Course:	Higher Diploma in Science in Data Analytics - HDSDA_SEP24
Module:	Business Intelligence
Lecturer:	Jorge Basilio
Project:	CA2
Submission Date:	30-04-2025

Student Name:	**Nikolaos Gkmpenompa**
Student Number:	**22213708**
Student Name:	Charleen Pierrette Mange
Student Number:	22207899
Student Name:	Yassin Kedir Simea
Student Number:	24164097

Table Of Contents

Online Retail Business Intelligence Solution

1. Introduction

1.1 Company Description

The company in this project is a retail establishment that sells directly to consumers using a business-to-consumer (B2C) model. It focuses on technology and consumer electronics such as MP3 players, DVDs, televisions, software, and digital cameras.

1.2 Project Scope Definition

The company is experiencing a decline in Sales performance over the last few months. Management has no visibility as to why this is the case.

Stakeholders need to identify:

- Which Products are underperforming.
- Which region/ Customers are dropping.
- Whether marketing campaigns are effective.
- How customer behaviour and satisfaction are changing.

The main issues include:

- Identify the root cause of the declining sales
- Uncover underperforming products or regions (Geographic Key).
- Predict future sales trends based on past behaviour.
- Optimise marketing spend.
- Enhance customer loyalty and retention.

The key questions that the Bi Solution must address are:

- Which product/ category is declining the most? This is to allow us to focus on the inventory, promotion, and product development.
- Which customer segment is experiencing decline? So, the company focuses on targeting retention campaigns to specific customer groups.
- Which regions are losing market share? This enables us to relocate resources and adjust operations.
- Are marketing efforts improving sales? This will optimise the budget towards high-performing campaigns.
- What are projected sales for next year? This is to allow better financial and operational planning.

2. Data Source Description

For this project we will be working with publicly available datasets from Microsoft. We selected the "Microsoft Contoso BI Demo Dataset" due to its detailed structure enabling us to represent a retail business and to conduct analysis. We chose specifically the "Contoso BI demo" backup file.

The row datasets can be downloaded from the url: https://www.microsoft.com/en-us/download/details.aspx?id=18279

To extract the datasets, we used Microsoft Azure SQL Server and SQL queries. Later we cleaned and tables using Python. The process is explained in detail in chapter 4. The tables that we worked with are shown below:

2.1 Customers_table.csv

Attribute	Description
CustomerID	ID for each customer.
CustomerName	Full name of the customer.
Gender	Gender of the customer (Male, Female).
EmailAddress	Email address of the customer.
Education	Highest level of education of the customer.
Occupation	Professional field of the customer.
Address	Street address of the customer.
GeographyKey	A key linking the customer to a geographic region.
Phone	Phone number of the customer.

2.2 Products_table.csv

Attribute	Description
ProductID	ID for each product.
ProductName	Name of the product including model and characteristics.
UnitCost	The cost that the company purchased the product.
UnitPrice	The selling price of the product to customers.

2.3 Sales_table.csv

Attribute	Description
SalesID	ID for each sales transaction.
SalesAmount	Amount of the sale.
Quantity	Number of units sold in each transaction.
ProductID	ID linking the sale to the corresponding product in the Products table.
ProductName	Name of the product sold.
CustomerID	ID linking the sale to the corresponding customer in the Customers table.
CustomerName	Name of the customer that made the purchase.
OrderDate	Date of each sales transaction.

2.4 Returns_table.csv

Attribute	Description
ReturnID	ID for each return transaction.
SalesID	ID for each sales transaction.
SalesAmount	Amount of the sale.
Quantity	Number of units sold in each transaction.
ProductID	ID linking the sale to the corresponding product in the Products table.
ProductName	Name of the product sold.
CustomerID	ID for each customer.
CustomerName	Full name of the customer.
OrderDate	Date of the original sales transaction.

2.5 Stock_table.csv

Attribute	Description
StockID	ID for each product in stock.
ProductID	ID of the corresponding product.
StockQuantity	Number of units of each product in stock.
UnitCost	The cost that the company purchased the product.

2.6 Expenses_table.csv

Attribute	Description
ExpenseID	ID for each expense.
Department	The department responsible for the expense.
Category	The type of expense (Training, Meals, Travel, etc.).
Year	The year the expense occurred.
Random_Boost1	A randomly generated variable for simulation purposes.
Random_Boost2	A second randomly generated variable for simulation purposes.
Amount	Amount of the expense.

3. Proposed Solution Design

The proposed Business Intelligence (BI) solution was designed to integrate data from multiple sources: sales, customer demographics, product catalogue, returns, expenses, and time. These are connected into a single model.

Using tools like Python, Power BI, MySQL we developed a centralised data model that allows our business to monitor sales performance, customer behaviour, product profitability, operational expenses, and forecast future trends.

The proposed solution will consist of multiple layers:

Data Integration Layer: Combines different source tables (Sales, Customers, Products, Returns, Expenses) through relationships, forming a star schema around a central Date table. A star schema is ideal for analytical reporting since it reduces relationships and improves query efficiency.

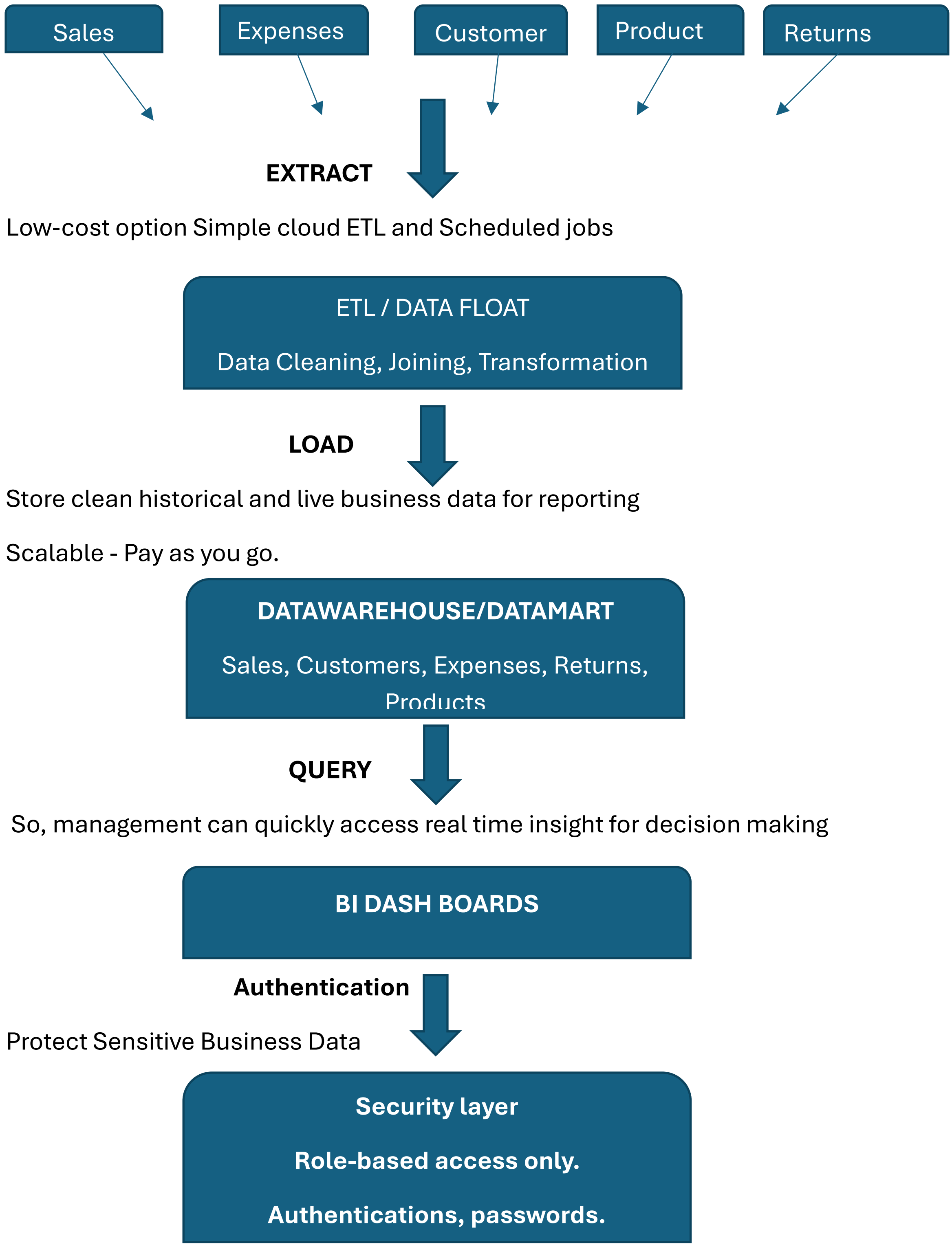
Data Storage Layer: All data will be loaded to Power BI to allows quick access, fast query performance and offline reporting.

Analytics Layer: Calculated columns and measures will be created to generate KPIs such as Total Sales, Total Expenses, Profit, Total Returns, Customer Churn Rate, etc. This will allow users to conduct analysis and get useful information.

Visualization Layer: Interactive dashboards, reports, and slicers will be developed to allow business users to filter data by Year, Department, Product, Customer Demographics, and be able to monitor important business trends.

3.1 Architectural Design for the Bi Solution

Operation system generation raw data



4. Technical Solution

The following application will be used

Design Stage	Application	Version
Data Storage	MySQL	
ETL Process	python	3.12.3
Dashboards and Reports	Power BI	2.142.928.0

4.1 Data Acquisition and Initial Staging

As mentioned in the second chapter of this report, for this project selected the "Microsoft Contoso BI Demo Dataset", and specifically the "Contoso BI demo" backup file.

"The Contoso BI Demo dataset is used to demonstrate DW/BI functionalities across the entire Microsoft Office product family. This dataset includes C-level, sales/marketing, IT, and common finance scenarios for the retail industry and support map integration. In addition, this dataset offers large volumes of transactions from OLTP and well-structured aggregations from OLAP, along with reference and dimension data." Microsoft

To extract the datasets, we used Microsoft Azure SQL Server and SQL queries. Later we cleaned and tables using Python. The process is explained in detail in chapter 4.

