End-to-End Data Cleaning, Transformation, and Relationship Modeling in PostgreSQL using Python.

1. Uploading Multiple CSVs to PostgreSQL using Python.
2. pip install pandas sqlalchemy psycopg2-binary
3. All files uploaded In PostgreSQL..
4. How to check for null value in a table.
5. Check the duplicate order id.
6. 2nd method check for duplicate value and show data.
7. Change the data type.
8. Remove Unnamed column.
9. Step 1 :- Delete rows where ALL columns are NULL.
10. Step 2 :- Delete rows where ANY column is NULL.
11. Delete Exact Duplicate Rows (All Column Match).

12. How to create a relationship across all tables.

1. Ensure Primary Keys Exist.

2. Each table should have a PRIMARY KEY, which uniquely identifies each row.

3. Step 1: Make sure `customer_id` in Customer Details is PRIMARY KEY.

14. Add Relationships

Uploading Multiple CSVs to PostgreSQL using Python.

```
import os
import pandas as pd
from sqlalchemy import create_engine
# Step 1: Define folder path containing CSVs
folder_path = r"D:\SQL Learning\Food delivery"
# Step 2: PostgreSQL connection string
conn_string = "postgresql+psycopg2://postgres:123456@localhost:5433/Sales"
engine = create_engine(conn_string)

# Step 3: Loop through CSV files in folder and upload to PostgreSQL
for file in os.listdir(folder_path):
  if file.endswith('.csv'):
    file_path = os.path.join(folder_path, file)
    table_name = os.path.splitext(file)[0]  # Remove .csv from filename for table name
    print(f"Uploading {file} as table '{table_name}'...")

# Load CSV into DataFrame
   df = pd.read_csv(file_path)
 # Upload to PostgreSQL
    df.to_sql(table_name, engine, index=False, if_exists='replace')
 print(f"Uploaded: {table_name}")
print(" All files uploaded successfully.")
```

All files uploaded In PostgreSQL..

To view All Table :-

```
Select * from "April_Sales_2023";
Select * from "March_Sales_2023";
Select * from "February_Sales_2023";
Select * from "January_Sales_2023";
Select * from "Customer Details";
Select * from "Resturant_Details";
Select * from "Food details";
```

How to check for null value in a table.

```
Select * from  "April_Sales_2023"
where "order_id" is null
or  "orderDate" is null
or "customer_id" is null
or  "Resturant_ID" is null
or  "Fooditem" is null
or "quantity" is null
or  "deliver_status" is null
or
Payment_method is null;
```

Check the duplicate order id.

```
Select  “order_id”, count(*) as count
From “April_Sales_2023”
Group by “order_id”
Having count (*) > 1;
```

2nd method check for duplicate value and show data.

```
Select  “order_id”,   "order_id",  "orderDate",  "customer_id",  "Resturant_ID",  "Fooditem",
"quantity", "deliver_status","payment_method", count(*) as Total_counts
From “April_Sales_2023”
Group  by  order_id”,   "order_id",  "orderDate",  "customer_id",  "Resturant_ID",  "Fooditem",
"quantity", "deliver_status","payment_method",
Having count(*) >1;
```

Change the data type.

```
Alter table “April_Sales_2023”
Alter column “orderdate” type Date using “orderdate”::Date;
```

March_Sales_2023 and February_Sales-2023 no error

Check the January_Sales_2023 table :-

```
Select * From “January_Sales_2023”;

where "order_id" is null
or
"orderDate" is null
or
"customer_id" is null
or
"Resturant_ID" is null
or
"Fooditem" is null
or
"quantity" is null
or
"deliver_status" is null
or
Payment_method is null;
```

Remove Unnamed column

```
Alter table “January_Sales_2023”
Drop column “Unnamed: 5”;
```

Step 1 :- Delete rows where ALL columns are NULL.

```
Delete from “January_sales_2023”
Where  "order_id" IS NULL
 AND "orderDate" IS NULL
 AND "customer_id" IS NULL
 AND "Resturant_ID" IS NULL
 AND "Fooditem" IS NULL
 AND "quantity" IS NULL
 AND "deliver_status" IS NULL
 AND "payment_method" IS NULL;
```

