E-commerce Order Fulfillment & Delivery Analytics Assignment Report

-- Sayan Das

1. Introduction

In this assignment, we integrated SAP ERP data into a SQL-based data warehouse to enable robust reporting and analytics. The goal was to extract data from multiple SAP modules—covering sales orders, shipments, customer information, and carrier details—and transform it into a relational data model that supports advanced analysis in Power BI. This approach allows the e-commerce company ("ShopX") to monitor order processing times, delivery performance, and carrier efficiency, thereby optimizing fulfillment and improving customer satisfaction.

2. ETL Process Overview

Data Extraction

Data was extracted from an Excel file containing SAP ERP sheets:

- **KNA1:** Customer master data
- **LFA1:** Vendor (carrier) data
- **VBAK:** Sales order header
- **VBAP:** Sales order items
- **LIKP:** Delivery (shipment) header
- **LIPS:** Delivery (shipment) items
- **VTTK & VTTP:** Shipment status details

Data Transformation

The transformation process was designed to build a robust relational data model with the following tables:

- **customers:** Created from KNA1, this table stores customer details.
- **sap_customers:** A duplicate of customers, to retain raw SAP customer information if needed.
- **orders:** Derived from VBAK, representing order header information.
- **order_items:** Derived from VBAP, capturing the item-level details of each order.
- **shipments:** Created from LIKP, holding shipment header data.
- **shipment_items:** Derived from LIPS, containing details about individual shipment items.
- **carriers:** Populated from LFA1, storing carrier/vendor details.
- **delivery_status:** Created by merging VTTK and VTTP, this table tracks shipment statuses and related timestamps.
- **delivery_analytics:** An aggregated table that computes performance metrics such as delivery time and on-time status by joining orders and shipments.

Key transformation tasks included:

- Renaming columns to standardized lower-case names with underscores.
- Converting date fields to proper datetime formats.
- Merging related data sources (e.g., joining VBAK with VBAP, LIKP with LIPS, VTTK with VTTP) to create comprehensive tables.

- Validating data quality through primary key uniqueness and referential integrity checks.

Data Validation

Before loading the data into MySQL, we performed several validation checks:

- **Primary Key Uniqueness:** Ensured that key fields (or composite keys) are unique for tables like orders, shipments, and delivery_status. (A warning was issued for duplicate shipment numbers in delivery_status, which can be addressed by aggregating or removing duplicates based on business rules.)
- **Foreign Key Consistency:** Verified that foreign keys (e.g., customer_id in orders) exist in the corresponding parent tables.

Data Load

The validated data was loaded into a MySQL database, using SQLAlchemy, with table names stored in lower-case. This ensures consistency and avoids issues with case sensitivity in SQL queries.

3. Representative Code Snippet

Below is a representative code snippet from the Python notebook that demonstrates the extraction, transformation, validation, and loading process:

```
import pandas as pd
from sqlalchemy import create_engine
import numpy as np

# Define Excel file path and load sheets
excel_file = 'SAP-DataSet.xlsx'
xls = pd.ExcelFile(excel_file)
print("Available sheets:", xls.sheet_names)

# Extract data from SAP sheets
df_kna1 = pd.read_excel(excel_file, sheet_name='KNA1')
df_lfa1 = pd.read_excel(excel_file, sheet_name='LFA1')
df_vbak = pd.read_excel(excel_file, sheet_name='VBAK')
df_vbap = pd.read_excel(excel_file, sheet_name='VBAP')
df_likp = pd.read_excel(excel_file, sheet_name='LIKP')
df_lips = pd.read_excel(excel_file, sheet_name='LIPS')
df_vttk = pd.read_excel(excel_file, sheet_name='VTTK')
df_vttp = pd.read_excel(excel_file, sheet_name='VTTP')
```