The Contoso BI Demo Dataset consist of the following datasets:

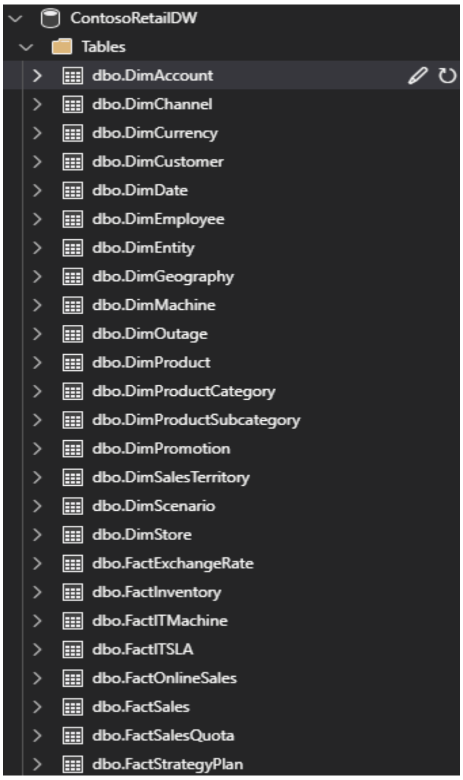


Figure 4.1 Contoso Retail Dataset list

The following tables were selected for data extraction:

dbo.DimCustomer
dbo.FactSales
dbo.FactOnlineSales
dbo.DimProduct
dbo.FactInventory
dbo.FactITMachine.

To restore and process the Contoso BI Demo dataset we used the Microsoft Azure SQL Server application.

"Microsoft Azure SQL Server (commonly called Azure SQL Database) is a cloud-based version of Microsoft's SQL Server database engine. It allows storing, managing, and retrieving data just like a traditional SQL Server, but without needing to maintain hardware. " Microsoft Azure.

After installing Microsoft Azure SQL Server, we connected to our "localhost" server and then selected the Contoso BI Demo dataset (ContosoRetailDW) which we saved in the backup folder in the directory of the application.

The SQL queries that were used to create the tables and their results are presented below:

4.1.1 Customer_Dim

The following query was used to select the top 1000 customers from the **DimCustomer** table:

```
SELECT TOP 1000 * FROM DimCustomer;
```

Below is a sample of the table:

Results Messages

	CustomerKey	GeographyKey	CustomerLabel	Title	FirstName	MiddleName	LastName	NameStyle	BirthDate	MaritalStatus	Suffix	Gender	EmailAddress	Ye
1	1	680	11000	NULL	Jon	V	Yang	0	1966-04-08	M	NULL	M	jon24@adventure-works.com	9
2	2	692	11001	NULL	Eugene	L	Huang	0	1965-05-14	S	NULL	M	eugene10@adventure-works.com	6
3	3	493	11002	NULL	Ruben	NULL	Torres	0	1965-08-12	M	NULL	M	ruben35@adventure-works.com	6
4	4	519	11003	NULL	Christy	NULL	Zhu	0	1968-02-15	S	NULL	F	christy12@adventure-works.com	7
5	5	706	11004	NULL	Elizabeth	NULL	Johnson	0	1968-08-08	S	NULL	F	elizabeth5@adventure-works.com	8
6	6	478	11005	NULL	Julio	NULL	Ruiz	0	1965-08-05	S	NULL	M	julio1@adventure-works.com	7
7	7	509	11006	NULL	Janet	G	Alvarez	0	1965-12-06	S	NULL	F	janet9@adventure-works.com	7
8	8	568	11007	NULL	Marco	NULL	Mehta	0	1964-05-09	M	NULL	M	marco14@adventure-works.com	6
9	9	425	11008	NULL	Rob	NULL	Verhoff	0	1964-07-07	S	NULL	F	rob4@adventure-works.com	6
10	10	492	11009	NULL	Shannon	C	Carlson	0	1964-04-01	S	NULL	M	shannon38@adventure-works.com	7
11	11	478	11010	NULL	Jacquelyn	C	Suarez	0	1964-02-06	S	NULL	F	jacquelyn20@adventure-works.com	7
12	12	478	11011	NULL	Curtis	NULL	Lu	0	1963-11-04	M	NULL	M	curtis9@adventure-works.com	6

Figure 4.1.1 Customer_Dim sample

4.1.2 Product_Dim

The following query was used to select the columns from the **DimProduct** table to pull basic product information like product name, cost, and price:

```
SELECT
    p.ProductKey,
    p.ProductName,
    p.UnitCost,
    p.UnitPrice
FROM DimProduct p;
```

A sample of the table is shown below:

	ProductKey	ProductName	UnitCost	UnitPrice
1	1	Contoso 512MB MP3 Player E51 Silver	6.62	12.99
2	2	Contoso 512MB MP3 Player E51 Blue	6.62	12.99
3	3	Contoso 1G MP3 Player E100 White	7.40	14.52
4	4	Contoso 2G MP3 Player E200 Silver	11.00	21.57
5	5	Contoso 2G MP3 Player E200 Red	11.00	21.57
6	6	Contoso 2G MP3 Player E200 Black	11.00	21.57
7	7	Contoso 2G MP3 Player E200 Blue	11.00	21.57
8	8	Contoso 4G MP3 Player E400 Silver	30.58	59.99
9	9	Contoso 4G MP3 Player E400 Black	30.58	59.99
10	10	Contoso 4G MP3 Player E400 Green	30.58	59.99
11	11	Contoso 4G MP3 Player E400 Orange	30.58	59.99
12	12	Contoso 4GB Flash MP3 Player E401 Blue	35.72	77.68

Figure 4.1.2 Product_Dim sample

4.1.3 Sales_Dim

The following query was used to select the columns from the **FactOnlineSales** table and to join the **DimProduct** and **DimCustomer** tables to pull product names and customer details linked to each online sale:

```sql
SELECT
    s.SalesOrderNumber,
    s.SalesAmount,
    s.SalesQuantity AS Quantity,
    p.ProductName,
    c.CustomerKey,
    c.FirstName + ' ' + c.LastName AS CustomerName
FROM FactOnlineSales s
JOIN DimProduct p ON s.ProductKey = p.ProductKey
JOIN DimCustomer c ON s.CustomerKey = c.CustomerKey;
```

A sample of the table is shown below:

	SalesOrderNumber	SalesAmount	Quantity	ProductName	CustomerKey	CustomerName
68	20070125315548	152.00	1	Proseware Ink Jet Instant PDF Sheet-Fed Scanne…	4549	Morgan Brooks
69	20070125513944	285.00	1	The Phone Company Touch Screen Phones Infrared…	2945	Andrew Moore
70	20070125513949	285.00	1	The Phone Company Touch Screen Phones Infrared…	2950	Dwayne Ramos
71	20070125513951	285.00	1	The Phone Company Touch Screen Phones Infrared…	2952	Dylan Hughes
72	20070125712981	66.6235	1	MGS Flight Simulator X M250	1982	Joe Sanz
73	20070125712983	66.6235	1	MGS Flight Simulator X M250	1984	Gabriella Cart…
74	20070125713015	66.6235	1	MGS Flight Simulator X M250	2016	Rebecca Parker
75	20070125211636	103.55	1	SV 16xDVD M330 Black	637	Kimberly Cox
76	20070125211638	103.55	1	SV 16xDVD M330 Black	639	Antonio Powell
77	20070125211643	103.55	1	SV 16xDVD M330 Black	644	Naomi Munoz
78	200702095CS430	3.1825	1	Contoso In-Line Coupler E180 Silver	18762	NULL
79	200702095CS430	3.1825	1	Contoso In-Line Coupler E180 Silver	18762	NULL

Figure 4.1.3 Sales_Dim sample

4.1.4 Returns_Dim

The following query was used to select the columns from the **FactSales** table and to join the **DimProduct** and **DimCustomer** tables to pull product names and customer details linked to each sale (return):

```sql
SELECT
```

```
    sr.SalesKey,
    sr.SalesAmount,
    sr.ReturnQuantity,
    sr.ReturnAmount,
    sr.UnitCost,
    sr.UnitPrice,
    p.ProductName,
    c.CustomerKey,
    c.FirstName + ' ' + c.LastName AS CustomerName
FROM FactSales sr
JOIN DimProduct p ON sr.ProductKey = p.ProductKey
JOIN DimCustomer c ON sr.SalesKey = c.CustomerKey;
```

A sample of the table is shown below:

Results Messages

	SalesKey	SalesAmount	ReturnQuantity	ReturnAmount	UnitCost	UnitPrice	ProductName
1	6	13999.65	0	0.00	183.94	399.99	Contoso Air conditioner 8000BTU M0320 Silver
2	7	843.60	0	0.00	68.06	148.00	A. Datum Slim Digital Camera M180 Green
3	8	4399.912	0	0.00	229.93	499.99	Adventure Works 32" LCD HDTV M130 Black
4	9	1702.575	0	0.00	33.32	72.45	MGS King& Myths: The Age Collection M340
5	10	1702.80	0	0.00	50.47	99.00	Adventure Works LCD17 E200 Black
6	11	128.00	0	0.00	16.31	32.00	MGS Age of Empires Expansion: The Rise of Rome 2009 E181
7	12	2290.00	0	0.00	116.75	229.00	Proseware Projector 480p LCD12 Black
8	13	2832.00	0	0.00	78.19	236.00	Proseware Slim-Design Fax Machine with Answering System X…
9	14	639.488	0	0.00	25.47	49.96	Contoso Carrying Case E312 Blue
10	15	430.00	0	0.00	21.92	43.00	MGS Age of Mythology: Gold Edition 2009 E144
11	16	1245.911	0	0.00	71.37	139.99	Contoso Microwave 1.0CuFt E0110 Black
12	17	2609.91	0	0.00	96.08	289.99	Contoso DVD 15-Inch Player Portable L200 Silver

Figure 4.1.4 Returns_Dim sample

4.1.5 Stock_Dim

The following queries were used to first select the top 1000 rows from the **FactInventory** table and then to pull the columns product key, safety stock quantity, and unit cost:

```
SELECT TOP 1000 * FROM FactInventory;

SELECT
    f.ProductKey,
    f.SafetyStockQuantity,
    f.UnitCost
FROM FactInventory f;
```

A sample of the table is shown below:

Results Messages

	ProductKey	SafetyStockQuantity	UnitCost
1	2086	9	403.53
2	643	9	77.72
3	18	6	50.56
4	1587	90	8.27
5	2269	12	15.29
6	1480	9	65.77
7	2379	9	183.94
8	736	6	54.26
9	1457	9	86.91
10	904	3	38.74
11	2172	3	204.64
12	1217	6	255.68

Figure 4.1.5 Returns_Dim sample

4.1.6 Expenses_Dim

The following query was used to select columns from the **FactITMachine** table and to pull machine cost details, filtering to only include records where the cost amount is not zero:

```
SELECT
    f.ITMachineKey,
    f.MachineKey,
    f.Datekey,
    f.CostAmount,
    f.CostType,
    f.ETLLoadID,
    f.LoadDate,
    f.UpdateDate
FROM FactITMachine f
WHERE f.CostAmount IS NOT NULL;
```

The steps above created the raw data for Customers, Products, Sales, Returns, Stock, and Expenses, forming the source datasets for our retail business.

