2024/2025

TUNIS BUSINESS SCHOOL
UNIVERSITY OF TUNIS

BUSINESS INTELLIGENCE AND DATABASE MANAGEMENT SYSTEMS

PERFORMANCE ANALYSIS PROJECT REPORT

PROFESSOR :
PROF. AMENI AZOUZ

AUTHORS :

MERIEM HOUISSA SARRA AKROUTI NOUR CHABBOUH

TABLE OF CONTENT

IKEA PERFORMANCE DATA ANALYSIS

TUNIS BUSINESS SCHOOL
UNIVERSITY OF TUNIS

INTRODUCTION

COMPANY INTRODUCTION

IKEA is a Swedish home furnishings brand founded in 1943, known for affordable, stylish, and functional products. With a presence in over 50 countries, IKEA is synonymous with flat-pack furniture and sustainable living solutions.

PROJECT MISSION

At **IKEA**, the mission to inspire people to live a better everyday life is at the heart of everything. Through this Business Intelligence project, we share that same vision—aspiring to empower IKEA with deeper insights and better analysis to enhance its performance and customer impact.
By analyzing patterns of **sales**, **inventory**, **products**, and **stores**, we aim to uncover trends and correlations that impact **operational performance** and **customer preferences**. Through relevant data modeling and visualization, this report highlights key insights, uncovering **opportunities for growth**, enhancing **strategic decision-making**, and supporting IKEA's mission of delivering value while adapting to a dynamic market.

BI ANALYSTS

MERIEM HOUISSA

SARRA AKROUTI

NOUR CHABBOUH

TOOLS & SOFTWARES USED

EXECUTION

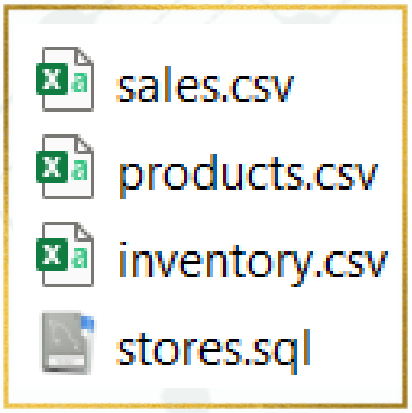
Phase1 : Data Gathering

Step1: Data Collection

- We started by collecting data from **different sources.**
- We found unsorted data of Ikea from 2019 to 2023 through different regions.

Step2: Initial Data Review

- The collected datasets are as follow



- Some Data observations :

```
CREATE TABLE IKEAstores (
    store_id VARCHAR(10) PRIMARY KEY,
    store_name VARCHAR(255),
    city VARCHAR(100),
    country VARCHAR(100)
);

-- Insert the data
INSERT INTO IKEAstores (store_id, store_name, city,
('IK-1', 'IKEA Hanoi', 'Hanoi', 'Vietnam'),
('IK-2', 'IKEA Anaheim', 'Anaheim', 'United States')
('IK-3', 'IKEA Ashburn', 'Ashburn', 'United States')
('IK-4', 'IKEA Atlanta', 'Atlanta', 'United States')
```

from stores.sql

product_id	product_name	category	subcategory	unit_pice
P-1	BILLY Bookcase	Furniture	Storage	59.9
P-2	LACK Coffee Table	Furniture	Surface	29.9
P-3	MALM Bed Frame	Bedroom	Bedding	19
P-4	KALLAX Shelf Unit	Furniture	Storage	79.9
P-5	POÄNG Armchair	Living Room	Seating	89.9
P-6	FÄRGRIK Dinnerwar	Kitchen	Dining	24.9
P-7	MICKE Desk	Furniture	Surface	59.9

from products.csv

iventory_id	product_id	current_stock	reorder_level
1	P-46	5	15
2	P-53	5	15
3	P-45	6	15
4	P-57	6	15
5	P-62	6	15

from inventory.csv

Phase2: DATA PREPARATION

Step1: Data Cleaning

Once the data was extracted, the following step involved transforming it into a **usable format** for the Data Warehouse. This process required unifying the data by converting it from multiple formats (csv and Sql) into a consistent format (JSON).

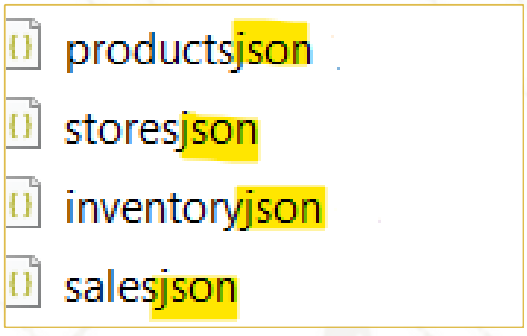
We used Pandas Library for the files transformations :

```
from google.colab import files
import pandas as pd
from google.colab import drive
drive.mount('/content/drive')
```

As the conversion of products.csv to json

```
import pandas as pd
file_path = '/content/drive/My Drive/ikea/ikea_products.csv'
products = pd.read_csv(file_path)
products.to_json('productsjson.json', orient='records', date_format='iso')
files.download('productsjson.json')
```

We did the same thing for all other files to finally get this output :



While cleaning the data , we **standardized data format** into YYYY-MM-DD as follow :

```
import pandas as pd
# load JSON files into pandas Dataframes
sales_df = pd.read_json('salesjson.json')
inventory_df = pd.read_json('inventoryjson.json')
products_df = pd.read_json('productsjson.json')
stores_df = pd.read_json('storesjson.json')
```

```
# Standardize date format to YYYY-MM-DD in sales
def date_format(date_series):
 return pd.to_datetime(date_series).dt.strftime('%y-%m-%d')
sales_df['order_date'] = date_format(sales_df['order_date'])
```

Then , we **removed the duplicates** from each table we've extracted

```
# Remove duplicates from each DataFrames
sales_df_no_duplicates = sales_df.drop_duplicates()
inventory_df_no_duplicates = inventory_df.drop_duplicates()
products_df_no_duplicates = products_df.drop_duplicates()
stores_df_no_duplicates = stores_df.drop_duplicates()
```