Data Transformation

1. Customers and SAP_Customers Tables

Use the KNA1 sheet to build the customers table

```
customers_df = df_kna1.copy()
customers_df.rename(columns={
    'Customer ID': 'customer_id',
    'Customer Name': 'customer_name',
    'Country': 'country',
    'Region': 'region',
    'City': 'city',
    'Postal Code': 'postal_code',
    'Street Address': 'street_address',
    'Phone Number': 'phone_number',
    'Email Address': 'email_address',
    'Language': 'language',
    'Tax Number': 'tax_number',
    'Customer Group': 'customer_group',
    'Sales Organization': 'sales_organization',
    'Distribution Channel': 'distribution_channel',
    'Division': 'division'
}, inplace=True)
sap_customers_df = customers_df.copy()  # Optional duplicate
```

2. Orders and Order_Items Tables

Orders: Derived from VBAK (order header).
Order_Items: Derived from VBAP (order item details).

```
orders_df = df_vbak.copy()
orders_df.rename(columns={
    'Sales Document': 'order_id',
    'Order Date': 'order_date',
    'Customer ID': 'customer_id',
    'Order Type': 'order_type',
    'Sales Organization': 'sales_organization',
    'Distribution Channel': 'distribution_channel',
    'Division': 'division',
    'Order Status': 'order_status'
}, inplace=True)
orders_df['order_date'] = pd.to_datetime(orders_df['order_date'], errors='coerce')

order_items_df = df_vbap.copy()
order_items_df.rename(columns={
    'Sales Document': 'order_id',
    'Item Number': 'item_number',
    'Material Number': 'product_id',
    'Quantity': 'order_quantity',
    'Net Price': 'unit_price',
    'Item Status': 'item_status',
    'Delivery Date': 'delivery_date'
}, inplace=True)
order_items_df['delivery_date'] = pd.to_datetime(order_items_df['delivery_date'], erro
rs='coerce')
```

3. Shipments and Shipment_Items Tables

Shipments: Based on LIKP (delivery header).
Shipment_Items: Based on LIPS (delivery items).

```
shipments_df = df_likp.copy()
shipments_df.rename(columns={
    'Delivery Number': 'shipment_id',
    'Delivery Date': 'shipment_date',
    'Sales Document': 'order_id',
    'Shipping Point': 'shipping_point',
    'Shipping Type': 'shipping_type',
    'Delivery Status': 'delivery_status',
    'Shipping Status': 'shipping_status',
    'Route': 'route',
    'Delivery Priority': 'delivery_priority',
    'Customer ID': 'customer_id'
}, inplace=True)
shipments_df['shipment_date'] = pd.to_datetime(shipments_df['shipment_date'], errors='
coerce')

# %%
shipment_items_df = df_lips.copy()
shipment_items_df.rename(columns={
    'Delivery Number': 'shipment_id',
    'Item Number': 'item_number',
    'Material Number': 'product_id',
    'Delivered Quantity': 'shipped_quantity',
    'Net Price': 'unit_price',
    'Delivery Status': 'delivery_status',
    'Customer ID': 'customer_id',
    'Sales Document': 'order_id',
    'Sales Item': 'sales_item',
    'Delivery Date': 'delivery_date'
}, inplace=True)
shipment_items_df['delivery_date'] = pd.to_datetime(shipment_items_df['delivery_date']
, errors='coerce')
```

4. Carriers Table

Use the LFA1 sheet as carriers.

```
carriers_df = df_lfa1.copy()
carriers_df.rename(columns={
    'Vendor Number': 'carrier_id',
    'Vendor Name': 'carrier_name',
    'Country': 'country',
    'Region': 'region',
    'City': 'city',
    'Postal Code': 'postal_code',
    'Street Address': 'street_address',
    'Phone Number': 'phone_number',
    'Email Address': 'email_address',
    'Language': 'language',
    'Tax Number': 'tax_number',
    'Payment Terms': 'payment_terms'
}, inplace=True)
```

