

Report

IT-300

### Business Intelligence and Database Management Systems



Business Intelligence Research



Authors:

Jihene Gazzeh

Farah Jaouadi

Nour Elloumi

Jawher Manai

# Contents

1. [**Introduction**](#_heading=h.30j0zll) **1**
2. [**Implementation**](#_heading=h.3znysh7) **1**
   1. [Data Gathering](#_heading=h.tyjcwt) 1
   2. [Data Preparation](#_heading=h.1t3h5sf) 1
   3. [Data Storage](#_heading=h.2s8eyo1) and Modeling 3
      1. Modeling 3
      2. [Fact](#_heading=h.17dp8vu) 5
      3. [Dimensions](#_heading=h.3rdcrjn) 5
   4. [Data Visualization](#_heading=h.26in1rg) 5
3. [**Conclusion**](#_heading=h.lnxbz9) **5**

**List of Figures**

[1 Data Warehouse Schema](#_heading=h.35nkun2) 4

# Introduction

This business intelligence project is dedicated to analyzing the sales performance of an e-commerce platform. The core focus lies in understanding the dynamics between a diverse range of products. We aim to gain deeper insights into the performance of this online shop by focusing on its top-selling products, monitoring stock levels, and evaluating overall revenue. These insights will significantly contribute to our understanding of the business, empowering us to make effective decisions that will support the shop's growth and expansion.

# Implementation

## Data Gathering

## For the data gathering phase we conducted thorough research and identified a suitable dataset available on GitHub that matches the project requirements. [Link](https://github.com/JannikArndt/PostgreSQLSampleDatabase/tree/master/data)

## Data Preparation

The data preparation began with the extraction of information from various Excel files, such as 'Product.xlsx and 'articles.xlsx.' Each extraction is facilitated by the 'extract\_data' function using Python(PyCharm), which constructs local file paths and reads data into Pandas DataFrames. The 'convert\_to\_numeric,' 'convert\_to\_datetime,' and 'convert\_to\_string' functions ensure consistent data types across columns.

In the context of product-related data (Product.xlsx), the script handles null values, formats date and time, removes unnecessary columns, and cleans zip codes. Additionally, eliminates duplicate entries. Special attention is given to resolving issues with datetime formatting and replacing '\\N' values.

The analysis of stock-related data (stock.xlsx) follows a similar structure. It includes cleaning steps, handling missing values, converting columns to datetime format, and performing outlier detection using the Z-score test. Detected outliers are processed using the 'process\_outlier' function.

Each script concludes by saving the cleaned and processed data to new Excel files. The provided Python scripts offer a comprehensive approach to handling data quality and integrity across all the datasets.

Below are some examples showcasing the manipulation performed:

The extract\_data function is designed to read an Excel file and return data as a Pandas DataFrame:

def extract\_data(directory\_path, file\_name):  
 local\_file\_path = os.path.join(directory\_path, file\_name)  
 data = pd.read\_excel(local\_file\_path, engine='openpyxl')  
 return data



The convert\_to\_datetime function attempts to convert a column in a DataFrame to datetime:

def convert\_to\_datetime(dataframe, column\_name):  
 try:  
 dataframe[column\_name] = pd.to\_datetime(dataframe[column\_name], errors='coerce')  
 except (ValueError, TypeError):  
 print(f"Conversion to datetime for column '{column\_name}' failed.")



The process\_outlier function identifies outliers in a specified column based on Z-scores. It calculates the Z-score for the specified column and identifies values that exceed the defined threshold. If outliers are found, the function removes the corresponding rows from the DataFrame and returns the data without outliers:

def process\_outlier(tmp, column\_name, threshold=1.96):  
 tmp\_col = tmp[column\_name]  
 z = np.abs(stats.zscore(tmp\_col))  
 sub = z >= threshold  
 min\_val = tmp\_col.min()  
 max\_val = tmp\_col.max()  
 if sub.sum() == 0:  
 print(f'No instances found with z-score above the threshold ({threshold}).')  
 else:  
 for i in tmp\_col[sub].index:  
 print(  
 f'The {Fore.LIGHTMAGENTA\_EX}{column\_name}{Style.RESET\_ALL} column ranges from {min\_val} to {max\_val}\n\nUnique values from the {sub.sum()} {Fore.LIGHTMAGENTA\_EX}{column\_name}{Style.RESET\_ALL} instances: {np.unique(tmp\_col[i]).round(2)}{Style.RESET\_ALL}')  
 tmp = tmp[~sub]  
 return tmp

