TUNIS BUSINESS SCHOOL
UNIVERSITY OF TUNIS

Report

IT 300

Business Intelligence and Database

Management Systems

Business Intelligence Mini-Project
Adidas US Sales

Authors:

Eslem Sebri

Oumayma Abayed

Submitted to:

Professor Amani Azzouz

Contents

1. Introduction

In today's competitive retail environment, leveraging data to make informed decisions is crucial for success.
This project aims to analyze Adidas sales data from 2020 to 2021 to provide actionable insights that support strategic decision-making.
By implementing a Business Intelligence (BI) dashboard, the project facilitates the evaluation of key metrics such as regional sales performance, product profitability, and customer preferences.
The results will help guide strategic actions such as opening or closing stores, optimizing marketing strategies, expanding the product catalog, and planning territorially.
The ultimate objective is to equip stakeholders with a unified and accessible platform for comprehensive data analysis.

2. Implementation

2.1. Data Gathering

2.2. First Dataset:

Type: Excel
Source: Kaggle
https://www.kaggle.com/datasets/heemalichaudhari/adidas-sales-dataset

2.3. Second Dataset:

Type: CSV
Source: Kaggle
https://www.kaggle.com/datasets/ahmedabbas757/dataset?select=data_sales+%281%29.csv

2.4. Data Preparation

Using pandas library from python

Loaded Data:

- Excel files loaded using the Python's pandas library.
- CSV files processed with Python's pandas library.

Unified Column Structures:

- Removed the "Operating Margin" column from the Excel data.
- Ensured identical columns in both datasets.

Data Cleaning:

- Unified date formats to MM-DD-YYYY.
- Standardized price-per-unit formats.
- Converted "Units Sold" column to integer type.
- Recalculated "Total Sales" for consistency.
- Cleaned the "Operating Profit" column.
- Removed duplicate entries and null values.
- Removed the retailer id column since they are not unique so they cannot be used as Primary Keys .

Merged Datasets:

- Combined the datasets into a single, unified dataset. (CSV)

Sorted Data:

- Ordered rows chronologically.

2.5. Data Storage

2.5.1. Storage

- **Database Tool:** MySQL, interfaced with Python using SQLAlchemy.

- **Setup Commands:**
  pip install sqlalchemy
  pip install mysql-connector-python
- **MySQL Code:**

```
USE Adidas_Sales;
-- 1. Create Retailer Dimension Table
CREATE TABLE retailer_dim (
    retailer_id INT PRIMARY KEY,
    retailer_name VARCHAR(255)
);

-- 2. Create Date Dimension Table
CREATE TABLE date_dim (
    date_id INT PRIMARY KEY AUTO_INCREMENT,
    invoice_date DATE,
    year INT,
    month INT,
    day INT
);

-- 3. Create Region Dimension Table
CREATE TABLE region_dim (
    region_id INT PRIMARY KEY AUTO_INCREMENT,
    region_name VARCHAR(255),
    state VARCHAR(255),
    city VARCHAR(255)
```

```
);

-- 4. Create Product Dimension Table
CREATE TABLE product_dim (
   product_id INT PRIMARY KEY AUTO_INCREMENT,
   product_name VARCHAR(255)
);

-- 5. Create Sales Method Dimension Table
CREATE TABLE sales_method_dim (
   sales_method_id INT PRIMARY KEY AUTO_INCREMENT,
   sales_method_name VARCHAR(255)
);

-- 6. Create Sales Fact Table
CREATE TABLE sales_fact (
   sales_id INT PRIMARY KEY AUTO_INCREMENT,
   retailer_id INT,
   date_id INT,
         region_id INT,
   product_id INT,
   sales_method_id INT,
   price_per_unit FLOAT,
   units_sold INT,
   total_sales FLOAT,
   operating_profit FLOAT,
   FOREIGN KEY (retailer_id) REFERENCES retailer_dim(retailer_id),
   FOREIGN KEY (date_id) REFERENCES date_dim(date_id),
   FOREIGN KEY (region_id) REFERENCES region_dim(region_id),
   FOREIGN KEY (product_id) REFERENCES product_dim(product_id),
   FOREIGN KEY (sales_method_id) REFERENCES
sales_method_dim(sales_method_id)
);
```

2.5.2. Fact: Sales_Fact

Sales_id: PK
retailer_id: FK
Date_id: FK
Region_id: FK
Product_id: FK
Sales_method_id: FK
price_per_unit
units_sold
total_sales
operating_profit

2.5.3. Dimensions

2.5.4. **Retailer Dimension:** `retailer_dim:` Retailer details.

retailer_id: PK
retailer_name

2.5.5. **Date Dimension:** `date_dim:` Temporal data.