5. Delivery_Status Table

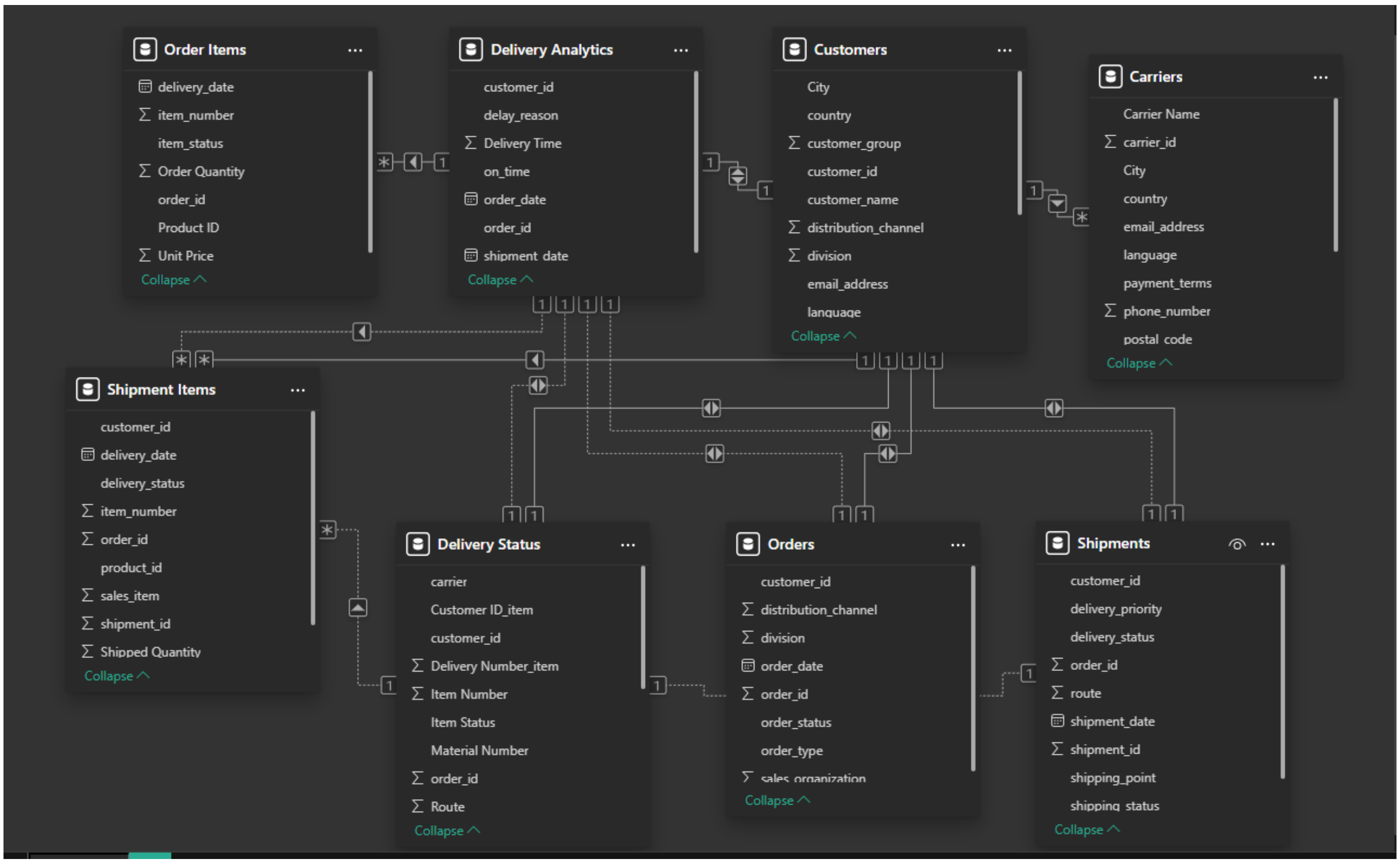
Merge VTTK (shipment header) and VTTP (shipment items) to build the delivery_status table. Because both sheets have common columns (e.g. "Shipment Date"), suffixes are applied. We use the header values for key fields by renaming:

- "Shipment Date_header" → "shipment_date"
- "Sales Document_header" → "order_id"
- "Delivery Number_header" → "shipment_id"
- "Customer ID_header" → "customer_id"

```
delivery_status_df = pd.merge(df_vttk, df_vttp, on='Shipment Number', how='inner',
                              suffixes=('_header', '_item'))
# Rename columns using header values
delivery_status_df.rename(columns={
    'Shipment Number': 'shipment_number',
    'Shipment Date_header': 'shipment_date',
    'Shipment Status': 'shipment_status',
    'Carrier': 'carrier',
    'Sales Document_header': 'order_id',
    'Delivery Number_header': 'shipment_id',
    'Customer ID_header': 'customer_id'
}, inplace=True)
delivery_status_df['shipment_date'] = pd.to_datetime(delivery_status_df['shipment_date
'], errors='coerce')
delivery_status_df.drop_duplicates(inplace=True)

# Data Validation: Check primary key uniqueness
def check_primary_key_uniqueness(df, key_columns, table_name):
    duplicates = df.duplicated(subset=key_columns)
    if duplicates.any():
        print(f"WARNING: Duplicates in {table_name} for key columns {key_columns}:")
        print(df[duplicates][key_columns])
    else:
        print(f"Primary key check passed for {table_name}.")

def check_foreign_key(child_df, child_key, parent_df, parent_key, table_name):
    missing = set(child_df[child_key].dropna()) - set(parent_df[parent_key].dropna())
    if missing:
        print(f"WARNING: In {table_name}, the following {child_key} values are missing in
parent table: {missing}")
    else:
        print(f"Foreign key check passed for {table_name} ({child_key}).")

# delivery_status_df.drop_duplicates(subset=['shipment_number'], inplace=True)

# Primary key validations
check_primary_key_uniqueness(customers_df, ['customer_id'], 'customers')
check_primary_key_uniqueness(orders_df, ['order_id'], 'orders')
check_primary_key_uniqueness(order_items_df, ['order_id', 'item_number'], 'order_items')
check_primary_key_uniqueness(shipments_df, ['shipment_id'], 'shipments')
check_primary_key_uniqueness(shipment_items_df, ['shipment_id', 'item_number'],
'shipment_items')
check_primary_key_uniqueness(carriers_df, ['carrier_id'], 'carriers')
check_primary_key_uniqueness(delivery_status_df, ['shipment_number'], 'delivery_status')
check_primary_key_uniqueness(delivery_analytics_df, ['order_id'], 'delivery_analytics')

# Foreign key validations
check_foreign_key(orders_df, 'customer_id', customers_df, 'customer_id', 'orders')
check_foreign_key(order_items_df, 'order_id', orders_df, 'order_id', 'order_items')
check_foreign_key(shipments_df, 'order_id', orders_df, 'order_id', 'shipments')
check_foreign_key(shipment_items_df, 'order_id', orders_df, 'order_id', 'shipment_items')
check_foreign_key(delivery_status_df, 'order_id', orders_df, 'order_id',
'delivery_status')
check_foreign_key(delivery_analytics_df, 'customer_id', customers_df, 'customer_id',
'delivery_analytics')

# Load Data into MySQL
username = 'root'
password = '12345'
host = 'localhost'
port = '3306'
database = 'case1'
engine = create_engine(f'mysql+pymysql://{username}:{password}@{host}:{port}/{database}')
orders_df.to_sql('orders', con=engine, if_exists='replace', index=False)
```

```
print("Data loaded to MySQL database successfully.")
```