  
The load\_data\_to\_postgre loads the processed data into Postgre:

def load\_data\_to\_postgres(filtered\_data, table\_name, database\_url):

try:

engine = create\_engine(database\_url, poolclass=QueuePool)

filtered\_data.to\_sql(table\_name, engine, if\_exists='replace', index=False)

print (f'Data has been loaded into the table {table\_name} in PostgreSQL.')

except SQLAlchemyError as e:

print (f'Error: {e}')

## Data Storage and Modeling

#### Modeling

**Facts:**

* **Profit**
* **Quantity sold**
* **Revenue**

**Dimensions:**

We identified eight dimensions for this data:

* **Gender** (Products Table)
* **Category** (Products Table)
* **Time** (Products Table)
* **Color** (Articles Table)
* **Price** (Articles Table)
* **Product** (Products Table)
* **Stock** (Stock Table)
* **Size** (Articles Table)

In our process for managing the Fact and Dimension tables, we integrated two tools: Microsoft SQL Server Management Studio (SSMS) and Alteryx. Initially, we used SSMS to create and design our Fact and Dimension tables, establishing the star schema of our data structure (find schema below).

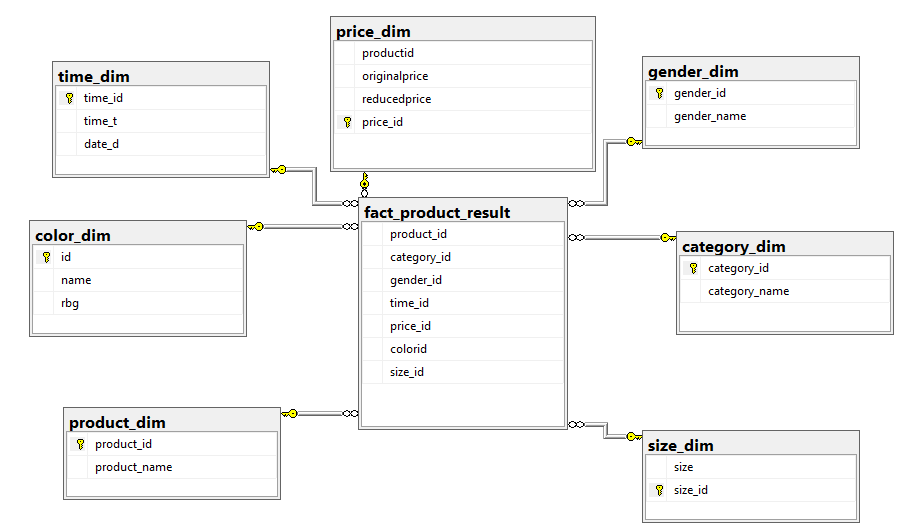


Figure 1: Data Warehouse Schema

Once we defined our schema, we employed Alteryx to populate our tables. In this process, we linked our primary Excel file (Products.xlsx) with its corresponding dimensions. Using **Alteryx**'s tools, we utilized Alteryx components to organize and manipulate the data according to our requirements.

The key tools used:

 **Select:** Enabled us to choose and extract specific columns for each dimension.

 **Summarize:** Allowed grouping of data.

 **Multi-Row Formula:** Allowed automatic incrementation of values (dimension ids).

if [Row-1:reducedprice]!=[reducedprice]

then [Row-1:price\_id]+1

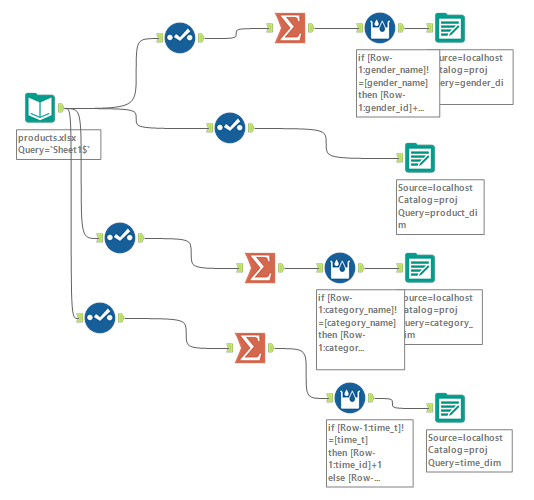
else [Row-1:price\_id]

