Retail Orders Analysis

**Prepared by:** Anvitha Shetty  
**Project Title:** Retail Orders Analysis and Business Insights  
**Tools & Technologies:**

**Python** (Pandas, SQLAlchemy)

**SQL Server**

**Kaggle Dataset**

**Date:** June 2025

## Project Overview

This project analyzes retail sales data to:

* Extract actionable insights on sales, revenue, and profits.
* Identify top-performing products, categories, and regions.
* Compare year-over-year and month-over-month growth.
* Highlight areas for potential improvement and revenue optimization.

**Objective:**  
Help stakeholders make data-driven decisions by converting raw transactional data into a clean, structured, and insightful dataset.

## Package Installation and Importing

### Package Installation

#%pip install pandas

#%pip install sqlalchemy

#%pip install pyodbc

* These lines install three key packages:
  + **pandas**: Enables powerful data manipulation and analysis.
  + **sqlalchemy**: Provides an SQL toolkit and Object Relational Mapper (ORM) for managing databases.
  + **pyodbc**: Enables connections to relational databases via ODBC drivers.
* The %pip syntax is used for installation in interactive environments like Jupyter notebooks.

**Note**: These lines are commented out (#) intentionally because installations are usually done only once and may not be required every time the script is run.

### Importing Packages

import kaggle

import pandas as pd

import sqlalchemy as sal

This section sets up the environment by:

* Installing required packages (if needed).
* Importing the core libraries required for data downloading, processing, and database interactions.
* import kaggle: Enables access to the Kaggle API for downloading datasets and submitting results.
* import pandas as pd: Provides data structures (DataFrame, Series) and methods for data wrangling.
* import sqlalchemy as sal: Enables database connectivity, allowing the script to build and run SQL queries through a Pythonic interface.

## Dataset Download, Extraction, and Loading

This section downloads the dataset from Kaggle, extracts it from a ZIP archive, and loads it into a Pandas DataFrame. It also gives an initial glance at the data.

### Downloading the Dataset from Kaggle

!kaggle datasets download ankitbansal06/retail-orders -f orders.csv

**What this does:**

* Uses Kaggle’s CLI tool (kaggle datasets download) to fetch the dataset named retail-orders uploaded by ankitbansal06.
* The -f orders.csv parameter specifies downloading the orders.csv file from the dataset.

⚡️ **Note**: You must have configured Kaggle API credentials beforehand (i.e., .kaggle directory with kaggle.json) for this command to work.

### Extracting the ZIP File

import zipfile

zip\_ref = zipfile.ZipFile('orders.csv.zip')

zip\_ref.extractall()

zip\_ref.close(

**What this does:**

* import zipfile: Imports the built-in Python module for working with ZIP files.
* zip\_ref = zipfile.ZipFile('orders.csv.zip'): Opens the ZIP archive.
* zip\_ref.extractall(): Extracts all files from the ZIP archive to the current working directory.
* zip\_ref.close(): Closes the ZIP archive, releasing resources.

### Loading the CSV into a Pandas DataFrame

df = pd.read\_csv('orders.csv')

df.head(20)

**What this does:**

* Reads the orders.csv file into a Pandas DataFrame named **df**.
* Displays the first 20 rows of the dataset using **head()** for a quick inspection of its structure and contents.

## Data Inspection and Cleaning

This section inspects the data for column data types, applies necessary transformations, and cleans the column names and data. It also handles null or placeholder values.

### Checking Initial Data Types

df.dtypes

**What this does:**

* Displays the data types of all columns in the DataFrame df.
* This allows you to understand how columns have been interpreted (e.g., object, int64, float64, etc.).

### Converting order\_date to DateTime

df['order\_date'] = pd.to\_datetime(df['order\_date'], format='%Y-%m-%d')

df.dtypes

**What this does:**

* Converts the order\_date column from a string (object) to a DateTime object, making it usable for time-based operations.
* Displays the updated column data types to confirm the change.

### Renaming Columns

df.columns = df.columns.str.lower()

df.columns = df.columns.str.replace(' ', '\_')

df.columns

**What this does:**

* Converts all column names to **lowercase** for consistency.
* Replaces spaces in column names with **underscores** (\_) for better compatibility with coding conventions.
* Displays the updated column names.

### Viewing Unique Ship Mode Values

df['Ship Mode'].unique()

**What this does:**

* Displays unique values in the Ship Mode column. This is useful for understanding the different shipment methods available in the dataset.

### Handling Null Values

df = pd.read\_csv('orders.csv',na\_values=['Not Available','unknown'])

df['Ship Mode'].unique()

**What this does:**

* Reloads the dataset, this time treating strings 'Not Available' and 'unknown' as **missing values** (NaN), making the dataset cleaner for analysis.
* Displays unique values of the Ship Mode column after this operation, confirming the null value handling.

## Feature Engineering and Cleanup

This section focuses on creating new calculated columns (discount, sale\_price, and profit) to make the data more insightful and actionable. It then removes redundant columns (cost\_price, discount\_percent, and list\_price) that are no longer required.