Step2: Data transformation

We have calculated the **revenue** of each record in sales and added it as a new column into the sales table :

```
# Calculate revenue and add it as a new column
sales_df['revenue'] = sales_df['qty'] * sales_df['unit_price'] * (1 - sales_df['discount_percentage'] )

total_revenue = sales_df['revenue'].sum()
# Add the total revenue row to the DataFrame
total_revenue_row = pd.DataFrame({'qty': ['Total'], 'unit_price': [None], 'discount_percentage': [None],
                                  'revenue': [total_revenue]})
sales_df = pd.concat([sales_df, total_revenue_row], ignore_index=True)
print(sales_df)
```

Output :

	order_id	order_date	product_id	qty	discount_percentage	unit_price	\
0	IN-897	21-08-22	P-75	4	0.5	249.00	
1	IN-4441	23-03-17	P-134	2	0.3	29.99	
2	IN-4746	23-10-02	P-131	4	0.5	699.00	
3	IN-5734	22-09-20	P-4	4	0.6	79.99	
4	IN-6742	21-11-25	P-40	4	0.5	34.99	
...	...	...	...	...	...	...	
50452	IN-9246	21-11-17	P-137	3	0.3	79.99	
50453	IN-10568	22-07-04	P-4	4	0.4	79.99	
50454	IN-11497	22-05-25	P-107	2	0.2	199.00	
50455	IN-15707	23-03-04	P-172	1	0.1	899.00	
50456	NaN	NaN	NaN	Total	NaN	NaN	

	store_id	revenue
0	IK-1	4.980000e+02
1	IK-1	4.198600e+01
2	IK-1	1.398000e+03
3	IK-1	1.279840e+02
4	IK-1	6.998000e+01
...	...	...
50452	IK-101	1.679790e+02
50453	IK-101	1.919760e+02
50454	IK-101	3.184000e+02
50455	IK-101	8.091000e+02
50456	NaN	1.222055e+07

Since the data was too extensive , we decided to narrow it down and analyze the sales records of **IKEA of the year 2023** and we sorted it by date

```
import pandas as pd

# Assuming sales_df is already loaded with your data

# Convert the 'order_date' column to datetime format if it's not already
sales_df['order_date'] = pd.to_datetime(sales_df['order_date'])

# Filter the data for the year 2023
sales_2023 = sales_df[sales_df['order_date'].dt.year == 2023]

# Sort the filtered data by 'order_date' in ascending order
sales_2023_sorted = sales_2023.sort_values(by='order_date', ascending=True)

# Display the sorted DataFrame
print(sales_2023_sorted)
```

Output :

```
       order_id order_date product_id qty  discount_percentage  unit_price
31054  IN-44633 2023-01-19       P-11   1                  0.1       24.99
26804  IN-43488 2023-01-19       P-18   4                  0.4       45.00
12180  IN-47618 2023-01-19       P-18   2                  0.2       45.00
4394   IN-44672 2023-01-19       P-23   3                  0.4        9.99
23389  IN-47796 2023-01-19       P-22   3                  0.3      159.00
...         ...        ...       ...  ..                  ...         ...
49048  IN-14479 2023-12-23      P-110   4                  0.4      299.00
7059   IN-38614 2023-12-23       P-13   4                  0.6      699.00
4889   IN-40420 2023-12-23       P-22   1                  0.3      159.00
23525  IN-40001 2023-12-23       P-24   1                  0.1       49.99
39130  IN-19630 2023-12-23      P-145   4                  0.4      179.00

      store_id   revenue
31054    IK-23    22.491
26804    IK-19   108.000
12180     IK-9    72.000
4394      IK-4    17.982
23389    IK-17   333.900
...        ...       ...
49048    IK-94   717.600
7059      IK-5  1118.400
4889      IK-4   111.300
23525    IK-17    44.991
39130    IK-46   429.600

[1641 rows x 8 columns]
```

We added a **stock status column** into the inventory table since it will be relevant later on with the sales :

```python
import pandas as pd
# Display the first few rows of the inventory data
print("Original Inventory Table:")
print(inventory_df.head())

# Check if the required columns exist
if 'current_stock' in inventory_df.columns and 'reorder_level' in inventory_df.columns:
    # Add the stock_status column
    inventory_df['stock_status'] = inventory_df['current_stock'] / inventory_df['reorder_level']

    # Handle cases where reorder_level is 0 to avoid division by zero
    inventory_df['stock_status'] = inventory_df['stock_status'].replace([float('inf'), -float('inf')], 0)
    inventory_df['stock_status'] = inventory_df['stock_status'].fillna(0)

    # Save the updated table (optional)
    inventory_df.to_csv('/content/updated_inventory.csv', index=False)

    print("\nUpdated Inventory Table with 'stock_status':")
    print(inventory_df)
```

Original Inventory Table:

	iventory_id	product_id	current_stock	reorder_level
0	1	P-46	5	15
1	2	P-53	5	15
2	3	P-45	6	15
3	4	P-57	6	15
4	5	P-62	6	15

Updated Inventory Table with 'stock_status':

	iventory_id	product_id	current_stock	reorder_level	stock_status
0	1	P-46	5	15	0.333333
1	2	P-53	5	15	0.333333
2	3	P-45	6	15	0.400000
3	4	P-57	6	15	0.400000
4	5	P-62	6	15	0.400000
..	...	...	...	...	...
174	175	P-67	160	50	3.200000
175	176	P-170	166	50	3.320000
176	177	P-5	166	50	3.320000
177	178	P-91	169	50	3.380000
178	179	P-166	170	50	3.400000

- If stock_status < 1 : This can help identify products that may run out of stock soon and need immediate restocking.
- If stock_status > 1 : This could help in reducing excess inventory costs by moving surplus products or adjusting future order quantities.