Date_id: PK
Invoice_Date
Year
Month
Day

2.5.6. **Region Dimension:** `region_dim:` Geographical locations.

Region_id: PK
Region_name
State
City

2.5.7. **Product Dimension:** `product_dim:` Product information.

Product_id: PK
product_name

2.5.8. **Sales Method Dimension:** `sales_method_dim:` Types of sales methods.

Sales_method_id: PK
Sales_method_name

2.6. Data Mapping:

Initially, we attempted to load all the data into our data warehouse at once, but the **challenge of limited available memory** forced us to adopt a **chunking approach** instead:

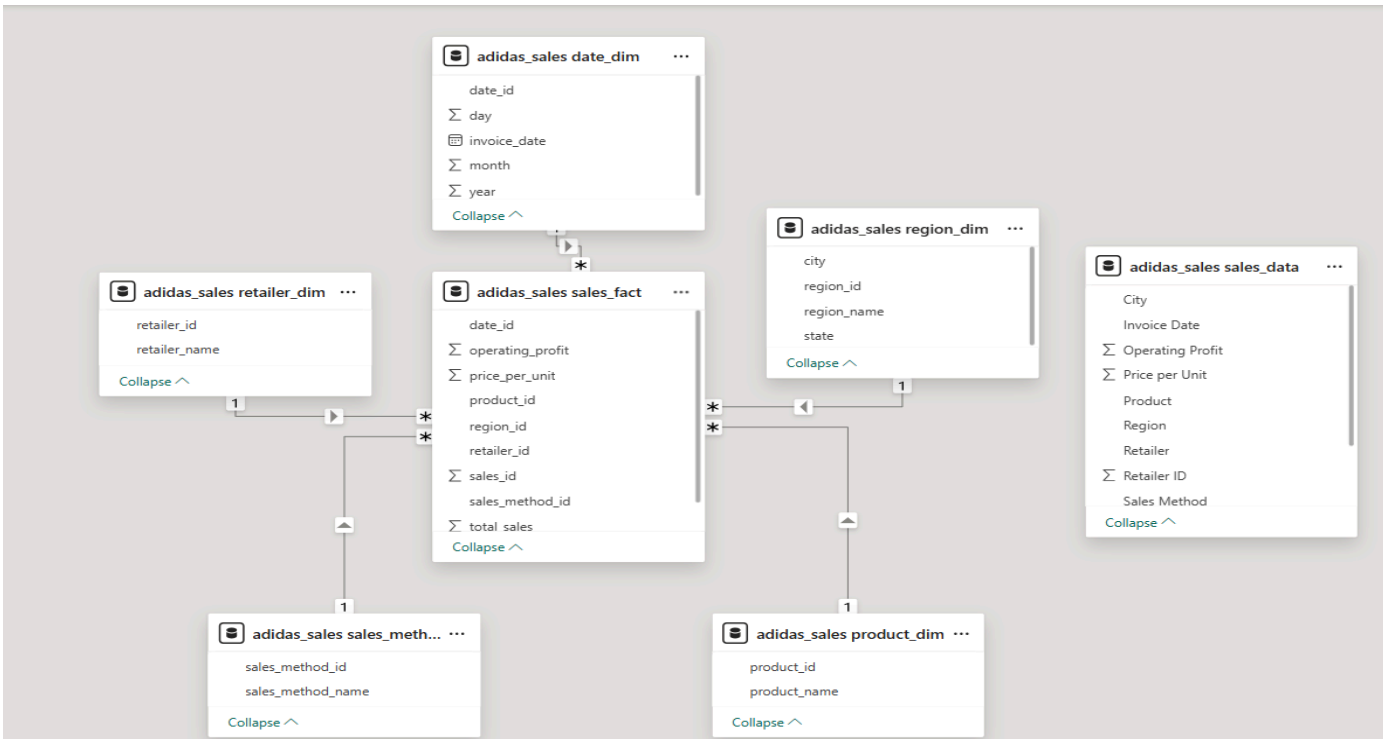
First attempt:

```
# Map foreign key IDs for each dimension
fact_data = sales_data.copy()

# 1. Map retailer_id
retailer_dim = pd.read_sql('SELECT * FROM retailer_dim', con=engine)
fact_data = fact_data.merge(retailer_dim, how='left', left_on='Retailer',
right_on='retailer_name')

# 2. Map date_id
date_dim = pd.read_sql('SELECT * FROM date_dim', con=engine)
fact_data = fact_data.merge(date_dim, how='left', left_on='Invoice Date',
right_on='invoice_date')
```

```
# 3. Map region_id
region_dim = pd.read_sql('SELECT * FROM region_dim', con=engine)
fact_data = fact_data.merge(region_dim, how='left', left_on=['Region', 'State',
'City'],
                            right_on=['region_name', 'state', 'city'])

# 4. Map product_id
product_dim = pd.read_sql('SELECT * FROM product_dim', con=engine)
fact_data = fact_data.merge(product_dim, how='left', left_on='Product',
right_on='product_name')

# 5. Map sales_method_id
sales_method_dim = pd.read_sql('SELECT * FROM sales_method_dim', con=engine)
fact_data = fact_data.merge(sales_method_dim, how='left', left_on='Sales Method',
right_on='sales_method_name')

# Select and rename the required columns for sales_fact
fact_table = fact_data[['retailer_id', 'date_id', 'region_id', 'product_id',
'sales_method_id',
                        'Price per Unit', 'Units Sold', 'Total Sales', 'Operating
Profit']]
fact_table.columns = ['retailer_id', 'date_id', 'region_id', 'product_id',
'sales_method_id',
                      'price_per_unit', 'units_sold', 'total_sales',
'operating_profit']

# Insert data into the fact table
fact_table.to_sql('sales_fact', con=engine, if_exists='append', index=False)
print("sales_fact table populated!")
```