Step 1 :- Delete rows where ANY column is NULL.

```
Delete from “January_sales_2023;
Where  "order_id" is null
   OR "orderDate" IS NULL
  OR "customer_id" IS NULL
  OR "Resturant_ID" IS NULL
  OR "Fooditem" IS NULL
  OR "quantity" IS NULL
  OR "deliver_status" IS NULL
  OR "payment_method" IS NULL;
```

Check the duplicate Order id.

```
Select “order_id” count(*) as count
From “January_Sales_2023
Group by “order_id”
Having count (*) > 1 ;
```

Delete Exact Duplicate Rows (All Column Match)

```
Delete from “January_Sales_2023” a
Using “January_Sales_2023” b
Where a.ctid  > b.ctid
AND a."order_id" = b."order_id"
  AND a."orderDate" = b."orderDate"
  AND a."customer_id" = b."customer_id"
  AND a."Resturant_ID" = b."Resturant_ID"
  AND a."Fooditem" = b."Fooditem"
  AND a."quantity" = b."quantity"
  AND a."deliver_status" = b."deliver_status"
  AND a."payment_method" = b."payment_method";
```

Change the datatype

```
alter table "January_Sales_2023"
alter column "orderDate" Type date using "orderDate"::date;
```

Check the Food Details Table

```
Select * from “Food details”
Where  “Fooditem” is null
Or
“Food” is null
Or
“type” is null;
```

Delete the null value

```
Delete from “Food details”
Where “Fooditem” is null;
```

Remove Unnamed Column

```
Alter table “Food details”
Alter column “Unnamed: 1:”  ;
```

Check the Duplicate value

```
Select “Fooditem” , count (*) as count
From “Food details”
Group by “Fooditem”
Having count (*) >  1;
```

Delete Exact Duplicate Rows (All Column Match)

```
Delete from “Food details” a
Using  “Food details" b
Where a.ctid > b.ctid
AND a."Fooditem" = b."Fooditem"
 AND a."Food" = b."Food"
 AND a."Type" = b."Type";
```

Check the customer table

```
Select * from “Customer Details”
Where  "customer_id" is null
   or
   "customer_name" is null
   or
    "member_Type" is null;
```

Delete Null value

```
Delete from “Customer Details”
where
"customer_id" is null
or
"customer_name" is null
or
"member_Type" is null;
```

Check the duplicate value

```
Select “customer_id” count (*) as count
From “Customer Details”
Group by “customer_Id”
Having count (*) > ;
```

Check the Resturant_Details

```
Select * from Resturant_Details
Where "Resturant_Code" is null
Or
"Resturant_Name” is null
Or
"Resturant_type" ;
```

Check the duplicate value

```
Select "Resturant_Code" count (*) as  count
From "Resturant_Details"
Group by "Resturant_Details"
Having count (*) > 1;
```

Delete duplicate value Resturant Details table

```
Delete  from "Resturant_Details" a
Using  "Restyrabt_Details" b
Where a.ctid > b.ctid
And a.Resturant_Code" = b.Resturant_Code"
and a."Resturant_Name" = b."Resturant_Name"
and a."Resturant_type" = b."Resturant_type";
```

How to create a relationship across all tables

1.Ensure Primary Keys Exist

Each table should have a PRIMARY KEY, which uniquely identifies each row.

Step 1: Make sure `customer_id` in Customer Details is PRIMARY KEY

```
Alter table "Customer_Details"
Add constraint  pk _customer_id primary key ("customer_id");
```

Add relationship January_Sales_2023 and Customer_Details.

```
Alter table "January_Sales_2023"
Add constraint fk_customer_id
Foreign key ("Customer_id")
References "Customer Details"  ("customer_id");
```

Add relationship February_Sales_2023 and Customer_Details.

```
Alter table "February_Sales_2023"
Add constraint fk_customer_id
Foreign key ("customer_id")
References "Customer Details" ("customer_id");
```

Add relationship March_Sales_2023 and Customer_Details.

```
Alter table "March_Sales_2023"
Add constraint fk_customer_id
Foreign key ("customer_id")
References  "Customer Details" ("customer_id");
```