From the sales Table , we computed the **average quantity sold per order** as it will reveal customer preferences later on :

```
import pandas as pd
# Group by order_id and calculate total quantity sold per order
order_totals = sales_2023_sorted.groupby('order_id')['qty'].sum()

# Calculate the average quantity per order
avg_quantity_per_order = order_totals.mean()
```

Based on the average computed beforehand , we added another column to the sales table entitled ‘compare to average’ that will attribute either ‘below’ or ‘above’ average in order to help with the customer preference conclusions :

```
# Add a descriptive column based on comparison to the average
sales_2023_sorted['compare_to_avg'] = sales_2023_sorted['total_quantity_per_order'].apply(
    lambda x: 'Above Average' if x > avg_quantity_per_order else 'Below Average'
)
# Print the updated DataFrame
print(f"Average quantity per order in 2023: {avg_quantity_per_order:.2f}")
print(sales_2023_sorted.head())
```

```
Average quantity per order in 2023: 2.55
   order_id order_date product_id qty  discount_percentage  unit_price
0  IN-44633 2023-01-19       P-11   1                 0.1       24.99
1  IN-43488 2023-01-19       P-18   4                 0.4       45.00
2  IN-47618 2023-01-19       P-18   2                 0.2       45.00
3  IN-44672 2023-01-19       P-23   3                 0.4        9.99
4  IN-47796 2023-01-19       P-22   3                 0.3      159.00

  store_id  revenue compare_to_avg
0    IK-23   22.491  Below Average
1    IK-19  108.000  Above Average
2     IK-9   72.000  Below Average
3     IK-4   17.982  Above Average
4    IK-17  333.900  Above Average
```

Step3: Schema Design

Fact table

Sales_id
Product_id
Store_id
Order_date_id
Inventory_id
Quantity_sold
Revenue
Discount_percentage
Compare_to_avg

Dimension Tables

Products_dimension :

Product_id
Product_name
Category
Subcategory
Unit_price

So Our schema consists of
1 fact table + 4 dimensions

It will be a **snowflake schema** as it:

- Reduces redundancy through data normalization.
- Ensures consistency with referential integrity via Foreign Keys.
- Scales easily for new attributes or dimensions.
- Supports detailed, multidimensional analysis and queries.
- Minimizes storage usage for large datasets

Inventory_dimension :

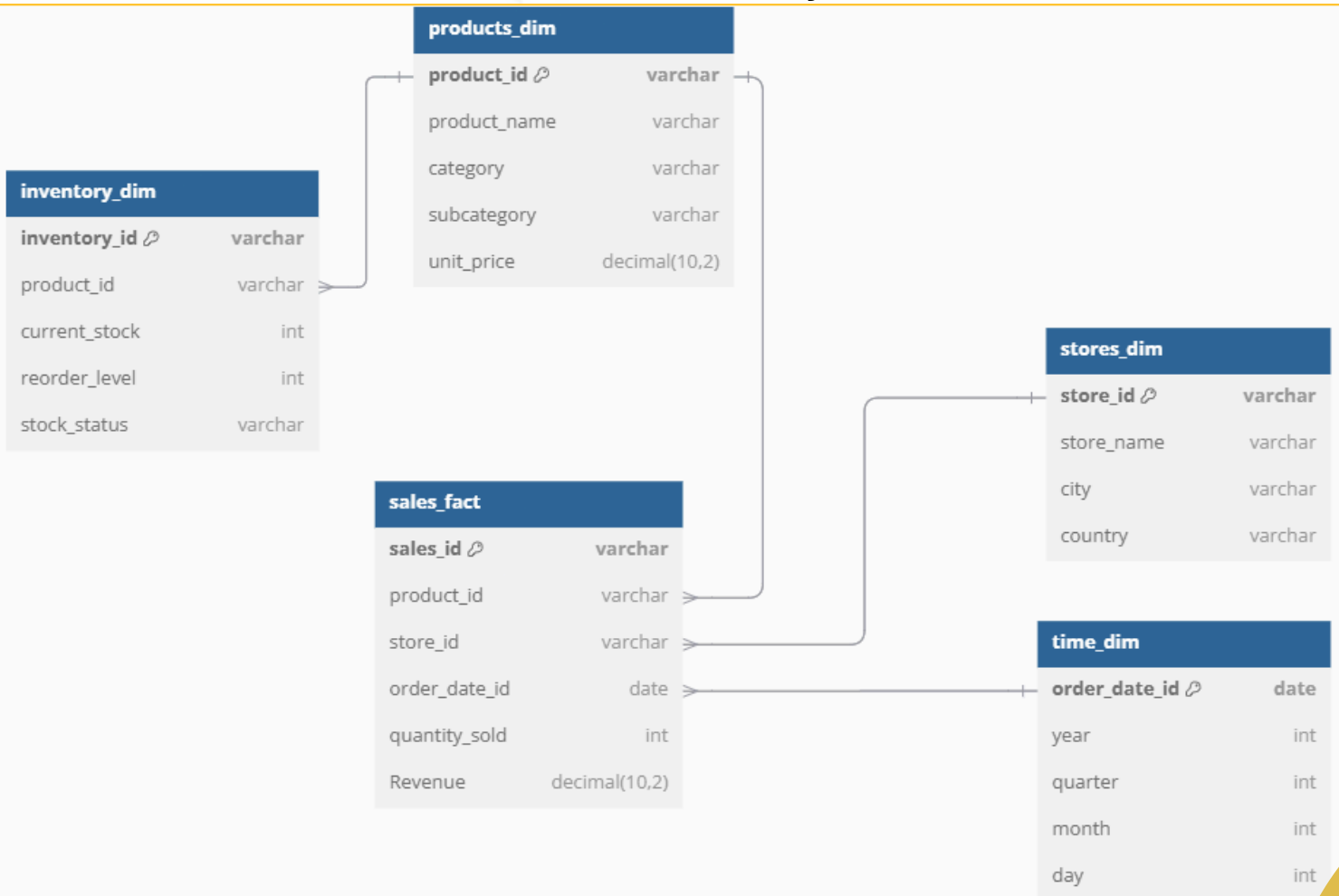
Inventory_id
Product_id
Current_stock
Reorder_level
Stock status