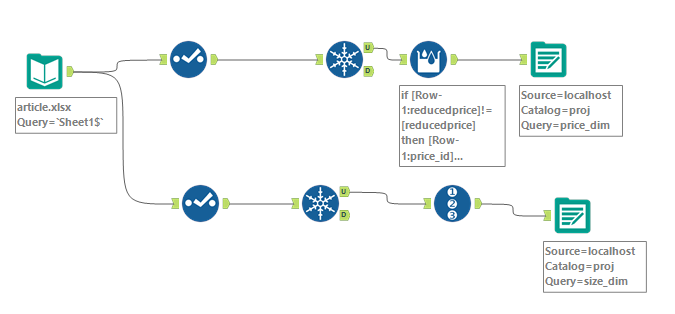
endif

 **Record ID**: Same function as Multi-Row Formula (increment)

  **Unique:** Separates data into two streams, duplicate and unique rows, based on the columns you choose.

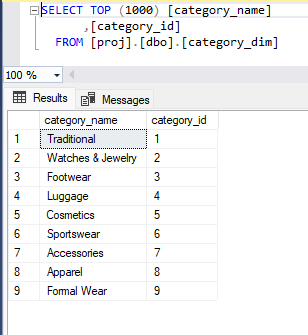
Here's an illustrative example of the process. The accompanying pictures provide a visual walkthrough of the steps that led to the population of the dimension tables.





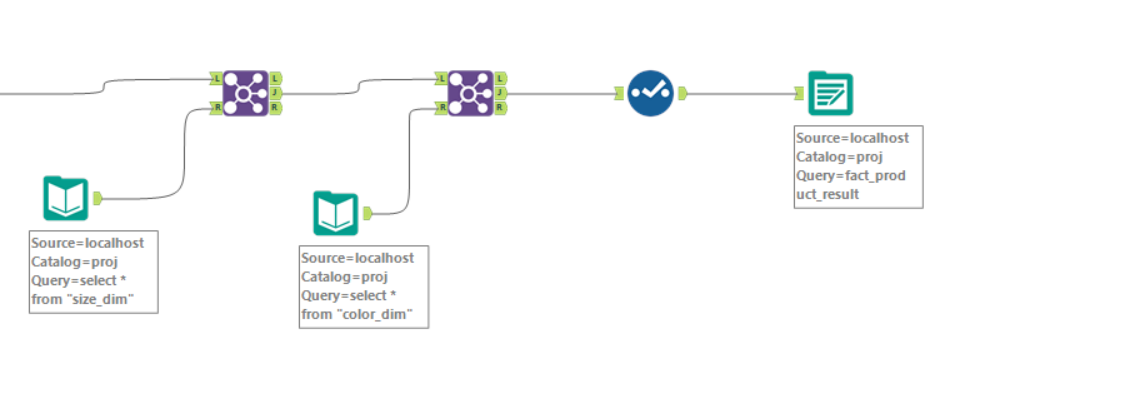
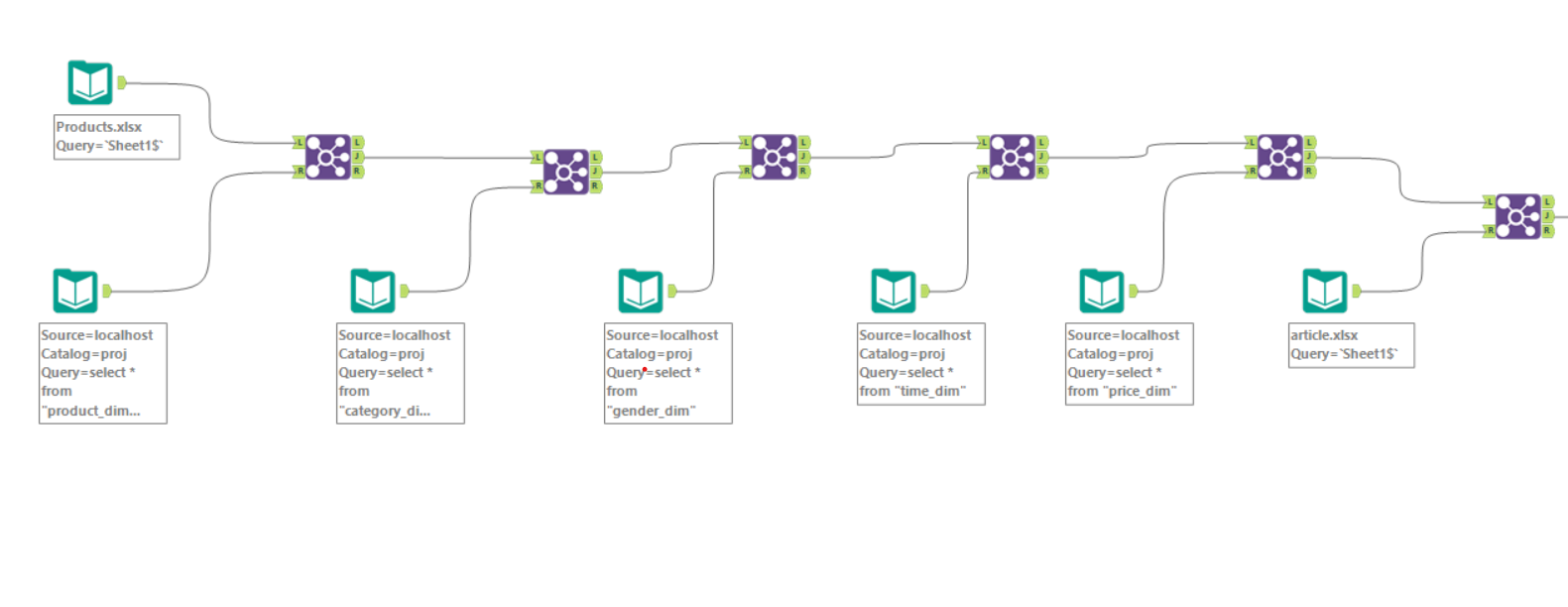