### Deriving New Columns

df['discount']=df['list\_price']\*df['discount\_percent']\*.01

df['sale\_price']= df['list\_price']-df['discount']

df['profit']=df['sale\_price']-df['cost\_price']

df

**What this does:**

* **discount**: Calculates the discount amount based on list\_price and discount\_percent.
* **sale\_price**: Determines the final sale price by subtracting the discount from the list price.
* **profit**: Computes the total profit for each transaction by subtracting the cost price from the sale price.

### Dropping Redundant Columns

df.drop(columns=['cost\_price', 'discount\_percent', 'list\_price'], inplace=True)

df.columns

**What this does:**

* Removes columns that have already been used to compute new metrics:
  + cost\_price
  + discount\_percent
  + list\_price
* Displays the updated column names to confirm their removal.

### Viewing Final Output

df.head(20)

**What this does:**  
Displays the first 20 rows of the modified dataset, allowing for a quick review of:

* New columns (discount, sale\_price, and profit).
* Final column structure.

## Loading the Final Dataset to SQL Server

This section connects to a Microsoft SQL Server database using SQLAlchemy and loads the final cleansed dataset (df) into a table named df\_orders. This allows for further relational database operations and persistent storage of results.

### Establishing a Connection to SQL Server

import sqlalchemy as sal

# Build the connection string

connection\_string = (

    "mssql+pyodbc://ASUS\_ROG\\SQLEXPRESS/master?"

    "driver=ODBC+Driver+17+for+SQL+Server"

)

# Create engine and connect

engine = sal.create\_engine(connection\_string)

conn = engine.connect()

**What this does:**

* Constructs a SQLAlchemy **connection string** that specifies:
  + The database engine (mssql+pyodbc).
  + The server (ASUS\_ROG\SQLEXPRESS).
  + The database (master).
  + The ODBC driver (ODBC Driver 17 for SQL Server).
* Uses sal.create\_engine() to create a database engine.
* Establishes a live database **connection** (conn) to run SQL queries.

### Loading the Final DataFrame into SQL Server

#load the data into sql server using append option

df.to\_sql('df\_orders', con=conn , index=False, if\_exists = 'append')

**What this does:**

* Uses the **to\_sql()** method to load the Pandas DataFrame (df) into a SQL table named **df\_orders**.
* index=False: Excludes the Pandas index from being added as a column.
* if\_exists='append': Appends the data to an existing table named df\_orders. If the table doesn’t exist, it will be created.

## SQL Scripts for Final Analysis

This section contains **SQL queries** designed to:

* Create the final table structure.
* Perform in-depth analyses on sales, revenue, and profit.
* Identify trends across products, regions, and months.

### Creating the Final Table

CREATE TABLE df\_orders (

order\_id INT PRIMARY KEY,

order\_date DATE,

ship\_mode VARCHAR(20),

segment VARCHAR(20),

country VARCHAR(20),

city VARCHAR(20),

state VARCHAR(20),

postal\_code VARCHAR(20),

region VARCHAR(20),

category VARCHAR(20),

sub\_category VARCHAR(20),

product\_id VARCHAR(50),

quantity INT,

discount DECIMAL(7,2),

sale\_price DECIMAL(7,2),

profit DECIMAL(7,2)

);

**What this does:**  
Creates the df\_orders table schema in SQL Server, matching the structure of your final DataFrame.

**Note**: This is commented out because it's typically run only once to initialize the database.

### Viewing the Final Table

SELECT \* FROM df\_orders;

**What this does:**  
Displays all data within the df\_orders table for review and validation.

### Top 10 Highest Revenue Generating Products

SELECT TOP 10 product\_id,

sum(sale\_price) AS sales

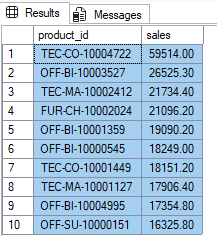
FROM df\_orders

GROUP BY product\_id

ORDER BY sales DESC;

**What this does:**  
Identifies the **top 10 products** with the highest total sales across the dataset.

Based on the analysis, the top 10 products with the highest total sales are:



**Insight:**  
These products have generated the highest revenue and should be prioritized for marketing, promotion, and inventory planning.

### Top 5 Highest Selling Products in Each Region

WITH RankedProducts AS (

SELECT region, product\_id,

SUM(sale\_price) AS total\_sales,

ROW\_NUMBER() OVER (PARTITION BY region ORDER BY SUM(sale\_price) DESC) AS sales\_rank

FROM df\_orders

GROUP BY region, product\_id

)

SELECT region, product\_id, total\_sales

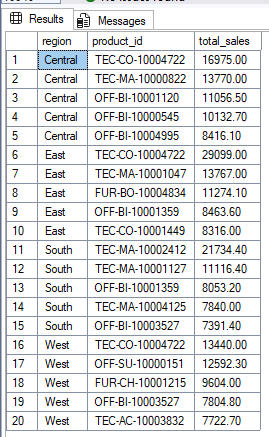
FROM RankedProducts

WHERE sales\_rank <= 5

ORDER BY region, total\_sales DESC;

**What this does:**  
Displays the **top 5 products** by sales for each region, making it ideal for targeted marketing and inventory decisions.