Stores_dimension

Store_id
Store_name
City
Country

Time_dimension :

Order_date_id
Year
Quarter
Month
Day



Step4: Data Loading

After identifying our fact and dimension tables, the next step is to use Python for creation :

1- Fact table

```
import pandas as pd

# Step 1: Rename columns to align with the fact table schema
sales_fact = sales_2023_sorted.copy()

sales_fact.rename(columns={
    'order_id': 'sales_id',              # Primary Key (PK)
    'qty': 'quantity_sold',              # Number of units sold
}, inplace=True)

# Step 2: Adjust column selection to include 'Qt sold compared to_avg'
columns_to_keep = [
    'sales_id',                          # Primary Key (PK)
    'product_id',                        # Foreign Key (FK) to products_dim
    'store_id',                          # Foreign Key (FK) to stores_dim
    'order_date',                        # Foreign Key (FK) to time_dim
    'quantity_sold',                     # Number of units sold
    'unit_price',                        # Unit price of product
    'revenue',                           # Total revenue
    'discount_percentage',               # Discount applied
    'compare_to_avg'                     # Descriptive attribute
]

# Check if all columns exist before selection
missing_columns = [col for col in columns_to_keep if col not in sales_fact.columns]
if missing_columns:
    print(f"Missing columns: {missing_columns}")
else:
    # Select relevant columns for the fact table
    sales_fact = sales_fact[columns_to_keep]

    # Step 3: Save the fact table as a CSV file
    sales_fact.to_csv('sales_fact.csv', index=False)

    # Step 4: Download the CSV file to your PC
    from google.colab import files
    files.download('sales_fact.csv')
```

2-Dimension Tables

```
# Step 1: Verify the columns in products_df
print(products_df.columns)

# Step 2: Select and create the products dimension table
products_dim = products_df[['product_id', 'product_name', 'category', 'subcategory', 'unit_pice']]

# Step 3: Save the products dimension table to a CSV file
products_dim.to_csv('products_dimension.csv', index=False)

# Step 4: Download the products dimension CSV file
from google.colab import files
files.download('products_dimension.csv')

print("Products dimension table saved and downloaded!")
```

```
Index(['product_id', 'product_name', 'category', 'subcategory', 'unit_pice'], dtype='object')
Products dimension table saved and downloaded!
```

```
# Step 1: Verify the columns in stores_df
print(stores_df.columns)

# Step 2: Select and create the stores dimension table
stores_dim = stores_df[['store_id', 'store_name', 'city', 'country']]

# Step 3: Save the stores dimension table to a CSV file
stores_dim.to_csv('stores_dimension.csv', index=False)

# Step 4: Download the stores dimension CSV file
from google.colab import files
files.download('stores_dimension.csv')

print("Stores dimension table saved and downloaded!")
```

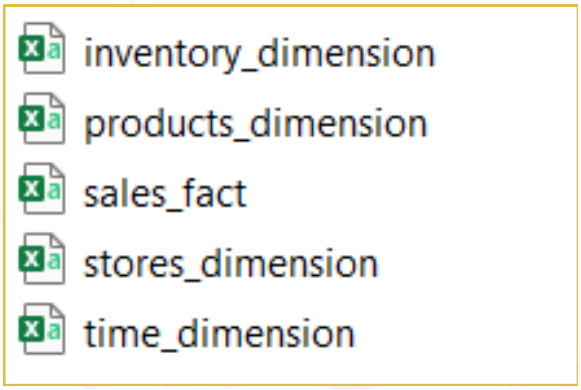
```
Index(['store_id', 'store_name', 'city', 'country'], dtype='object')
Stores dimension table saved and downloaded!
```

```
[ ]
# Step 1: Verify the columns in inventory_df
print(inventory_df.columns)

# Step 2: Select and create the inventory dimension table
inventory_dim = inventory_df[['iventory_id', 'product_id', 'current_stock', 'reorder_level', 'stock_status']]

# Step 3: Save the inventory dimension table to a CSV file
inventory_dim.to_csv('inventory_dimension.csv', index=False)

# Step 4: Download the inventory dimension CSV file
from google.colab import files
files.download('inventory_dimension.csv')

print("Inventory dimension table saved and downloaded!")
```

```
Index(['iventory_id', 'product_id', 'current_stock', 'reorder_level',
       'stock_status'],
      dtype='object')
Inventory dimension table saved and downloaded!
```

```python
import pandas as pd

# Step 1: Make sure 'order_date' is in datetime format
sales_2023_sorted['order_date'] = pd.to_datetime(sales_2023_sorted['order_date'])

# Step 2: Extract the required components (year, quarter, month, day)
time_dim = pd.DataFrame()

# Create the order_date_id as a unique identifier (you can use the index or a custom ID if needed)
time_dim['order_date_id'] = sales_2023_sorted['order_date'].dt.date.unique()

# Extract year, quarter, month, and day
time_dim['order_date'] = pd.to_datetime(time_dim['order_date_id'])
time_dim['year'] = time_dim['order_date'].dt.year
time_dim['quarter'] = time_dim['order_date'].dt.quarter
time_dim['month'] = time_dim['order_date'].dt.month
time_dim['day'] = time_dim['order_date'].dt.day

# Step 3: Save the time dimension table to a CSV file
time_dim.to_csv('time_dimension.csv', index=False)

# Step 4: Download the time dimension CSV file
from google.colab import files
files.download('time_dimension.csv')
```

=> We finally obtained this final output :



The data has been cleaned, transformed, and loaded, making it ready for the next phase!