4.2 Data Cleaning and Preparation

Once dummy datasets were acquired, a MySQL database named *businessintelligence_staging_db* was created in MySQL Workbench to serve as the initial repository for raw data. The query to create staging area in MySQL was as follows:

```
create database BusinessIntelligence_Staging_DB;
use BusinessIntelligence_Staging_DB;
create table Customers (CustomerKey int, GeographyKey int, CustomerLabel varchar(100),
        Title varchar (5), FirstName varchar(50), MiddleName varchar(50),
        LastName varchar (50), NameStyle int, BirthDate date, MaritalStatus varchar (1),
        Suffix varchar(10), Gender varchar(1), EmailAddress varchar(100), YearlyIncome decimal(10,2),
        TotalChildren int, NumberChildrenAtHome int, Education varchar(30), Occupation varchar(30),
        HouseOwnerFlag int, NumberCarsOwned int, AddressLine1 varchar(50), AddressLine2 varchar(50),
        Phone int, DateFirstPurchase date, CustomerType varchar(10), CompanyName varchar(4),
        ETLLoadID int, LoadDate date, UpdateDate date);

create table products ( ProductKey int, ProductName varchar(150),
        UnitCost decimal(10, 2), UnitPrice decimal(10, 2));

create table sales(SalesOrderNumber int, SalesAmount decimal (10, 2), Quantity int,
        ProductName varchar(150), CustomerKey int, CustomerName varchar (50));

create table returns (SalesKey int, SalesAmount decimal (10,2), ReturnQuantity int,
        ReturnAmount decimal (10, 2), UnitCost decimal(10, 2), UnitPrice decimal(10,2),
        ProductName varchar(150), CustomerKey int, CustomerName varchar(50));

create table stock (ProductKey int, SafetyStockQuantity int, UnitCost decimal(10,2));

create table expenses (Expense_ID int, Department varchar(100), Category varchar(50),
        Year int, Amount decimal(10, 2));
```

The datasets were then imported into *businessintelligence_staging_db* using bulk insert operations, ensuring all original fields, including inconsistencies such as mixed date formats, duplicate entries, and missing values were preserved. This staging layer acted as a temporary storage area, allowing for further extraction without altering the source data. The raw data in *businessintelligence_staging_db* included multiple tables with relational dependencies, such as sales orders linked to customer IDs and product, which would later be processed for analytical purposes. The raw datasets were also imported into Python for extraction and transformation. Python was selected over MySQL for data wrangling due to its superior flexibility and ease of use in handling extract-transform workflows.

The process began by importing essential Python libraries for data manipulation. This approach streamlined the data preparation pipeline while maintaining compatibility with downstream downstream database operations.

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from pandas import DataFrame, Series
from datetime import timedelta
import os
from scipy import stats
from sqlalchemy import create_engine
import mysql.connector
import warnings
warnings.filterwarnings('ignore')
```

Concurrently, we created a MySQL data warehouse named BusinessIntelligence_warehouse to store the cleaned data. After processing and cleaning the datasets in Python, they were exported to this warehouse. The following query was used to establish the data warehouse:

```sql
create database BusinessIntelligence_warehouse;

create table customers (CustomerID int primary key, CustomerName varchar(50), Gender varchar(1),
        EmailAddress varchar(150), Education varchar(150), Occupation varchar(150),
        Address varchar(150), GeographyKey int, Phone int
 );
create table products(ProductID int primary key, ProductName varchar(150),
        UnitCost decimal(10,2), UnitPrice decimal(10,2)
 );
create table sales (SalesID int primary key, SalesAmount decimal(10,2), Quantity int,
        ProductID int, ProductName varchar(150), CustomerID int, CustomerName varchar(150),
        OrderDate date,

        FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
        FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
 );
create table returns (ReturnID int primary key, SalesID int, SalesAmount decimal (10, 2),
        Quantity int, ProductID int, ProductName varchar(150), CustomerID int,
        CustomerName varchar(150), OrderDate date,

        FOREIGN KEY (SalesID) REFERENCES Sales(SalesID),
        FOREIGN KEY (ProductID) REFERENCES Products(ProductID),
        FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
 );
create table stock (StockID int primary key, ProductID int, StockQuantity int,
        UnitCost decimal(10, 2),

        FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
 );
create table espenses(Id int primary key, MachineKey int, ITMachineKey int, DateKey date,
        CostAmount decimal (10, 2), CostType decimal (10, 2));
```

As shown below, the process of optimising datasets and transferring the cleaned data into data warehouse in SQL was done for each of table using python.

4.2.1 Customers Dataset

The Customers dataset was imported and cleaned by removing unnecessary columns, renaming and reorganising some fields, combining first and last names into a full name, making the gender labels clearer, and then saving the cleaned dataset into a new CSV file.

```python
#importing dataset
customers = DataFrame()
customers = pd.read_csv('C:/Users/maxwe/Desktop/BI_Project/Customers_Dim.csv')

#dropping unwanted columns
customers = customers.drop(columns=['CustomerLabel', 'Title', 'MiddleName', 'NameStyle', 'BirthDate',
                                    'MaritalStatus', 'Suffix', 'YearlyIncome', 'BirthDate', 'TotalChildren',
                                    'NumberChildrenAtHome', 'HouseOwnerFlag', 'NumberCarsOwned', 'AddressLine2',
                                    'DateFirstPurchase', 'CustomerType', 'CompanyName', 'ETLLoadID', 'LoadDate',
                                    'UpdateDate'])

#renaming "CustomerKey to "CustomerID"
customers = customers.rename(columns={"CustomerKey": "CustomerID", "AddressLine1": "Address"})

#creating CustomerName by combining FirstName and LastName
customers['CustomerName'] = customers['FirstName'] + ' ' + customers['LastName']

#dropping the original FirstName and LastName columns
customers = customers.drop(['FirstName', 'LastName'], axis=1)

#moving 'CustomerName' to the 2nd position and 'GeographyKey' to the 7th
cols = customers.columns.tolist()
cols.insert(1, cols.pop(cols.index('CustomerName')))
cols.insert(7, cols.pop(cols.index('GeographyKey')))
customers = customers[cols]

#replacing 'Gender' values for clarity
customers['Gender'] = customers['Gender'].replace({
    'F': 'Female',
    'M': 'Male'
})

#exporting clean dataset to Mysql workbench
engine = create_engine('mysql+mysqlconnector://root:password@localhost/BusinessIntelligence_warehouse')

customers.to_sql(name='customers', con=engine, if_exists='replace', index=False)
```

A sample of the cleaned customers table is shown below. It was exported from python to the Mysql data warehouse:

```sql
6 • SELECT * FROM businessintelligence_warehouse.customers;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Fetch rows:

CustomerID	CustomerName	Gender	EmailAddress	Education	Occupation	Address	GeographyKey	Phone
1	Jon Yang	Male	jon24@adventure-works.com	Bachelors	Professional	3761 N. 14th St	680	1 (11) 500 555-0162
2	Eugene Huang	Male	eugene10@adventure-works.com	Bachelors	Professional	2243 W St.	692	1 (11) 500 555-0110
3	Ruben Torres	Male	ruben35@adventure-works.com	Bachelors	Professional	5844 Linden Land	493	1 (11) 500 555-0184
4	Christy Zhu	Female	christy12@adventure-works.com	Bachelors	Professional	1825 Village Pl.	519	1 (11) 500 555-0162
5	Elizabeth Johnson	Female	elizabeth5@adventure-works.com	Bachelors	Professional	7553 Harness Circle	706	1 (11) 500 555-0131
6	Julio Ruiz	Male	julio1@adventure-works.com	Bachelors	Professional	7305 Humphrey Drive	478	1 (11) 500 555-0151

Figure 4.1.6 customers_warehouse_table.sql sample

4.2.2 Products Dataset

The Products dataset was imported and cleaned by renaming the column Product Key to Product ID for consistency and then saved the cleaned dataset into BusinessIntelligence_warehouse in Mysql.

```python
#importing dataset
products = DataFrame()
products = pd.read_csv('C:/Users/maxwe/Desktop/BI_Project/Products_Dim.csv')

#renaming "CustomerKey to "CustomerID"
products = products.rename(columns={"ProductKey": "ProductID"})

#exporting clean dataset to Mysql workbench
engine = create_engine('mysql+mysqlconnector://root:password@localhost/BusinessIntelligence_warehouse')

products.to_sql(name='products', con=engine, if_exists='replace', index=False)
```

A sample of the cleaned products table is shown below. It was exported from python to the Mysql data warehouse:

```sql
SELECT * FROM businessintelligence_warehouse.products;
```

ProductID	ProductName	UnitCost	UnitPrice
1	Contoso 512MB MP3 Player E51 Silver	6.62	12.99
2	Contoso 512MB MP3 Player E51 Blue	6.62	12.99
3	Contoso 1G MP3 Player E100 White	7.40	14.52
4	Contoso 2G MP3 Player E200 Silver	11.00	21.57
5	Contoso 2G MP3 Player E200 Red	11.00	21.57