The following table highlights the **Top 5 Products** by total sales in each region, making it ideal for **regional marketing** and **stock optimization**.



**Insight:**  
Each region has its own best-selling products, making this table valuable for:

* **Regional Marketing Strategies** — Targeting top-performing products.
* **Inventory Optimization** — Ensuring popular products have adequate stock across different regions.

### Month-Over-Month Growth Comparison (2022 vs 2023)

WITH MonthlyGrowth AS (

SELECT

YEAR(order\_date) AS order\_year,

MONTH(order\_date) AS order\_month,

SUM(sale\_price) AS total\_sales

FROM df\_orders

GROUP BY YEAR(order\_date), MONTH(order\_date)

)

SELECT

order\_month,

SUM(CASE WHEN order\_year = 2022 THEN total\_sales ELSE 0 END) AS sales\_2022,

SUM(CASE WHEN order\_year = 2023 THEN total\_sales ELSE 0 END) AS sales\_2023

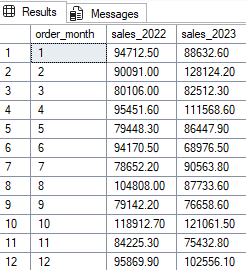
FROM MonthlyGrowth

GROUP BY order\_month

ORDER BY order\_month;

**What this does:**  
Compares total sales month-by-month across the years 2022 and 2023, providing insights into **seasonality** and **annual performance trends**.

The table below compares total sales (sale\_price) for each month across **2022** and **2023**. This provides valuable insights into **seasonal trends** and **yearly growth**:



**Insights:**

* Strong performance in **October** across both years, indicating a **peak sales period**.
* Notable growth in **February 2023** compared to 2022 (increase of ~38,000), suggesting potential effectiveness of promotions, new products, or pricing strategies.
* A slight drop in certain months like **June** and **September** in 2023 compared to 2022, indicating areas for targeted sales improvement.

### Best Month for Each Category

WITH CategoryMonthSales AS (

SELECT

category,

YEAR(order\_date) AS order\_year,

MONTH(order\_date) AS order\_month,

SUM(sale\_price) AS total\_sales

FROM df\_orders

GROUP BY category, YEAR(order\_date), MONTH(order\_date)

),

RankedSales AS (

SELECT

category,

order\_year,

order\_month,

total\_sales,

ROW\_NUMBER() OVER (PARTITION BY category ORDER BY total\_sales DESC) AS sales\_rank

FROM CategoryMonthSales

)

SELECT category, order\_year, order\_month, total\_sales

FROM RankedSales

WHERE sales\_rank = 1

ORDER BY category;

**What this does:**  
Identifies the **month of highest sales for each category** across available years.

This table identifies the **highest sales month** for each category across the years available in the dataset:



**Insights:**

* **Furniture** peaked in October 2022.
* **Office Supplies** had its best month in February 2023.
* **Technology** surged in October 2023, making it the highest-performing category and month in this comparison.

### Sub-Category Profit Growth (2023 vs 2022)

WITH ProfitByYear AS (

SELECT

sub\_category,

YEAR(order\_date) AS order\_year,

SUM(profit) AS total\_profit

FROM df\_orders

GROUP BY sub\_category, YEAR(order\_date)

),

Growth AS (

SELECT

sub\_category,

SUM(CASE WHEN order\_year = 2022 THEN total\_profit ELSE 0 END) AS profit\_2022,

SUM(CASE WHEN order\_year = 2023 THEN total\_profit ELSE 0 END) AS profit\_2023,

SUM(CASE WHEN order\_year = 2023 THEN total\_profit ELSE 0 END)

- SUM(CASE WHEN order\_year = 2022 THEN total\_profit ELSE 0 END) AS profit\_growth

FROM ProfitByYear

GROUP BY sub\_category

)

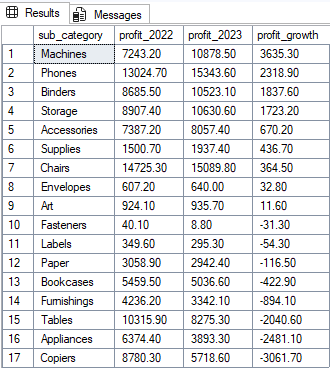
SELECT sub\_category, profit\_2022, profit\_2023, profit\_growth

FROM Growth

ORDER BY profit\_growth DESC;

**What this does:**  
Highlights the **sub-categories** that achieved the highest profit growth between **2022** and **2023**, making it ideal for identifying growth hotspots.

This table showcases the **profit change** for each sub-category between **2022** and **2023**:



**Insights:**  
 **Top Growth Sub-Categories:**

* **Machines** (+₹3,635.30): Strong upward trend
* **Phones** (+₹2,318.90): Steady growth indicating increased demand
* **Binders** (+₹1,837.60): Significantly higher profitability

**Declining Sub-Categories:**

* **Copiers** (-₹3,061.70): Notable drop in profitability
* **Appliances** (-₹2,481.10): Decrease suggests pricing, demand, or cost challenges
* **Tables** (-₹2,040.60): Requires review of sales or pricing strategies