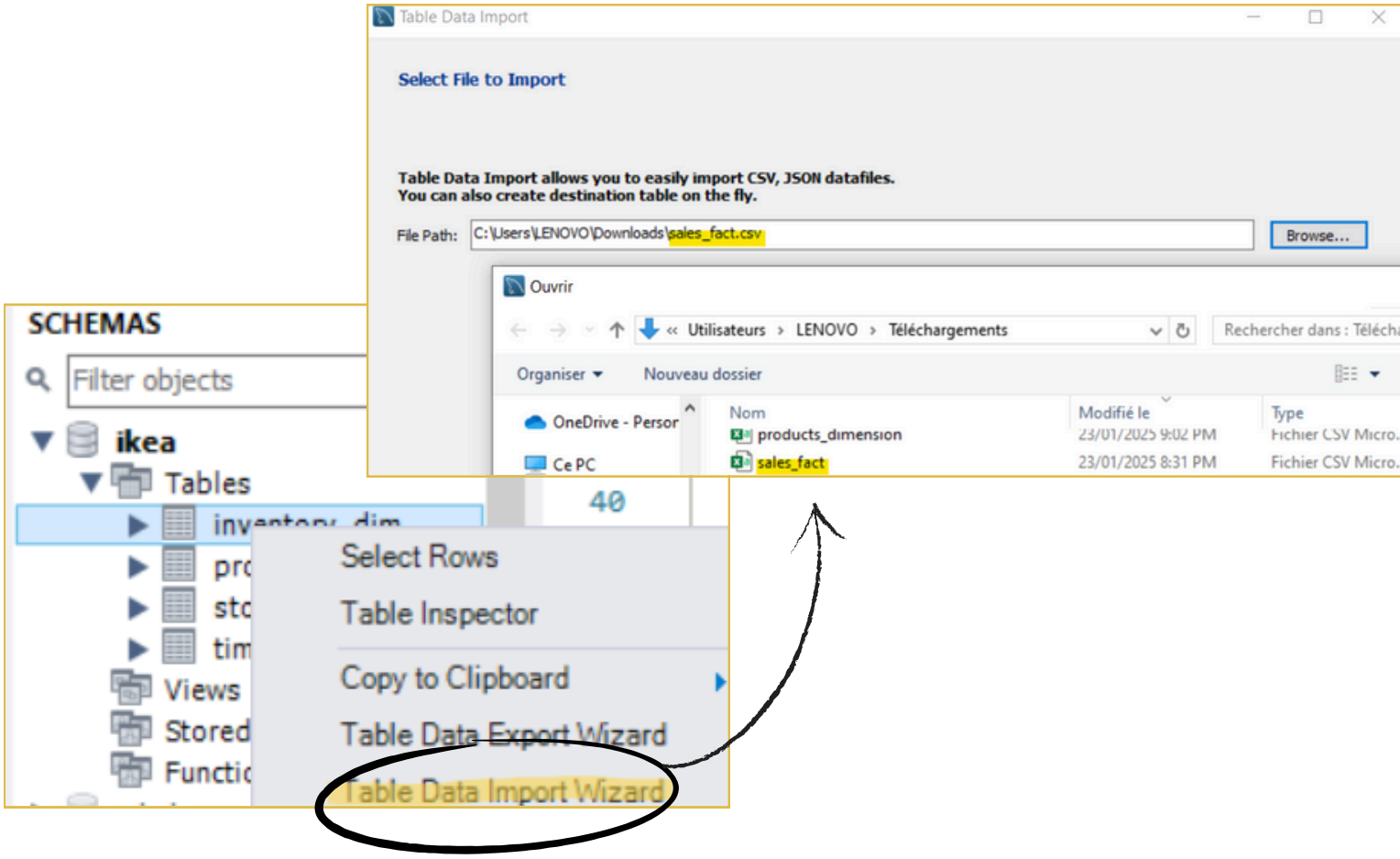
Phase 3: Data Storage

For data storage, we relied on MySQL as our primary database management system due to its reliability and scalability.
To efficiently manage and interact with the database, we used its user-friendly interface for executing queries, organizing data, and performing administrative tasks.

```
create database ikea ;
use ikea ;
CREATE TABLE sales_fact (
    sales_id varchar(100) PRIMARY KEY,
    product_id varchar(10),
    store_id varchar(10) ,
    order_date_id date ,
    quantity_sold INT,
    Revenue DECIMAL(10, 2),
    FOREIGN KEY (product_id) REFERENCES products_dim(product_id),
    FOREIGN KEY (store_id) REFERENCES stores_dim(store_id),
    FOREIGN KEY (order_date_id) REFERENCES time_dim(order_date_id)
);

  CREATE TABLE products_dim (
      product_id varchar(10) PRIMARY KEY,
      product_name VARCHAR(255) NOT NULL,
      category VARCHAR(100),
      subcategory VARCHAR(100),
      unit_price DECIMAL(10, 2)
  );
  select * from products_dim ;
```

```sql
CREATE TABLE stores_dim (
    store_id varchar(10) PRIMARY KEY,
    store_name VARCHAR(255) NOT NULL,
    city VARCHAR(100),
    country VARCHAR(100)
);
drop table time_dim;
CREATE TABLE time_dim (
    order_date_id date  PRIMARY KEY,
    year INT,
    quarter INT,
    month INT,
    day INT
);


CREATE TABLE inventory_dim (
    inventory_id varchar(100) PRIMARY KEY,
    product_id varchar(10),
    current_stock INT,
    reorder_level INT,
    stock_status VARCHAR(50),
    FOREIGN KEY (product_id) REFERENCES products_dim(product_id)
);
```

We imported the csv files of the sales fact table and the dimension tables extracted via python to the created tables in project bi database using the Table Data Import Wizard in MySQL