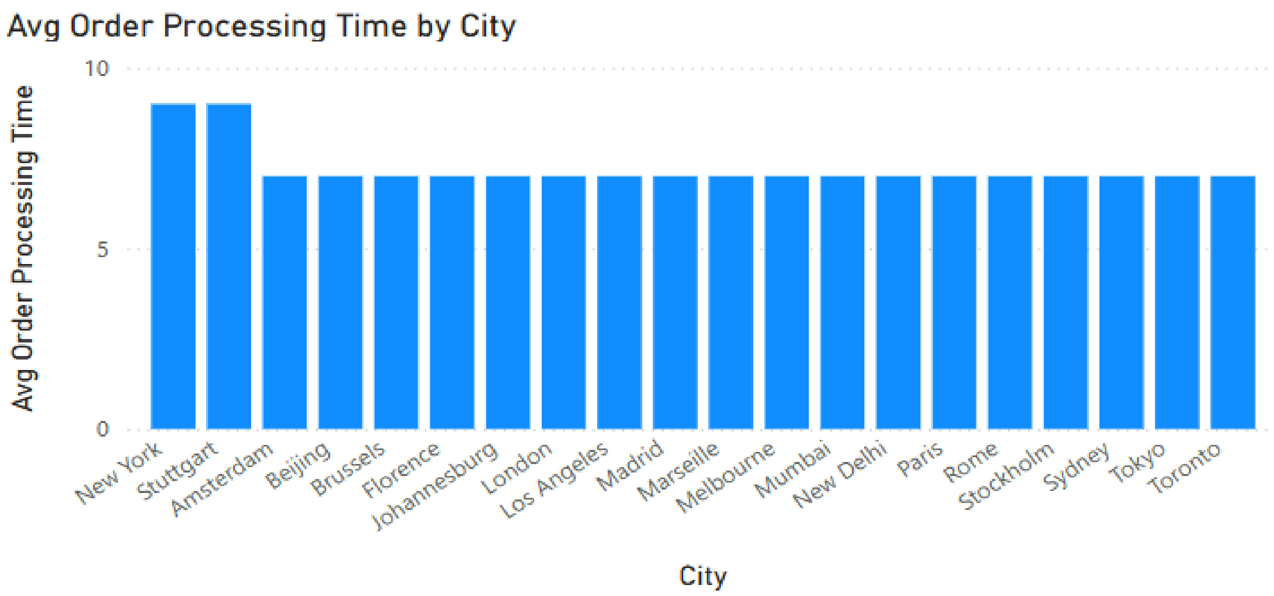
SQL Schema



4. Power BI Reporting and Analytics

After loading the data into the MySQL database, the next step is to build Power BI reports and dashboards to visualize key performance metrics. The following KPIs and graphs are planned for the Power BI dashboard:

- **Order Processing Time: Difference between delivery date and order date**
  - *Minimum time – 7* days
  - *Maximum time – 9 days*

