End-to-End Sales Data Analysis using Python & SQL Server

"Comprehensive Sales and Order Analysis Using Python & SQL"

Overview

I developed an end-to-end data analysis pipeline using Kaggle datasets, Python, and SQL Server. The project began with downloading the dataset through the Kaggle API, followed by extensive data cleaning and pre-processing in Python using Pandas. After transforming and structuring the data, I loaded it into SQL Server for further analysis. I used SQL queries to extract insights, perform aggregations, and analyse trends across different categories and time periods. The project demonstrates my ability to work across data acquisition, processing, storage, and analytical stages using real-world tools and workflows.

Dataset Structure

The order table contains the following columns:

```
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)
);
```

Key Business Questions & Analysis

- **Which products generate the highest revenue?**
  Analysis: Aggregate sales by product_id, sort descending and select top N (SUM(sale_price), GROUP BY, ORDER BY). Use results to prioritize stocking and supplier negotiations.
- **What are the top 5 selling products in each region?**
  Analysis: Aggregate revenue by region, product_id, then use ROW_NUMBER() OVER (PARTITION BY region ORDER BY rev_gen DESC) to rank and filter top 5 per region. Use for region-specific merchandising and promotions.
- **How do monthly sales compare between 2022 and 2023 (MoM comparison)?**
  Analysis: Derive order_year and order_month, aggregate sales, then pivot year values using SUM(CASE WHEN order_year=... THEN sales ELSE 0 END) to compare same months across years. Use for performance tracking and anomaly detection.
- **For each category, which month produced the highest sales?**
  Analysis: Group by category and formatted month (FORMAT(order_date,'yyyyMM')), sum sales, then rank with ROW_NUMBER() partitioned by category to pick the top month. Use to inform seasonal campaign timing.
- **Which sub-category showed the highest profit growth in 2023 vs 2022?**
  Analysis: Aggregate sales/profit by sub_category and year, compute absolute and percentage growth ((2023-2022)/2022)*100, then order to find the biggest gainers. Use to identify fast-growing lines and reallocate marketing spend.

