product_sales_analysis

November 26, 2025

1 ETL Pipeline: Sales Data Analysis

Project Goal: Transform raw product sales data into a clean, structured format ready for SQL analysis and Power BI visualization. **Steps:** Data Cleaning -> Feature Engineering -> Data Validation -> SQL Loading.

Importing necessary libraries and loading a raw dataset.

[19]:
```
import pandas as pd
import numpy as np

df = pd.read_csv("product_sales_dataset_final.csv")
df.head()
```

[19]:

	Order_ID	Order_Date	Customer_Name	City	State	Region	Country	Category	Sub_Category	Product_Name	Quantity	Unit_Price	Revenue	Profit
0	1	08-23-23	Bianca Brown	Jackson	Mississippi	South	United States	Accessories	Small Electronics	Phone Case	3	201.01	603.03	221.49
1	2	12-20-24	Jared Edwards	Grand Rapids	Michigan	Centre	United States	Accessories	Small Electronics	Charging Cable	4	74.30	297.20	97.09
2	3	01-29-24	Susan Valdez	Minneapolis	Minnesota	Centre	United States	Clothing & Apparel	Sportswear	Nike Air Force 1	1	68.19	68.19	25.47
3	4	11-29-24	Tina Williams	Tallahassee	Florida	South	United States	Clothing & Apparel	Sportswear	Adidas Tracksuit	3	209.64	628.92	231.38
4	5	09-21-23	Catherine Gordon	Baltimore	Maryland	East	United States	Accessories	Bags	Backpack	1	216.63	216.63	42.46

Initial inspection of data.

[20]:
```
df.describe()
```

[20]:
```
            Order_ID       Quantity    Unit_Price        Revenue   \
count  200000.000000  200000.000000  200000.000000  200000.000000
mean   100000.500000       1.854000     382.855615     712.038725
std     57735.171256       1.100536     276.870235     742.471556
min         1.000000       1.000000      17.030000      17.030000
25%     50000.750000       1.000000     162.760000     229.187500
50%    100000.500000       1.000000     303.545000     464.880000
75%    150000.250000       2.000000     562.252500     881.302500
max    200000.000000      11.000000    1432.000000    9014.250000

             Profit
count  200000.000000
mean      157.743041
std       155.689581
min         3.920000
25%        59.210000
50%       109.530000
75%       199.402500
max      2763.720000
```

[21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 14 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Order_ID       200000 non-null  int64
 1   Order_Date     200000 non-null  object
 2   Customer_Name  200000 non-null  object
 3   City           200000 non-null  object
 4   State          200000 non-null  object
 5   Region         200000 non-null  object
 6   Country        200000 non-null  object
 7   Category       200000 non-null  object
 8   Sub_Category   200000 non-null  object
 9   Product_Name   200000 non-null  object
 10  Quantity       200000 non-null  int64
 11   Unit_Price    200000 non-null  float64
 12   Revenue       200000 non-null  float64
 13   Profit        200000 non-null  float64
dtypes: float64(3), int64(2), object(9)
memory usage: 21.4+ MB
```

Checking for any null values.

[22]:
```
df.isna().sum()
```

[22]:
```
Order_ID         0
Order_Date       0
Customer_Name    0
City             0
State            0
Region           0
Country          0
Category         0
Sub_Category     0
Product_Name     0
Quantity         0
 Unit_Price      0
 Revenue         0
 Profit          0
dtype: int64
```

[23]:
```
df.shape
```

[23]:
```
(200000, 14)
```

Converting "Order_Date" from object to datetime format.

[24]:
```
df["Order_Date"] = pd.to_datetime(df["Order_Date"], format="%m-%d-%y")
df["Order_Date"]
```

[24]:
```
0        2023-08-23
1        2024-12-20
2        2024-01-29
3        2024-11-29
4        2023-09-21
             ...
199995   2023-08-15
199996   2023-10-17
199997   2023-12-03
199998   2023-12-08
199999   2024-12-13
Name: Order_Date, Length: 200000, dtype: datetime64[ns]
```

Creating additional columns from "Order_Date".

[25]:
```
df["Year"] = df["Order_Date"].dt.year
df["Month"] = df["Order_Date"].dt.month
df["Month_Name"] = df["Order_Date"].dt.month_name()
df["Day"] = df["Order_Date"].dt.day