Once imported , the data is finally loaded into the Data Warehouse

Example for products_dim output :

```
select * from products_dim ;
```

product_id	product_name	category	subcategory	unit_price
P-1	BILLY Bookcase	Furniture	Storage	59.99
P-10	VARIERA Shelf Insert	Kitchen	Organizers	6.99
P-100	INREDA Box	Furniture	Storage	12.99
P-101	HÃ„LLINGE Shelf	Furniture	Storage	35.00
P-102	SAMLA Box	Storage	Storage	4.99
P-103	RÃ–NNAN Storage Cabinet	Furniture	Storage	59.99
P-104	UPPLEVA TV Unit	Living Room	Storage	349.00
P-105	KALLAX Shelf	Furniture	Storage	79.99

Example for inventory_dim `SELECT * FROM inventory_dim ;`

inventory_id	product_id	current_stock	reorder_level	stock_status
1	P-46	5	15	0.3333333333333333
10	P-114	8	15	0.5333333333333333
100	P-49	69	30	2.3
101	P-136	70	30	2.3333333333333335
102	P-137	70	30	2.3333333333333335
103	P-139	70	30	2.3333333333333335
104	P-176	70	30	2.3333333333333335
105	P-20	70	30	2.3333333333333335

Example for sales_fact `SELECT * FROM sales_fact ;`

sales_id	product_id	store_id	order_date_id	quantity_sold	Revenue
IN-43488	P-18	IK-19	2023-01-19	4	108.00
IN-44633	P-11	IK-23	2023-01-19	1	22.49
IN-44672	P-23	IK-4	2023-01-19	3	17.98
IN-47618	P-18	IK-9	2023-01-19	2	72.00
IN-47796	P-22	IK-17	2023-01-19	3	333.90
IN-48234	P-14	IK-25	2023-01-19	2	47.98
IN-49729	P-25	IK-16	2023-01-19	3	73.50
IN-45202	P-14	IK-6	2023-01-20	2	47.98

Example for stores_dim `SELECT * FROM stores_dim ;`

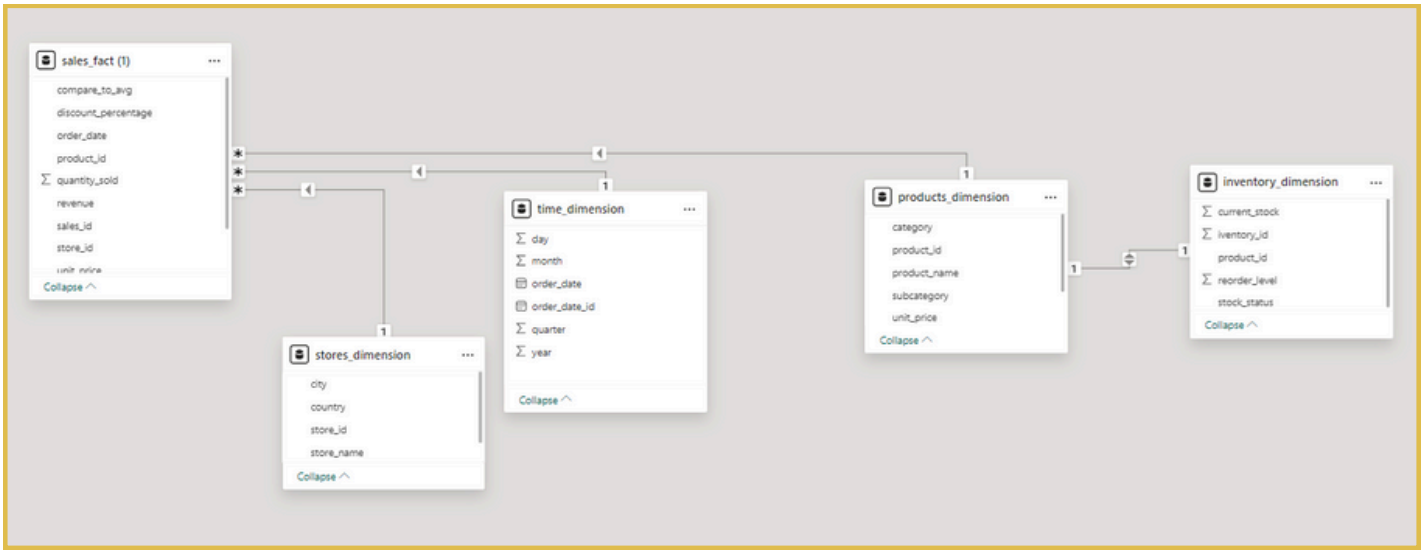
store_id	store_name	city	country
IK-1	IKEA Hanoi	Hanoi	Vietnam
IK-10	IKEA Eugene	Eugene	United States
IK-100	IKEA Baulkham Hills	Baulkham Hills	Australia
IK-101	IKEA Brisbane	Brisbane	Australia
IK-102	IKEA Perth	Perth	Australia
IK-103	IKEA Sydney	Sydney	Australia
IK-104	IKEA Buenos Aires	Buenos Aires	Argentina
IK-11	IKEA Harrisburg	Harrisburg	United States

Example for time_dim `SELECT * FROM time_dim ;`

order_date_id	year	quarter	month	day
2023-01-19	2023	1	1	19
2023-01-20	2023	1	1	20
2023-01-21	2023	1	1	21
2023-01-22	2023	1	1	22
2023-01-23	2023	1	1	23
2023-02-19	2023	1	2	19
2023-02-20	2023	1	2	20
2023-02-21	2023	1	2	21

Phase4 : Data Visualization

We imported the data into **Power BI**, using it not only for better visualization and insights but also as a platform to implement a **ROLAP** approach with a snowflake schema as the data model.
This schema organizes data into a central fact table linked to multiple dimension tables, enabling structured and efficient analysis. While the data was not connected to Power BI through a live connection, the dashboard mirrors the relational database structure, providing flexibility for dynamic and detailed analytics



Some measures were performed using **DAX** (Data Analysis Expressions)

Total Revenue :

```
Total Revenue = SUMX(
    'sales_fact (1)',
    'sales_fact (1)'[unit_price] * 'sales_fact (1)'[quantity_sold] * (1 - 'sales_fact (1)'[discount_percentage]))
```

