursor.fetchall() # to fetch rows only
columns = [col[0] for col in cursor.description]


# Let's understand what we have done here :)
"""
>>> cursor.execute
        This prep. both the data and metadata

>>> Now what is cursor.description?
    - This helps us to get the col^s metadata only

>>> What Does cursor.description Look Like?
    - It's just a list of tuples & each tuple represents one col^m
    - example: cursor.description
[
  ('order_details_id', 3, None, None, None, None, None),
  ('order_id', 3, None, None, None, None, None),
  ('order_date', 10, None, None, None, None, None),
  ('order_time', 11, None, None, None, None, None),
  ('item_id', 3, None, None, None, None, None)
]

>>> And each tuple contains 7 elements which is

                (
                  name,          ← index 0
                  type_code,     ← index 1
                  display_size,  ← index 2
                  internal_size, ← index 3
                  precision,     ← index 4
                  scale,         ← index 5
                  null_ok        ← index 6
                )


        # Which means:

            Col[0] - Col Name
            Col[1] - Data Type Code - SQL internal code and not human frienc
            Col[2] - Other metadata - low level database info.
                And most importantly pandas doesnt need this.


Now I guess you have got an idea why we have choosen Col[0]

    For each column metadata tuple, extract the column name.
        So that the output becomes
            ['order_details_id', 'order_id', 'order_date', 'order_time', 'it
"""
```

```
order_details_df = pd.DataFrame(rows, columns=columns)
```

In [37]:
```
# basic sanity check

order_details_df.head()
```

Out[37]:

	order_details_id	order_id	order_date	order_time	item_id
0	1	1	2023-01-01	0 days 11:38:36	109.0
1	2	2	2023-01-01	0 days 11:57:40	108.0
2	3	2	2023-01-01	0 days 11:57:40	124.0
3	4	2	2023-01-01	0 days 11:57:40	117.0
4	5	2	2023-01-01	0 days 11:57:40	129.0

In [39]:
```
# Same way we are doing the same with the menu table

cursor.execute("SELECT * FROM menu_items;")

rows = cursor.fetchall()
columns = [col[0] for col in cursor.description]

menu_items_df = pd.DataFrame(rows, columns=columns)
```

In [40]:
```
# Basic sanity checkup to ensure everything is loaded
# Now, schema item_id is commeon for the both tables
# Which means item_id is the primary key for orders & foreign for menu table

menu_items_df.head()
```

Out[40]:

	menu_item_id	item_name	category	price
0	101	Hamburger	American	12.95
1	102	Cheeseburger	American	13.95
2	103	Hot Dog	American	9.00
3	104	Veggie Burger	American	10.50
4	105	Mac & Cheese	American	7.00

In [41]:
```
# Lets join everything

query = """
SELECT
    *
FROM order_details od
JOIN menu_items mi
  ON od.item_id = mi.menu_item_id;
"""
raw_df = pd.read_sql(query, conn)
```

In [84]:
```
# basic sanity check

raw_df.head(10)
```

Out[84]:

	order_details_id	order_id	order_date	order_time	item_id	menu_item_id	item_na
0	1	1	2023-01-01	0 days 11:38:36	109	109	Kor Beef B
1	2	2	2023-01-01	0 days 11:57:40	108	108	Tofu
2	3	2	2023-01-01	0 days 11:57:40	124	124	Spagh
3	4	2	2023-01-01	0 days 11:57:40	117	117	Chic Bu
4	5	2	2023-01-01	0 days 11:57:40	129	129	Mushr Ra
5	6	2	2023-01-01	0 days 11:57:40	106	106	Fre F
6	7	3	2023-01-01	0 days 12:12:28	117	117	Chic Bu
7	8	3	2023-01-01	0 days 12:12:28	119	119	Chic T
8	9	4	2023-01-01	0 days 12:16:31	117	117	Chic Bu
9	10	5	2023-01-01	0 days 12:21:30	117	117	Chic Bu

In [85]:
```
# freezing the working schema

raw_df['order_date'] = pd.to_datetime(raw_df['order_date']) # time series ar
raw_df['order_hour'] = raw_df['order_time'].dt.components.hours # int for pe
raw_df['price'] = raw_df['price'].astype(float) # converted into float for r
```

In [86]:
```
# basic sanity check

raw_df.head(10)
```

Out[86]:

	order_details_id	order_id	order_date	order_time	item_id	menu_item_id	item_na
0	1	1	2023-01-01	0 days 11:38:36	109	109	Kor Beef B
1	2	2	2023-01-01	0 days 11:57:40	108	108	Tofu
2	3	2	2023-01-01	0 days 11:57:40	124	124	Spagh
3	4	2	2023-01-01	0 days 11:57:40	117	117	Chic Bu
4	5	2	2023-01-01	0 days 11:57:40	129	129	Mushr Ra
5	6	2	2023-01-01	0 days 11:57:40	106	106	Fre F
6	7	3	2023-01-01	0 days 12:12:28	117	117	Chic Bu
7	8	3	2023-01-01	0 days 12:12:28	119	119	Chic T
8	9	4	2023-01-01	0 days 12:16:31	117	117	Chic Bu
9	10	5	2023-01-01	0 days 12:21:30	117	117	Chic Bu

In [83]:
```
print(raw_df.dtypes)
```