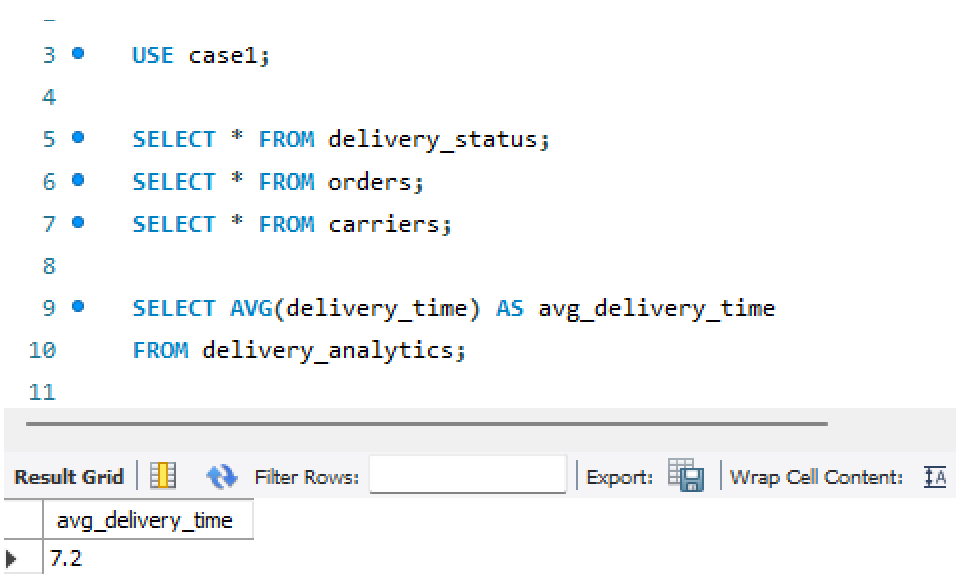


On-

- **Time Delivery Rate: Percentage of deliveries that arrive on time**

- *None of the packages was delivered on time.*

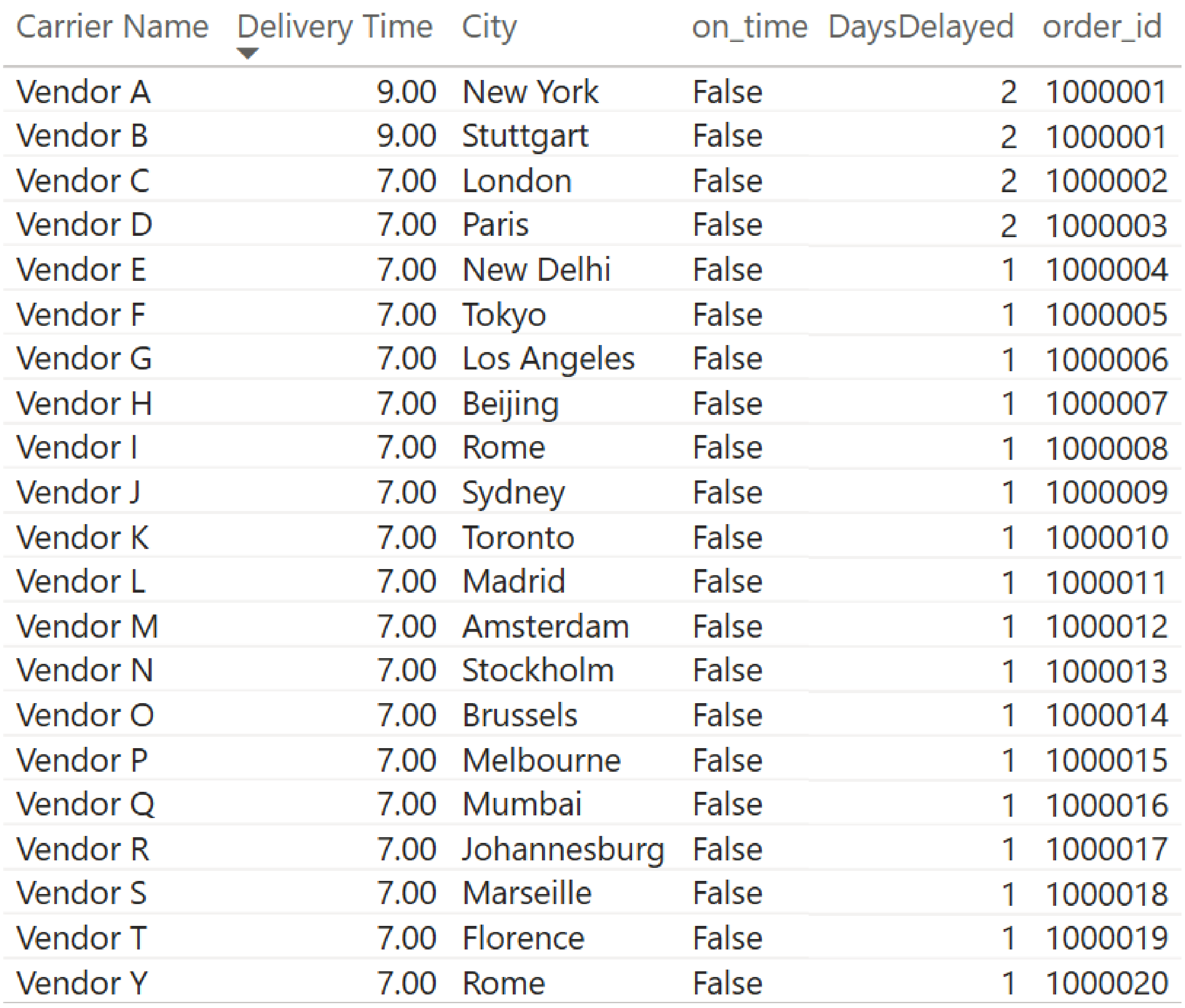
- **Average Delivery Time:**



```
USE case1;

SELECT * FROM delivery_status;
SELECT * FROM orders;
SELECT * FROM carriers;

SELECT AVG(delivery_time) AS avg_delivery_time
FROM delivery_analytics;
```