Add relationship April_Sales_2023 and Customer_Details.

```
Alter table “April_Sales_2023”

Add constraint fk_customer_id

Foreign key  (“customer_id”)

References “Customer Details” (“customer_id”);
```

Change data type

```
Alter table “January_Sales_2023”
Alter column “Resturant_ID” type bigint using  “Resturant_ID”::bigint;

alter table "February_Sales_2023"
alter column "Resturant_ID" type bigint using  "Resturant_ID"::bigint;

alter table "April_Sales_2023"
alter column "Resturant_ID" type bigint  USING "Resturant_ID"::bigint ;

alter table "March_Sales_2023"
alter column "Resturant_ID" type bigint using "Resturant_ID"::bigint;
```

Add relationship January_Sales_2023 and Resturant_details .

```
Alter table “January_Sales_2023”

Add constraint fk_Resturant_Code

Foreign key (“Resturent_ID)

References “Resturant_details”  (“Resturent_Code”);
```

Make sure ` Fooditem` in Food details is PRIMARY KEY.

```
alter table "Food details"
add constraint pk_Fooditem primary key ("Fooditem");
```

Change data type

```
alter table "April_Sales_2023"
alter column "Fooditem" type bigint  USING "Fooditem"::bigint ;

alter table "March_Sales_2023"
alter column "Fooditem" type bigint using "Fooditem"::bigint;

alter table "February_Sales_2023"
alter column "Fooditem" type bigint using "Fooditem"::bigint;

alter table "January_Sales_2023"
alter column "Fooditem" type bigint using "Fooditem"::bigint;
```

Add relationship January_Sales_2023 and Food details .

```
alter table "January_Sales_2023"
add constraint fk_Fooditem
foreign key ("Fooditem")
References "Food details" ("Fooditem");
```

Top 20 SQL Insights Questions Based on Your Project.

1. Total Orders (Transactions)

```
SELECT COUNT(order_id) AS total_transactions FROM "January_Sales_2023"
UNION ALL
SELECT COUNT(order_id) FROM "February_Sales_2023"
UNION ALL
SELECT COUNT(order_id) FROM "March_Sales_2023"
UNION ALL
SELECT COUNT(order_id) FROM "April_Sales_2023";
```

2. Total Quantity Sold

```
SELECT SUM(quantity) AS total_quantity FROM "January_Sales_2023"
UNION ALL
SELECT SUM(quantity) FROM "February_Sales_2023"
UNION ALL
SELECT SUM(quantity) FROM "March_Sales_2023"
UNION ALL
SELECT SUM(quantity) FROM "April_Sales_2023";
```

3. Average Quantity per Order

```
SELECT ROUND(AVG(quantity), 2) AS avg_quantity FROM "January_Sales_2023"
UNION ALL
SELECT ROUND(AVG(quantity), 2) FROM "February_Sales_2023"
UNION ALL
SELECT ROUND(AVG(quantity), 2) FROM "March_Sales_2023"
UNION ALL
SELECT ROUND(AVG(quantity), 2) FROM "April_Sales_2023";
Select * from "January_Sales_2023";
alter table "January_Sales_2023"
alter column "quantity" type bigint using "quantity"::bigint;
```

4. Monthly Sales Quantity

```
SELECT 'January' AS month, SUM(quantity) FROM "January_Sales_2023"
UNION ALL
SELECT 'February', SUM(quantity) FROM "February_Sales_2023"
UNION ALL
SELECT 'March', SUM(quantity) FROM "March_Sales_2023"
UNION ALL
SELECT 'April', SUM(quantity) FROM "April_Sales_2023";
```

5. Monthly Transactions

```
SELECT 'January' AS month, COUNT(order_id) FROM "January_Sales_2023"
UNION ALL
SELECT 'February', COUNT(order_id) FROM "February_Sales_2023"
UNION ALL
SELECT 'March', COUNT(order_id) FROM "March_Sales_2023"
UNION ALL
SELECT 'April', COUNT(order_id) FROM "April_Sales_2023";
```