```
order_details_id             int64
order_id                     int64
order_date           datetime64[ns]
order_time          timedelta64[ns]
item_id                      int64
menu_item_id                 int64
item_name                   object
category                    object
price                      float64
order_hour                   int64
dtype: object
```

In [56]:
```
raw_df.isnull().sum()
```

Out[56]:
```
order_details_id    0
order_id            0
order_date          0
order_time          0
item_id             0
menu_item_id        0
item_name           0
category            0
price               0
order_hour          0
dtype: int64
```

In [82]:
```
# Item popularity

item_popularity = (
    raw_df
    .groupby(['item_name', 'category'])
    .size()
    .reset_index(name='item_order_count')
)
```

In [60]:
```
# Order Value analysis

order_value = (
    raw_df
    .groupby('order_id')
    .agg(
        total_spend=('price', 'sum'),
        items_bought=('item_name', lambda x: ', '.join(x))
    )
    .reset_index()
)
```

In [62]:
```
# Catagory performances

category_performance = (
    raw_df
    .groupby('category')
    .agg(
        category_total_orders=('item_name', 'count'),
        category_total_revenue=('price', 'sum'),
        category_avg_price=('price', 'mean')
    )
    .reset_index()
)
```

In [76]:
```
# final dataframe for d.viz

final_df = final_df[[
    'order_id',
    'order_date',
    'order_hour',
    'category',
    'item_name',
    'price',
```

```
    'item_order_count',
    'total_spend',
    'category_total_orders',
    'category_total_revenue',
    'category_avg_price'
]]
```

In [77]:
```
final_df.columns
```

Out[77]:
```
Index(['order_id', 'order_date', 'order_hour', 'category', 'item_name',
       'price', 'item_order_count', 'total_spend', 'category_total_orders',
       'category_total_revenue', 'category_avg_price'],
      dtype='object')
```

In [78]:
```
assert final_df.isnull().sum().sum() == 0
```

In [81]:
```
print(final_df.shape)
```

```
(12097, 11)
```

In [87]:
```
# final csv file to save/download

final_df.to_csv("restaurant_analytics_final.csv", index=False)
```

In [91]:
```
from email.message import EmailMessage
import smtplib
import os
from datetime import datetime

# Config to send the email

APP_PASSWORD = os.getenv("GMAIL_APP_PASSWORD")

if not APP_PASSWORD:
    raise EnvironmentError("GMAIL_APP_PASSWORD environment variable not set"

now = datetime.now().strftime("%Y-%m-%d %H:%M")


subject = f"KPI Report | {now}"

email_body = f"""
Hi, Pranab

Please find attached the KPI report.

Generated at: {now}

Regards,
Automated Analytics Pipeline
"""

# Read CSV file

with open("restaurant_analytics_final.csv", "r", encoding="utf-8") as f:
    csv_data = f.read()
```

```
# Email Setup

msg = EmailMessage()
msg["Subject"] = subject
msg["From"] = "luxevistahub@gmail.com"
msg["To"] = "career.pranab@gmail.com"

msg.set_content(email_body)

msg.add_attachment(
    csv_data.encode("utf-8"),
    maintype="text",
    subtype="csv",
    filename="daily_kpis.csv"
)



# SMPT Connection
# what is SMPT?
"""
a technical standard for transmitting electronic mail (email) over a network
SMTP allows computers and servers to exchange data regardless of their under
"""

server = smtplib.SMTP_SSL("smtp.gmail.com", 465)
server.login("luxevistahub@gmail.com", APP_PASSWORD)


"""
#Login with the sender id
server.login(
    "luxevistahub@gmail.com",
    "abcd abcd abcd abcd".replace(" ", "")

    ---Incase you are using this for your person staff and app password does
        In such scenerio you can keep your app password public

"""
server.send_message(msg)
server.quit()

print("Email sent successfully with CSV attachment")
```

```
Email sent successfully with CSV attachment
```

> Next project on cron Scheduling