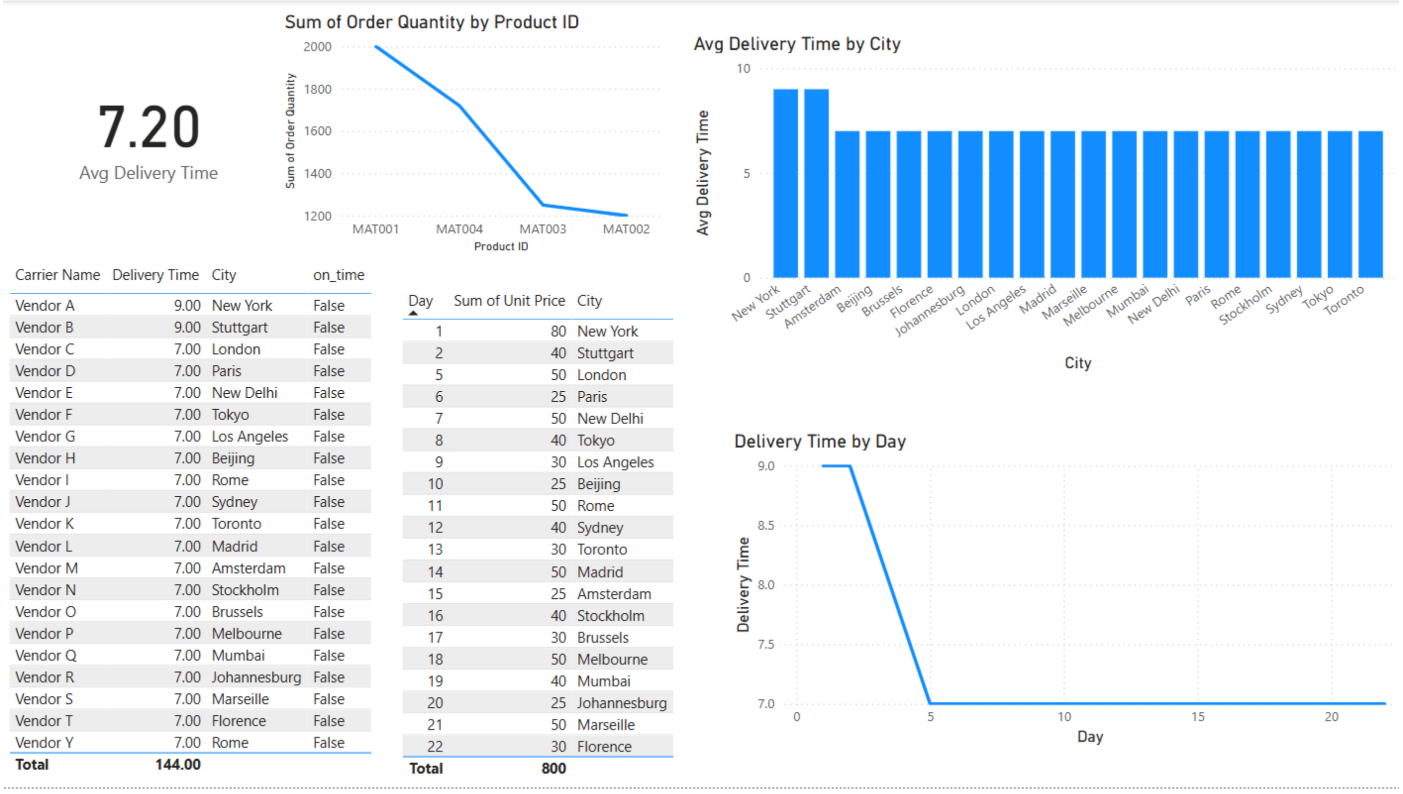
avg_delivery_time
7.2

- **Carrier Performance: Carrier's performance by aggregating the average delivery time**