Average quantity per order :

```
1  Average Qty per Order = DIVIDE(SUM('sales_fact (1)'[quantity_sold]), COUNT('sales_fact (1)'[order_date]))
```

Total number of categories :

```
1  Categories = DISTINCTCOUNT(products_dimension[category])
2
```

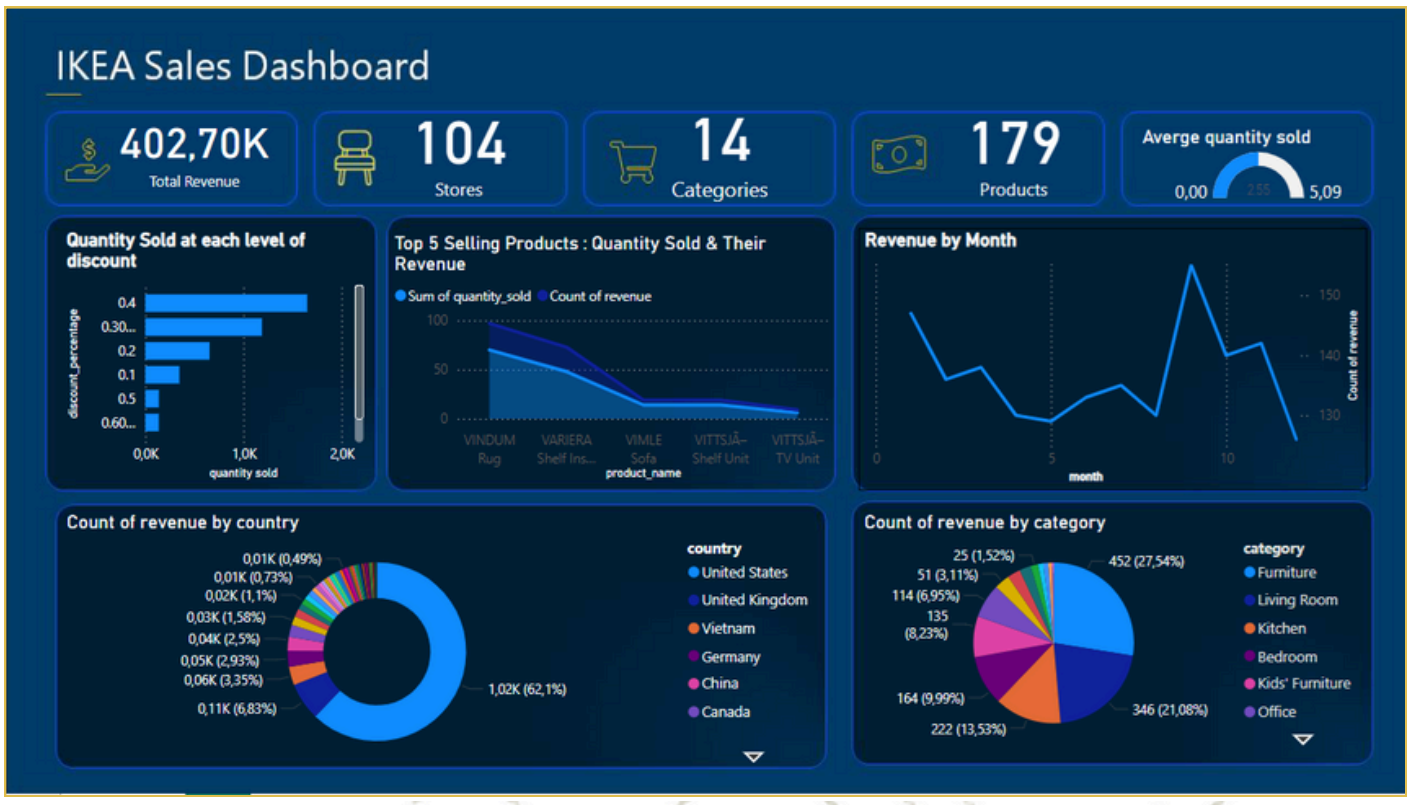
Total number of Products :

```
1  Products = COUNTROWS(products_dimension)
```

Total number of Stores :

```
1  Stores = COUNTROWS(stores_dimension)
```

Drawing insights from **IKEA** Dashboard :



INSIGHTS :

1. Overall Performance Metrics:

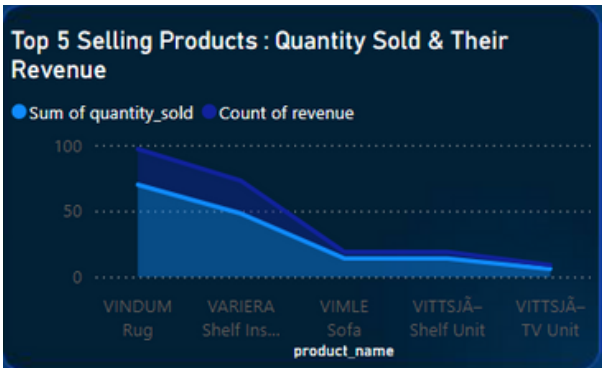
Total Revenue: IKEA generated **402.70K** in total revenue.
Number of Stores: The sales data spans **104** stores.
Categories and Products: There are **14** categories and **179** distinct products being analyzed.
Average Quantity Sold: The average quantity sold per order is 2.55 units.

2. Discount Levels and Sales:

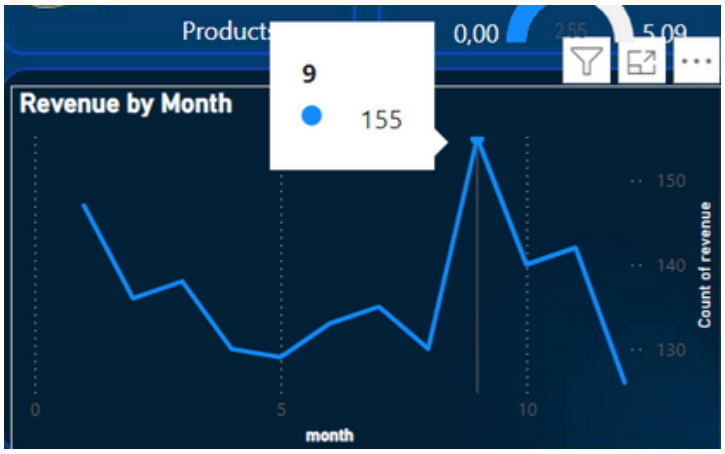
The Bar Chart indicates that as the discount percentage **increases**, the quantity sold also tends to **increase**, showing customers respond well to discounts .