df[["Order_Date", "Year", "Month", "Month_Name", "Day"]].head(1)
```

[25]:

	Order_Date	Year	Month	Month_Name	Day
0	2023-08-23	2023	8	August	23

Striping whitespace to prevent errors during downstream analysis.

[26]:
```
df.columns = df.columns.str.strip()
```

Standarizing categorical data to Title Case and removing extra spaces.

[27]:
```
text_columns = ["City", "State", "Region", "Category", "Sub_Category",␣
 ↪"Product_Name"]
for column in text_columns:
    df[column] = df[column].str.strip().str.title()
df.columns
```

[27]:
```
Index(['Order_ID', 'Order_Date', 'Customer_Name', 'City', 'State', 'Region',
       'Country', 'Category', 'Sub_Category', 'Product_Name', 'Quantity',
       'Unit_Price', 'Revenue', 'Profit', 'Year', 'Month', 'Month_Name',
       'Day'],
      dtype='object')
```

Checking for data integrity, recalculating "Revenue"

[28]:
```
expected_revenue = df["Quantity"] * df["Unit_Price"]
diff = (expected_revenue.round(2) - df["Revenue"].round(2)).abs()
error_count = (diff > 0).sum()
print(error_count)
```

```
0
```

Checking for any negative values.

[29]:
```
negative_qty = df[df["Quantity"] < 0]
print(f"The number of negative quantity transactions: {len(negative_qty)}")
negative_price = df[df["Unit_Price"] < 0]
print(f"The number of negative price transactions: {len(negative_price)}")
loss_making = df[df["Profit"] < 0]
print(f"The number of loss making transactions: {len(loss_making)}")
```

```
The number of negative quantity transactions: 0
The number of negative price transactions: 0
The number of loss making transactions: 0
```

Creating "Margin" metric to calculate relative profitability.

[30]:
```
df["Margin"] = (df["Profit"] / df["Revenue"]) * 100
df["Margin"] = df["Margin"].round(2)
df["Margin"]
```

[30]:
```
0         36.73
1         32.67
2         37.35
3         36.79
4         19.60
```

```
          …
199995    32.89
199996    19.63
199997    48.01
199998    40.90
199999    27.28
Name: Margin, Length: 200000, dtype: float64
```

Checking for data integrity of "Region".

[31]:
```
df["Region"].unique().size
```

[31]:
```
4
```

Dropping unneccessary column "Country".

[32]:
```
df.drop(columns=["Country"])
df.head()
```

[32]:

	Order_ID	Order_Date	Customer_Name	City	State	Region	Country	Category	Sub_Category	Product_Name	Quantity	Unit_Price	Revenue	Profit	Year	Month	Month_Name	Day	Margin
0	1	2023-08-23	Bianca Brown	Jackson	Mississippi	South	United States	Accessories	Small Electronics	Phone Case	3	201.01	603.03	221.49	2023	8	August	23	36.73
1	2	2024-12-20	Jared Edwards	Grand Rapids	Michigan	Centre	United States	Accessories	Small Electronics	Charging Cable	4	74.30	297.20	97.09	2024	12	December	20	32.67
2	3	2024-01-29	Susan Valdez	Minneapolis	Minnesota	Centre	United States	Clothing & Apparel	Sportswear	Nike Air Force 1	1	68.19	68.19	25.47	2024	1	January	29	37.35
3	4	2024-11-29	Tina Williams	Tallahassee	Florida	South	United States	Clothing & Apparel	Sportswear	Adidas Tracksuit	3	209.64	628.92	231.38	2024	11	November	29	36.79
4	5	2023-09-21	Catherine Gordon	Baltimore	Maryland	East	United States	Accessories	Bags	Backpack	1	216.63	216.63	42.46	2023	9	September	21	19.60

Defining connection details for the local PostgreSQL instance. Creating Database Engine using SQLAlchemy to establish the connection string.

[34]:
```
from sqlalchemy import create_engine

username = "postgres"
password = "GXaled71ma76"
host = "localhost"
port = "5432"
```

```
database = "product_sales"

engine = create_engine(f"postgresql+psycopg2://{username}:{password}@{host}:
  ↪{port}/{database}")

table_name = "sales"
df.to_sql(table_name, engine, if_exists="replace", index=False)
print(f"Data successfully loaded into table {table_name} in database␣
  ↪{database}")
```

```
Data successfully loaded into table sales in database product_sales
```