Figure 4.2.2 products_warehouse_table.sql sample

4.2.3 Sales Dataset

The Sales dataset was imported and cleaned by removing rows with missing customers or zero quantity, renaming and adjusting columns, filtering customers to match our customer list, randomly sampling 6000 sales, assigning random order dates across the years 2022, 2023, and 2024, linking each sale to the correct Product ID, reorganising the columns, and exporting the cleaned dataset into BusinessIntelligence_warehouse in Mysql.

```python
#importing dataset
sales = pd.read_csv('C:/Users/maxwe/Desktop/BI_Project/Sales_Dim.csv')

#dropping rows where "Quantity" value = 0
sales.drop(sales[sales['Quantity'] == 0].index, inplace=True)

#dropping rows where "Customer" value = NaN
sales = sales.dropna(subset=["CustomerName"])

#renaming "SalesOrderNumber" to "SalesOrderID"
sales = sales.rename(columns={"SalesOrderNumber": "SalesID", "CustomerKey": "CustomerID"})

#dropping rows where "CustomerKey" value <= 1000 in order to match the customers list in our Customers table
sales.drop(sales[sales['CustomerID'] > 1000].index, inplace=True)

#randomly sample 6000 rows from the Sales dataframe
sales = sales.sample(n=6000, random_state=42)

sales = sales.reset_index(drop=True)

#calculating Sales DataFrame of 6000 rows
total_rows = len(sales)

#calculating number of rows per year for 6000 in total
orders_in_2022 = int(total_rows * 0.39)
orders_in_2023 = int(total_rows * 0.37)
orders_in_2024 = total_rows - orders_in_2022 - orders_in_2023

#generating random dates for each year
dates_2022 = pd.to_datetime(np.random.choice(
    pd.date_range(start='2022-01-01', end='2022-12-31'), size=orders_in_2022))

dates_2023 = pd.to_datetime(np.random.choice(
    pd.date_range(start='2023-01-01', end='2023-12-31'), size=orders_in_2023))

dates_2024 = pd.to_datetime(np.random.choice(
    pd.date_range(start='2024-01-01', end='2024-12-30'), size=orders_in_2024))

#concatenating the dates
all_dates = np.concatenate([dates_2022, dates_2023, dates_2024])
np.random.shuffle(all_dates)

#setting a new column for the dates and sorting by OrderDate
sales['OrderDate'] = all_dates
sales = sales.sort_values('OrderDate').reset_index(drop=True)

#assigning a unique sales id for every order
sales['SalesID'] = range(1, len(sales) + 1)

#adding a ProductID column matching the IDs from each ProductName in the products table
sales = sales.merge(products[['ProductID', 'ProductName']], on='ProductName', how='inner')
print("Merged rows:", len(sales))

#moving 'ProductID' new column to the 4th position
cols = sales.columns.tolist()
cols.insert(3, cols.pop(cols.index('ProductID')))
sales = sales[cols]

#exporting clean dataset to Mysql workbench
engine = create_engine('mysql+mysqlconnector://root:password@localhost/BusinessIntelligence_warehouse')
products.to_sql(name='products', con=engine, if_exists='replace', index=False)
```

A sample of the cleaned sales table is shown below. It was exported from python to the Mysql data warehouse:

```sql
SELECT * FROM businessintelligence_warehouse.sales;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Fetch rows:

SalesID	SalesAmount	Quantity	ProductID	ProductName	CustomerID	CustomerName	OrderDate
1	14.36	1	1666	MGS Hand Games for Office worker L299 Yellow	995	Leah Hu	2022-01-01 00:00:00
2	56.10	1	1708	MGS Collector's M160	536	Devin Nelson	2022-01-01 00:00:00
3	7.99	1	1700	SV Hand Games women M40 Red	913	Rachael Sai	2022-01-01 00:00:00
4	296.40	1	1074	A. Datum SLR Camera M142 Orange	141	Javier Alvarez	2022-01-01 00:00:00
5	5.50	1	1679	MGS Hand Games for kids E300 Silver	368	Calvin Nara	2022-01-01 00:00:00
6	70.67	1	904	SV 40GB USB2.0 Portable Hard Disk E400 Silver	244	Robin Alvarez	2022-02-01 00:00:00
7	175.20	1	1577	SV DVD Recorder L200 Black	439	Jenny Nara	2022-02-01 00:00:00

Figure 4.2.3 sales_warehouse_table.sql sample

4.2.4 Returns Dataset

The Returns dataset was imported and cleaned by randomly selecting 1000 rows from the sales table to be used as the Returns table (using random state), adding Return ID column as the first column with a unique value for every row then saving the cleaned dataset into BusinessIntelligence_warehouse in MySQL.

```python
#importing dataset
returns = pd.read_csv('C:/Users/maxwe/Desktop/BI_Project/Returns_Dim.csv')
#randomly selecting 1000 rows from the sales table to be used as returns table (using random_state)
returns = sales.sample(n=1000, random_state=42)

#adding ReturnID as the 1st column
returns.insert(0, 'ReturnID', range(1, len(returns) + 1))

#exporting clean dataset to Mysql Workbench
engine = create_engine('mysql+mysqlconnector://root:password@localhost/BusinessIntelligence_warehouse')
returns.to_sql(name='returns', con=engine, if_exists='replace', index=False)
```

A sample of the cleaned returns table is shown below. It was exported from python to the Mysql data warehouse:

```sql
SELECT * FROM businessintelligence_warehouse.returns;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Fetch rows:

ReturnID	SalesID	SalesAmount	Quantity	ProductID	ProductName	CustomerID	CustomerName	OrderDate
1	1783	399.47	1	153	Adventure Works 26" 720p LCD HDTV M140 Silver	121	Beth Jiménez	2022-10-10 00:00:00
2	3918	422.97	1	153	Adventure Works 26" 720p LCD HDTV M140 Silver	443	Joseph Harris	2023-09-18 00:00:00
3	222	113.05	1	182	SV 16xDVD M320 Silver	791	Timothy Gonzalez	2022-04-02 00:00:00
4	2136	179.00	1	535	WWI LCD19W M100 White	867	Lucas Scott	2022-02-12 00:00:00
5	5225	13.51	1	1678	MGS Hand Games for Office worker L299 Red	429	Deanna Perez	2024-06-21 00:00:00
6	1169	39.64	1	896	SV Keyboard E90 White	862	Katherine Carter	2022-10-07 00:00:00
7	880	999.00	1	1831	Litware Washer & Dryer 15.5in E150 White	127	Shaun Carson	2022-05-23 00:00:00

Figure 4.2.4 returns_warehouse_table.sql sample

4.2.5 Stock Dataset

The Stock dataset was imported and cleaned by renaming columns, filtering products to match the product list in the Products table, grouping by product and aggregating stock quantities by summing and unit costs by averaging, adding a Stock ID column as the primary key, and then saving the cleaned dataset into BusinessIntelligence_warehouse in Mysql.

```python
#importing dataset
stock = DataFrame()
stock = pd.read_csv('C:/Users/maxwe/Desktop/BI_Project/Supplies_Dim.csv')

#renaming "ProductKey" to "ProductID" to "SafetyStockQuantity" to "StockQuantity"
stock = stock.rename(columns={"ProductKey": "ProductID", "SafetyStockQuantity": "StockQuantity"})

#dropping rows where "ProductKey" value <= 2517 in order to match the products list in our Products table
stock.drop(stock[stock['ProductID'] > 2517].index, inplace=True)

#grouping by "ProductKey" and aggregating "StockQuantity" by summing and "UnitCost" by mean value
stock = stock.groupby('ProductID', as_index=False).agg({
    'StockQuantity': 'sum',
    'UnitCost': 'mean'  # or 'first' is similar results
})

#adding StockID in the table as primary key
stock.insert(0, 'StockID', range(1, len(stock)+1))

#exporting clean dataset to Mysql workbench
engine = create_engine('mysql+mysqlconnector://root:password@localhost/BusinessIntelligence_warehouse')
stock.to_sql(name='stock', con=engine, if_exists='replace', index=False)
```

A sample of the cleaned stock table is shown below. It was exported from python to the Mysql data warehouse:

```sql
SELECT * FROM businessintelligence_warehouse.stock;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

StockID	ProductID	StockQuantity	UnitCost
1	1	41810	6.62
2	2	39842	6.62
3	3	40920	7.40
4	4	39028	11.00
5	5	39216	11.00
6	6	40850	11.00
7	7	38978	11.00
8	8	33472	30.58

Figure 4.2.5 stock_warehouse_table.sql sample

4.2.6 Expenses Dataset

The Expenses dataset was imported and cleaned by parsing the Date Key column to convert it to a datetime format, replacing years in the Date Key according to a mapping for 2007–2009, dropping unnecessary columns, and exporting the cleaned dataset into BusinessIntelligence_warehouse in Mysql.

```python
#importing dataset
expenses = pd.read_excel('C:/Users/maxwe/Desktop/BI_Project/Expenses_Dim.csv')

#parsing DateKey column to change from string to datetime value
expenses['Datekey'] = pd.to_datetime(expenses['Datekey'])

#replacing years in DateKey
expenses['Datekey'] = expenses['Datekey'].apply(lambda x: x.replace(year={
    2007: 2022,
    2008: 2023,
    2009: 2024
}.get(x.year, x.year)))

#droping unwanted columns
expenses = expenses.drop(columns=['ETLLoadID', 'LoadDate', 'UpdateDate'])

#exporting clean dataset to Mysql workbench
engine = create_engine('mysql+mysqlconnector://root:password@localhost/BusinessIntelligence_warehouse')
expenses.to_sql(name='expenses', con=engine, if_exists='replace', index=False)
```

A sample of the cleaned expenses table is shown below. It was exported from python to the Mysql data warehouse:

```sql
SELECT * FROM businessintelligence_warehouse.expenses;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Expense_ID	Department	Category	Year	Random_Boost1	Random_Boost2	Amount
1	Finance	Training	2022	269	128	1128
2	Sales	Training	2023	146	226	1872
3	Operation	Meals	2023	15	40	1555
4	IT	Travel	2023	146	-1	1645
5	Finance	Travel	2022	250	68	1068
6	Finance	Meals	2023	218	249	1967

Figure 4.2.6 expenses_warehouse_table.sql sample

5. Dashboards & Reports

In this chapter, we present the reports and dashboards created in Microsoft Power BI to answer the questions outlined in the project. The goal is to visualise key insights from the datasets and effectively communicate findings to the stakeholders.