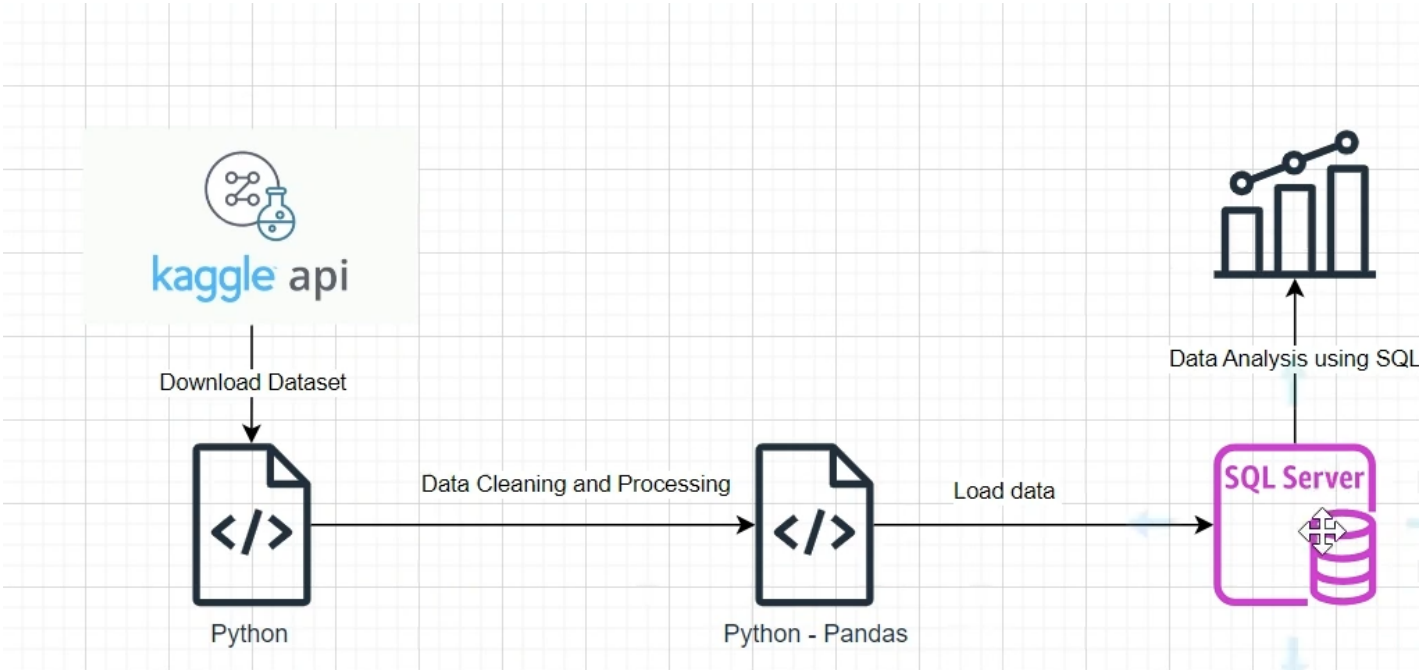
Expected Outcomes

- Actionable list of **top revenue products** and **region-specific top sellers** for inventory and promotion prioritization.
- Month-over-month and year-over-year comparisons to reveal **growth/decline trends** and seasonal patterns.
- Identification of **high-growth sub-categories** to focus marketing, pricing, and procurement decisions.
- A reusable SQL/Python workflow (ETL → cleaned table → analysis queries) to support regular reporting and faster decision cycles.
- Clear KPIs (revenue, profit growth, % change) to measure impact of future campaigns or operational changes.

Technologies & Methods Used

- **Languages / Tools:** Python (Pandas), SQL (T-SQL / SQL Server), Kaggle API (data acquisition).
- **Database:** SQL Server (tables loaded via Python connectors).
- **Techniques / SQL features:** Aggregations (SUM, GROUP BY), conditional aggregation (CASE), window functions (ROW_NUMBER()), derived tables/subqueries, date functions (YEAR, MONTH, FORMAT) and sorting/TOP selection.
- **Data tasks:** Data cleaning & preprocessing (null handling, date formatting, type casting) in Pandas, ETL to SQL Server, analytic querying for insights.
- **Deliverables:** SQL scripts, cleaned dataset, README/GitHub note, and summary reports with recommended actions.

Project



1. Importing Required Libraries

- Installed and imported necessary libraries such as `kaggle`, `pandas`, `zipfile`, and `sqlalchemy` for data handling, manipulation, and SQL integration.

```
[1]: #import libraries
     !pip install kaggle
```

```
Requirement already satisfied: kaggle in c:\users\sushil kumar\anaconda3\lib\site-packages (1.7.4.5)
Requirement already satisfied: bleach in c:\users\sushil kumar\anaconda3\lib\site-packages (from kaggle) (4.1.0)
Requirement already satisfied: certifi>=14.05.14 in c:\users\sushil kumar\anaconda3\lib\site-packages (from kaggle) (2024.8.30)
Requirement already satisfied: charset-normalizer in c:\users\sushil kumar\anaconda3\lib\site-packages (from kaggle) (3.3.2)
Requirement already satisfied: idna in c:\users\sushil kumar\anaconda3\lib\site-packages (from kaggle) (3.7)
Requirement already satisfied: protobuf in c:\users\sushil kumar\anaconda3\lib\site-packages (from kaggle) (4.25.3)
Requirement already satisfied: python-dateutil>=2.5.3 in c:\users\sushil kumar\anaconda3\lib\site-packages (from kaggle) (2.9.0.post0)
Requirement already satisfied: python-slugify in c:\users\sushil kumar\anaconda3\lib\site-packages (from kaggle) (5.0.2)
```

2. Downloading the Dataset

- Downloaded the `orders.csv` dataset from Kaggle using Kaggle API.
- Extracted the CSV file from the downloaded zip file.

```
[2]: # Download the datasets
     import kaggle

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

```
Dataset URL: https://www.kaggle.com/datasets/ankitbansal06/retail-orders
License(s): CC0-1.0
Downloading orders.csv to C:\Users\SUSHIL KUMAR\Desktop\PYTHON SQL


  0%|          | 0.00/200k [00:00<?, ?B/s]
100%|##########| 200k/200k [00:00<00:00, 103MB/s]
```

```
[5]: #extract file from zip file
     import zipfile
     zip_ref = zipfile.ZipFile('orders.csv.zip')
     zip_ref.extractall() # extract file to dir
     zip_ref.close() # close file
```

3. Loading Data and Handling Null Values

- Loaded the CSV data into a pandas DataFrame.
- Handled null and unknown values by replacing `'Not Available'` and `'unknown'` with NaN.