Second attempt code (*the chunking method*) can be found in the Star_Schema.py file

2.7. Data Visualization



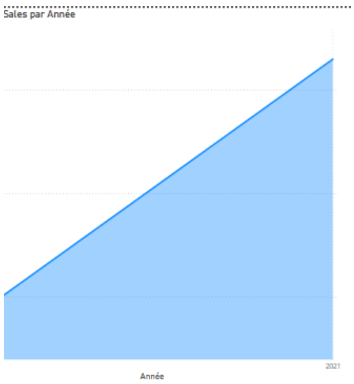
We chose the star schema for our data warehouse design because it offers simplicity, efficiency, and ease of use, particularly for analytical and reporting purposes. The star schema organizes data into a central fact table (sales_fact) surrounded by dimension tables, making it highly intuitive and user-friendly. This design allows for faster query performance due to its denormalized structure, reducing the need for complex joins. It is also well-suited for business intelligence applications as it simplifies data aggregation, making it easier to slice and dice data across various dimensions, such as time, product, region, retailer, and sales method.

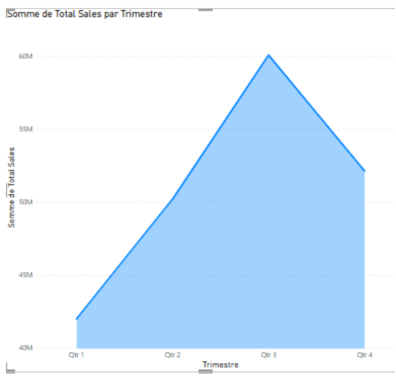
Additionally, the star schema aligns with the requirements of our project by ensuring a clear separation between transactional data in the fact table and descriptive attributes in the dimension tables, which enhances data clarity and scalability for future growth.