5.1 Dataset Loading

This chapter is focused on importing the cleaned tables in Power BI from the SQL Server, creating relationships between the tables, and implementing the necessary transformations.

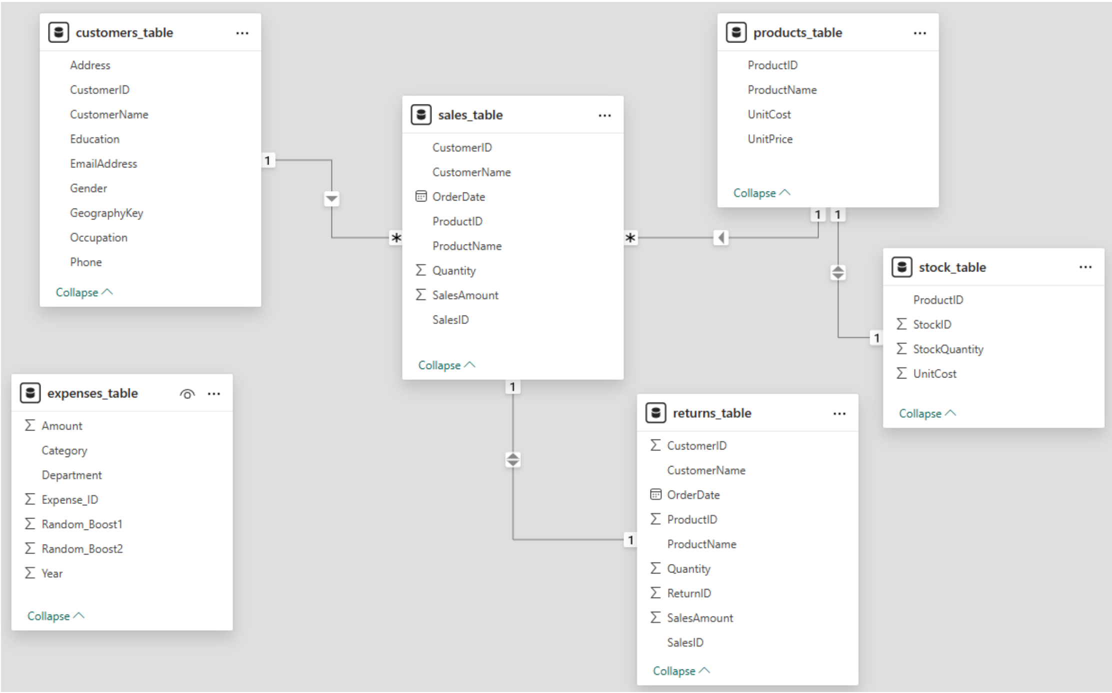


Figure 5.1: Initial structure and relationships of the model

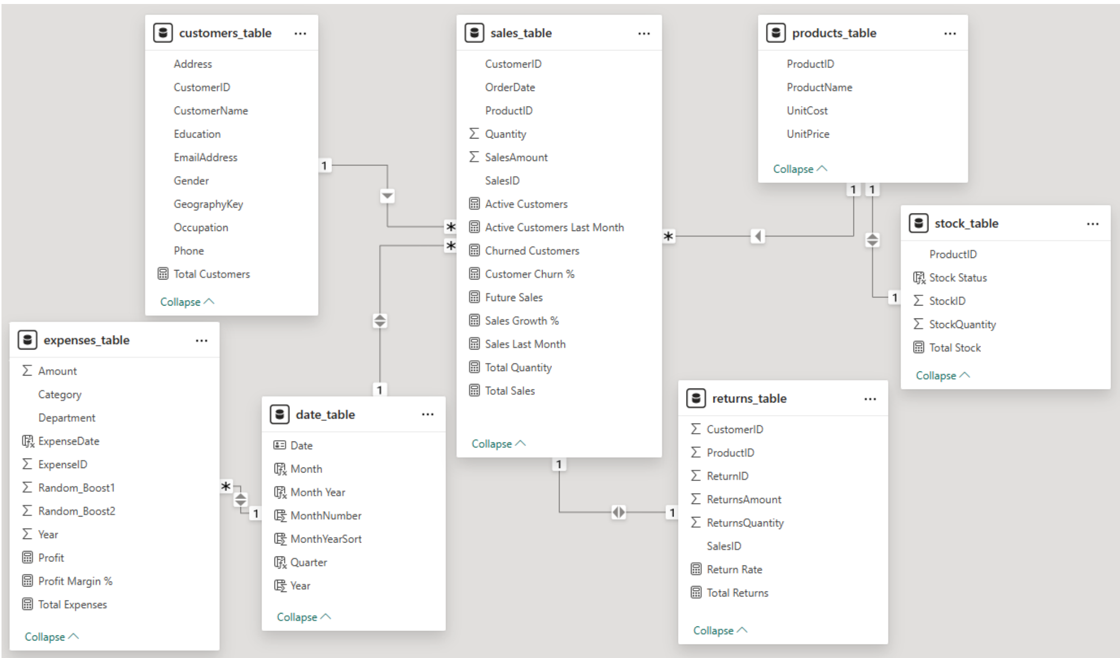


Figure 5.2: Final structure and relationships of the model

Between each table we created relationships using key IDs, a new Date table was also created, we removed duplicate columns, cleaned and standardised data and created new calculated columns and measures.

5.1.1 Sales Table

The Sales table initially contained the following transactional data: **CustomerID**, **CustomerName**, **OrderDate**, **ProductID**, **ProductName**, **Quantity**, **SalesAmount** and **SalesID**.

In the final model, the Sales table was improved with new calculated measures and columns to allow for a deeper business analysis.

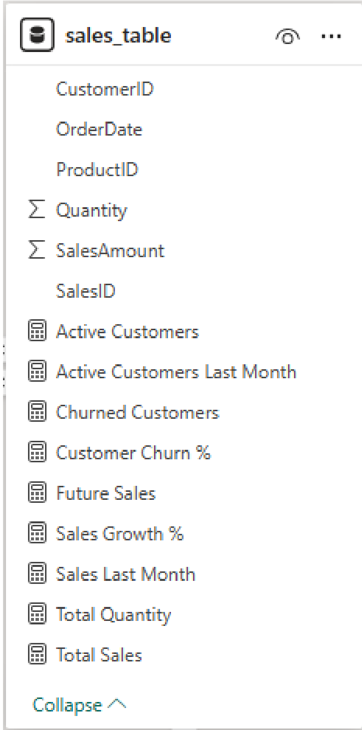


Figure 5.1.1 Sales Table

The following changes were made: Active Customers (Measure) counts the number of unique customers who made purchases in a selected period. Active Customers Last Month (Measure) counts customers from the previous month for churn analysis. Churned Customers (Measure) calculates the number of customers who stopped purchasing compared to the previous month. Customer Churn % (Measure) calculates the churn rate as a percentage of customers lost relative to the previous active customers. Future Sales (Measure) shows future sales based on historical data and trends (used for forecasting). Sales Growth % (Measure) measures the percentage change in sales compared to previous periods. Sales Last Month (Measure) tracks sales performance in the last month for comparison purposes. Total Quantity (Measure) summarises the total units of products sold. Total Sales (Measure) summarises the total sales amount over the selected period. Additionally, the Sales table was connected to the Date table via OrderDate, enabling time-based analysis like monthly patterns, time period comparisons, averages and forecasts.

5.1.2 Customers Table

The Customers table initially contained basic customer demographic and identification data: **Address**, **CustomerID**, **CustomerName**, **Education**, **EmailAddress**, **Gender**, **GeographyKey**, **Occupation,** and **Phone.**

In the final model, the Customers table was improved to allow for customer analysis and filtering based on demographics.

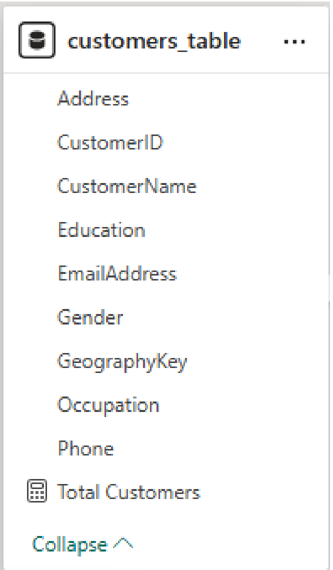


Figure 5.1.2 Customers Table

A new measure was created, **Total Customers** (Measure) to calculate the total number of unique customers in the dataset, used in KPIs and slicers. Additionally, a relationship to the Sales table was created where **CustomerID** from the Customers table was linked to **CustomerID** in the Sales table, allowing customer demographics to be connected directly to purchasing behavior. Similarly, a relationship to the Returns table was made so customer returns could be tracked by connecting the **CustomerID** fields. Customer fields such as **Gender**, **Education**, and **Occupation** were used to create dynamic slicers on the report pages, enabling users to analyse sales, returns, and churn by filtering customer groups.

5.1.3 Products Table

The Products table initially contained basic product-related information: **ProductID**, **ProductName**, **UnitCost**, and **UnitPrice**.

In the final model, even though there were no measures or calculations added, the Products table was improved to enable product profitability analysis and inventory management.

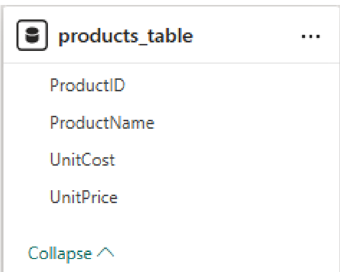


Figure 5.1.3 Products Table

A relationship was created between the Products table and the Sales, Returns, and Stock tables through the **ProductID** field, enabling easy integration of product sales, return rates, and stock levels. This relationship allows users to analyse product performance and profitability across different areas. Product fields like **ProductName** were used in slicers and visualisations to allow users to filter KPIs, sales, and returns based on specific products.