```
[11]: df['Ship Mode'].unique()
```

```
[11]: array(['Second Class', 'Standard Class', 'Not Available', 'unknown',
             'First Class', nan, 'Same Day'], dtype=object)
```

```
[12]: #read data from the file and handle null values
      df = pd.read_csv('orders.csv',na_values=['Not Available','unknown'])
      df['Ship Mode'].unique()
```

```
[12]: array(['Second Class', 'Standard Class', nan, 'First Class', 'Same Day'],
            dtype=object)
```

4. Data Cleaning

- Standardized column names: converted to lowercase and replaced spaces with underscores.

```
[15]: #rename columns names and make them lower case and replace space with underscore
      #df.rename(columns={'Order Id':'order_id', 'City':'city'})
      df.columns=df.columns.str.lower()
      df.columns=df.columns.str.replace(' ','_')
      df.head(5)
```

[15]:

	order_id	order_date	ship_mode	segment	country	city	state	postal_code	region	category	sub_category	product_id
0	1	2023-03-01	Second Class	Consumer	United States	Henderson	Kentucky	42420	South	Furniture	Bookcases	FUR-BO-10001798
1	2	2023-08-15	Second Class	Consumer	United States	Henderson	Kentucky	42420	South	Furniture	Chairs	FUR-CH-10000454

5. Feature Engineering

- Derived new columns:
  - `discount` = list_price × discount_percent × 0.01
  - `sale_price` = list_price − discount
  - `profit` = sale_price − cost_price
  -

```
[17]: #derive new columns discount, sale price and profit
      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
```

6. Data Type Conversion

- Converted the `order_date` column from object/string type to datetime format.

```
[21]: #convert order date from object data type to datetime
      df['order_date']=pd.to_datetime(df['order_date'],format="%Y-%m-%d")
      df.dtypes
```

```
[21]: order_id                    int64
      order_date          datetime64[ns]
      ship_mode                   object
      segment                     object
      country                     object
```

8. Loading Data into SQL Server

- Created a connection to SQL Server using `SQLAlchemy`.
- Loaded the processed data into a SQL table `df_orders` using both `replace` and `append` options.

```
[25]: #load the data into sql server using replace option
      import sqlalchemy as sal
      engine = sal.create_engine(
          r"mssql://DESKTOP-AI51EOV\SQLEXPRESS/master?driver=ODBC+DRIVER+17+FOR+SQL+SERVER"
      )

      conn=engine.connect()
```

```
[27]: #load the data into sql server using append option
      df.to_sql('df_orders', con=conn , index=False, if_exists = 'append')
```

```
[27]: 38
```

9. Outcome

- Successfully cleaned, transformed, and loaded the retail orders dataset into SQL Server.
- Derived metrics like `discount`, `sale_price`, and `profit` for further business insights and reporting.

SQL Analysis on Retail Orders Dataset

After cleaning and loading the dataset into SQL Server, several business questions were addressed using SQL queries.

1. Top 10 Highest Revenue Generating Products

- Calculated total revenue (`sale_price`) for each product and retrieved the top 10 products contributing the highest revenue.

Outcome: Identified the most profitable products for business prioritization.

```
--find top 10 highest reveue generating products

select top 10 product_id, sum(sale_price) as rev_gen
from df_orders
group by product_id
order by rev_gen desc
```

	product_id	rev_gen
1	TEC-CO-10004722	59514.00
2	OFF-BI-10003527	26525.30
3	TEC-MA-10002412	21734.40
4	FUR-CH-10002024	21096.20
5	OFF-BI-10001359	19090.20
6	OFF-BI-10000545	18249.00
7	TEC-CO-10001449	18151.20
8	TEC-MA-10001127	17906.40
9	OFF-BI-10004995	17354.80
10	OFF-SU-10000151	16325.80

2. Top 5 Highest Selling Products in Each Region

- Used `ROW_NUMBER()` with partitioning by region to find the top 5 products in terms of revenue for each region.

Outcome: Highlighted regional product performance for targeted marketing.