Illustrative example showcasing the structure of one of the dimension tables:



After successfully populating the dimension tables, we employed Alteryx once again to fill the Fact table using the same process.

**Join:** The Join tool combines two data streams based on common fields or record position.



After populating our fact table using Alteryx, we utilized an MDX query in Excel to generate a cube. The MDX code incorporates two selected measures: 'Sum of reducedprice' and 'Sum of count.' The dimensions employed in the code include time, size, gender, color name, category, and name. Additionally, the Hierarchize function is utilized to organize the specified set of members into a unified hierarchy, while the DrilldownLevel function facilitates the drill-down into the specified levels of the hierarchies, allowing for a more granular analysis within the cube.

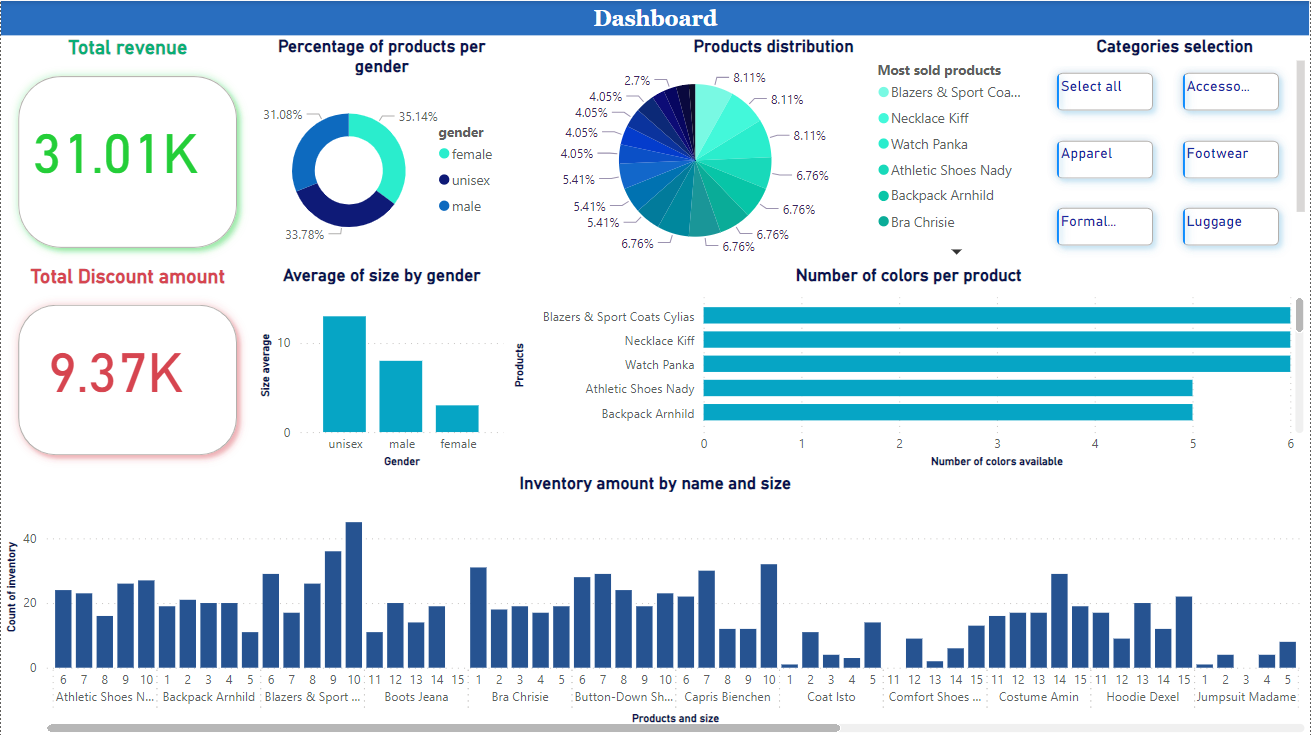
The code:

SELECT {[Measures].[Sum of reducedprice],[Measures].[Sum of count]} DIMENSION PROPERTIES PARENT\_UNIQUE\_NAME,MEMBER\_VALUE,HIERARCHY\_UNIQUE\_NAME ON COLUMNS , NON EMPTY Hierarchize({DrilldownLevel({[Range].[name].[All]},,,INCLUDE\_CALC\_MEMBERS)}) DIMENSION PROPERTIES PARENT\_UNIQUE\_NAME,MEMBER\_VALUE,HIERARCHY\_UNIQUE\_NAME ON ROWS FROM [Model] WHERE ([Range].[time].[All],[Range].[size].[All],[Range].[gender].[All],[Range].[color\_name].[All],[Range].[category].[All]) CELL PROPERTIES VALUE, FORMAT\_STRING, LANGUAGE, BACK\_COLOR, FORE\_COLOR, FONT\_FLAGS

## Data Visualization:

For the data visualization we used Power Bi and extracted the following insights:

* The total revenue from sales is $31,010, including a detailed breakdown by product, gender, and category.
* Blazers and Sports Coats contribute a total revenue of $1,590.
* The total discount amount is $9,370, plus identification of specific discounts applied per product and gender.
* The discount amount for Blazers and Sports Coats is $896.40.
* The most sold product: Blazers & Sport Coats, Necklace Kiff, and Watch Panka 8.11%
* The number of colors per product. Blazers & Sport Coats have the highest color variation with 6 different colors.
* The percentage of products per gender: 35.14% female, 33.78% unisex, 31.08% male.
* The average size distribution across genders: with an average of 13 unisex sizes, 8 for males, and 3 for females
* Stock levels by product name.



# Conclusion

In conclusion, the detailed sales analysis reveals a total revenue of $31,010, with standout contributions such as Blazers and Sports Coats generating $1,590. The comprehensive examination of discounts, color variations, product distribution by gender, and average size distribution offers valuable insights into our e-commerce.

**Recommendations for Improvement:**

1. The current discount rate, accounting for 30.216% of total revenue, is very high. Therefore, the company should consider reducing the discount rate to increase profits.
2. Enhance product appeal by introducing varied color options to products with a limited color range.
3. Consider eliminating the luggage category as it has a limited contribution to overall profit