3. Top 5 Selling Products:

Products like **VINDUM Rug** and **VARIERA Shelf** are the leaders in both quantity sold and revenue.
This highlights popular products that drive sales .



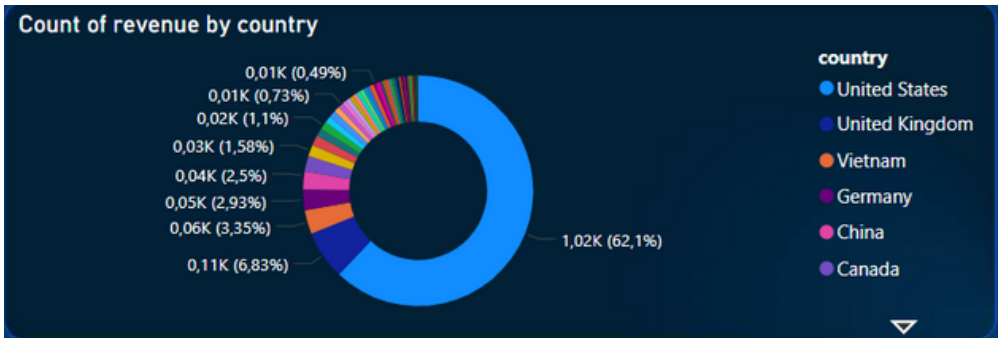
4. Revenue by Month:



The line graph shows **fluctuations** in monthly revenue, with noticeable peaks and dips.
The line graph shows that the peak of the monthly revenue (155k) is detected at September 2023 (month 9).
=>This could suggest seasonal trends or the impact of **promotions/campaigns.**

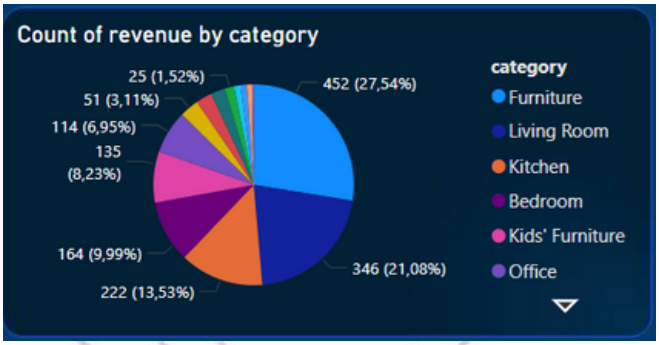
5. Revenue Distribution by Country:

The pie chart shows the **United States** contributes the largest share of revenue (62.1%), followed by **United Kingdom**(6.83%) and other countries. This emphasizes the importance of the US market for IKEA .



6. Revenue by Category :

While **Furniture** leads with 27.54% of revenue, categories like **Office** and **Kids**' Furniture appear to contribute minimally .



RECOMMENDATIONS :

1. Strategic Decisions on Inventory

Promote Best-Selling Products: Products like **VINDUM Rug** and **VARIERA Shelf** should be promoted more heavily across all stores .

For products with low sales volumes,the manager should analyze their performance alongside purchase patterns based on the average and maximum purchase quantities computed. If the product is underperforming , the manager can take action and introduce targeted promotions such as pairing low-performing products with popular ones in “Buy 3, get 1 free” deals) to boost visibility and sales.

2. Sales Performance Improvements

While **Furniture** leads with 27.54% of revenue, categories like Office and Kids' Furniture appear to contribute minimally. The manager could Introduce **promotions or explore partnerships** with schools or workplaces to boost sales in these segments

Adjust Discount Strategies: The manager should consider increasing discounts for high-demand products that are not yet sold in large quantities to boost their sales volume.

3. Store Operations and Expansion

Store Closures or Restructuring: Reviewing underperforming stores in regions like **China** and **Germany,** the manager should consider closing, restructuring, or reallocating resources to higher-performing regions.

Geographic Expansion: Given the high sales in the **United States**, consider expanding operations in this region and looking into other regions with similar potential, such as the **UK** or **Vietnam.**

4. Revenue and Profitability Strategies

Restock Profitable Products: Focus on maintaining stock levels of high-revenue products, especially **VINDUM Rug** and **VARIERA Shelf**, to prevent stockouts and capitalize on demand. The manager can regularly check the stock status column (over or below 1) , inserted during the transformation phase to ensure that the demand and supply of the popular products are met .

5. KPI Monitoring

Establish KPIs for Sales and Profitability: Implement **KPIs** such as total sales per store and average revenue per product. Track these regularly to adjust strategies as needed . For example , each season , the manager can set an average total sales per store that should be tracked . At the end of the set period , If the total sales meet the aspiration put beforehand , no regulations are needed. Otherwise, The manager should react and change the store marketing or sales strategy accordingly.

CONCLUSION

This Business Intelligence project provided a detailed analysis of **IKEA's performance**, covering data collection, transformation, visualization, and actionable recommendations. For example , we highlighted key metrics to identify top-performing categories like Furniture and opportunities to improve underperforming segments like Office and Kids' Furniture. Insights into seasonal trends, discount strategies, and inventory optimization were provided, along with regional and store-level evaluations for targeted improvements.
In a nutshell, our data-driven recommendations support IKEA's mission to improve everyday life and its vision to be a sustainable, people-centric retailer, helping the company navigate challenges and drive growth.