Carrier Name	Delivery Time	City	on_time	DaysDelayed	order_id
Vendor A	9.00	New York	False	2	1000001
Vendor B	9.00	Stuttgart	False	2	1000001
Vendor C	7.00	London	False	2	1000002
Vendor D	7.00	Paris	False	2	1000003
Vendor E	7.00	New Delhi	False	1	1000004
Vendor F	7.00	Tokyo	False	1	1000005
Vendor G	7.00	Los Angeles	False	1	1000006
Vendor H	7.00	Beijing	False	1	1000007
Vendor I	7.00	Rome	False	1	1000008
Vendor J	7.00	Sydney	False	1	1000009
Vendor K	7.00	Toronto	False	1	1000010
Vendor L	7.00	Madrid	False	1	1000011
Vendor M	7.00	Amsterdam	False	1	1000012
Vendor N	7.00	Stockholm	False	1	1000013
Vendor O	7.00	Brussels	False	1	1000014
Vendor P	7.00	Melbourne	False	1	1000015
Vendor Q	7.00	Mumbai	False	1	1000016
Vendor R	7.00	Johannesburg	False	1	1000017
Vendor S	7.00	Marseille	False	1	1000018
Vendor T	7.00	Florence	False	1	1000019
Vendor Y	7.00	Rome	False	1	1000020

Full Report-



7.20
Avg Delivery Time

Carrier Name	Delivery Time	City	on_time
Vendor A	9.00	New York	False
Vendor B	9.00	Stuttgart	False
Vendor C	7.00	London	False
Vendor D	7.00	Paris	False
Vendor E	7.00	New Delhi	False
Vendor F	7.00	Tokyo	False
Vendor G	7.00	Los Angeles	False
Vendor H	7.00	Beijing	False
Vendor I	7.00	Rome	False
Vendor J	7.00	Sydney	False
Vendor K	7.00	Toronto	False
Vendor L	7.00	Madrid	False
Vendor M	7.00	Amsterdam	False
Vendor N	7.00	Stockholm	False
Vendor O	7.00	Brussels	False
Vendor P	7.00	Melbourne	False
Vendor Q	7.00	Mumbai	False
Vendor R	7.00	Johannesburg	False
Vendor S	7.00	Marseille	False
Vendor T	7.00	Florence	False
Vendor Y	7.00	Rome	False
Total	**144.00**		

Day	Sum of Unit Price	City
1	80	New York
2	40	Stuttgart
5	50	London
6	25	Paris
7	50	New Delhi
8	40	Tokyo
9	30	Los Angeles
10	25	Beijing
11	50	Rome
12	40	Sydney
13	30	Toronto
14	50	Madrid
15	25	Amsterdam
16	40	Stockholm
17	30	Brussels
18	50	Melbourne
19	40	Mumbai
20	25	Johannesburg
21	50	Marseille
22	30	Florence
Total	**800**	

5. Conclusion

In summary, we have:

- **Extracted** raw SAP ERP data from an Excel file containing multiple sheets.
- **Transformed** the data into a structured relational model that includes tables for customers, orders (header and items), shipments (header and items), carriers, delivery_status, and delivery_analytics.
- **Validated** the data for completeness, uniqueness, and referential integrity.
- **Loaded** the clean, transformed data into a MySQL database.
- **Prepared** the environment for Power BI reporting by creating key measures and KPIs (such as Order Processing Time, On-Time Delivery Rate, Average Delivery Time, and Carrier Performance).

We found that orders were always delivered late and Vendor A,B,C and D were worse than rest.
ShopX needs to work on improving their delivery time.