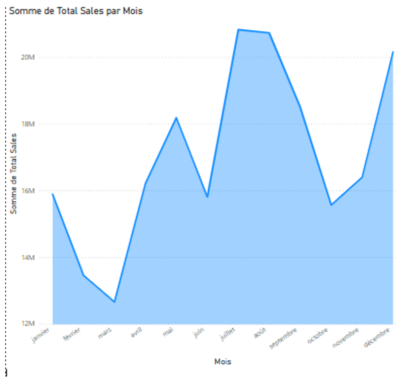
3. Conclusion

3.1. Sales Analysis

- What is the total sales generated over time?

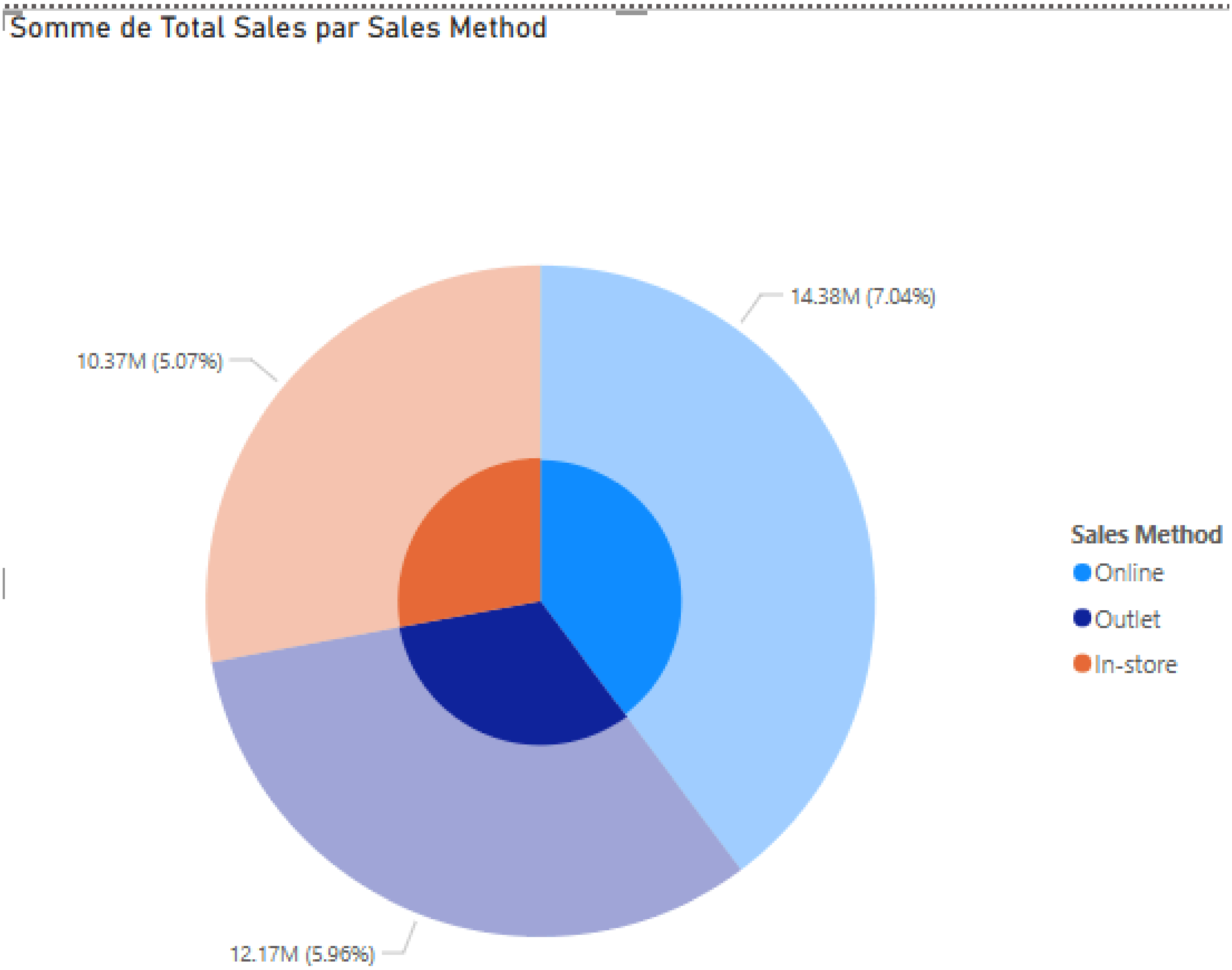




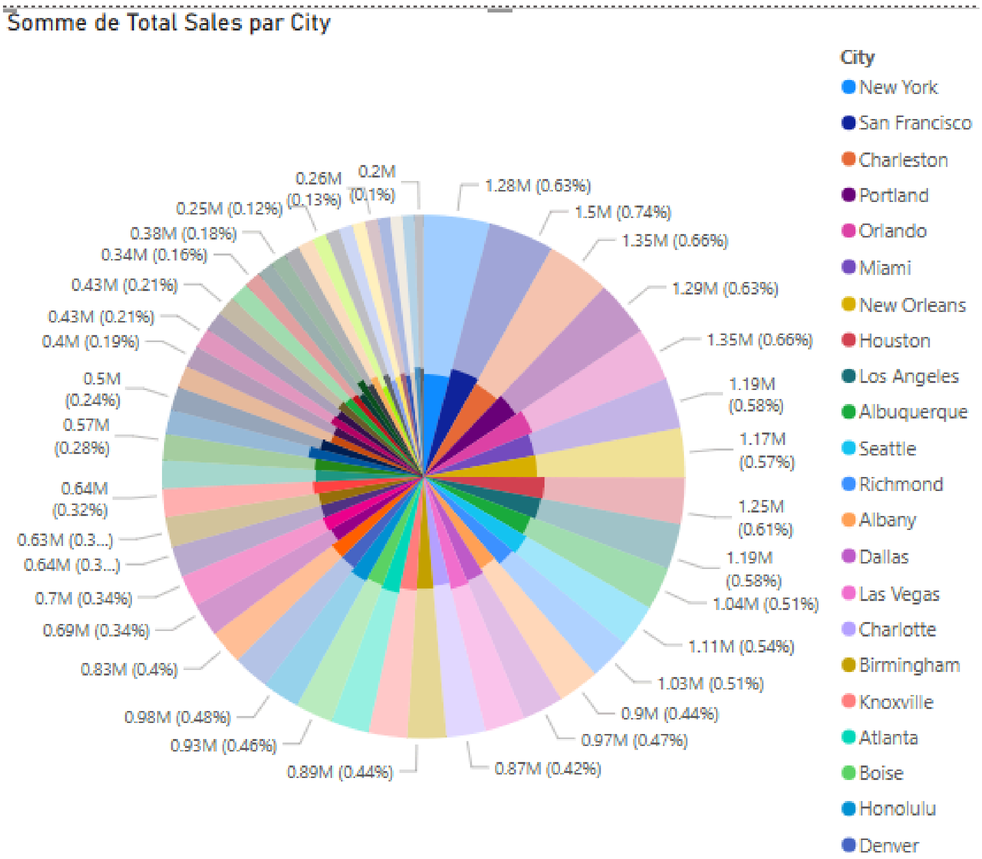


3.2. Customer Analysis*

- Which sales method generates the highest sales?