```
--find top 5 highest selling products in each region
select * from(
	select *,ROW_NUMBER() over(partition by region order by rev_gen desc) as rn from(
		select region, product_id, sum(sale_price) as rev_gen
		from df_orders
		group by region, product_id)A) B
	where rn<=5
```

	region	product_id	rev_gen	rn
1	Central	TEC-CO-10004722	16975.00	1
2	Central	TEC-MA-10000822	13770.00	2
3	Central	OFF-BI-10001120	11056.50	3
4	Central	OFF-BI-10000545	10132.70	4
5	Central	OFF-BI-10004995	8416.10	5
6	East	TEC-CO-10004722	29099.00	1
7	East	TEC-MA-10001047	13767.00	2
8	East	FUR-BO-10004834	11274.10	3
9	East	OFF-BI-10001359	8463.60	4
10	East	TEC-CO-10001449	8316.00	5
11	South	TEC-MA-10002412	21734.40	1
12	South	TEC-MA-10001127	11116.40	2
13	South	OFF-BI-10001359	8053.20	3
14	South	TEC-MA-10004125	7840.00	4
15	South	OFF-BI-10003527	7391.40	5
16	West	TEC-CO-10004722	13440.00	1
17	West	OFF-SU-10000151	12592.30	2
18	West	FUR-CH-10001215	9604.00	3
19	West	OFF-BI-10003527	7804.80	4
20	West	TEC-AC-10003832	7722.70	5

3. Month-over-Month Sales Growth Comparison (2022 vs 2023)

- Compared monthly sales between 2022 and 2023 using conditional aggregation.

Outcome: Analyzed sales trends and growth across years.

```
--find month over month growth comparison for 2022 and 2023 sales eg : jan 2022 vs jan 2023

select order_month,
       sum(case when order_year=2022 then sales else 0 end) as sales_2022,
       sum(case when order_year=2023 then sales else 0 end) as sales_2023
from(
       select  year(order_date) as order_year ,MONTH(order_date)as order_month,sum(sale_price)
as sales
       from df_orders
       group by year(order_date),MONTH(order_date))A
group by order_month
```

	order_month	sales_2022	sales_2023
1	1	94712.50	88632.60
2	2	90091.00	128124.20
3	3	80106.00	82512.30
4	4	95451.60	111568.60
5	5	79448.30	86447.90
6	6	94170.50	68976.50
7	7	78652.20	90563.80
8	8	104808.00	87733.60
9	9	79142.20	76658.60
10	10	118912.70	121061.50
11	11	84225.30	75432.80
12	12	95869.90	102556.10

4. Month with Highest Sales for Each Category

- Identified which month in the dataset recorded the highest sales for each category.

Outcome: Helped determine peak months for each product category.

```sql
--for each category which month had highest sales
select * from(
select *, ROW_NUMBER() over(PARTITION BY category ORDER BY sales DESC) AS rn
from(
	select FORMAT(order_date, 'yyyyMM') AS order_year_month,category,SUM(sale_price) as
sales
	from df_orders
	group by FORMAT(order_date, 'yyyyMM'),category)A) B

where rn=1
```

	order_year_month	category	sales	rn
1	202210	Furniture	42888.90	1
2	202302	Office Supplies	44118.50	1
3	202310	Technology	53000.10	1

5. Sub-Category with Highest Profit Growth (2023 vs 2022)

- Calculated year-over-year profit growth for each sub-category and found the sub-category with maximum growth.

Outcome: Provided insights on high-growth sub-categories for strategic focus.

```sql
--which sub category had highest growth by profit in 2023 compare to 2022

select top 1*, (sum_sales_2023-sum_sales_2022)*100/sum_sales_2022 as perc_growth
from(
select sub_category,
sum(case when order_year=2022 then sales else 0 end)as sum_sales_2022,
sum(case when order_year=2023 then sales else 0 end)as sum_sales_2023
from(
	select sub_category,YEAR(order_date) AS order_year,SUM(sale_price) AS sales
	from df_orders
	group by sub_category, YEAR(order_date))A
group by sub_category) B
order by (sum_sales_2023-sum_sales_2022)*100/sum_sales_2022 desc
```

	sub_category	sum_sales_2022	sum_sales_2023	perc_growth
1	Supplies	16140.70	28917.40	79.158276