6. Month-on-Month Growth

```
-- This is a sample of growth from Jan to Feb

SELECT   'Feb over Jan' AS period,
ROUND(((feb.total - jan.total) / jan.total) * 100, 2) AS growth_percentage
FROM
   (SELECT COUNT(*) AS total FROM "February_Sales_2023") feb,
   (SELECT COUNT(*) AS total FROM "January_Sales_2023") jan;
```

7. Top 10 Customers by Quantity

```
SELECT customer_id, SUM(quantity) AS total_quantity
FROM "January_Sales_2023"
GROUP BY customer_id
ORDER BY total_quantity DESC
LIMIT 10;
```

8. Customer Segmentation by Member_Type

```
SELECT cd."member_Type", COUNT(js."order_id") AS total_orders
FROM "January_Sales_2023" js
JOIN "Customer Details" cd ON js."customer_id" = cd."customer_id"
GROUP BY cd."member_Type";
```

9. Repeat Customers Count

```
SELECT COUNT(*) FROM (
  SELECT "customer_id" FROM "January_Sales_2023"
  GROUP BY "customer_id"
  HAVING COUNT("order_id") > 1
) AS repeat_customers;
```

10. Top 10 Food Items by Quantity

```
SELECT fd."Food", SUM(js.quantity) AS total_quantity
FROM "January_Sales_2023" js
JOIN "Food details" fd ON js."Fooditem" = fd."Fooditem"
GROUP BY fd."Food"
ORDER BY total_quantity DESC
LIMIT 10;
```

11. Food Type-wise Sales

```
SELECT fd."Type", SUM(js.quantity) AS total_quantity
FROM "January_Sales_2023" js
JOIN "Food details" fd ON js."Fooditem" = fd."Fooditem"
GROUP BY fd."Type";
```

12. Food Preference by Membership Type

```
SELECT cd."member_Type", fd."Type", SUM(js."quantity") AS total_quantity
FROM "January_Sales_2023" js
JOIN "Customer Details" cd ON js."customer_id" = cd."customer_id"
JOIN "Food details" fd ON js."Fooditem" = fd."Fooditem"
GROUP BY cd."member_Type", fd."Type";
```

13. Top 5 Restaurants by Orders

```
SELECT "Resturant_ID", COUNT(*) AS total_orders
FROM "January_Sales_2023"
GROUP BY "Resturant_ID"
ORDER BY total_orders DESC
LIMIT 5;
```

14. Average Order Quantity per Restaurant

```
SELECT 'Resturant_ID', ROUND(AVG(quantity), 2) AS avg_quantity
FROM "January_Sales_2023"
GROUP BY "Resturant_ID";
```

15. Restaurant-wise Cancelled Orders

```
SELECT "Resturant_ID", COUNT(*) AS cancelled_orders
FROM "January_Sales_2023"
WHERE deliver_status = 'Cancelled'
GROUP BY "Resturant_ID";
```

16. Payment Method Usage Share

```
SELECT "payment_method", COUNT(*) AS total_orders
FROM "January_Sales_2023"
GROUP BY "payment_method";
```

17. Delivery Status Distribution

```
SELECT "deliver_status", COUNT(*) AS total_orders
FROM "January_Sales_2023"
GROUP BY "deliver_status";
```

18. Cancelled Order % per Month

```
SELECT
  'January' AS month,
  ROUND(
    (SELECT COUNT(*) FROM "January_Sales_2023" WHERE deliver_status =
'Cancelled')::numeric / COUNT(*) * 100, 2
  ) AS cancellation_rate
FROM "January_Sales_2023";
```

19. Food Item vs Quantity vs Month

```
SELECT 'January' AS month, fd."Food", SUM(js.quantity) AS total_quantity
FROM "January_Sales_2023" js
JOIN "Food details" fd ON js."Fooditem"= fd."Fooditem"
GROUP BY fd."Food"
ORDER BY total_quantity DESC;
```

20. Restaurant Performance by Month

```
SELECT 'January' AS month,"Resturant_ID", COUNT(*) AS total_orders
FROM "January_Sales_2023"
GROUP BY "Resturant_ID";
```