5.1.4 Stock Table

The Stock table initially contained inventory information: **ProductID**, **StockID**, **StockQuantity**, and **UnitCost**. In the final model, the Stock table was enhanced to support stock level tracking and improve inventory-related analysis.

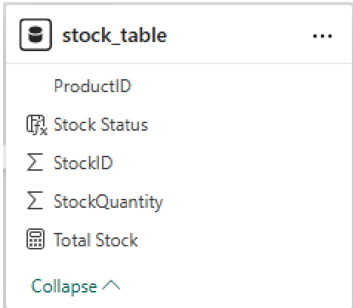


Figure 5.1.4 Stock Table

A relationship was created between the Stock table and the Products table via the **ProductID** field. This allowed stock levels to be linked directly to product information and sales performance. A new calculated column, **Stock Status**, was added to classify inventory based on stock levels (e.g., In Stock, Low Stock, Out of Stock). Additionally, a measure **Total Stock** (Measure) was created to sum the available stock quantity across all products. These additions allow users to easily monitor stock levels, identify low inventory products, and manage stock supply more effectively.

5.1.5 Returns Table

The Returns table initially included return-related data: **CustomerID**, **ProductID**, **ReturnID**, **SalesAmount**, **SalesID**, **Quantity**, and **OrderDate**.

In the final model, the Returns table was improved to enable return analysis and its formation with customer and product behaviours.

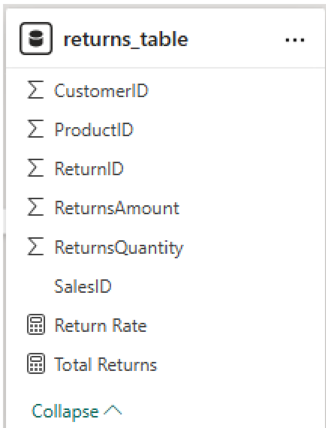


Figure 5.1.5 Returns Table

Relationships were established between the Returns table and the Sales table using the **SalesID** field, ensuring each return could be traced back to its original sale. Additionally, relationships were created between the Returns table and both the Customers and Products tables through **CustomerID** and **ProductID**, respectively. Additionally, new calculated columns and measures were added. **Total Returns** (Measure) to calculate the sum of returned sales amount. **Return Rate** (Measure) to calculate the percentage of sales that were returned, helping to monitor customer satisfaction levels and problematic products.

These improvements allowed the reporting to include return trends, return rates by customer group or product, and to analyse the impact of returns on overall profitability.

5.1.6 Expenses Table

The Expenses table initially contained operational expense data: **ExpenseID**, **Department**, **Category**, **Year**, **Random_Boost1**, **Random_Boost2**, and **Amount**.

In the final model, the Expenses table was reshaped to enable expense tracking and profitability analysis.

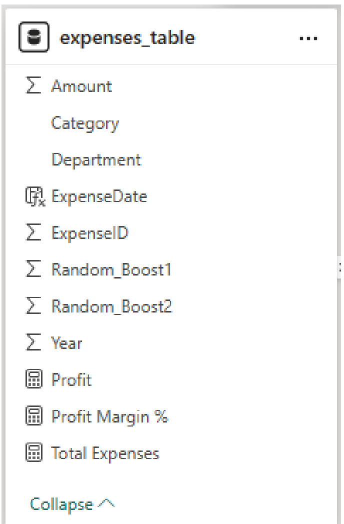


Figure 5.1.6 Expenses Table

A relationship was created between the Expenses table and the Date table via the **Year** field, aligning expense data with the same timeline used for sales and returns analysis.

A new measure **Total Expenses** (Measure) was created to calculate the total operational expenses, which was crucial for profitability calculations. The Department and Category fields were used in slicers and charts to allow users to break down expenses by department (e.g., HR, IT, Finance) and by expense category (e.g., Meals, Travel, Software). Finally, integrating the Expenses table allowed the dashboards to show net profit after deducting costs and gave deeper insights into operational spending patterns.

5.1.7 Date Table

The Date table is left last in this report, but it was a very important addition to our model. It was created to provide a consistent timeline for all time-based analysis across the model.

In the final model, the Date table role is to align sales, returns, and expenses chronologically for accurate reporting and forecasting.

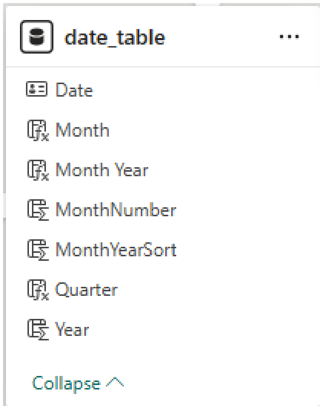


Figure 5.1.7 Date Table

The Date table was created using DAX to generate a continuous list of dates covering the range of the Sales and Expenses tables. Key columns such as **Year**, **Month Name**, **Month Number**, **Quarter** were added, from the **Date** column, to support flexible time-based slicing and aggregation. Relationships were established between the Date table and the Sales table using **OrderDate**. Between the Date table and the Returns table also using **OrderDate**. And between the Date table and the Expenses table using **Year**. This ensured that all facts (sales, returns, and expenses) could be analysed consistently across time.

The Date table also enabled time calculations, such as annual growth, moving averages, and forecasting future trends in sales and profit.

5.2 Reports & Dashboards

5.2.1 Sales Performance Over Time

This report page shows the overall sales trends over time, focusing on both Total Sales and Total Returns over time.

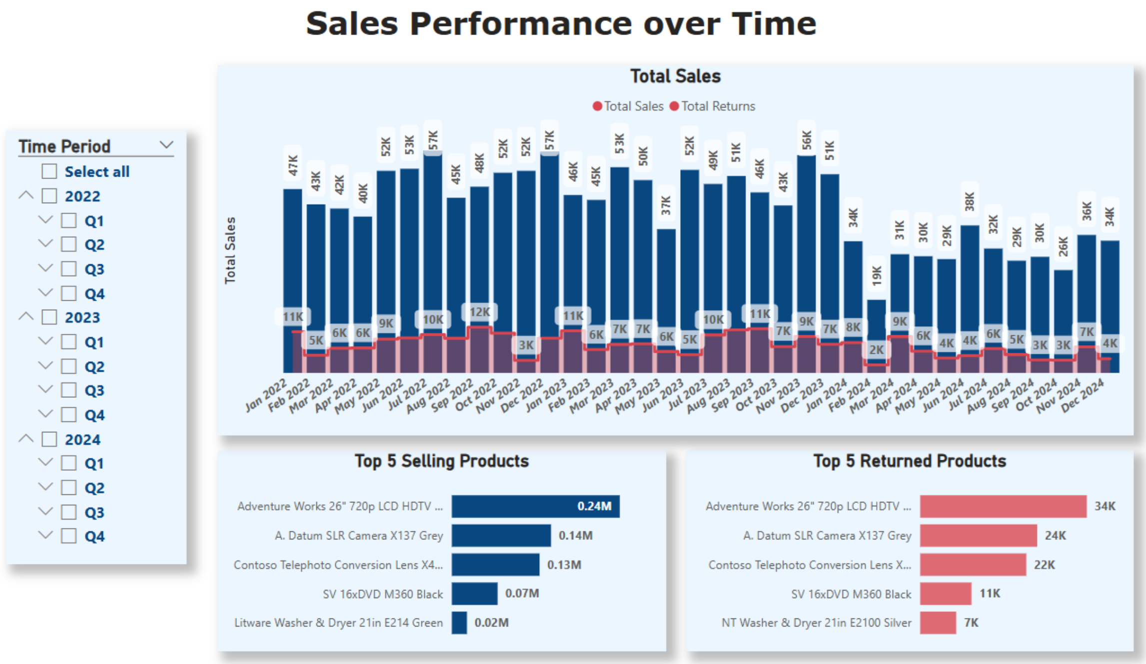


Figure 5.2.1 Sales Performance over Time Dashboard

The main clustered column chart shows the monthly Total Sales (blue bars) compared to Total Returns (red bars) from January 2022 to December 2024 (3 years period). The data labels help users identify the exact sales and returns values, to easily spot variations and patterns.

The slicer panel on the left lets users filter the data by Year, Quarter, or Month, enabling dynamic time-based analysis.

The two horizontal bar charts on the bottom, summarise product information. The “Top 5 Selling Products” present the products bringing the highest sales revenue. The “Top 5 Returned Products” show the products with the most customer returns, enabling return rate analysis.

With the help of this dashboard the users can track sales growth, identify high or low return times, and investigate the best-performing and most troubled products for the selected period.

5.2.2 Sales Performance per Customer Group

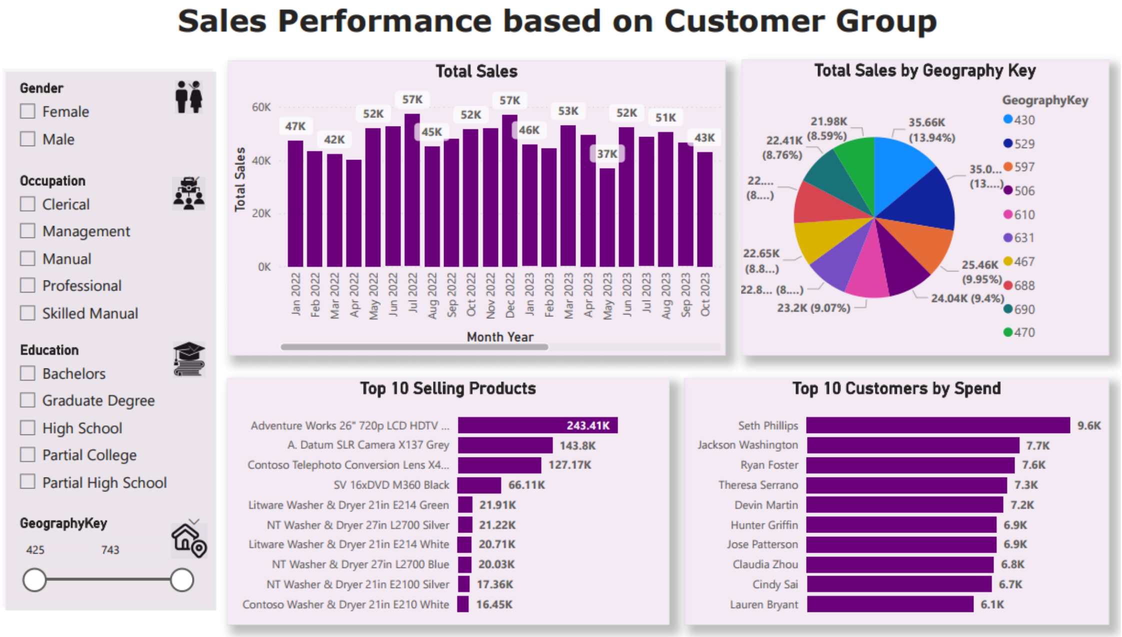


Figure 5.2.2 Sales Performance on Customer Group

The clustered column chart shows the monthly Total Sales (blue bars) compared to Total Returns (red bars) from January 2022 to December 2024 (3 years period). The pie chart presents the percentage of the total sales amount to each GeographyKey (Area Code).