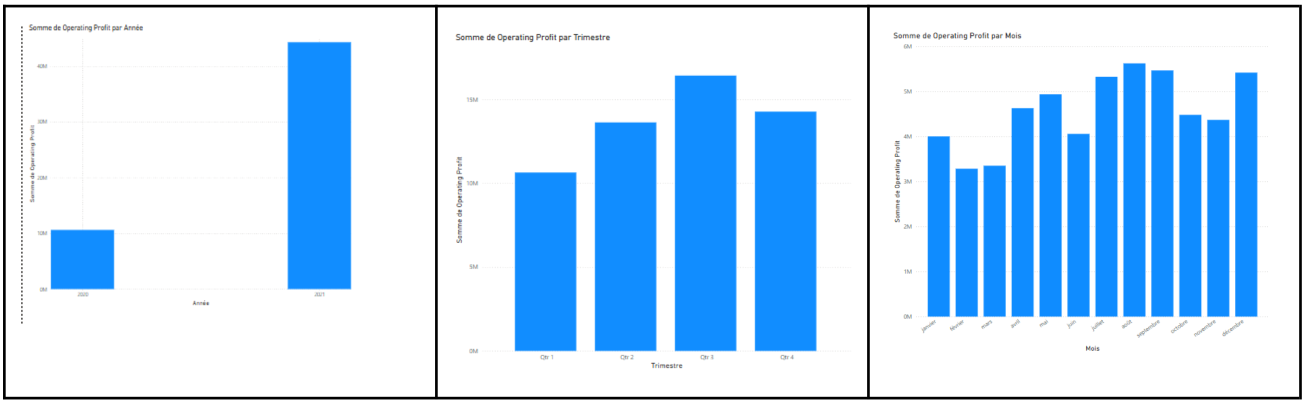
3.3. Geographical Analysis



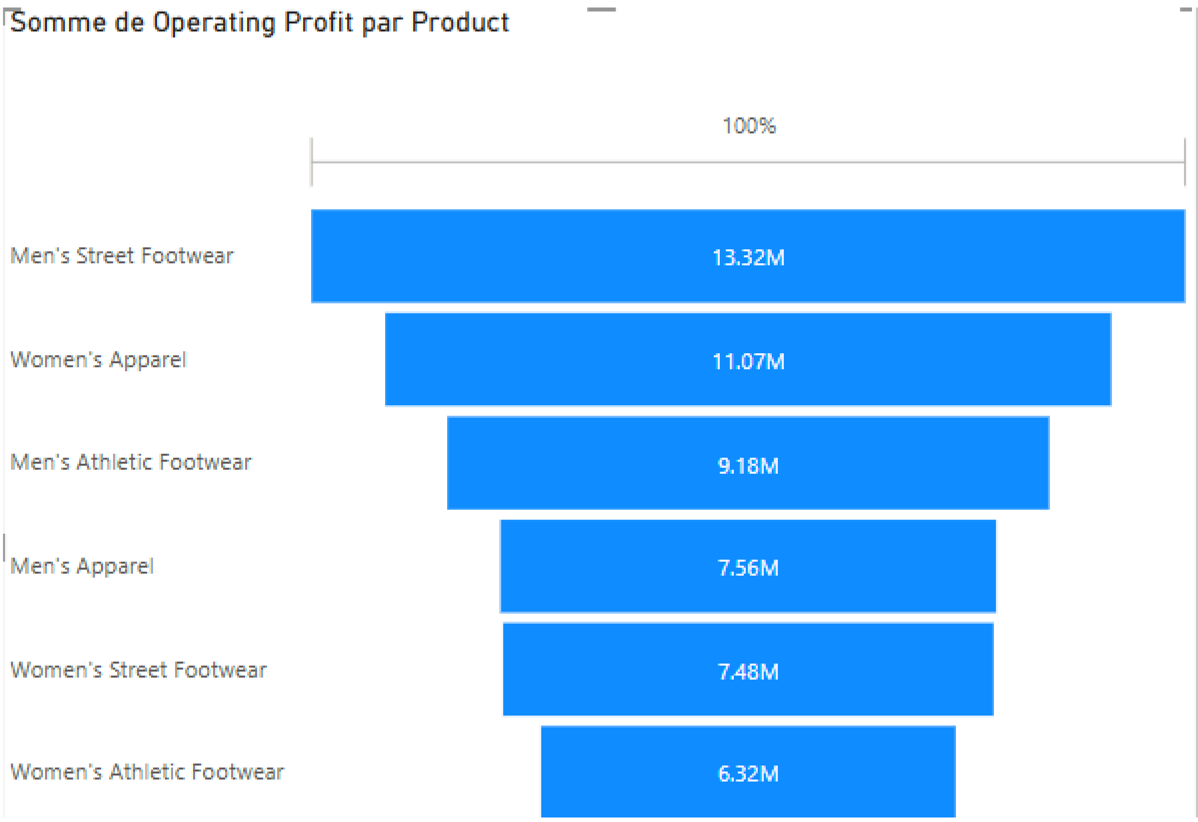
- What are the key markets in terms of region, state, and city?

3.4. Profitability Analysis

- What is the total sales generated over time?



- What is the profit margin for different products?



- How does profitability vary across different sales methods?