The slicer panel on the left lets users filter the data by Year, Quarter, or Month, enabling dynamic time-based analysis.

The two horizontal bar charts on the bottom, summarise product information. The “Top 5 Selling Products” present the products bringing the highest sales revenue. The “Top 5 Returned Products” show the products with the most customer returns, enabling return rate analysis.

With the help of this dashboard the users can track sales growth, identify high or low return times, and investigate the best-performing and most troubled products for the selected period.

5.2.3 Customer Information

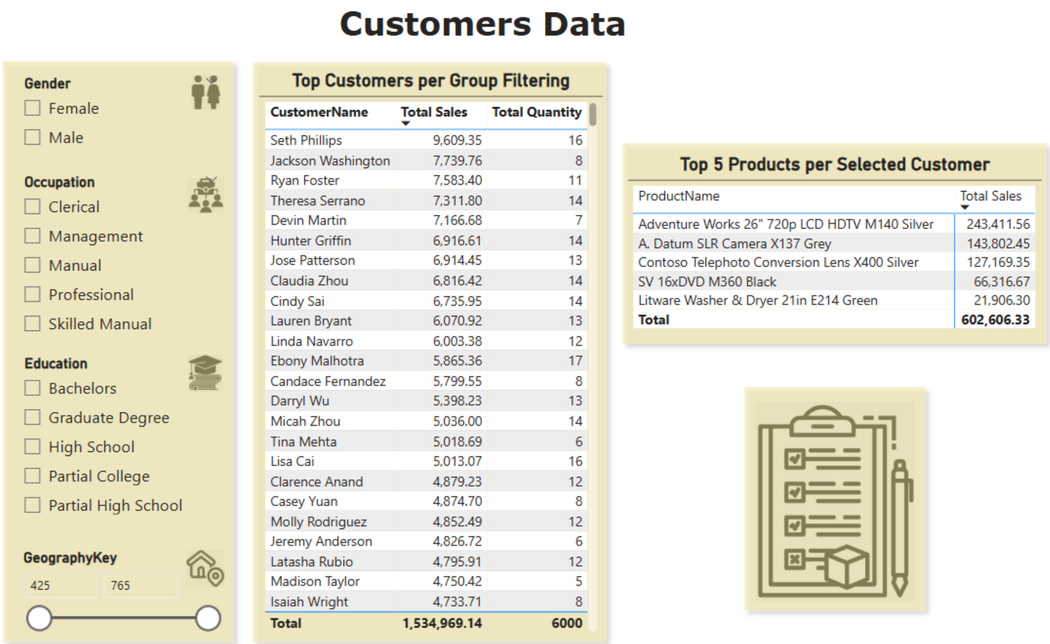


Figure 5.2.3 Customer Information

The slicer panel on the left lets users filter the data by customer group, Gender, Occupation, Education, or Area Code, enabling dynamic customer group-based analysis.

The table in the center shows a list of the top customers based on spend according to the filters in the slicer. The table on the right shows the top 5 products purchased by the selected customer in the cantered table.

With this dashboard the users have information on the products purchased by each customer personally. Personalised product preference will allow targeted customer campaigns such as advertisements and discounts.

5.2.4 Returns & Profitability

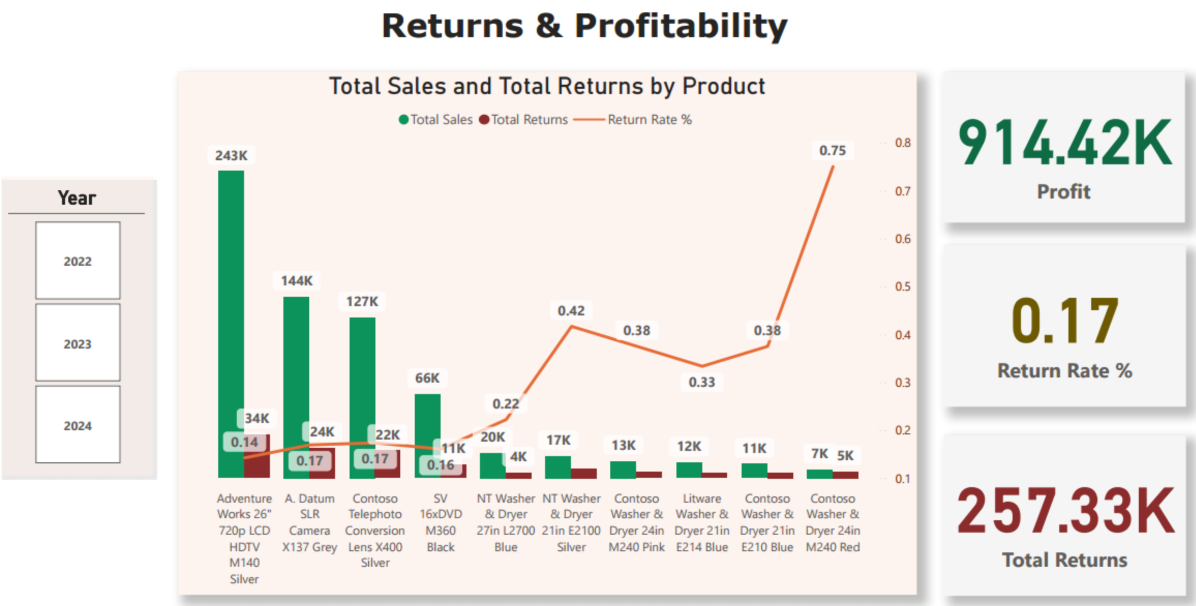


Figure 5.2.4 Returns & Profitability

The slicer panel on the left lets users filter the data by Year, enabling dynamic time-based analysis. The main clustered column chart provides visualisation of the top product's selling total amount (green column), the returned total amount (red column) for each year filtered in the slicer, and the return rate percentage (orange line).

The three card on the right give users a quick view of the Profit, the Return Rate and the Total Returns. The "Profit" card is calculated from expenses table measure: **Profit = [Total Sales] - [Total Expenses] - [Total Returns]**. The "Return Rate" card from the return table measure: **Return Rate = DIVIDE(SUM(returns_table[ReturnsQuantity]), SUM(sales_table[Quantity])).** The "Total Returns" card from the returns table again: **Return Rate = DIVIDE(SUM(returns_table[ReturnsQuantity]), SUM(sales_table[Quantity]))**

With this dashboard the users have information on the products sales and returns and their impact on the company's revenue.

5.2.5 Expenses & Profitability

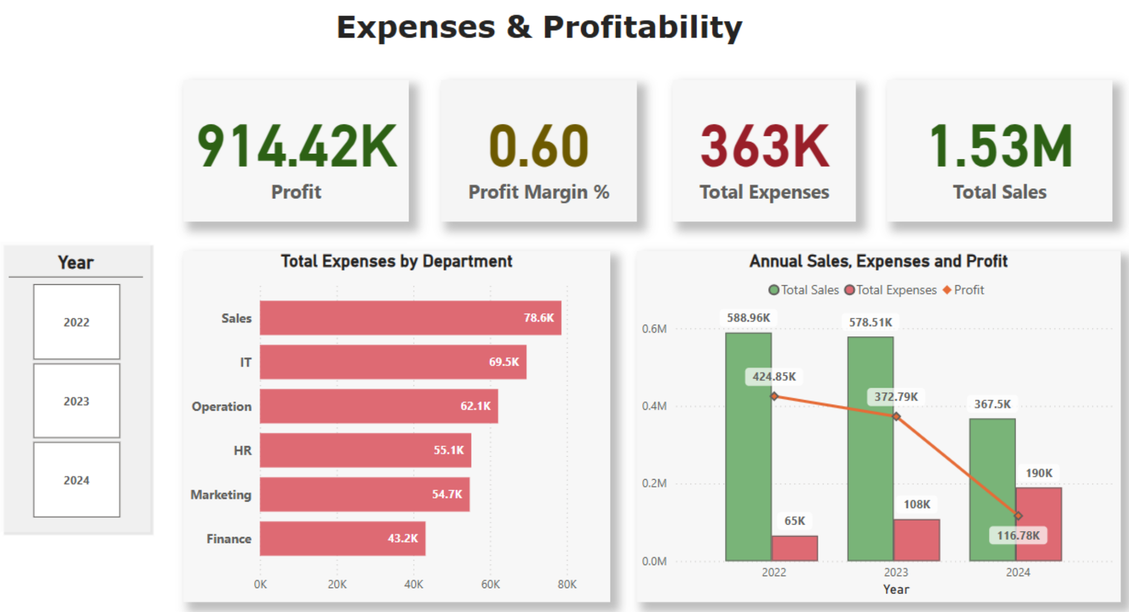


Figure 5.2.5 Expenses & Profitability

The slicer panel on the left lets users filter the data by Year, enabling dynamic time-based analysis.

The four cards on top give users a quick view of the Profit, Profit Margin percentage, Total Expenses, and Total Sales. The “Profit” card is calculated from the expenses table measure: **Profit = [Total Sales] - [Total Expenses] - [Total Returns]**. The “Profit Margin %” card is calculated from the expenses table measure: **Profit Margin % = DIVIDE([Profit], [Total Sales])**. The “Total Expenses” card from the expenses table measure: **Total Expenses = SUM(expenses_table[Amount]).** The “Total Sales” card from the sales table measure: **Total Sales = SUM(sales_table[SalesAmount])**

The clustered bar chart on the left shows the total expenses amount for each department. The line and clustered column chart on the right shows the total sales (green column) in comparison with the annual total expenses (red column) and the annual profit (orange line).

5.2.6 Company Departments

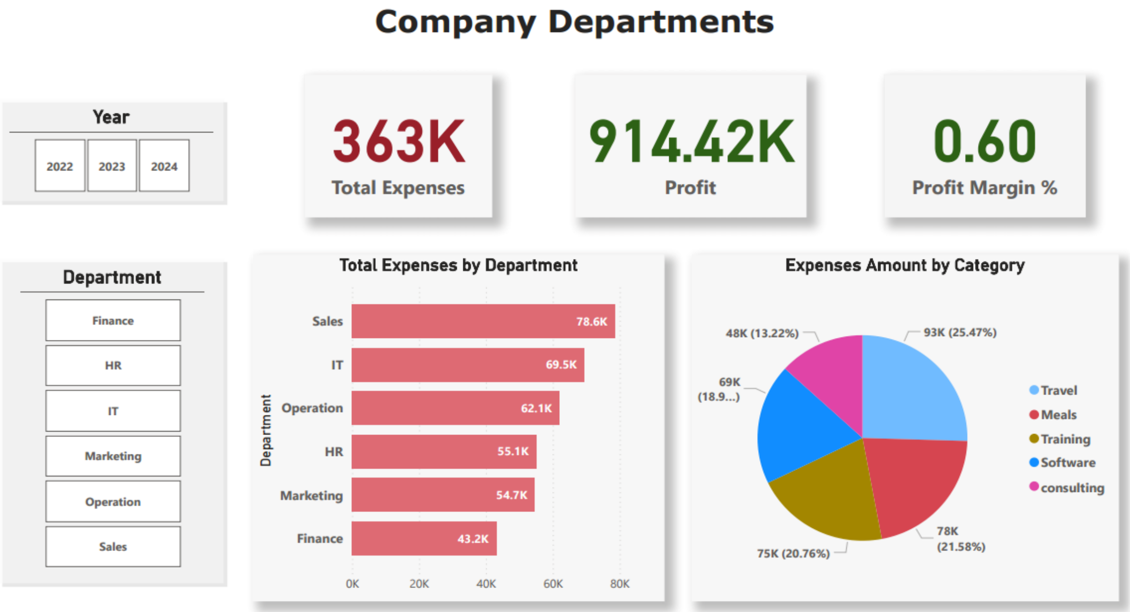


Figure 5.2.6 Company Departments

The slicer panels on the left lets users filter the data by Year, and by Department.

The three cards on top give users a quick view of the Total Expenses, Profit and Profit Margin percentage.

The clustered bar chart on the left shows the total expenses amount for each department. The pie chart shows the percentage of the Category of the expenses for each department.

5.2.7 Predictive Analysis

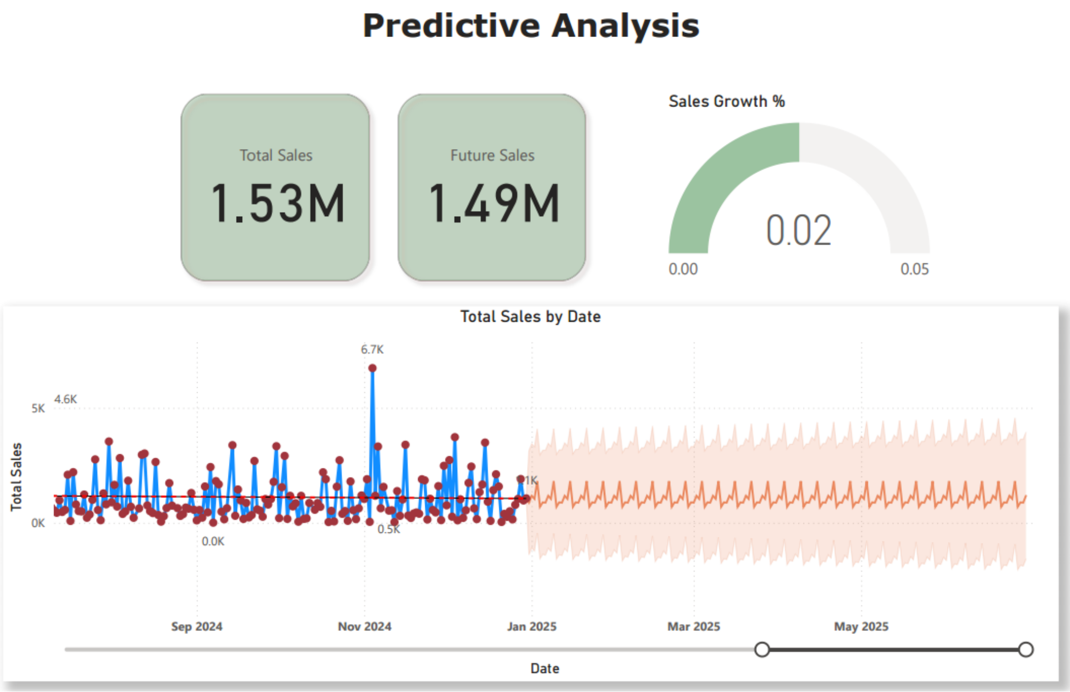


Figure 5.2.7 Predictive Analysis

The two cards present the Total Sales for the past three years (left) and the predictive future sales (right). The gauge on top right shows the sales growth percentage.

The line chart presents the future sales as estimated by the predictive analysis from Power BI.

5.2.7 KPI Overview

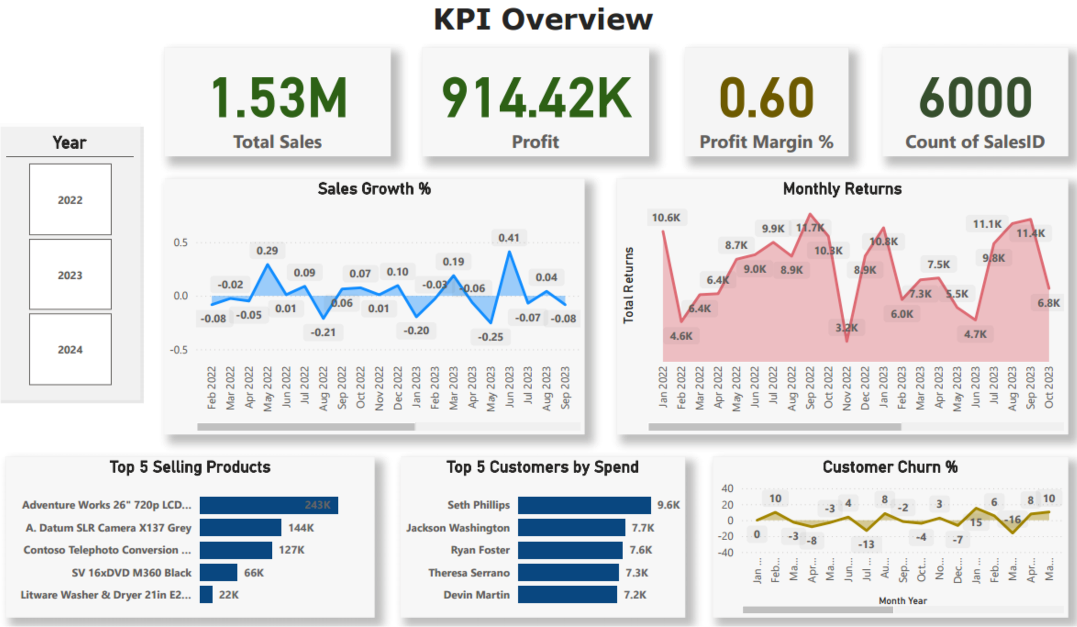


Figure 5.2.8 KPI Overview

The last dashboard includes cards four cards on top give users a quick view of the Total Sales, Profit, Profit Margin percentage, and Count of Sales (ID)

The line charts below show the Sales Growth percentage and the Monthly Returns.

The clustered bar charts on the bottom show the Top 5 Selling Products and the Top 5 Customers by Spend.

Lastly, the line chart on the bottom shows the Customer Churn percentage.

6. Conclusion

This project merged several datasets into a simplified, effective star schema. Created visual dashboards and dynamic KPIs to monitor sales, profit, customer behaviour and costs, and allowed for real-time filtering of temporal, departmental, and demographic data. Finally, it provided business stakeholders with a scalable, decision-supporting tool.

Power Bi visuals and cards have been crucial to derive comprehensive insights into the NickYasChar company's general performance:

KPIs

The KPI overview card on the KPI dashboard delivers a concise snapshot on key business metrics, including Total revenue, gross profit margin, return rate (%), Top product performance and Top customers. The sales figure reflects solid demand across many products but elevated high returns percentage for some items calls for concern. This may be due to quality, misleading description or issues delivery issues that are not under management's radar and needs to be investigated.

Profitability

The report shows a consistent decline in sales by year for the 3 years and therefore denotes an unhealthy state for the business. At the same time, total business expenses have been on the rise over the years. This may be why profit has been reducing.

Customer churn

This is the percentage loss of existing customers, and for this company has remained fairly constant between 7 to 15% on a monthly basis which is an indication that customers are loyal to the company. But measures such as launching customer programs to reduce this to a minimum.

Top selling product

Top selling products also show low return rate making them priority items for future promotion and investment.

Predicted future sales

Prediction shows a further decline in future sales, and it is therefore important to take proactive data-driven action to minimise the impact and maybe reverse the trend.

- Consider promotion and discount for some products to boost sales and minimize stock wastage.
- Explore new social media platforms to increase our presence. If local markets are already saturated. Consider expanding to other areas where we are not yet present.

References

Microsoft Contoso BI Demo Dataset for Retail Industry.: https://www.microsoft.com/en-us/download/details.aspx?id=18279

Microsoft Azure SQL Server: https://azure.microsoft.com/en-gb/explore/